Inteligencia Artificial

Informe Final: Car Sequencing Problem

Daniel Alejandro Martínez Castro

December 1, 2022

Evaluación

Mejoras 2da Entrega (10%):	_____
Código Fuente (10%):	_____
Representación (15%):	_____
Descripción del algoritmo (20%):	_____
Experimentos (10%):	_____
Resultados (10%):	_____
Conclusiones (20%):	_____
Bibliografía (5%):	_____
Nota Final (100):	_____

Abstract

Car Sequencing Problem (CS) es un problema en el que se ordenan autos en una línea de producción, con el fin de instalar opciones, estas se instalan en estaciones de trabajo las cuales tienen una capacidad definida, cada auto pertenece a una clase la cual tiene un set de opciones predefinido y una demanda. El objetivo de este problema para una instancia es encontrar una secuencia que sea factible o declarar la instancia como sin solución, aunque, existen enfoques que permiten que se presente como solución la que cometa menos errores. En este escrito se explica el problema en detalle para luego mostrar las diferentes aproximaciones que se han realizado a este problema con el pasar del tiempo, luego se expone un modelo matemático, después se implementa un algoritmo de resolución del problema usando backtracking para experimentar sobre el rendimiento de este algoritmo para después presentar los resultados de esta experimentación con elementos visuales concluyendo posteriormente.

1 Introducción

Car sequencing problem (CS) es un problema en el que se busca ordenar una cantidad de autos de diferentes clases en una línea de producción para satisfacer la demanda, cada auto tiene asociada una clase la cual tiene definida un set opciones que se le deben instalar en diferentes estaciones de trabajo, estas estaciones tienen una capacidad asociada.

El CS es considerado un CSP donde las soluciones son del tipo SI/NO. SI en el caso que se ha encontrado una solución posible que satisface las capacidades de las estaciones y No en el caso contrario, aunque también existen modelos que permiten que una restricción pueda ser violada en este caso se usa como solución la que tenga la menor cantidad de violaciones.

Este problema es de particular interés para la industria automotriz por sus amplias aplicaciones prácticas, debido a que la industria se encuentra en crecimiento constante y en esta ocurren cambios mayores cada muy poco tiempo, como lo son los cambios tecnológicos, además de que se manejan procesos muy complejos por lo que se hace necesario tener un método para poder organizar la producción y con esto satisfacer la demanda cambiante de la actualidad.

Se presenta un posible algoritmo para encontrar las soluciones del CS usando backtracking el cual corresponde a una técnica completa de busca en un espacio de búsqueda, este algoritmo instancia todas las posibles soluciones del problema para volver atrás si se violan restricciones, este algoritmo retorna la primera solución encontrada o la que menos violaciones tenga en el caso de que la instancia no tenga solución.

En este documento se encuentra estructurado siguiente manera, en la sección 2 se define el problema, explicando sus constantes, variables y restricciones. En la sección 3 se detalla el estado del arte en el cual se resume investigaciones realizada por múltiples investigadores, y sus enfoques en torno a resolver el problema. En la sección 4 se presenta el modelo matemático general en el cual se formalizan todos los aspectos del problema. En la sección 5 se presentan las representaciones de las soluciones usando un algoritmo basado en backtraking. En la sección 6 se detalla la implementación de este algoritmo, posteriormente se explican los procedimientos usados para la experimentación en la sección 7, para después exponer resultados de estos mediante tabla y gráficos, para finalmente concluir en la sección 9.

2 Definición del Problema

CS un problema en que se busca ordenar autos que tienen asociada una clase en una linea de producción de autos, en la cual se le deben instalar un set de opciones a cada clase de autos, para instalar estas opciones se tiene estaciones de trabajo, una por cada opción, las que tienen una capacidad definida. El objetivo de este problema es minimizar la cantidad de violaciones de capacidad de las estaciones satisfaciendo la demanda de cada una de las clases de autos [3].

Para la resolución de CS normalmente se usan 3 restricciones:

1. Satisfacer la demanda de todas las clases.
2. No sobrepasar la capacidad de una estación de trabajo en ninguna ventana de tiempo.
3. Asociar los autos que una clase en un slot definido con las opciones que se instalan en él.

