Resumen

En una línea de montaje de productos mixtos, aquella que es responsable de la producción de diversas variaciones de un mismo producto (modelos), se presenta, entre otros, un problema de secuenciación consistente en obtener el orden en que la línea producirá los diferentes modelos. Este tipo de línea se conoce en la literatura como *mixed-model assembly line* y una de las formulaciones posibles de la secuenciación es un problema de optimización conocido como problema de Monden o ORV (del inglés, *output rate variation*). Este problema tiene como objetivo encontrar una secuencia de producción de los distintos modelos de manera que el consumo de piezas o componentes sea lo más regular posible, entendiendo por regular una secuencia de producción que diste lo menos posible de una secuencia ideal formada por modelos promedio.

Tanto el sistema de producción como el objetivo del problema de secuenciación resultan interesantes en sistemas just-in-time, en los que se pretende reducir, entre otros, los stocks intermedios y repartir cargas de trabajo de la forma más homogénea posible.

En este proyecto se propone la aplicación de un algoritmo de tipo Branch-and-bound para su resolución. Un procedimiento Branch-and-bound, o de enumeración implícita, consiste en la exploración de un árbol con todas las soluciones factibles que permite elegir la mejor entre ellas. Desarrollar todo el árbol de soluciones sería lento e ineficiente, por tanto se incluyen diversos procedimientos de cota y dominancia, que detectan cuándo una rama del árbol (formada por un conjunto de soluciones) no puede llevar a una solución mejor que la mejor conocida e impiden que ésta se siga explorando.

Para que el método exacto sea lo más eficiente posible, el procedimiento presentado incluye diversas cotas, tanto superiores como inferiores, y reglas de dominancia. Entre ellas, como cotas inferiores se destacan la asimilación del problema de secuencias regulares a un problema de transporte, así como una de las principales novedades de este trabajo respecto a procedimientos anteriores, que consiste en la separación del problema por opciones (uso de componentes) y la resolución exacta de los subproblemas obtenidos mediante programación dinámica. Otra novedad que aporta este trabajo son las reglas de dominancia presentadas, que incluyen un procedimiento que trata de encontrar mejores secuencias para la rama construida mediante una heurística *goal-chasing*, una estructura de memoria que almacena soluciones parciales, permitiendo detectar soluciones equivalentes a otras exploradas anteriormente, y reglas basadas en la explotación de la reversibilidad del problema.

Además se utiliza un procedimiento inicial que permite obtener una solución de partida. Este procedimiento está basado en un procedimiento de programación dinámica acotada. Éste resuelve el grafo asociado al problema ORV mediante programación dinámica, pero limitando el número de estados desarrollados por etapa a un ancho de ventana, convirtiéndose así en un procedimiento heurístico.

Los experimentos computacionales realizados muestran que el procedimiento presentado resuelve instancias de tamaños diversos: desde 10 hasta 200 productos a secuenciar, de 5 a 27 modelos o variaciones y de 3 a 9 opciones que caracterizan los distintos modelos. Además, puede solucionar problemas que incluyan modelos cuyo consumo para ciertas opciones no sea binario.

Los resultados obtenidos en pruebas de 1 hora con el algoritmo propuesto muestran que, para cinco colecciones de problemas usadas, con un total de 3442 instancias, se puede demostrar el óptimo en el 98.93% de los casos (para un total de 3405 instancias). También se ha demostrado la efectividad de las heurísticas implementadas, que son capaces de demostrar el óptimo para todas las instancias de tres de las cinco colecciones, y para las instancias más pequeñas dentro de las dos restantes.

Las cotas implementadas, una basada en la separación del problema por opciones y una por asimilación con un problema de transporte, también han demostrado su efectividad, dando valores cercanos a la solución óptima, especialmente para las instancias de mayor tamaño y menor número de opciones.

El proyecto presentado es un trabajo de investigación de tipo teórico, realizado según la normativa referente a los trabajos de fin de máster (TFM). Según el apartado 1.2 de la mencionada normativa el TFM debe contener los elementos característicos que sean pertinentes de un proyecto o trabajo científico teniendo en cuenta su carácter profesional, de investigación o mixto. Dada la naturaleza teórica del proyecto, se considera que los elementos característicos a desarrollar son: el estado del arte (antecedentes en la literatura), la definición de objetivos, alcance y limitaciones del proyecto, propuesta de resolución y metodología, tratamiento informático (implementación), resultados experimentales, análisis de resultados, conclusiones y bibliografía. Los apartados de análisis económico, planos, presupuesto, manuales de uso, impacto ambiental y pliego de condiciones se han omitido por no ser relevantes en un proyecto de esta índole.

Sumario

Glosario

A continuación, se presenta un glosario con la intención de aclarar la terminología empleada a lo largo de este documento:

A

Aportación: Se entiende por aportación el incremento en el valor de la función objetivo que supone la secuenciación de un modelo concreto en un instante, teniendo en cuenta la secuencia con la que se llega al instante.

B

***Backtrack*:** Acción de volver atrás una posición en una secuencia (solución parcial), para tratar de encontrar una secuencia mejor. En el árbol de exploración consiste en retornar al vértice padre y explorar alguno de los descendientes del padre; en caso de no existir descendientes, acceder al padre de éste y explorar alguno de sus sucesores y así sucesivamente.

BDP: (del inglés, *Bounded Dynamic Programming*), se trata de un procedimiento basado en resolver el problema por programación dinámica, limitando el número de estados a desarrollar en cada etapa del programa dinámico, a través de un parámetro, ancho de ventana *w,* y desarrollando únicamente los *w* estados con mejor cota. De este modo, el procedimiento es más rápido, pero en los casos en que el número de estados en alguna etapa supere el ancho de ventana, puede que no llegue demostrar la optimalidad de la solución, convirtiéndose en un procedimiento heurístico.

Branch-and-bound: Este algoritmo consiste en un proceso de enumeración implícita. Se trata de un procedimiento exacto empleado para resolver problemas de optimización. Consiste en generar un árbol de posibles soluciones, desarrollando (ramificando) aquellos vértices (soluciones parciales) que puedan conducir a la solución óptima, y no desarrollando (podando) los que no pueden llevar a ésta.

C

Código en paralelo: El código en paralelo, o la computación en paralelo, es una técnica de computación en la que se realizan cálculos de forma simultánea siempre y cuando se disponga de múltiples núcleos por procesador o múltiples procesadores. En el primer caso, que es el que se ha empleado en este trabajo, cada una de las tareas que deben procesarse en paralelo se conocen como hilos, o *threads*, que trabajarán de forma simultánea excepto en caso que haya peligro de sobrescribir información: si diversos *threads* comparten una o más variables, no debe permitirse leer y escribir sobre la variable a la vez, para evitar información errónea. Para ello, es necesario incluir cerrojos (*locks*) en las partes del programa que no puedan realizarse en paralelo. El tipo de programación en paralelo empleada en el presente trabajo se conoce como SPMD (*Single Process, Multiple Data*), ya que ejecuta el mismo proceso con diferentes datos simultáneamente mediante el uso de diversos hilos: uno por cada set de datos.

Composición (de una secuencia): La composición de una secuencia es un vector u de dimensión igual al número de modelos que considera el problema, donde cada componente u_i corresponde al número de unidades del modelo *i* que se han secuenciado hasta el instante *t*.

Cota inferior: En problemas de minimización, se considera cota inferior de un problema a cualquier medida que indique un valor mínimo que puede tomar una solución de ese problema. Cualquier solución, por lo tanto, debe de tener un valor igual o superior a cualquier cota inferior para una misma instancia de un problema. En caso de encontrarse una solución cuyo valor de la función objetivo coincida con el valor de la cota inferior más restrictiva (la mayor), se puede asegurar que la solución obtenida es óptima.

Cota superior: De manera similar a la cota inferior, en problemas de minimización una cota superior indica un valor máximo que puede tomar la solución óptima. Obviamente, el valor de cualquier solución es una cota superior, y cualquier solución de valor superior puede ser descartada.

D

Demanda (o plan de producción): En el problema tratado, se desean producir distintos modelos según una secuencia lo más regular posible. Como dato de partida del problema se requerirá, entre otros, la demanda o plan de producción para cada uno de los modelos, es decir, la cantidad de unidades que hay que producir de cada modelo.

E

Estado: Un estado queda definido por la producción acumulada hasta el instante actual, es decir, por la cantidad de cada uno de los modelos que han sido producidos hasta el momento. Esto puede representarse mediante un vector de tantas componentes como tipos (o modelos) a producir, en que cada una de estas componentes toma por valor la cantidad que se ha fabricado del modelo correspondiente. Por ejemplo, suponiendo un problema de dos modelos, considerando un estado asociado a la etapa 7 habiendo fabricado 5 unidades del primer modelo y 2 unidades del segundo, éste se puede representar con el vector [5,2].

Estado complementario: Para un cierto estado, definido por el instante en que se encuentra y la composición de la secuencia para cada modelo, puede definirse su estado complementario como el estado que corresponde al instante final menos el instante actual, y cuya composición es, para cada modelo, el plan de producción total para ese modelo, menos la producción acumulada hasta el instante actual (es decir, las unidades que faltan por secuenciar de cada modelo). Por ejemplo: suponiendo un problema de 2 modelos, que tienen una demanda de 7 y 8, respectivamente, y un estado en que se ha llegado a la etapa 4 habiendo secuenciado un modelo 1 y tres modelos 2 (es decir, con una composición u=[1,3]), el estado complementario de éste sería llegar a la etapa 11 con cualquier secuencia que cumpla la composición $\bar{u} =$ [6,5]. Esta definición se ve con más detalle en el apartado 4.1.

Etapa: En un programa dinámico se llama etapa a cada uno de los momentos en que es necesario tomar una decisión. Para cada etapa, hay diversas situaciones en que puede encontrarse el sistema, dependiendo de las decisiones tomadas anteriormente; a estas situaciones se las conoce como estados.

F

G

***Goal-chasing*:** El procedimiento *goal-chasing* es una heurística voraz que se emplea para la resolución del problema de secuencias regulares. Consiste en secuenciar en cada instante *t* el modelo *m* que genere una menor aportación a la función objetivo en ese instante. Se trata de un mecanismo voraz (*greedy*), ya que toma la decisión localmente óptima en cada instante, pero esto no suele llevar a una solución globalmente óptima.

H

I

Instante: Se llama instante a cada una de las divisiones temporales en las que se divide el problema. En el caso del programa dinámico coincide con las etapas, ya que es necesario tomar una decisión en cada instante, y tiene que haber tantas etapas como instantes, ya que es necesario considerar una etapa inicial.

J

Just in Time: El Just in Time (JiT) es una estrategia de producción centrada en reducir los stocks y buffers intermedios. El problema de secuencias regulares se presenta de manera destacable en las industrias que emplean este sistema.

K

L

Línea de montaje: Una línea de montaje es una manera de organizar la producción industrial en continuo. Consiste en una cinta transportadora o similar, que transporta el producto inacabado o las piezas, pasando por diversas estaciones de trabajo, donde se realiza una parte del proceso de producción.

Línea de montaje *mixed-model:* Supóngase un sistema de producción en continuo que fabrica un mismo producto no homogéneo, con distintos modelos que consisten en una o varias modificaciones del producto básico. Si las diferencias entre distintos modelos son significativas, pero no lo suficientemente grandes como para suponer cambios estructurales en la línea de montaje, entonces todos los modelos pueden ser producidos en una misma línea. Si además, los modelos son tan similares que pueden ser producidos de forma alterna sin requerir tiempo de

preparación entre un modelo y otro distinto, la línea en que se producen se conoce como línea de montaje *mixed-model*.

M

Modelo: Variación del mismo producto definida por el uso o consumo de una serie de opciones (que podrían considerarse piezas que pueden incluirse en distintas cantidades para distintos modelos).

N

Ñ

O

Opciones: En el problema tratado, la regularidad de una secuencia se valora en relación a la regularidad en el uso o consumo de opciones. El uso de cada una de las opciones es también lo que diferencia los distintos modelos. Podría considerarse que las opciones son piezas que distintos modelos usan en distintas cantidades, de manera que el objetivo es encontrar una secuencia que regularice el consumo de piezas.