La primera y tercera restricción son duras por lo que se deben cumplir en todas las instancias del problema, en cambio la segunda restricción es blanda por lo que puede ser violada, pero manteniendo estas violaciones en lo más mínimo posible.

Las constantes para la resolución de este problema consisten en el número de opciones, número de clases, cantidad de autos totales a fabricar, demanda de cada una de las clases, set de opciones que se le deben instalar a cada clase y capacidad de las estaciones de trabajo la cual se representa de manera porcentual por otro lado las variables de este problema consisten en una lista S en la cual cada slot representa una clase de auto que se fabricara en ese intervalo de tiempo, y una matriz X en la cual las columnas representan las opciones que se le instalan a un auto en un intervalo de tiempo.

Este problema se ha probado ser NP-duro por Kis [8], lo que significa que puede que no se pueda resolver con algoritmos en tiempo polinomial. El espacio de búsqueda corresponde a todas las posibles permutaciones de diferentes clases de vehículos, pero la dificultad no solo

depende del espacio de búsqueda, sino también de la tasa de uso de cada una clase de auto, debido a mientras más se use una clase de auto más difícil es que se cumpla con la restricción de capacidad de las estaciones de trabajo [9].

Existen múltiples variantes del CS, las cuales expanden el caso base usando demanda parcial incierta para resolver el problema, añadiendo así nuevas clases de autos con diferentes opciones a las ya definidas las cuales requieren estaciones de trabajo especializadas, el objetivo de este problema además de minimizar la cantidad de errores es también minimizar los costos que implica fabricar estos nuevos autos [1]. Otras variaciones de este problema implican añadir líneas de producción con lo cual además de los objetivos base del problema se añade el de balancear la carga en estas líneas.

3 Estado del Arte

El problema Car sequencing Problem (CS) fue definido por primera vez en 1986 por Parello et al. [10]. Este problema fue definido de la misma forma que se presenta en secciones anteriores a esta, las soluciones al problemna en este modelo consisten en encontrar un orden en los autos, en donde, todas las restricciones sean satisfechas o declarar la instancia como sin solución.

Como se expuso anteriormente este problema es NP-duro esto fue propuesto por Gent [5] y luego comprobado por Kis [8], por lo tanto esto significa que no exite un algoritmo que pueda solucionar el problema en tiempo polinomial.

En 1994 Hindi y Ploszaski [7] desarrollaron un algoritmo greedy, es estos algoritmos se inicia con una secuencia vacía y luego se le añaden autos iterativamente en relación con una función greedy, el algoritmo cual probó ser efectivo encontrando soluciones óptimas en tiempo bajos, además podía detectar que la instancia no tenía solución también en tiempos bajos en la mayoría de los casos. Pero para instancias grandes se concluyó que era necesaria la inclusión de algoritmos heurísticos auxiliares. Este trabajo cimentó las bases para las investigaciones futuras sobre el tema.

En 1995 Warwick et al.[13] propuso un Genetic Algorithm (GA) para resolver este problema. Estos consisten en técnicas que exploran el espacio de búsqueda mediante una simulación de evolución, esto se logra combinando estructuras de datos que obtuvieron buen rendimiento minimizando o maximizando la función objetivo, las soluciones de algoritmos GA convergen a valores cercanos a soluciones óptimas.

La mayoría de los trabajos que resuelven el CS usando Constraint Programing consideran el resultado final una solución o indican que la instancia no tiene solución ya que todas las restricciones se deben cumplir para que una solución candidata se considere solución final, por lo tanto, es un modelo exacto. En 2001 Bergen et al. [2] propuso un modelo que incluye tanto restricciones duras como restricciones blandas, el cual funcionaba bien con instancias pequeñas, pero se quedaba atrás con instancias más grandes o complejas. Desde este punto de vista después se desarrollaron algoritmos usando heurísticas de ordenamiento con el fin de reducir el espacio de búsqueda, estos acercamientos ponen las clases de autos más difícil o con más demanda al principio de la secuencia para así trabajar en base a ellos.