ORV: El problema de secuencias regulares, u ORV (del inglés *Output Rate Variation*), es un problema de secuenciación en que pretende encontrarse el orden óptimo en el que secuenciar distintos modelos en una línea de montaje *mixed-model*, con el objetivo de que esta secuencia sea lo más regular posible. Esta regularidad viene referida, en este problema, al uso que hace cada modelo de distintas opciones (por ejemplo, piezas que varían de un modelo a otro), de manera que el objetivo será que la suma de consumos de estas opciones en cada instante de tiempo se aleje lo menos posible de la tasa ideal acumulada de cada opción en cada instante.

P

Plan de producción: véase *Demanda*.

Producto: En el contexto industrial, se entiende como producto el resultado de la transformación de ciertas materias primas, que mediante esta transformación adquiere un cierto valor añadido. En este trabajo se considera que el producto tratado puede dividirse en diversos modelos, según algunas modificaciones respecto al producto base.

Programación dinámica: La programación dinámica es una técnica de resolución que consiste en dividir el problema en problemas idénticos de menor tamaño. Se basa en calcular las soluciones óptimas de estos subproblemas y, sabiendo la solución óptima de todos los subproblemas, encontrar la solución óptima del problema global.

Q

R

S

Secuencia (de una solución): Una secuencia es un vector de dimensión igual al número de instantes (*T*, en caso de ser una secuencia solución; si corresponde a un número de etapas menor, se trata de una subsecuencia), donde cada componente *t* tiene el valor correspondiente al modelo a fabricar en el instante *t*.

Secuencia simétrica: Una secuencia A se dice que es simétrica a otra, B, si el primer modelo de la secuencia A es el último de la secuencia B, el segundo modelo de A es el penúltimo de la secuencia B, y así repetidamente para todos los instantes de tiempo.

Sdq (del inglés, *Squared Deviation*): Esta es otra forma de llamar al valor de la función objetivo: representa el sumatorio de las diferencias entre los acumulados de producción de las opciones y las tasas ideales acumuladas. Para evitar valores negativos, cada una de estas diferencias se elevará al cuadrado.

Subsecuencia: Una subsecuencia se representa mediante un vector de dimensión *t*, asociado a una etapa *t* ($1 \leq t < T$). En este vector se anota, para cada instante, el modelo que se ha secuenciado en ese instante. Es importante destacar la diferencia entre una secuencia y su composición, ya que la composición solamente indica el número de unidades de cada modelo que se ha realizado hasta el momento.

T

Tasa ideal: Corresponde a la utilización de opciones de un modelo promedio. Esta tasa se calcula para cada opción, como el consumo total de esa opción (sumando el consumo de esa opción por parte de cada modelo, multiplicado por la demanda del modelo), dividido entre el número total de instantes.

Tasa ideal acumulada: Cantidad que idealmente debería haberse consumido de cada una de las opciones en un instante de tiempo. Se obtiene multiplicando la tasa ideal de la opción por el instante de tiempo.

Tipo: véase *Modelo.*

U

V

W

X

Y

Z

Nomenclatura y notación

En este apartado se resume la notación matemática empleada a lo largo de este trabajo. Alguna nomenclatura concreta que se emplea solamente en el apartado correspondiente se presenta con detalle en el apartado que la concierne.

Un conjunto de elementos (que pueden no estar ordenados) $e_1, e_2, e_3, \dots$ se representa $E = \{e_1, e_2, e_3, \dots\}$ y su cardinalidad se anota $|E|$. En este trabajo se emplearán también símbolos de teoría de conjuntos: conjunto vacío (∅), unión de conjuntos (∪), intersección de conjuntos (∩), complemento de un conjunto (\) y pertenencia a un conjunto (∈).

Una matriz de dos dimensiones *m*x*n* se representa mediante una letra mayúscula, mientras que sus componentes se representan por la misma letra en minúsculas con el número de fila en el primer subíndice y el número de columna en el segundo subíndice:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

De manera similar, un vector *x* de dimensión *n* se anota como $x = [x_1, x_2, \dots, x_n]$, y un vector que se represente como $\bar{x}$, corresponde a un estado complementario del vector *x* (ver *Estado complementario* en el Glosario).

Considerando **R** el conjunto de los números reales y **Z** el conjunto de los números naturales, también cabe tener en cuenta las siguientes definiciones:

$\lfloor z \rfloor$ representa el entero por defecto de *z*, definido como $\max\{k \in \mathbf{Z} : k \leq z\}$, siempre que $x \in \mathbf{R}$.

$\lceil z \rceil$ representa el entero por exceso de *z*, definido como $\min\{k \in \mathbf{Z} : k \geq z\}$, siempre que $x \in \mathbf{R}$.

Un grafo *G* queda definido por un conjunto de vértices $V=\{v_1, v_2, \dots\}$ y una serie de subconjuntos de dos vértices llamados arcos. Un camino (*w*) de longitud *k* en *G* es una secuencia de vértices conectados por arcos: $w=[v_1, v_2, \dots, v_k]$. El camino es cerrado si $k>1$ y $v_k=v_1$. Un camino cerrado cuyos únicos vértices repetidos son el primero y el último, se conoce como circuito o ciclo.

Antecedentes

Actualmente, los consumidores disponen de múltiples variantes para un producto, que se producen en una misma cadena de montaje con el objetivo de minimizar costes. Estas líneas mixed-model no sólo presentan un problema de equilibrado de líneas, véase Thomopoulos (1967), sino que además debe decidirse la secuencia de producción de los distintos modelos. Suponiendo un tiempo de ciclo conocido (sea dado o calculado mediante un método de equilibrado), el orden en que se producen los modelos afecta no sólo a la carga de trabajo por estación de la línea, sino también a la salida de los diversos productos, las tasas de consumo de los componentes y los stocks intermedios.

La secuencia óptima para este tipo de líneas dependerá del objetivo perseguido: por ejemplo, reducir los costes de preparación, reducir los costes de capacidad, reducir los buffers intermedios, regularizar el consumo de componentes, utilizar las estaciones de trabajo de forma eficiente, reducir el tamaño de la línea, etc. Estos objetivos, y otros que se han estudiado en la literatura, pueden encontrarse referenciados en los estudios de Yano y Bolat (1989), Ghosh y Gagnon (1989) y Kubiak (1993).

El presente trabajo tiene como objetivo estudiar la secuencia que regularice el consumo de opciones, de manera que el suministro de componentes sea regular y pueda gestionarse JiT. Esto minimiza los stocks de seguridad o sobredimensionamientos en el abastecimiento. Este problema se conoce como ORV (Output Rate Variation), según Kubiak y Sethi (1991), o problema de Monden (1983), que lo plantea originalmente como método de secuenciación en el contexto del sistema de fabricación de Toyota.

El planteamiento del problema es similar al del Car Sequencing Problem, ver Parrello et al. (1986), que también busca la secuencia de fabricación para un producto que considera diversas opciones, pero que tiene en cuenta otras restricciones tecnológicas, correspondientes al número máximo de veces que puede aplicarse cada opción en un cierto periodo de tiempo. En el CSP se pretende comprobar la factibilidad de la instancia dadas estas restricciones. Otro problema similar es el PRV (Product Rate Variation), véase Miltenburg (1989), cuya única diferencia con el problema que se plantea en este trabajo es que el objetivo no es regularizar el consumo de componentes, sino regularizar la producción de los distintos modelos.

La literatura que ha estudiado este problema se ha centrado en procedimientos heurísticos: Monden (1983) propuso dos métodos de secuenciación *goal-chasing* usados por la Toyota Motor Corporation: GC1 y GC2. Las heurísticas *goal-chasing* son procedimientos heurísticos voraces, que secuencian según una regla de prioridad.

Por su parte, Miltenburg (1989) propuso tres procedimientos de secuenciación (M-A1, M-A3H1 y M-A3H2). El primero de ellos considera las etapas por separado y encuentra el mejor modelo a secuenciar en cada una de ellas (el resultado final puede dar una secuencia infactible, en cuyo caso puede usarse el valor obtenido como cota inferior). La segunda y la tercera aplican M-A1, para posteriormente revisar la solución y así dar una secuencia válida.

Sumichrast y Russell (1990) desarrollaron un procedimiento de programación entera mixta capaz de encontrar la secuencia óptima para el problema ORV, y en su estudio compararon los resultados de éste con los de 5 heurísticas ya existentes. Sin embargo, a pesar de que para los problemas pequeños (se considera una instancia pequeña la que no supera las 20 unidades totales por secuenciar) el procedimiento exacto encontraba la secuencia óptima, resultaba ser demasiado lento para su aplicación en problemas de tamaño real.

En este mismo estudio, las cinco heurísticas expuestas se probaron y compararon, llegándose a la conclusión de que la heurística M-A3H2 es la que obtiene mejores resultados para las instancias más grandes, aunque las heurísticas más sencillas trabajan de manera más rápida.

Bautista et al.(1996), proponen un procedimiento exacto en el que el problema se resuelve mediante un procedimiento de programación dinámica acotado (BDP), que da soluciones óptimas para las instancias de tamaño pequeño, y puede aplicarse como un procedimiento heurístico, dando soluciones buenas para instancias de mayor tamaño, como puede comprobarse en el presente trabajo.

Otro enfoque heurístico fue propuesto por Leu, Huang y Russell (1997), y posteriormente por Erel et al. (2007). Su aportación fue un procedimiento de beam search que, a pesar de que no podía demostrar optimalidad para el ORV, superaba los resultados obtenidos por las heurísticas *goal-chasing*. Jin y Wu (2002) proponen otro método heurístico que mejora los métodos *goal-chasing* con una heurística menos voraz.

A diferencia de estos, el procedimiento que se propone en este trabajo es un algoritmo exacto. Inicialmente, utiliza un *goal-chasing* y la BDP desarrollada por Bautista, Companys y Corominas (1996) para obtener una cota superior (una solución factible), calcula diversas cotas inferiores, una de ellas ya conocida (la que asimila el problema de ORV a un problema de transporte), y otra que no se había visto previamente en la literatura (en la que separa el problema en subproblemas según opciones utilizadas y se resuelven mediante programación dinámica). Tras esto, se inicia un procedimiento de branch-and-bound que combina una estrategia depth-first con una best-first, y que utiliza como mecanismos de poda las cotas mencionadas, así como diversas reglas de dominancia basadas en simetría y la formulación de programación dinámica asociada al problema.

El algoritmo presentado obtiene resultados muy buenos, que superan los resultados publicados anteriormente, en las pruebas computacionales, en los cuales destaca su capacidad de resolver de forma óptima la totalidad del juego de instancias de la literatura para consumos no binarios. En general, para todas las colecciones de instancias usadas, es capaz de resolver de forma óptima 3405 de 3442 instancias.

1. Introducción

En este apartado, se definen los conceptos básicos a conocer para presentar el problema, así como los métodos de resolución que se emplean en este trabajo. Con esto se pretende dar una visión general del problema y las estrategias que se aplican para su resolución.

1.1. Líneas de montaje mixed-model

Actualmente, los productos de gran consumo se producen en su gran mayoría en líneas de montaje, en las cuales la producción es homogénea. A pesar de esto, hay algunos productos (por ejemplo, los automóviles), en los que los consumidores pueden elegir entre una gran cantidad de variantes (por ejemplo, con o sin aire acondicionado, con o sin elevalunas eléctrico, etc.).

Para producir este tipo de modelos con modificaciones en un entorno industrial a gran escala sin incurrir en los gastos que supondría construir diversas líneas de montaje, todos los modelos se producen en una misma línea. Esto puede hacerse de dos maneras: con una línea de montaje multimodelo o con una línea de montaje mixed-model.

Con una línea mutimodelo se producen los diferentes modelos por lotes, ya que las diferencias entre los modelos son lo suficiente significativas como para que sea necesario reconfigurar la línea tras cada uno de estos lotes. En estas líneas no se puede intercalar la fabricación de diferentes modelos, ya que el tiempo de preparación entre un modelo y otro es demasiado elevado.

En el caso de las líneas de montaje mixed-model se supone que los modelos son lo suficientemente similares como para no necesitar tiempo de preparación (o este existe, pero resulta negligible frente al tiempo de proceso del producto) y pueden ser producidos de forma intercalada, sin preparación entre un modelo y otro. Esto suele plantearse como la producción de un modelo básico al que se le añaden o no una serie de opciones o partes, de forma que cada tipo de modelo a producir queda definido por el uso que tiene de cada una de las opciones disponibles. Este trabajo se centra en este caso y en la secuenciación regular para las líneas mixed-model, que se conoce en la literatura como Monden Problem o ORV (Output Rate Variation).

1.2. Definición del problema de secuenciación regular (ORV)

Supongamos una cadena de montaje con 3 modelos a producir y un plan de producción (d_i) igual a las unidades que deben producirse en el horizonte temporal que se estudia (por ejemplo,

en un día). Cada producto se diferencia de los otros dos por la presencia o no de una serie de opciones, como se ve en la siguiente tabla:

Modelo	p_1	p_2	p_3	p_4	d_i
1	X	X			3
2		X	X	X	3
3	X		X		2

Tabla 1: Ejemplo de instancia para el caso binario

Si se puede fabricar un modelo por cada unidad de tiempo, el tiempo necesario para satisfacer el plan de producción es de 8 unidades de tiempo.

En esta tabla se supone que el consumo de opción *j* por parte del modelo *i* (c_{ij}) es binario, marcando con una cruz si el modelo consume esa opción o no. Cabe decir que se ha escogido este ejemplo para que resulte sencillo e ilustrativo, pero el procedimiento presentado es capaz de resolver instancias con consumos no binarios (aunque siempre enteros).

Esto presenta un problema asociado a buscar una secuencia para fabricar las unidades de distintos modelos, basándose en algún criterio concreto. El objetivo del problema ORV es encontrar una secuencia tal que el consumo de las opciones sea equilibrado, de manera que se minimice la variación en las tasas de consumo de las distintas partes u opciones respecto a la tasa ideal. Se entiende por tasa ideal de una opción *j* (r_j) el consumo por unidad de tiempo que realizaría el producto promedio de esa opción, e indica las unidades por unidad de tiempo que habría que fabricar de todos los modelos para que la opción p estuviera equilibrada. Se obtiene dividiendo el consumo total de la opción *j* (suma de la demanda de cada modelo multiplicada por el consumo de la opción *j*) entre el número total de productos a secuenciar. Para el caso del ejemplo, las tasas serían:

$$r_1 = \frac{\sum_{\forall i} d_i \cdot c_{i1}}{T} = \frac{3 \cdot 1 + 3 \cdot 0 + 2 \cdot 1}{8} = \frac{5}{8}$$

$$r_2 = \frac{\sum_{\forall i} d_i \cdot c_{i2}}{T} = \frac{3 \cdot 1 + 3 \cdot 1 + 2 \cdot 0}{8} = \frac{6}{8}$$

$$r_3 = \frac{\sum_{\forall i} d_i \cdot c_{i3}}{T} = \frac{3 \cdot 0 + 3 \cdot 1 + 2 \cdot 1}{8} = \frac{5}{8}$$

$$r_4 = \frac{\sum_{\forall i} d_i \cdot c_{i4}}{T} = \frac{3 \cdot 0 + 3 \cdot 1 + 2 \cdot 0}{8} = \frac{3}{8}$$

Nuestro objetivo, entonces, es encontrar una solución en que el consumo de todas las opciones en cada instante sea lo más cercano posible a estas tasas ideales. Sin embargo, cualquier solución que asigne un solo modelo en cada instante *t*, cumpliendo que al final del horizonte *T*

todos los modelos han cumplido con su plan de producción, sería válida. Pongamos un ejemplo con una solución cualquiera del ejemplo anterior como la secuencia mostrada en la tabla 2.

t	1	2	3	4	5	6	7	8
Modelo	1	1	1	2	2	2	3	3
$\mathbf{p_1}$	X	X	X				X	X
$\mathbf{p_2}$	X	X	X	X	X	X		
$\mathbf{p_3}$				X	X	X	X	X
$\mathbf{p_4}$				X	X	X		

Tabla 2: Ejemplo de secuencia

La secuencia de la tabla 2 dista mucho del objetivo descrito, ya que el consumo de cada opción está acumulado en uno o dos tramos de la secuencia. Por ejemplo, si la opción p_1 fueran aires acondicionados, el resultado sería que habría un periodo de tiempo en que toda esa parte de la línea estaría parada, y el stock de aires acondicionados se acumularía, mientras que en el resto de periodos, la línea estaría saturada y se consumirían todas las piezas asociadas a esa opción.

Para poder medir la discrepancia que se ha observado intuitivamente, es necesario evaluar de alguna manera su evolución. Para calcular este término de evaluación, podemos usar las tasas previamente calculadas. Por ejemplo, en el instante 3 se han producido 3 unidades con opción 1, sin embargo, según la tasa ideal calculada, deberían haberse producido 3·(5/8)=15/8 (algo menos de 2), por tanto, la diferencia es de 9/8. Es importante indicar que en el último periodo temporal, esta diferencia es siempre 0, ya que todos los elementos se han producido.

Partiendo de este hecho, para evaluar la calidad de una secuencia se propone sumar los cuadrados de estas diferencias para todos los periodos temporales y para cada una de las opciones, según la fórmula:

$$\sum_{t=1}^{T}\sum_{j=1}^{P}\left(\sum_{i=1}^{M}\sum_{\tau=1}^{t} X_{i\tau}\cdot c_{ij} - r_j\cdot t\right)^2 \qquad (1)$$

Donde $X_{i\tau}$ valdrá 1 o 0, dependiendo de si el modelo m se ha secuenciado en el instante τ o no. En el caso del ejemplo, se calcula el valor de la función objetivo para cada opción (entre corchetes) y luego se obtiene el total sumando el valor de cada opción:

$[(1-5/8)^2+(2-10/8)^2+(3-15/8)^2+(3-20/8)^2+(3-25/8)^2+(3-30/8)^2+(4-35/8)^2]+[(1-6/8)^2+(2-12/8)^2+(3-18/8)^2+(4-24/8)^2+(5-30/8)^2+(6-36/8)^2+(6-42/8)^2]+[(0-5/8)^2+(0-10/8)^2+(0-15/8)^2+(1-20/8)^2+(2-25/8)^2+(3-30/8)^2+(4-35/8)^2]+[(0-3/8)^2+(0-6/8)^2+(0-9/8)^2+(1-12/8)^2+(2-15/8)^2+(3-18/8)^2+(3-21/8)^2]=21,8125$

Elevar al cuadrado las diferencias no sólo evita que las diferencias negativas compensen las positivas, sino que también penaliza más las diferencias más grandes. Por ejemplo, en el caso anterior si la diferencia se calculara en valor absoluto ésta es de 15/8=1.875, ésta correspondería al mismo valor que tener tres diferencias de 5/8: 5/8+5/8+5/8=15/8. Sin embargo, usando la diferencia al cuadrado, el valor de la función objetivo para el caso expuesto es de 225/64=3.52. Y esto no corresponde al mismo valor que tener tres diferencias de 5/8: 25/64+25/64+25/64=1.17. Se observa, entonces, que al usar diferencias cuadráticas se penalizan más las diferencias más grandes, que el hecho de que haya muchas diferencias pequeñas. En el contexto industrial también tiene sentido penalizar las diferencias más grandes que una gran cantidad de diferencias pequeñas, ya que una diferencia muy grande puede suponer problemas graves de rotura o de acumulación de stock, mientras que las diferencias pequeñas son más fáciles de paliar.

Así pues, el problema puede definirse de la manera siguiente: dados m productos (o modelos), con una demanda d_i cada uno (o plan de producción), que usan un total de p opciones con un consumo c_{ij} correspondiente para cada opción y modelo ($1 \leq i \leq m$; $1 \leq j \leq p$), se busca una secuencia de producción de los distintos modelos de manera que se minimice la diferencia cuadrática entre el consumo real acumulado de las opciones, y la tasa ideal acumulada que debería haberse alcanzado en cada instante.

1.3. Descripción de un procedimiento de programación dinámica

Los procedimientos basados en programación dinámica se usan para resolver eficientemente problemas de búsqueda o de optimización que presentan superposición de subproblemas y subestructuras óptimas, Bellman (1954).

Por un lado, se dice que un problema presenta superposición de subproblemas si puede ser dividido en subproblemas, que no pueden solucionarse por separado, sino que para encontrar la solución de uno de ellos, puede requerirse la solución de alguno de los anteriores. Un algoritmo para solucionar un problema de este tipo suele ser recursivo y debe ser capaz de solucionar el mismo subproblema una y otra vez, basándose en la solución del mismo problema. Por ejemplo, el enésimo número de una serie de Fibonacci se calcula sumando los dos números anteriores en la serie. Por tanto, se trata de un problema que presenta superposición de subproblemas: el problema de calcular el enésimo número de Fibonacci requiere la solución de (n-1) y de (n-2), del mismo modo que para calcular el número (n-1) de la serie requiere del cálculo del (n-2) y el (n-3). Se observa entonces que el cálculo de cualquier número de la serie de Fibonacci puede descomponerse en una repetición de los números de Fibonacci de 1 y 0, que son conocidos de antemano:

NFib(4) = NFib(3) + NFib(2) = (NFib(2) +NFib(1)) + (NFib(1) + NFib(0)) = [(NFib(1) + NFib(0)) + NFib(1)] + NFib(1) + NFib(0)

Por otro lado, el concepto de subestructura óptima indica que la solución global del problema puede ser construida mediante soluciones óptimas de subproblemas. Por ejemplo, dado un problema en que se busca la ruta óptima entre un punto *A* y un punto *C* en un grafo, si la solución óptima calculada pasa por un tercer punto, *B*, se puede asegurar que la ruta *A*-*B* y la ruta *B*-*C* son a su vez, rutas parciales óptimas.

Combinando estos dos conceptos, que en el ejemplo de diseño de rutas corresponde a dividir un camino en caminos más pequeños, y demostrar la optimalidad de un camino por la optimalidad de estos caminos más pequeños, se obtiene la base teórica para un procedimiento basado en programación dinámica.

A efectos prácticos, un procedimiento de programación dinámica para la resolución de problemas de optimización combinatoria consta de diversos componentes, que se definirán a continuación: etapa, estado, transición y función de recurrencia.

Se define una etapa como cada momento en que debe tomarse una decisión: en el caso del problema, cada instante corresponde a una etapa, ya que hay que decidir qué modelo debe ser secuenciado en cada posición de la secuencia.

Por su parte, se entiende como estado la situación en que nos podemos encontrar antes de tomar una decisión, que debe aportar toda la información particular necesaria para tomar la decisión. En el presente problema, la primera etapa siempre corresponderá a un solo estado: el estado en que no hay ningún producto secuenciado, y la última etapa corresponde a un solo estado: el estado en que la producción de cada modelo corresponde a su plan de producción. Y un estado intermedio cualquiera quedará definido por la composición de modelos secuenciados hasta el instante actual.

Una transición se define como cada una de las decisiones que pueden tomarse en cada estado de una etapa y que dan como resultado un estado de la etapa siguiente.

Por último, la función de recurrencia es la función que se aplica de forma reiterada para evaluar los estados y determinar la mejor transición desde el estado inicial hasta el estado final. Pueden distinguirse dos casos del programa dinámico, según se estudien los estados del primer instante al último (*forwards*), o al revés (caso *backwards*).

Hay diversas maneras de implementar programa dinámico, pero en el caso de este trabajo se opta por el método denominado de las dos listas. Poniendo por ejemplo el caso *forward*, el procedimiento seguiría el siguiente esquema:

Paso 1: Se crea la primera etapa en la primera lista (lista 1), correspondiente a un único estado en que no hay nada secuenciado. Se crea una segunda lista vacía (lista 2).

Paso 2: Se exploran los estados que hay en lista 1, uno a uno. Se buscan todos los estados que pueden originarse a partir del estado almacenado en la lista 1 que se está explorando. Debe comprobarse si alguno de estos estados ya existe en la lista actual (lista 2), es decir, si se ha

almacenado un estado idéntico anteriormente en la lista 2. En caso negativo, y si no hay ningún mecanismo de cota, se almacenan todos los estados en la lista 2. Si hay estados idénticos, sólo tiene que almacenarse uno de ellos: el que presenta mejor valor acumulado de la función de recurrencia (lo que en el cálculo de caminos extremos se conoce como etiqueta o *label*). Adicionalmente, si se dispone del valor de una solución (cota superior), si algún estado presenta una cota inferior de mayor valor a la mejor solución encontrada por el momento, no tiene que almacenarse.

Paso 3: Se vacía la lista 1, la lista 2 se almacena en la 1, y la 2 se vacía.

Paso 4: Mientras queden estados en la lista 1, ir al el paso 2. Si la lista 1 está vacía FIN.