Solnon en 2001 [12] propuso un algoritmo Ant Colony Optimization (ACO) para CSP. En el cual se utiliza el comportamiento de las hormigas con el fin de encontrar una solución, estas hormigas forman una solución dejando un rastro de feromonas que luego iterativamente las hormigas posteriores siguen para mejorar la solución, estas feromonas eventualmente se evaporan. Este

algoritmo se ha mejorado con el tiempo incluyendo técnicas de Local Search y Greedy Heuristics [9].

Múltiples algoritmos de Local Search (LS) se han planteado algunos más centrados en CS que otros, esto algoritmos inician con una solución candidata y luego iterativamente se mueven hacia una solución vecina de la inicial, solución vecina es un set de soluciones potenciales que varían de la solución actual por lo mínimo posible un ejemplo de esto es Puchta y Gottlieb [11] en 2002. En 2003 Puchta y Gottlieb [6] compararon los rendimientos de algoritmos LS y ACO para este problema, en estas pruebas los algoritmos que usan ACO se mostraron levemente superiores a los que usan LS para instancias grandes del problema, pero los autores recalcan que con mejoras LS puede mejorar. En 2008 Fliedner and Boysen [4] desarrollaron un algoritmo Branch and Bound exacto, este explota los aspectos estructurales del CS para así reducir la complejidad del problema.

Las tendencias actuales sobre la investigación del CS más que continuar desarrollando técnicas más eficientes son de expandir el modelo para poder así usarlo en casos más específicos de CS, como lo son añadir múltiples líneas de producción o incluir demanda parcial o incierta en el modelo para así asimilarlo más a la realidad. También se están haciendo estudios prácticos que miden la utilidad de múltiples algoritmos que usa mezclas de heurísticas para así compararlos en estos entornos.

4 Modelo Matemático

Para resolver este problema se utiliza un modelo matemático que minimiza la cantidad de violaciones de capacidad de las estaciones de trabajo, para esto es necesario la restricción ligada a la capacidad sea blanda. Para la creación de este modelo se usó el trabajo de Dincbas [3] para el planteo de las restricciones, y el de Lin para la función objetivo [9]. Considerando las opciones o, número de autos a producir j y que existen k clases de autos.

4.1 Constantes

- $D[k]$ = demanda de autos de clase k.
- $P[i]$ = máximo número de autos que un bloque permite para cada opción i.
- $Q[i]$ = tamaño del bloque para cada opción i.
- $M[i][k] = 1$, si la clase k necesita la opción i. 0, caso contrario.
- C = {1, 2, ..., k}, O = {1, 2, ..., i}, N = {1, 2, ..., j}, n = total de autos a producir.

4.2 Variables

- $S[j]$ = clase de auto k asignado al slot j.
- $X[i][j] = 1$, si el auto clase k en en slot j tiene instalada la opción i. 0, caso contrario.

4.3 Función Objetivo

$$min \sum_{i \in O} \sum_{j \in N} X[i][j] \cdot max\{ \sum_{j'=j}^{min\{j+Q[i]-1,n\}} X[i][j'] - P[i], 0\} \quad (1)$$

Esta función objetivo minimiza la cantidad de violaciones en relación con la restricción de capacidad de las estaciones de trabajo. para esto se crea una ventana de tamaño $Q[i]$ la cual

se va moviendo por la línea de producción revisando que la cantidad de veces que se instala la opción i en esa ventana sea menor o igual a $P[i]$, para luego sumar todas las violaciones de cada una de las opciones.

4.4 Restriciones

$$\sum_{j'=j}^{min\{j+Q[i]-1,n\}} X[i][j'] \leq P[i], \forall j \in N, \forall i \in O \tag{2}$$

$$\sum_{i \in N} max\{-1 * (|S[i] - k| - 1), 0\} = D[k], \forall k \in C \tag{3}$$

$$M[i][S[j]] = X[i][j], \forall j \in N, \forall i \in O \tag{4}$$

La inecuación 2 corresponde a la restricción de capacidad, esta crea una ventana de largo $Q[i]$ en una de las opciones i de O, contando la cantidad de veces que se instala esta opción en esta ventana, esta cantidad tiene que ser menor o igual a $P[i]$ para así estar cumpliendo la capacidad de la máquina que instala la opción i, esta ventana se va moviendo hasta que no hay más autos, para luego revisar lo explicado anteriormente con las siguientes opciones. La ecuación 3 corresponde a la restricción que asegura que se cumpla la demanda para cada una de las clases, esta restriccoón básicamente cuenta la cantidad de ocurrencias de k en S, $\forall k \in C$, y revisa que sea igual a $D[k]$. La ecuación 4 representa la restricción de liga las opciones de un auto que se produce en el slot $S[j]$ a la matriz $X[i][j]$. Por lo tanto si un vehículo clase k se produce en un slot j y esa clase necesita la opción $M[i][k]$, esta opción debe estar en $X[i][j]$.