El problema ORV puede plantearse como un programa dinámico, como se verá en el apartado 3.2, ya que presenta superposición de problemas y subestructuras óptimas. Otra característica destacable del modelo de programación dinámica asociado al ORV, es que no presenta aleatoriedad y tiene un número finito de etapas, por tanto, puede ser reformulado como un problema de caminos mínimos, como también se detalla en el apartado 3.2.

1.4. Descripción de un procedimiento Branch-and-bound

Un algoritmo Branch-and-bound es un procedimiento de enumeración implícita que se emplea para la resolución de problemas de optimización combinatoria.

La idea general es que la solución óptima de un problema de optimización podría encontrarse generando todas las soluciones factibles, de las cuales se tomará la solución que minimice el valor de la función objetivo. La construcción del conjunto de soluciones factibles se organiza según un árbol que consta de vértices que representan subconjuntos de soluciones y arcos que representan la división del subconjunto padre en subconjuntos más pequeños. Estos descendientes siempre representan el total de soluciones que representaba el padre. Finalmente, los vértices terminales (es decir, que no tienen sucesores, también llamados vértices hoja) corresponden a las soluciones del problema.

En problemas de secuenciación como el ORV normalmente el subconjunto de soluciones queda definido por una subsecuencia y los arcos consisten en secuenciar una unidad adicional al final de la subsecuencia.

De este modo, si no se incluyen elementos adicionales, el árbol no sería más que un método para construir y representar todas las soluciones factibles. Pero la construcción del árbol completo no resulta eficiente y, en muchas ocasiones, ni siquiera es factible, por lo que mientras se ramifica cada vértice también es conveniente comprobar si la rama que se está desarrollando puede dar una solución mejor que la actual, esto es, que alguna de las soluciones comprendidas

en su subconjunto de soluciones puede ser mejor que la mejor solución conocida. En caso contrario, la rama se "corta", proceso denominado como poda y se pasa a desarrollar otra rama.

Para llevar a cabo la poda de vértices correctamente (es decir, para decidir si hay que ramificar o no un vértice determinado) es necesario evaluar una cota inferior de la solución total que puede aportar: para ello, se suma el valor de la función objetivo acumulada hasta el vértice actual con una cota inferior de la aportación para el resto del problema. De este modo, si en un paso del algoritmo, el valor de esta cota es superior a la mejor solución encontrada hasta el momento, el vértice puede dejar de desarrollarse.

Otra opción frecuente es la aplicación de varias reglas de dominancia que permitirían eliminar ciertos vértices, ya que el subconjunto de soluciones que representan es equivalente o peor que otro subconjunto (por ejemplo, es interesante eliminar subconjuntos que corresponden al mismo estado en el programa dinámico).

Resumidamente, un algoritmo Branch-and-bound es un procedimiento iterativo que repite tres pasos: ramificación, reducción (que incluye cálculo de cotas, poda, y reglas de dominancia) y selección del siguiente vértice a explorar.

- Empezando por el espacio entero de soluciones, la ramificación subdivide el subespacio de soluciones correspondiente al vértice actual en dos o más subespacios que explorar en iteraciones siguientes.
- Se comprueba la función (o funciones) de cota, comparando el resultado con la mejor solución conocida hasta el momento. Si se concluye que el subespacio no puede contener la solución óptima, se descarta. Si no se ha descartado, se verifica que este subespacio no está dominado. En caso contrario, se almacena.
- De los subespacios almacenados, se escoge uno de acuerdo con algún criterio y se ramifica. Este paso y el anterior se repiten hasta que no quedan subespacios almacenados.

Dependiendo de la secuencia que se siga para la exploración de subespacios, se puede clasificar el procedimiento en:

- Búsqueda primera en profundidad (o DFS, del inglés *depth-first-search*): Sólo se desarrolla una rama del árbol de soluciones hasta llegar a un vértice terminal, ya sea porque se ha encontrado una solución final o porque el vértice ha sido descartado. La búsqueda continua entonces por la primera rama que así lo permita (la de mayor profundidad). Normalmente se organiza como una búsqueda láser (secuencial), de modo que solamente se mantiene en memoria el vértice correspondiente al subconjunto de soluciones que se está explorando.

- Búsqueda primero en anchura (o *breadth-first-search*): En este caso, se exploran los subconjuntos de soluciones siguiendo un orden de prioridad que prima los subconjuntos de soluciones en que el número de ramificaciones sea menor.

- Búsqueda *Best-First*: En una búsqueda *Best-First* se explora el árbol desarrollando en cada paso el mejor vértice teórico, es decir: el subconjunto de soluciones con mejor valor de un indicador (en la mayoría de casos, una cota).

Existen métodos intermedios, como el procedimiento desarrollado en el presente trabajo, que empieza realizando una búsqueda *depth-first* de tipo láser (secuencial), hasta cumplir un límite (que puede ser temporal o de número de vértices desarrollados). Una vez la búsqueda llega al límite, los vértices descendientes del último vértice explorado son creados, y almacenados en memoria, junto con su cota parcial. Tras esto, se comprueba el estado de la memoria: si está muy llena, se continúa con una búsqueda tipo *depth-first*. Si queda espacio en memoria, se realiza una búsqueda *best-first*, que selecciona el vértice con mejor cota inferior de entre todos los vértices que se tienen almacenados. A partir del vértice seleccionado, se realiza diversas búsquedas *depth-first* hasta llegar al límite, para almacenar los nodos descendientes, y repetir así el proceso. Este proceso, aunque podría hacerse de forma totalmente secuencial, realiza la búsqueda *depth-first* en paralelo, de manera que diversos agentes trabajan en paralelo, cada agente seleccionando un nodo y desarrollandolo de forma paralela.

2. Alcance y limitaciones del proyecto

En este proyecto se pretende diseñar e implementar un algoritmo exacto de tipo Branch& Bound que dé solución a cualquier instancia de un problema ORV (aunque para instancias grandes no se puede garantizar que esta sea óptima, ya que el algoritmo tiene un límite temporal), siempre que el uso de las opciones por parte de cada modelo pueda expresarse con un número entero, y, así, no está limitado al caso de consumo binario. Naturalmente, podrá resolver cualquier problema que pueda ser expresado o simplificado como un problema de ORV, en que las únicas restricciones sean las correspondientes al cumplimiento de la demanda y que sólo pueda secuenciarse un producto en cada instante de tiempo.

Para resolver el problema será necesario que la instancia procure la siguiente información: el número total de productos a fabricar, que también corresponde al número de periodos temporales, el número total de opciones y el número total de modelos. También se requerirá el plan de producción de los modelos a fabricar, así como el consumo de cada opción por parte de cada modelo.

El procedimiento implementado dará como solución la mejor secuencia encontrada, así como el valor de función objetivo que se obtiene con esta secuencia (que se conoce también como sdq). También se obtendrá como resultado un indicador de optimalidad de la solución, en relación con la mejor cota inferior obtenida.

El trabajo presentado aporta diversas novedades, que se listan a continuación: la separación del problema en varios subproblemas según las opciones consumidas, para solucionarlos de manera óptima mediante programación dinámica y usar combinaciones de estas soluciones para obtener cotas; el uso de la simetría del problema y de las cotas obtenidas para el cálculo de un programa dinámico acotado más eficiente; la aplicación de diversas reglas de dominancia basadas en la simetría del problema, en la obtención de soluciones heurísticas y en el almacenamiento de vértices desarrollados.

3. Formulación matemática del problema

Hay diversas maneras de formular matemáticamente el problema considerado, por lo cual se divide este apartado entre la formulación según programación entera, en que se emplea la nomenclatura de Jin y Wu (2002), y la formulación según programación dinámica, en que se emplea la adaptación al problema realizada por Bautista et al. (1996). En el apartado de programación entera, además, se compara la formulación del problema tratado con el problema de transporte.

3.1. Programación entera

En el problema considerado, se suponen conocidos los siguientes parámetros (dados por la instancia de datos):

Parámetros:

M productos (o modelos) ($1 \leq i \leq M$)

$d=[d_1, d_2, \ldots, d_M]$, plan de producción para los *M* productos

T tiempo total necesario y número total de unidades a secuenciar, definido como:

$$T = \sum_{i=1}^{M} d_i \qquad (2)$$

P opciones ($1 \leq j \leq P$)

C=$\begin{bmatrix} c_{11} & \cdots & c_{M1} \\ \vdots & \ddots & \vdots \\ c_{1P} & \cdots & c_{MP} \end{bmatrix}$, número de unidades de la opción *j* que se usan para cada producto *i*.

$\mathbf{r_j}$ tasa ideal por opción, definida como:

$$r_j = \sum_{i=1}^{M} \frac{d_i \cdot c_{ij}}{T} \qquad (3)$$

Variables:

$\mathbf{X_{it}} \in \{0,1\}$. Esta variable binaria tomará por valor 1 si se secuencia el producto *m* en el instante *t*, y 0 en caso contrario.

Función objetivo:

$$[MIN]\sum_{t=1}^{T}\sum_{j=1}^{P}\left(\sum_{i=1}^{M}\sum_{\tau=1}^{t}X_{i\tau}\cdot c_{ij}-r_j\cdot t\right)^2 \quad (4)$$

En la función objetivo se elevan al cuadrado las discrepancias con dos objetivos: primero, necesitamos que la función objetivo penalice cualquier discrepancia con la tasa ideal, tanto si es positiva, como si es negativa; y segundo, el cuadrado hace que se penalicen más las discrepancias más grandes. Al valor de la función objetivo para cada etapa se le denomina aportación.

Restricciones:

$$\sum_{t=1}^{T}X_{it}=d_i;\forall i \quad (5)$$

$$\sum_{i=1}^{M}X_{it}=1;\forall t \quad (6)$$

La primera restricción asegura que en una solución todos los modelos fabricados cumplen con su plan de producción. La segunda asegura que en cada instante de tiempo se esté fabricando un sólo modelo. Otra manera de escribir (4) sería:

$$[MIN]\sum_{t=1}^{T}\sum_{j=1}^{P}\left(\sum_{i=1}^{M}u_i\cdot c_{ij}-r_j\cdot t\right)^2 \quad (7)$$

Donde *U* es un vector de composición de modelos, es decir: para cada componente *i* en modelos, la componente U_i indica la cantidad de unidades del modelo *i* que han sido secuenciadas.

Esta formulación corresponde, casi completamente a la formulación de un problema de transporte, que presenta las mismas restricciones (5) y (6), pero con la función objetivo siguiente:

$$[MIN]\sum_{i=1}^{M}\sum_{t=1}^{T}X_{it}\cdot C_{it} \quad (8)$$

Donde la componente *i*,*t* de la matriz *C* corresponde al "coste" de asignar el modelo *i* en el instante *t*. Por esta razón, resulta útil asimilar ambos problemas para la obtención de una cota. Tanto la definición del problema de transporte como su aplicación en este trabajo se ven con más detalle en el apartado 5.1.4.

3.2. Programación dinámica

Para la resolución del problema mediante programación dinámica se plantean dos formulaciones: una tipo *forward* (en la que la función de recurrencia enlaza estados asociados a un instante con los asociados al instante siguiente) y otra tipo *backward* (en la que la función de recurrencia enlaza estados asociados a un instante con los asociados al instante anterior). Ambas comparten la misma definición de estado: un estado *S* se define como la composición hasta el instante en curso (*t*). Puede definirse de forma vectorial como $u = [u_1, u_2, \dots, u_M]$, con tantas componentes como modelos considere el problema. Estas componentes tienen que cumplir $\sum_{\forall i} u_i = t$

La formulación tipo *backward* considera una etapa inicial (*t*=0) correspondiente al instante *T*, que tiene un único estado asociado, en que la composición corresponde a la demanda total de cada modelo [d_1, d_2, …, d_M] y tiene un coste igual a 0. La etapa *t*=1 corresponde al instante de producción *T*-1 y así sucesivamente hasta la etapa *t*=*T*, que corresponde al instante de producción 0. Esta última etapa tiene un único estado, en el que la composición es igual a 0 para todos los productos.