5 Representación

Para la resolución del problema se definieron estructuras idénticas a las señaladas en el modelo matemático del CS, las cuales son los arrays; S[nAutos]; D[nClases]; P[nOpciones]; Q[nOpciones]; M[nClases][nOpciones]; X[nAutos][nOpciones]. Los arrays D, P, Q y M son llenados con el archivo que contiene la instancia a solucionar, donde; D, representa la demanda de cada clase de auto; P, representa la cantidad de autos que se le puede instalar una opción en una ventana de tiempo; Q, representa el largo de la ventana; M, es la matriz que liga los requerimientos de opciones con sus respectivas clases. X se utiliza internamente para verificar la factibilidad de una instancia de S y S corresponde a la representación del problema esta corresponde a una lista de largo del número de autos a producir y esta se utiliza para guardar la combinación actual de autos que se está probando, finalmente esta es la estructura que se retorna en el caso de encontrar una solución.

La instanciación de variables dentro de S donde el orden de inserción es de menor a mayor índice en S y el orden de instanciación en cada slot de S es el mismo que aparece en el archivo que contiene la instancia del problema. Por ejemplo: si se tiene; $D[5] = \{1, 1, 2, 2, 2, 2\}$; $S[10] = \{0, 1, 2, 2, 3, 3\}$. La siguiente variable en instanciarse corresponde a S[7] con lo cual se tendría $S = \{0, 1, 2, 2, 3, 3, 0\}$, pero como esto no es factible ya que se trasgrede la restricción de demanda por lo que se continua instanciando hasta llegar a un número factible $S = \{0, 1, 2, 2, 3, 3, 4\}$ en este caso.

Como se puede notar con el párrafo anterior y lo dicho en secciones anteriores para esta representación la restricción tanto de Capacidad como de Unión son Duras, lo que quiere decir que bajo ningún motivo se pueden trasgredir, en cambio, la restricción de Capacidad es Blanda lo que permite guardar soluciones no optimas de una instancia, esto aumenta el espacio de

búsqueda en comparación a un algoritmo que retorne SI/NO si existe una solución factible.

El espacio de búsqueda de esta representación corresponde a exactamente lo comprobado por Kis [8] lo cual corresponde a:

$$\frac{|D|!}{|D[1]|! \cdot |D[2]|! \cdot ... \cdot |D[k]|!} \qquad k \in C \tag{5}$$

por lo tanto en el ejemplo anterior en el que $D = \{1, 1, 2, 2, 2, 2\}$ el espacio de busqueda corresponde a: $\frac{3628800}{1 \cdot 1 \cdot 2 \cdot 2 \cdot 2 \cdot 2} = \frac{3628800}{16} = 226800$. Como se puede observar en el numerador de (5) existe un factorial del la cantidad de autos total a producir por lo que el espacio de busqueda crece extremadamente rápido al aumentar la cantidad de autos a producir.

En el caso de que una instancia tenga múltiples soluciones al no tener forma de compararlas en términos de calidad, la solución elegida para como oficial es la primera que se encuentra para así reducir tiempo de computación. Y en el caso de que una instancia no tenga solución se entrega la secuencia de S que tenga la menor cantidad de violaciones de capacidad de las estaciones de trabajo.

6 Descripción del algoritmo

Para la implementación del algoritmo de resolución del CS utilizando backtracking se utilizará el lenguaje de programación C, debido a la simpleza de este lenguaje en comparación con otros y por contar con funcionalidades útiles en este algoritmo. La idea básica de este algoritmo es generar secuencias válidas de S que cumplan con la restricción de demanda para luego comprobar si cumple las otras restricciones, en el caso de que no se cumplan se vuelve atrás para cambiar secuencialmente variables de S anteriormente instanciadas, hasta recorrer todo el espacio de búsqueda o encontrar una solución.