En una etapa cualquiera *t*, se supone un estado *S*, caracterizado por un vector $u = [u_1, u_2, \dots, u_M]$. En este caso, el número total de productos secuenciados debe corresponder a *T*-*t*. La función de recurrencia asociada al estado S es la siguiente:

$$f^t(S) = MIN\{f^{t-1}(S \cup \{i\}) + aportación(i)\}_{\forall i \in M; d_i \geq u_i > 0} \qquad (9)$$

Donde la aportación del modelo *i* es su contribución a la función objetivo al secuenciarlo en el instante *T*- *t*, y debe comprobarse para todos los modelos cuya composición en el estado anterior sea superior a 0. Es importante observar que al calcular las recurrencias el valor asociado a un vértice corresponde a la aportación restante desde el instante actual hasta el instante final, es decir: se obtiene el valor de la secuencia óptima desde la etapa en curso hasta la etapa final. Por este motivo, se prefiere utilizar esta formulación para el cálculo de cotas, ya que asocia a cada estado la aportación restante hasta el final.

La segunda formulación es la más utilizada para la resolución del problema y corresponde a un procedimiento *forward*. La etapa *t*=0 está asociada al estado en que no se han secuenciado unidades, y la etapa *t*=*T* al estado en que todas las unidades han sido secuenciadas.

El coste asociado a la etapa 0 es igual a 0 y en una etapa t, se considera un estado *S* definido por un vector $u = [u_1, u_2, \dots, u_M]$ que cumpla que el número total de productos secuenciados hasta el momento es igual al número de instantes de producción. La función de recurrencia asociada a ese estado *S* es la siguiente:

$$f^t(S) = MIN\{f^{t-1}(S \backslash \{i\}) + aportación(i)\}_{\forall i \in M; 0 \leq u_i < d_i} \qquad (10)$$

En este caso, una vez finalizado el cálculo de recurrencias, el valor asociado a un vértice corresponde a la aportación acumulada desde el instante inicial hasta el instante actual.

Para su resolución mediante programación dinámica es habitual reformular el problema como un grafo polietápico, de manera que encontrar la secuencia que minimice el valor de sdq sea equivalente a encontrar el camino mínimo entre dos nodos del grafo asociado. El grafo se construye de manera que los estados corresponden a los vértices del grafo y las transiciones a los arcos. A continuación se muestra el procedimiento a seguir para construir el grafo usando las recurrencias en sentido forward. En caso de querer construir el grafo en sentido *backwards* deben hacerse algunas modificaciones.

Para este efecto, el problema puede formularse como un grafo de *T*+1 niveles (donde *T* es el número total de etapas). Los vértices de cada etapa *t* (*t*=0,1,2,…,*T*) están definidos por vectores *u* , correspondientes a los distintos estados, que siempre deben cumplir:

$$\sum_{i=1}^{M} u_i = t \qquad (11)$$

Donde $0 \leq u_i \leq d_i$ se cumpla para todo modelo *i*.

Es decir, estos vectores indicarán para cada componente *i*, cuántos productos del modelo *i* se han secuenciado. La suma de la cantidad de productos secuenciados para cada etapa *t* debe ser igual a *t*, ya que se supone que se fabrica un producto por unidad de tiempo.

En el nivel 0 también habrá un vértice ficticio α, representado por el vector [0,0,…,0], que supondrá que aún no hay nada producido. De manera similar, en la etapa *T* se creará un último vértice ω, representado por el vector [d_1,d_2,…,d_M], que supondrá que se ha terminado de producir la demanda para todos los modelos.

El número de vértices en el resto de etapas no es conocido a priori (aunque podría determinarse mediante combinatoria), pero siempre alcanzará el máximo en la etapa $\lfloor (T+1)/2 \rfloor$, y dado que se tratará de un grafo simétrico (esta propiedad se ve con más detalle en el apartado 4.1.1. Simetría), el número de vértices en la etapa *t* siempre será el mismo que en la etapa *T*-*t*.

Se define un arco entre un vértice de una etapa *t* y un vértice de la etapa *t*+1 si para cada uno de los componentes del vector asociado al vértice de la etapa *t* y el vector asociado al vértice de la etapa *t*+1, todos los componentes son iguales excepto uno, que denominaremos *i*, que cumple que ${u^{t+1}}_i = {u^t}_i + 1$. A cada arco se le asigna un valor igual a la aportación de secuenciar el modelo m en la etapa *t*+1. Dado este grafo, encontrar una secuencia de modelos a producir que minimice la sdq es equivalente a buscar en el grafo un camino de mínimo coste entre α y ω.

La figura 1 muestra el grafo asociado a un problema de secuencias en que se considera una única opción con consumo binario, dando lugar, por tanto, a 2 modelos (el que incorpora la opción y el que no). El plan de producción de estos dos modelos es [4,2].

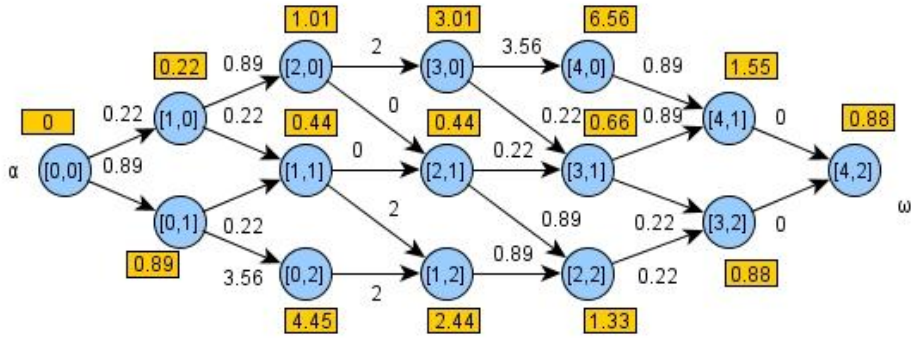

Figura 1: Grafo forward correspondiente al ejemplo

Análogamente, la figura 2 muestra el grafo generado usando las funciones de recurrencia en sentido backwards.

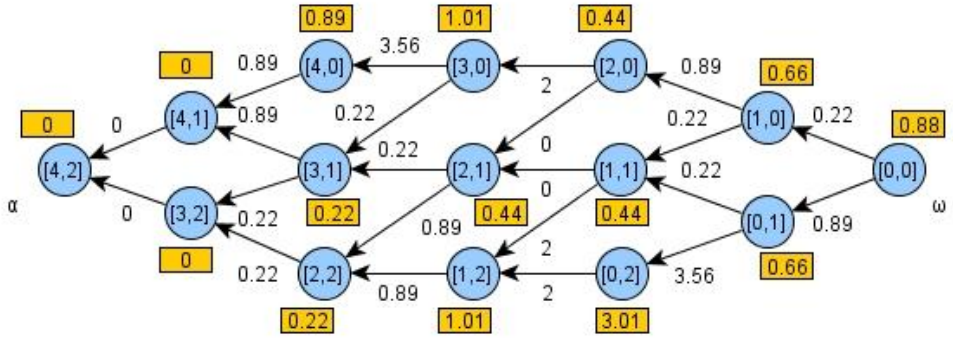

Figura 2: Grafo *backwards* correspondiente al ejemplo

El valor sobre cada arco *A*-*B* representa la aportación de la decisión que ha permitido pasar del estado *A* al *B*. Por ejemplo, en el caso *forward*, el arco que va del nodo [2,0] al [3,0] tiene como valor 2, porque esta es la aportación correspondiente a secuenciar una unidad del modelo 1 en el instante 3, partiendo del estado [2,0]. En el ejemplo, la tasa ideal para cada modelo es 4/6 y 2/6, respectivamente. La aportación del arco, entonces, se calcula como la diferencia al cuadrado de lo que debería haberse producido del modelo uno y lo que se ha producido, más la diferencia al cuadrado ídem para el producto dos:

$$\left(3 \cdot \frac{4}{6} - 3\right)^2 + \left(3 \cdot \frac{2}{6} - 0\right)^2 = 2$$

En los rectángulos se observa sobre ambos grafos la etiqueta (o *label*) de cada nodo. Esta corresponde al valor de función objetivo para cada nodo, es decir, la mínima sdq acumulada para ese estado. Por ejemplo, la etiqueta del nodo [2,1] en el grafo *forward* tiene como valor 0.44, ya que el camino de menor valor para llegar a este vértice tiene como aportaciones 0.22 (del vértice α a [1,0]),

0.22 (del vértice [1,0] al vértice [1,1]) y 0 (del vértice [1,1] al vértice [2,1]). Sumando los tres valores se obtiene como valor total de sdq acumulada 0.44.

4. Propiedades del problema

En este trabajo, para el cálculo de cotas y reglas de dominancia se aprovechan las propiedades del problema, tales como su simetría (que permite ahorrar gran número de cálculos), y su separabilidad por opciones (que permite dividir el problema en subproblemas más pequeños, y, por tanto, más fáciles de resolver). En este apartado, se comentan estas propiedades, su origen y el uso que se puede hacer de ellas.

4.1. Simetría

Como se ha comentado en el apartado 3.2, se deduce una cierta simetría de los grafos asociados al programa dinámico del problema. Esta propiedad era previamente conocida, véase Bautista et al. (1996) y Fliedner et al. (2010) y para este estudio, los resultados más importantes son: (1) La existencia de un estado complementario $\bar{A}$, asociado a la etapa *T-t*, a cada estado *A*, asociado a la etapa *t*, tal que las aportaciones desde el estado *A* hasta el estado final son iguales a las aportaciones desde el estado inicial a $\bar{A}$; (2) que los caminos desde los estados hasta los estados iniciales o finales tienen una aportación acumulada equivalente a los caminos desde los estados complementarios hasta los estados finales o iniciales, respectivamente, y (3) que una secuencia simétrica a otra tendrá el mismo valor de sdq.

La demostración de la propiedad de simetría se divide en dos partes:

- Primero, debe demostrarse que para todo estado *A* existe un estado complementario $\bar{A}$ y por tanto el grafo es simétrico.

Cada estado queda definido por un vector con componentes x_i entre 0 y d_i para cada modelo *i*, mientras que el estado complementario queda definido por un vector con componentes d_i-x_i para cada modelo. Debido al rango de valores del primer vector, el segundo vector también definirá un estado del programa dinámico, asociado a la etapa simétrica.

Además, la simetría implica que para cualquier camino *w* entre α y un estado *A* existe un camino $\overline{w}$ que va desde se $\bar{A}$ hasta ω compuesto por vértices asociados a los estados complementarios de los vértices del camino *w*.

- Segundo, para todo estado *A* su estado complementario $\bar{A}$ tiene la misma aportación y por tanto los caminos *w* y $\overline{w}$ definidos anteriormente tendrán la misma aportación acumulada.

Partiendo que el problema es separable, como se demuestra en el apartado siguiente, es suficiente con demostrar la simetría para una única opción, *j*.

Puede demostrarse que la aportación de cualquier estado de la etapa *t*, es idéntica a la aportación desde su estado complementario en la etapa *t'* tal como sigue:

Recordando la formulación de la función objetivo según (7), para la etapa *t*

$$\left(\sum_{i=1}^{M} U_i \cdot c_{ij} - r_j \cdot t\right)^2 \qquad (7')$$

Va a demostrarse que esta ecuación es equivalente a la que representa a su estado complementario:

$$\left(\sum_{i=1}^{M} \overline{U}_i \cdot c_{ij} - r_j \cdot t'\right)^2 \qquad (12)$$

Dado que el estado complementario cumple $t'=T-t$ y $\overline{U}_i=d_i-U_i$ y sabiendo que $r_j = \sum_{i=1}^{M} \frac{d_i \cdot c_{ij}}{T}$, según la definición (4), entonces pueden sustituirse las expresiones en el cálculo:

$$\left(\sum_{i=1}^{M} (d_i - U_i) \cdot c_{ij} - \sum_{i=1}^{M} \frac{d_i \cdot c_{ij}}{T} \cdot (T - t)\right)^2 \qquad (13)$$

$$\left(\sum_{i=1}^{M} d_i \cdot c_{ij} - U_i \cdot c_{ij} - \sum_{i=1}^{M} d_i \cdot c_{ij} - r_j \cdot t)\right)^2 \qquad (14)$$

Y finalmente, se demuestra que

$$\left(\sum_{i=1}^{M} -U_i \cdot c_{ij} + t\right)^2 = \left(\sum_{i=1}^{M} U_i \cdot c_{ij} - t\right)^2 \qquad (15)$$

Como los caminos w y $\overline{w}$ atraviesan vértices complementarios con idéntica aportación, la aportación acumulada de un estado será idéntica y sustituible por la de su complementario. Además no existirá ningún camino con mejor aportación acumulada, porque si no ese camino sería el escogido como camino original. Usando el mismo principio lógico se puede demostrar que la sdq de una secuencia es idéntica a la de su secuencia simétrica.

A efectos prácticos se desprende un problema de las anteriores demostraciones cuando se intenta trabajar con costes asociados a los arcos en lugar de a los vértices. A continuación, se ejemplifica la problemática:

En un problema de una opción y dos modelos, y plan de producción [4,2], uno de los vértices de la etapa 4 tiene una composición [3,1], que indica que ya se han secuenciado tres unidades del producto 1 y una del 2. En este caso, la composición complementaria para el estado complementario 2, sería [1,1].

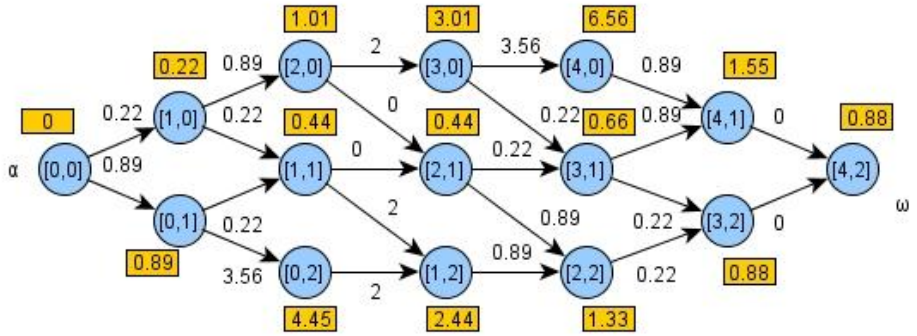

Figura 3: Ejemplo forward para la simetría del problema

El grafo asociado a este ejemplo corresponde a una representación según un procedimiento forward, y por tanto, la aportación de un nodo concreto *A* viene dada por el valor de los arcos entrantes (todos ellos tienen valor idéntico) a este nodo. Por tanto, la aportación asociada a la composición [3,1] es idéntica a la de la composición [1,1] e igual a 0.22.

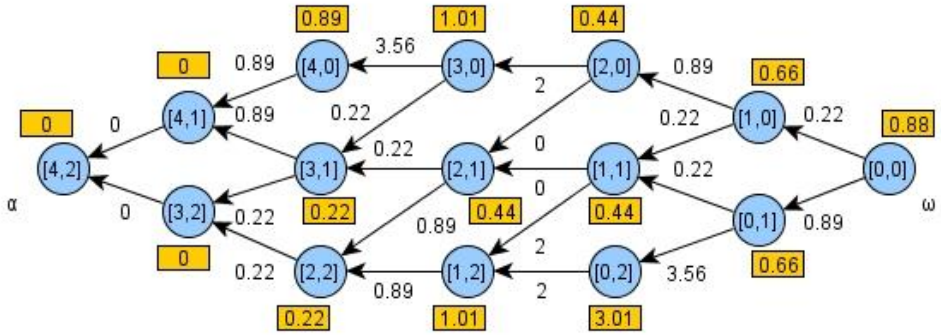

Figura 4: Ejemplo backwards para la simetría del problema

Como se observa en la figura 4, esta propiedad puede comprobarse con la misma facilidad en el grafo backwards. En este caso, la aportación de cada nodo también se anota en los arcos entrantes y para cualquier nodo es igual que la aportación de su complementario.

Al calcular la aportación acumulada aparece el problema indicado, ya que la aportación acumulada desde [0,0] hasta [3,1] más la aportación acumulada desde [0,0] hasta [1,1], debería corresponder a coste total desde [0,0] hasta [4,2]. Sin embargo los valores acumulados incluyen en dos ocasiones el valor de aportación del estado [3,1], y por tanto debería restarse en una ocasión o no ser sumado en dos ocasiones.

Lo mismo sucedería en el grafo asociado a la función de recurrencia backward, excepto que faltaría tener en cuenta en una ocasión la aportación del estado (o de su complementario).

Esta propiedad del problema puede usarse para calcular una cota de lo que queda por secuenciar desde un nodo cualquiera hasta el nodo ω en caso de combinarse con la propiedad de separabilidad de un problema, o para determinar directamente el valor el complementario de la secuencia si éste ya ha sido desarrollado con anterioridad. La aplicación de esta propiedad en procedimientos de acotación y simetría se detalla en los apartados 5.1.2. y 5.3.3.

4.2. Separabilidad

Otra de las propiedades que se usarán para el desarrollo de cotas en este algoritmo es la separabilidad del problema. Cualquier problema de secuencias regulares que tenga dos o más opciones puede separarse en diversos subproblemas por opciones. Por ejemplo, un problema de 4 opciones puede ser separado en cuatro subproblemas de una opción, así como en seis subproblemas de dos opciones (combinando las opciones 1 y 2, las 1 y 3, 1 y 4, 2 y 3, 2 y 4, 3 y 4) y cuatro problemas de tres opciones (1,2 y 3; 1,2 y 4; 1,3 y 4; 2,3 y 4).

Este hecho puede extraerse de la función objetivo, que puede ser reexpresada de la siguiente manera:

$$\sum_{t=1}^{T}\sum_{j=1}^{P}\left(\sum_{i=1}^{M}\sum_{\tau=1}^{t} X_{i\tau}\cdot c_{ij} - r_j\cdot t\right)^2 = \sum_{j=1}^{P}\left(\sum_{t=1}^{T}\left(\sum_{i=1}^{M}\sum_{\tau=1}^{t} X_{i\tau}\cdot c_{ij} - r_j\cdot t\right)^2\right) \qquad (16)$$

De manera que, calculando los valores de la función objetivo separadamente por opciones, y sumándolos, se obtiene el mismo resultado que usando la formulación original. Sin embargo, no se puede obtener una solución mediante las soluciones de cada opción por separado, ya que la restricción que obliga a cumplir con el plan de producción para cada modelo lo impide. A pesar de esto, esta separabilidad puede utilizarse en el cálculo de cotas, ya que la solución óptima de los problemas separados y sumados en combinaciones que no repitan ninguna opción será un valor inferior o igual al de la solución óptima, como se detalla en el apartado 5.3.2. Por ejemplo, la combinación entre el subproblema de las opciones 1, 3 y 4 y el subproblema para la opción 2 no genera una solución, pero sí que puede usarse para obtener una cota.

Para obtener el subproblema para una sola opción, se considerarán como modelos diferentes aquellos que tengan consumo diferente para la opción escogida. De esta manera, si varios modelos del problema original tenían el mismo consumo para la opción, se toman en consideración como si se tratara de un solo modelo y su plan de producción es igual a la suma de los planes de producción de los modelos del problema original. Para los subproblemas de 2 y 3 opciones, se hace de forma análoga.

A continuación se presentan varios ejemplos de cómo se obtendrían algunos de estos subproblemas, para un problema de cuatro opciones, véase tabla 3.

Problema global	Consumo Op. 1	Consumo Op. 2	Consumo Op. 3	Consumo Op. 4	Plan de producción
Modelo 1	1	0	2	0	3
Modelo 2	0	0	1	1	4
Modelo 3	0	1	1	1	4
Modelo 4	2	1	0	0	6
Modelo 5	1	1	0	2	2

Tabla 3: Ejemplo con 5 modelos y 4 opciones completo

Esta primera tabla presenta los datos del problema original. Para construir el subproblema de una opción (por ejemplo, la opción 3), se considera un modelo distinto por cada valor de consumo que tenemos para la opción 3. En este caso, hay tres modelos: el que consume cero, el que consume uno y el que consume dos. El plan de producción corresponde a la suma de los planes de producción del problema original de los modelos que tienen el mismo consumo. En este caso, el modelo con consumo cero tendrá un plan de producción igual a ocho (correspondientes al modelo 4 y 5 del problema original), de la misma forma, el modelo con consumo uno tendrá un plan de producción de ocho y el modelo con consumo dos tendrá un plan de producción de tres.

Problema opción 3	Consumo Op. 3	Plan de producción
Modelo 1	2	3
Modelo 2	1	8
Modelo 3	0	8

Tabla 4: Ejemplo del subproblema de una opción (opción 3) obtenido para el ejemplo anterior

Para crear el subproblema de las opciones dos y tres, se considera un modelo distinto cada combinación distinta entre consumos de las opciones dos y tres. Por ejemplo, en el caso tratado, los modelos 1, 2, 3 tienen consumos distintos para las opciones dos y tres, sin embargo, los modelos 4 y 5 ambos consumen una unidad de opción dos y ninguna de opción tres, por tanto, se consideran un total de cuatro modelos. De la misma manera que en el caso anterior, se calcula el plan de producción de los modelos, sumando los planes de producción de los modelos originales que sean equivalentes:

Problema opción 2 y 3	Consumo Op. 2	Consumo Op. 3	Plan de producción
Modelo 1	0	2	3
Modelo 2	0	1	4
Modelo 3	1	1	4
Modelo 4	1	0	8

Tabla 5: Ejemplo del subproblema de dos opciones (opc. 2 y opc. 3) obtenido para el ejemplo anterior

Finalmente el subproblema de tres opciones asociado a las opciones 1, 3 y 4, correspondería a los datos que se muestran en la tabla 6.

Problema opción 1, 3 y 4	Consumo Op. 1	Consumo Op. 3	Consumo Op. 4	Plan de producción
Modelo 1	1	2	0	3
Modelo 2	0	1	1	8
Modelo 3	2	0	0	6
Modelo 4	1	0	2	2

Tabla 6: Ejemplo del subproblema de tres opciones (opc. 1, 3 y 4) obtenido para el ejemplo anterior

Excepto los subproblemas de una opción, que resultan triviales de resolver, estos subproblemas son teóricamente igual de complejos que el problema original. En la práctica, como ya se ha indicado, los subproblemas de hasta tres opciones se pueden solucionar mediante un programa dinámico (cosa que para el problema completo no es eficiente, debido a la gran cantidad de nodos a desarrollar y almacenar). Las soluciones de estos problemas ya suponen una cota, pero es muy poco restrictiva, ya que no incluyen todas las opciones del problema. Para mejorar la cota, pueden combinarse las soluciones a los subproblemas de una, dos y tres opciones, siempre y cuando en una combinación no se repitan las opciones. Es decir, las combinaciones posibles para el ejemplo anterior de cuatro opciones son:

Subproblema [1,2] + Subproblema [3,4]

Subproblema [1,3] + Subproblema [2,4]

Subproblema [1,4] + Subproblema [2,3]

Subproblema [1,2,3] + Subproblema [4]

Subproblema [1,2,4] + Subproblema [3]

Subproblema [1,3,4] + Subproblema [2]

Subproblema [2,3,4] + Subproblema [1]

Naturalmente, también se puede sumar el valor de las soluciones para los cuatro subproblemas de una sola opción, pero el valor siempre será igual o peor al que se obtendrá con las anteriores. El apartado 5.1.3 analiza con más detalle los usos de la separabilidad del problema.

5. Componentes del algoritmo

En este apartado se exponen los componentes del algoritmo desarrollado, detallando la base teórica sobre la que se sustentan, los cálculos asociados a cada uno de ellos y, finalmente, cómo se unen para crear el algoritmo general.

Estos componentes incluyen cotas superiores, cotas inferiores y reglas de dominancia, así como el funcionamiento detallado del algoritmo de enumeración, incluyendo las estrategias de ramificación empleadas y las estrategias de poda.

5.1. Cotas inferiores

5.1.1. Cota por condición de integralidad

Esta primera cota se basa en que la cantidad de productos secuenciados en cualquier instante de tiempo es un número entero, mientras que la cantidad ideal ($r_j \cdot t$) es un número real que podría no ser entero.

Por tanto, se puede definir una aportación mínima, como la diferencia entre la cantidad ideal de una opción que debería secuenciarse en un instante y el entero más próximo, ya sea por exceso o por defecto:

$$MIN\left\{\left(\lfloor r_j \cdot t \rfloor - r_j \cdot t\right)^2; \left(\lceil r_j \cdot t \rceil - r_j \cdot t\right)^2\right\} \qquad (17)$$

Durante la ejecución del algoritmo de enumeración en el instante *t*, en el que quedan *T*-*t* unidades por secuenciar y hay *x* elementos secuenciados con la opción *j*, debe recalcularse esta aportación mínima para los instantes que se encuentran entre *t*+1 y *T*.