Algorithm 1 Backtracking solver

```
1:  function BT_SOLVER(S, D, P, Q, profundidad, nAutos, nClases, nOpciones)
2:      if profundidad = nAutos then
3:          nViolations ← calculate_violations(S, P, Q, nAutos, nOpciones)
4:          if nViolations = 0 then
5:              solution ← S
6:              end execution
7:          end if
8:      else
9:          for i ← 0; i < nClases; i ← i + 1 do
10:             if D[i] ≠ 0 then
11:                 D[i] ← D[i] − 1
12:                 S[profundidad] ← i
13:                 BT_SOLVER(S, D, P, Q, profundidad + 1, nAutos, nClases, nOpciones)
14:                 D[i] ← D[i] + 1
15:             end if
16:         end for
17:     end if
18: end function
```

Como se puede ver en el seudocódigo del algoritmo se utiliza recursión para implementar el backtracking, lo que junto a los punteros en C permiten un manejo óptimo de memoria y

mínimos cambios de valor de variables dentro de las estructuras que se utilizan para probar la factibilidad de instancias. En cuanto al manejo de las restricciones en el código; para la primera restricción (Demanda) antes de instanciar una clase en un slot de la estructura S se verifica que exista demanda para esta clase así evitando generar instancias imposibles de S con lo cual se reduce la cantidad de verificaciones de esta restricción a futuro, esto se realiza en la línea 10 del seudocódigo con un simple *if*; por otro lado para las otras dos restricciones (Capacidad, Unión) estas se comprueban dentro de la función *calculate_violations()*, la cual construye la matriz X a partir de S y M cuando S se tiene todos los slots ocupados por una clase para después usar esta matriz para contar el número de violaciones de cada una de las estaciones de trabajo en esta instancia S, para finalmente sumar todas estas para obtener el total de violaciones de esta instancia S, se puede apreciar que cuando el número que retorna esta función es 0 significa que S es una instancia que no viola ninguna de las restricciones por lo tanto se retorna S como solución y se finaliza la ejecución del programa.

En este programa la instanciación de variables se realiza en el array S, cada slot representa una clase de auto a producirse en ese slot de tiempo, en cuanto a las clases de los autos estas se instancian en el mismo orden en el cual se especifica en el archivo de entrada, por otro lado en cuanto al orden de instanciación en el tiempo este se realiza de izquierda a derecha en el array S, en otras palabras, en slot que representa el momento mas temprano se instancia primero, para después instanciar el segundo más temprano, y así, hasta llenar el array S por completo.

Debido a la estructura de las soluciones se implementó un mecanismo para guardar una solución alternativa en el caso de que no existan soluciones de la instancia del problema, para esto cada vez que se verifica una instancia S con *calculate_violations()* se compara lo que retorna con el menor valor obtenido en verificaciones anteriores, si es menor esta instancia se guarda como la óptima, en el caso contrario no se hace nada. Esto no se expone en el seudocódigo ya que añade complejidad innecesaria.

7 Experimentos

En el Car Sequencing Problem existen 2 variables que afectan el espacio de búsqueda; por un lado, es la cantidad total de autos a producir; y por otro la cantidad de autos de cada clase que se van a producir. Por lo que para esta etapa la experimentación se enfocara en estas variables, principalmente la cantidad total de autos a producir ya que esta es la que más afecta el crecimiento del espacio de búsqueda en este problema.

Las instancias con las cuales se realizará la experimentación corresponden a instancias con cantidad de opciones y cantidad de clases fijas, siendo estas 5 y 6 respectivamente, con esto se ira variando la cantidad de total de autos a producir desde 10 a 17 en intervalos de 1, Los valores para P y Q son 1/2, 2/3, 1/3, 2/5, 1/5 para cada estación de trabajo, en la tabla 1 se presenta la relación entre clases y opciones usada por esa clase. El indicador de calidad que se ocupara para medir el rendimiento del algoritmo en estas instancias corresponde al tiempo de ejecución del programa, el cual corresponde al tiempo en el que el algoritmo encuentra una solución o revisa todo el espacio de búsqueda y retorna la combinación de autos con la menor cantidad de violaciones de capacidad de las estaciones de trabajo.