Cabe destacar que si suponemos una serie de subproblemas de consumo binario de opciones, en que se tienen en cuenta las opciones por separado, y se pretende equilibrar el uso de una sola opción, el valor de esta cota de integralidad corresponde con la solución óptima al subproblema para cada opción. Además, esta cota es poco restrictiva y no se usa en el algoritmo implementado, excepto para instancias de gran tamaño, donde las cotas por separabilidad no pueden almacenarse en memoria.

5.1.2. Cota por simetría

Gracias a la simetría del problema, propiedad dada por la función objetivo y demostrada con anterioridad por Fliedner et al. (2010), y teniendo en cuenta la definición de nodo y etapa complementaria expuesta en el apartado 4.1., se puede afirmar lo siguiente:

- La aportación de un estado *A* es siempre idéntica a la aportación de su estado complementario $\bar{A}$.
- La aportación mínima de los estados que quedan por secuenciar desde un estado *A* asociado a una etapa *t* es idéntica a la aportación mínima desde el inicio hasta el estado $\bar{A}$ asociado a la etapa *T*-*t*.

Cabe destacar que usando la formulación de caminos basada en la función de recurrencia *forward*, el coste asociado a todos los arcos que llegan a ω es cero, por tanto, cuando nos referimos a la aportación acumulada para ir de α a un estado cualquiera (el valor que aparece en el label del nodo en cuestión), la etiqueta siempre incluye la aportación del estado en sí, es decir, el valor del arco entrante en el nodo. Esto implica que al calcular la cota total de cualquier nodo *A*, no se puede sumar directamente la aportación acumulada desde α hasta *A* (label de *A*) y la aportación desde α hasta $\bar{A}$ (label de $\bar{A}$), ya que ambos términos incluyen la aportación del nodo (que es idéntica para ambos). Por tanto, si se usa el grafo derivado del programa dinámico en el sentido forward, la cota por simetría deberá calcularse sumando la aportación acumulada desde α hasta el nodo del que procede *A* y la aportación desde α hasta $\bar{A}$.

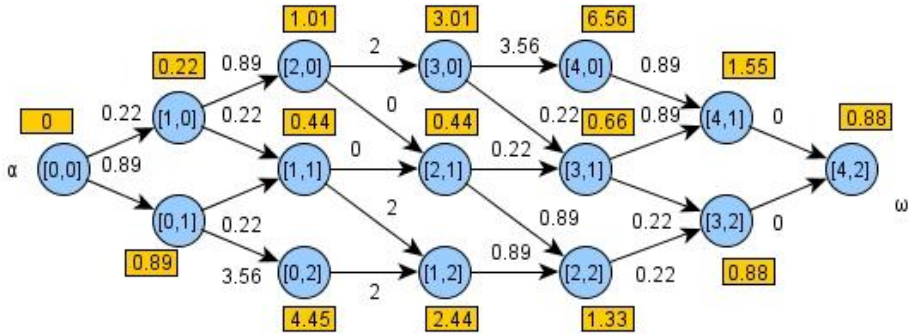

Figura 5: Ejemplo de grafo forward

Recordando el ejemplo del apartado 3.2. para un problema de una opción y dos modelos, con plan de producción [4,2], se toma como ejemplo un nodo en la etapa 4 X_4=[3,1] (que se está explorando al desarrollar el nodo [3,0]). Este vértice tiene como complementario el nodo [1,1], cuya aportación acumulada desde α es 0.44. Una cota del valor total de sdq que puede suponer el hecho de secuenciar por el nodo [3,1], viniendo de [3,0], es la suma del valor acumulado hasta [3,1] (sin incluirlo, por tanto la sdq acumulada hasta el nodo [3,0]) y una cota del valor que puede acumularse a partir de ese nodo (esto es, el valor de sdq acumulado para su nodo complementario, que también incluye la aportación del nodo [1,1]). Esto da una cota para el nodo [3,1] de 3.45.

Con el grafo *backwards*, también es posible calcular una cota por simetría de una manera muy similar. La diferencia es que, en ese caso la etiqueta para un nodo cualquiera no incluye la

aportación de ese estado, ya que la aportación de los estados se representa en los arcos entrantes, pero estos van en la dirección contraria. Por ejemplo, observando el grafo backwards del ejemplo, vemos que para el estado [2,2] el valor de la etiqueta sólo incluye la aportación de α hasta [3,2], mientras que su aportación (0.89), se encuentra en los arcos entrantes. Esto implica que, cuando se pretenda calcular la cota por simetría de un estado cualquiera *A*, hay que sumar el valor de sdq acumulado desde α hasta *A*, una cota de lo que queda por secuenciar (la sdq acumulada desde α hasta $\bar{A}$), y la aportación del nodo *A*.

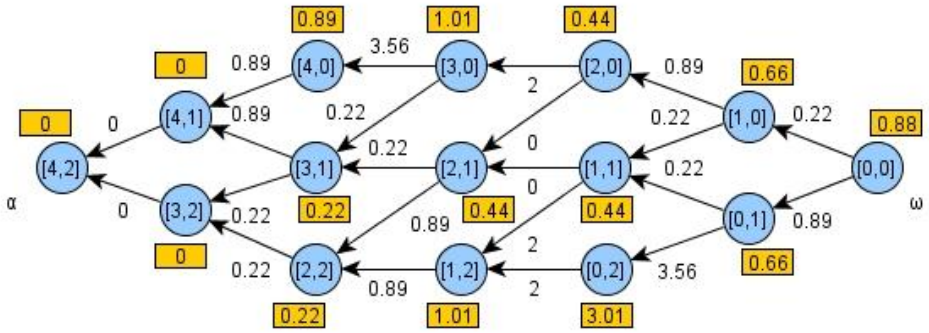

Figura 6: Ejemplo de grafo *backward*

Usando un ejemplo parecido al del caso forward, supongamos el estado [1,1], que tiene como complementario el estado [3,1]. Supongamos que se ha llegado al estado [1,1] por el camino más corto y que, por tanto, se aportación acumulada desde α hasta él es de 0.44. La cota por simetría del estado [1,1] sería la suma de esta aportación acumulada (que, como se ha detallado en el párrafo anterior, no incluye la aportación del estado [1,1]), más la aportación del estado [1,1] (anotada en los arcos entrantes, 0.22), y por último, la aportación acumulada de α hasta el estado complementario [3,1], 0.22. Esto da una cota de 0.88.

La cota por simetría sólo se utiliza para agilizar el cálculo de las cotas por separabilidad de opciones, que se detalla en el siguiente apartado.

Es importante indicar que el uso del término cota puede entenderse como un abuso del lenguaje ya que la simetría aporta el valor óptimo de la aportación remanente, aunque se esté utilizando como cota.

5.1.3. Cota por separabilidad del problema por opciones

Solucionar el problema global de manera exacta mediante programación dinámica es inviable para las instancias a partir de cierto tamaño, por la cantidad de nodos a explorar y

almacenar que esto supone. Sin embargo, es posible separar el problema en diversos subproblemas de 2 y 3 opciones, como se ha detallado en el apartado 4.1.2, y solucionarlos de forma óptima mediante programación dinámica. Tras ello, la solución de estos subproblemas puede usarse como cotas del problema global.

Las soluciones a estos subproblemas no sólo son una cota para el problema global, sino que además pueden ser combinadas para obtener cotas más restrictivas: por ejemplo, para un problema con cinco opciones, se puede sumar la solución correspondiente al subproblema para las opciones 1 y 2, con la solución del problema para las opciones 3, 4 y 5. Así, solucionando todos los subproblemas correspondientes a 2 y 3 opciones (cabe recordar que la solución óptima a los subproblemas con una sola opción en el caso de consumos binarios es la cota por integralidad asociada a ese subproblema), generando todas las combinaciones posibles entre ellos, calculando la suma de las soluciones para cada combinación y seleccionando la mejor de ellas, se obtiene una cota inferior por combinación de opciones separadas.

Estos subproblemas se solucionan mediante programación dinámica (PD), explorando un grafo asociado análogo al que se ha descrito en el apartado 3.2. El algoritmo desarrollado para la resolución de este problema encuentra el camino mínimo de forma *backward*, es decir, empezando por el último vértice ω y con cada paso hacia atrás, genera los vértices que pueden haber dado lugar al vértice actual, con una unidad menos secuenciada. Se utiliza este tipo de resolución ya que interesa conocer el valor de sdq acumulado desde un nodo hasta el final de la secuencia (cosa que en el grafo *backward* corresponde directamente al valor de aportación acumulado desde α hasta el nodo en cuestión).

Para la resolución del PD se usa el procedimiento de las dos listas, una correspondiente a la etapa actual y otra correspondiente a la etapa siguiente. El programa se inicia con etapa siguiente=etapa final (ω), un vértice en que se supone que no queda nada por secuenciar y cuya aportación es cero.

Para cada vértice de la etapa siguiente, se desarrollan todos los estados (vértices) anteriores que pueden desencadenar el estado actual. Se calcula la aportación del arco, se le suma la aportación del vértice sucesor y se guarda en la etapa actual. Cada vez que se desarrolla un vértice hay que comprobar si éste ya existía: en caso negativo hay que crearlo, en caso positivo, hay que comprobar la cota: si es mejor, se sustituye el existente por el nuevo, y si es peor, se descarta.

Cuando no quedan más vértices por desarrollar en la etapa siguiente, se iguala la etapa actual a la etapa siguiente y la etapa siguiente a conjunto vacío, para repetir el proceso.

Este procedimiento tiene como inconvenientes la gran cantidad de estados a almacenar (ya que será interesante guardar todos los vértices generados, así como su aportación, tanto para su uso dentro del propio procedimiento, como para uso posterior en el Branch-and-bound), así como la gran cantidad de vértices a explorar (al ser un procedimiento exacto, a no ser que se encuentre una manera de acotar los vértices a explorar, deben explorarse todos).

El primer problema puede paliarse mediante el uso de un código unívoco que represente lo que se lleva secuenciado de cada producto, que lleve a su vez asociado un valor de aportación. Este código puede representarse por un número entero, como se detalla en el apartado 6.1.3.

El segundo problema puede paliarse aplicando cotas que permitan determinar si el desarrollo de un vértice puede llevar o no a una solución mejor que una ya obtenida, evitando el desarrollo del vértice en caso contrario. Entonces, antes de calcular las cotas por separabilidad es interesante haber calculado alguna de las cotas superiores (ver el apartado 5.2), para tener un valor de cota superior que llamaremos UB. Al estudiar un vértice, si el valor de aportación de éste es igual o superior a UB, no es necesario desarrollarlo, ya que ningún estado equivalente durante el proceso de enumeración llevará a una solución mejor.

Pero esta cota inferior es muy poco restrictiva, ya que sólo se está calculando la cota para 2 o 3 opciones, y la aportación de 2 o 3 opciones será pequeña en comparación a una cota superior del problema completo. Para convertir la cota superior del problema global en una cota superior del subproblema que se está trabajando, es posible restarle a UB la mejor cota correspondiente a las opciones que no están presentes en el subproblema actual. Por ejemplo, si en un problema de 5 opciones se está calculando la cota del subproblema correspondiente a las opciones 2, 3 y 4, UB puede mejorarse restándole la cota correspondiente al subproblema con las opciones 1 y 5. El UB mejorado se identificará como UB'.

Con el objetivo de restringir aún más esta cota, se utiliza, como ya se ha adelantado en el apartado anterior, la cota por simetría. Sin embargo, ésta no puede aplicarse a todos los nodos, ya que se requieren los valores de aportación para los instantes desde T hasta $T/2$. Estos instantes corresponden a las primeras $T/2$ etapas en desarrollarse, siguiendo un procedimiento backwards. A partir de esta etapa, la cota parcial que se usa para decidir si un nodo se almacena o no, se calcula sumando la aportación desde α hasta el estado, un término con la aportación correspondiente a lo que queda por secuenciar (que es igual a la aportación desde α hasta el nodo complementario) y la aportación del estado que se está estudiando.

Como hasta la etapa $T/2$, no puede aplicarse la cota por simetría, pero sí puede calcularse una cota de lo queda hasta llegar a ω mediante las cotas de una sola opción o de integralidad, se aplican aquellas cotas que estén disponibles, tal como se detalla al final de este apartado. Este valor de cota siempre será igual o peor que el que dará la cota por simetría (ya que la cota por simetría siempre da el valor óptimo), por lo que sólo se aplicará en las primeras $T/2$ etapas. En el hipotético caso que un estado fuera descartado por cota, tal como se verá al final del apartado, no afectaría el resto de cálculos.