X	O_1	O_2	O_3	O_4	O_5
C_1	1	0	1	1	0
C_2	0	0	0	1	0
C_3	0	1	0	0	1
C_4	0	1	0	1	0
C_5	1	0	1	0	0
C_1	1	1	0	0	0

Table 1: Requerimientos de opciones de cada clase

La Demanda de cada de las clases y la cantidad de autos total de cada instancia se presentan en la siguiente tabla:

$nAutos$	nC_1	nC_2	nC_3	nC_4	nC_5	nC_6
10	1	1	2	2	2	2
11	1	2	2	2	2	2
12	2	2	2	2	2	2
13	2	2	2	2	2	3
14	2	2	2	2	3	3
15	2	2	2	3	3	3
16	2	2	3	3	3	3
17	2	3	3	3	3	3

Table 2: Demanda de cada clase y demanda total de autos para cada instancia

La razón por la que no se utilizaron las instancias de pruebas suministradas es debido a que el algoritmo de backtracking al ser una técnica completa esta debe recorrer todo el espacio de búsqueda, el cual corresponde a la ecuación 5, como se puede ver este espacio de búsqueda crece enormemente al aumentar la cantidad de autos a producir. Por lo que se decidió usar múltiples instancias pequeñas en las cuales el tiempo de ejecución de la instancia fuera razonable (máximo 12[h] o 43200[min]).

El hardware utilizado para el proceso de experimentación fue un procesador AMD Ryzen 5 5600X junto con 16[GB] de RAM a 2400MHz, el sistema operativo utilizado corresponde a Windows 10 en el cual se ejecutó una máquina virtual de Ubuntu 22.04.1 LTS utilizando wsl2.

8 Resultados

Las instancias especificadas en la sección anterior fueron ingresadas en el programa que implementa el algoritmo de resolución del car sequencing problem con backtracking, para obtener los resultados que se aprecian en la tabla 3.

nAutos	tiempo [s]
10	0.001
11	0.010
12	0.180
13	3.150
14	101.590
15	544.650
16	3122.750
17	14064.610

Table 3: Resultados de la experimentacion con las instacias

Como se puede ver con la instancia más pequeña el algoritmo soluciona la instancia muy rápido, pero este tiempo va creciendo rápidamente a medida que se aumenta la cantidad total de vehículos a producir, ya que, esta es la variable que esta relacionada con la explosión combinatoria en este problema como se ve en la ecuación 5. Como se puede ver solo aumentando 7 autos a la demanda total el tiempo que demora el algoritmo es 14064.610 [s] o 3.9[h], lo cual es un tiempo considerable para un aumento tan pequeño en la demanda total de autos. Debido a esto se decidió detener el aumento de vehículos en este punto ya que aumentar la demanda de vehículos en un vehículo más (18 autos) implicaría que tiempo de ejecución tomaría aproximadamente 5 veces mas que la última ejecución (17 autos) lo cual corresponde a 19[h] tiempo el cual transgrede las limitaciones propuestas en la sección anterior.

9 Conclusiones

A lo largo de este escrito se estudió el Car sequencing Problem (CS), problema de vital importancia para la industria manufacturera de automóviles ya que este permite optimizar el tiempo en el cual se fabrican autos lo que permite fabricar más autos y por lo tanto obtener más ganancias, se presentó una definición de este problema junto con todas las variables que influyen en la resolución de este, para después analizar la evolución de las técnicas para la resolución de este problema las cuales a un inicio tenían un enfoque general para problemas de satisfacción de restricciones, para después evolucionar a algoritmos más complejos con el uso de heurísticas que reducen el espacio de búsqueda con el fin de especializarse en este problema, más tarde se presentó un modelo matemático y finalmente una representación del problema con un algoritmo propuesto para la resolución de este problema.

A pesar de los fuertes avances a de finales de la década de los 90 e inicios de la primera década de los años 2000, el problema ha sufrido una disminución de los avances a partir finales de la década de los años 2000. Los focos principales de desarrollo a futuro son el uso de greedy algorithms suplementado con heurísticas como Local Search (LS) y Ant Colony Optimization (ACO), además de la focalización del problema para instancias más específicas del problema, haciéndolo así, más similar a las condiciones que se presentan en la realidad.

Se implemento una propuesta de solución en el lenguaje de programación C utilizando backtracking recursivo para la resolución del CS, haciendo las restricciones de demanda y unión duras, y la restricción de capacidad blandas. Todo esto para poder conseguir soluciones alternativas en caso de instancias del problema sin solución. Para instancias pequeñas de ordenes entre 10 y 30 autos a producir al algoritmo entrega resultados en tiempos aceptables, pero al aumentar la cantidad de autos a producir el tiempo de ejecución aumenta enormemente. Esto debido a la naturaleza del espacio de búsqueda este algoritmo es extremadamente ineficiente, ya que, incluso con instancias relativamente pequeñas con una cantidad total de autos a producir

de 100, el espacio de búsqueda es enorme, por lo que para instancias con cantidad de autos mayor a 30 no se recomienda usar esta técnica de resolución.

Después se experimentó con múltiples instancias en las cuales se variaba la cantidad de autos a producir de forma incremental, midiendo los tiempos de ejecución del programa creado. Con lo cual se puede concluir como ya se había demostrado antes, qué el uso de backtracking para la resolución de este problema es sumamente ineficiente, ya que el espacio de búsqueda crece de manera enorme al aumentar la cantidad de autos y que para instancias mas grandes es recomendable el uso de heurísticas para la resolución de este problema.

De cara a versiones futuras del algoritmo propuesto se tienen 2 opciones para mejorarlo; la primera opción sería eliminar que se guarde una solución alternativa en el caso de que una instancia no tenga soluciones haciendo con esto la restricción de capacidad dura, por lo tanto esta se revisaría cada vez que se instancia una variable con lo que las soluciones de una instancia serian de carácter SI/NO; La segunda es seguir con el modelo actual de guardar una solución alternativa pero revisando la restricción de capacidad de manera más frecuente así se lograría comparar instancias parciales que pueden tener más violaciones que otras ya revisadas, con lo que esta instancia se podría descartar y hacer backtracking inmediatamente.

10 Bibliografía

Indicando toda la información necesaria de acuerdo al tipo de documento revisado. Todas las referencias deben ser citadas en el documento.

References

[1] Joaquín Bautista-Valhondo, Manuel Chica, and Ignacio Moya. Car sequencing problem con flotas de vehículos especiales. presentación. 10 2016.

[2] Michael Bergen, Peter van Beek, and Tom Carchrae. Constraint-based vehicle assembly line sequencing. 2056:88–99, 06 2001.

[3] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car sequencing problem in constraint logic programming. *Proceedings of the European Conference on Artificial Intelligence*, pages 290–295, 08 1988.

[4] Malte Fliedner and Nils Boysen. Solving the car sequencing problem via branch & bound. *European Journal of Operational Research*, 191(3):1023–1042, 2008.

[5] Ian P. Gent and Toby Walsh. Csplib: A benchmark library for constraints. pages 480–481, 1999.

[6] Jens Gottlieb, Markus Puchta, and Christine Solnon. A study of greedy, local search and ant colony optimization approaches for car sequencing problems. *LNCS*, 2611, 06 2003.

[7] Khalil S. Hindi and Grzegorz Ploszajski. Formulation and solution of a selection and sequencing problem in car manufacture. *Computers & Industrial Engineering*, 26(1):203–211, 1994.

[8] Tamás Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32:331–335, 2004.

[9] Long Lin. A hybrid greedy algorithm for the car sequencing problem. *2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management*, pages 719–722, 2010.

[10] Bruce D. Parrello and Waldo C. Kabat. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2:1–42, 1986.

[11] M. Puchta and J. Gottlieb. Solving car sequencing problems by local optimization. *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, 2279:132–142, 01 2002.

[12] Christine Solnon. Solving permutation constraint satisfaction problems with artificial ants. 2000, 04 2001.

[13] Terry Warwick and Edward Tsang. Tackling car sequencing problems using a generic genetic algorithm. *Evolutionary Computation - EC*, 3:267–298, 1995.