En este procedimiento de programación dinámica se calcula la cota en dos ocasiones: cuando se pretende decidir si es eficiente continuar la búsqueda a partir de un estado determinado, y cuando se está explorando un estado que ya existe en el grafo, y sólo se debe almacenar el mejor nodo. En ambos casos, si el procedimiento se encuentra en una etapa posterior a $T/2$, se añade a la cota un término correspondiente a los nodos que quedan por desarrollar hasta ω.

Este término es el valor de sdq almacenado para el estado complementario (el nodo con la composición inversa, tal y como se ha detallado en el apartado 4.1.). Si el nodo complementario no fue almacenado, porque su valor de cota era superior a la mejor solución hasta el momento, entonces el nodo actual no debe desarrollarse, ya que se ha comprobado con anterioridad (al haber eliminado el estado complementario) que ningún estado con esa composición (total o parcial) puede llevar a una mejora en la solución conocida.

El algoritmo 1 presenta el pseudocódigo utilizado para la resolución de los subproblemas por separación de opciones.

Algoritmo 1: PD

Estado inicial=Plan de producción

A(Estado inicial)=0

Lista 1={Estado inicial}

Lista 2={∅}

Para cada instante *t*, de *T*-1 a 0

Para cada vértice *v* en la Lista 1

Para cada modelo m, cuya composición en el vértice *v* sea mayor que 0

Crear *Nuevo Vértice* con una unidad menos de modelo *m*, calcular aportación hasta etapa *t*+1 (A)

Buscar el *Nuevo Vértice* en la Lista 2

Si existe **Y** A menor a A (*Vértice almacenado*)

Cambiar A(*Vértice almacenado*)

Si no existe

Si (*t*≤*T*/2)**Y**(A ($\overline{Nuevo\ Vértice}$)+A(*Nuevo Vértice*))<UB

Guardar el *Nuevo Vértice* en la lista 2

Fin Si

Si (*t*>*T*/2)**Y**(A(*Nuevo Vértice*)+CotaOps($\overline{Nuevo\ Vértice}$)<UB)

Guardar el *Nuevo Vértice* en la lista 2

Fin Si

Fin Si

Fin Para

Fin Para

Lista 1=Lista 2

Lista 2={0}

Fin Para

Donde la operación **guardar** almacena en memoria el estado y su aportación acumulada, A; **A (*Vértice*)** indica el valor de aportación acumulada almacenado en memoria para el estado *Vértice*; **UB** indica la cota superior; y finalmente, **CotaOps(*Vértice*)** indica la cota inferior asociado al *Vértice* utilizando las cotas obtenidas mediante separabilidad de opciones.

CotaOps se basa en que una combinación de subproblemas genera una cota si no coincide ninguna de las opciones para ninguno de los subproblemas y se trata de una cota restrictiva si, además, cubre todas las opciones que representa el problema. Por ejemplo: la combinación entre el subproblema con las opciones 2 y 3, y el subproblema con las opciones 1, 4 y 5, genera una cota, que además es más restrictiva que la que genera la combinación de los subproblemas de opciones 2 y 3 y el de las opciones 1 y 4. Por el contrario, combinando los subproblemas con opciones 1, 2 y 3, y el que incluye las opciones 3, 4 y 5, no se generaría una cota, ya que la opción 3 se ha contado dos veces.

Estas combinaciones que generan una cota se usarán también al calcular cotas parciales para un vértice (solución parcial) en el branch-and-bound.

Para poder utilizar este tipo de cota en el cálculo de subproblemas con más opciones y del problema global, será necesario almacenar las listas generadas durante el algoritmo 1 para su consulta rápida cuando sea conveniente.

Para los problemas de hasta aproximadamente 7 opciones y 200 unidades, es viable almacenar todos los grafos asociados a los subproblemas y todas las combinaciones que generan esta cota con 2 o 3 opciones por subproblema. Para los problemas de 8 o más opciones resulta ineficiente y, en muchos casos, inviable por problemas de memoria, calcular cotas parciales para todas las combinaciones posibles de subproblemas, ya que el número de combinaciones puede ser muy elevado. Lo que se propone es, para los problemas de más de 7 opciones, calcular solamente las cotas de una sola opción, y aplicar éstas para el resto de cálculos.

5.1.4. Cota por asimilación a un problema de transporte

Como se ha expuesto en la modelización matemática, el problema comparte similitudes con un problema de transporte. El problema de transporte es un problema en que se pretende minimizar los costes de transportar unos productos desde una serie de puntos de oferta hasta unos puntos de demanda. El modelo lineal es el siguiente:

Datos:

n puntos de oferta

F=[F_1,F_2,…,F_n] Producción de cada punto de oferta

m puntos de demanda

M=[M_1,M_2,…,M_m] Demanda de cada punto de demanda

C matriz *nxm* de precios. El componente C_{ij} de la matriz corresponde al precio de transportar una unidad de producto del punto de oferta *i* hasta el punto de demanda *j*

Variables:

X matriz solución *nxm*, donde el componente X_{ij} corresponde a la cantidad de producto que debe transportarse del punto *i* al punto *j*

Función objetivo:

$$[MIN] \sum_{i=1}^{n} \sum_{j=1}^{m} X_{ij} \cdot C_{ij} \qquad (18)$$

Donde (18) minimiza los costes de transporte totales.

Restricciones:

$$\sum_{i=1}^{n} X_{ij} \geq M_j \; ; \; \forall j = 1, \dots, m \qquad (19)$$

La restricción (19) asegura que la demanda se cubre para todos los puntos de demanda.

$$\sum_{j=1}^{m} X_{ij} \leq F_i \; ; \; \forall i = 1, \dots, n \qquad (20)$$

La restricción (20) asegura que la cantidad total de producto que se transporta no es superior a la cantidad que se produce en los puntos de oferta.

$$X_{ij} \geq 0;\ \forall i = 1, \dots, n;\ \forall j = 1, \dots, m \qquad (21)$$

Y la restricción (21) asegura que todas las cantidades que se transportan son positivas.

Si se cumple que $\sum M_j = \sum F_i$, entonces las desigualdades de las restricciones (19) y (20) pueden sustituirse por igualdades estrictas.

Considérese ahora la siguiente adaptación del problema de transporte al problema de secuencias regulares:

Datos:

n número de modelos

P=[P_1,P_2,…,P_n]: Plan de producción por modelo

T número total de productos a secuenciar (se secuencia un producto por instante, por tanto, *T*=número de instantes)

C matriz de "costes" *n*x*T*. La componente C_{it} corresponde al "coste" de secuenciar un modelo tipo *i* en el instante *t*

Variables:

X matriz solución *n*x*T*. La componente X_{it} indica si el modelo *i* se secuencia en la posición *t*

Función objetivo:

$$[MIN] \sum_{i=1}^{n} \sum_{t=1}^{T} X_{it} \cdot C_{it} \qquad (21)$$

Restricciones:

$$X_{it} \geq 0;\ \forall i = 1, \dots, n;\ \forall t = 1, \dots, T \qquad (22)$$

$$\sum_{i=1}^{n} X_{it} = 1;\ \forall t = 1, \dots, T \qquad (23)$$

$$\sum_{t=1}^{T} X_{it} = P_i;\ \forall i = 1, \dots, n \qquad (24)$$

$$X_{it} \in \{0,1\};\ \forall i = 1, \dots, n;\ \forall t = 1, \dots, T \qquad (25)$$

Estas restricciones aseguran que la cantidad de modelos secuenciados cada instante sea siempre no negativa (22) y que en cada instante sólo se secuencie un modelo (23). Se añade una tercera restricción que asegure que al llegar a la última etapa, todos los modelos cumplan con el plan de producción (24). Por último, es necesario que todos los componentes de la matriz X sean binarios.

Puede apreciarse que la ecuación (19) corresponde a la ecuación (23), en el caso particular en que cada demandante requiere una única unidad de producto y que la ecuación (24) es idéntica a la ecuación (20) si se cumple en igualdad; tal como sucede en este caso ya que la suma de unidades a producir es igual al número de instantes de secuenciación. Como la ecuación (25) se cumplirá en todas las soluciones al modelo de transporte, las ecuaciones (21)-(25) son equivalentes a un problema de transporte (18)-(20).

Vemos entonces que el problema puede asimilarse a un problema de transporte con la salvedad de que la función objetivo del ORV no puede expresarse con la misma estructura que la del modelo de transporte. Esto nos obliga a buscar una cota de los coeficientes de coste que nos permita utilizar el modelo anterior de transporte para resolver el problema (en este caso, aportando una cota, ya que la función objetivo derivada es, a su vez, una cota de la función original).

Un primer método para obtener unas cotas de C_{it} consiste en igualarlas a las aportaciones mínimas asociadas a secuenciar un modelo *i* en el instante *t*. El problema es que resulta muy poco restrictiva, ya que ésta siempre corresponderá a que el valor de consumo total para cada opción sea igual al valor entero más cercano respecto a la tasa ideal acumulada del instante. Además, este cálculo no toma en cuenta la secuencia, por lo que, excepto en contadas ocasiones al principio de la secuencia, siempre es posible encontrar una combinación de unidades secuenciadas tales que se llega al instante *t*-1 con unos consumos de opciones que ocasionan una desviación mínima al secuenciar el modelo i en el instante *t*. Por tanto, el problema de transporte con los coeficientes C_{it} obtenidos según la técnica anterior (excepto pequeñas variaciones al principio de la secuencia), daría como cota un valor cercano al obtenido por las cotas de integralidad.

Uno de los defectos de la cota anterior está asociado a ignorar que es probable que, para llegar a una situación ideal en el instante *t*, se proceda de una situación peor, o que si se parte de una situación ideal se llegue a una situación peor. Para intentar paliar este defecto se propone un método de cálculo para los coeficientes que combine las mejores aportaciones combinadas para dos instantes consecutivos de la secuencia.

El método calculará para cada instante de tiempo y modelo la aportación asociada a secuenciar la unidad en un instante *t* más la aportación del instante *t*-1. Entre todas las combinaciones posibles, se seleccionará la de mínimo coste y C_{it} se igualará a la mitad de dicho valor, para tener en cuenta que se suma la aportación de dos instantes en vez de uno.

Pongamos un ejemplo para un modelo con dos opciones, tasa ideal de 0,3 y 0,4 respectivamente, calculando la aportación ideal para la posición 16 de la secuencia de un modelo con c_{1i} =1, c_{2i}=0.

En la cota básica el consumo ideal sería 0,3·16=4,8 para la opción uno y 0,4·16=6,4 para la opción dos, siendo, por tanto la posición ideal para secuenciar el modelo en el instante 16 sería que el consumo acumulado de opciones en el instante 15 fuera [4,6], ya que así el consumo acumulado en el instante 16 sería [5,6]. Además, siendo el consumo de opciones del instante 15 factible, el valor que podría otorgarse al coeficiente $c_{1,16}$ considerando únicamente la posición *t* de la secuencia sería:

$(5-4,8)^2 +(6-6,4)^2 =0,2$

En el caso de tener en cuenta las dos posiciones de la secuencia, en la posición 16 nos encontraríamos, idealmente, en una de estas cuatro situaciones de consumo acumulado de recursos: (4,6) ; (5,6) ; (4,7) ; (5,7).

Si bien es cierto que en esta posición hay más situaciones posibles respecto a los consumos acumulados, cualquiera de ellas resultaría en un valor de sdq peor. Puede demostrarse que el óptimo debe encontrarse entre dos valores, tal como se muestra posteriormente.

Analizando los casos veríamos que:

Si en el instante 16 se tiene un consumo acumulado de (4,6) en el instante 15 se tendrá un consumo acumulado de (3,6)

$[(4-4,8)^2+(6-6,4)^2]+ [(3-4,5)^2+(6-6)^2]=[0,64+0,16]+[2,25+0]=3,05$

Si en el instante 16 se tiene un consumo acumulado de (5,6) en el instante 15 se tendrá un consumo acumulado de (4,6)

$[(5-4,8)^2+(6-6,4)^2]+ [(4-4,5)^2+(6-6)^2]=[0,04+0,16]+[0,25+0]=0,45$

Si en el instante 16 se tiene un consumo acumulado de (4,7) en el instante 15 se tendrá un consumo acumulado de (3,7)

$[(4-4,8)^2+(7-6,4)^2]+ [(3-4,5)^2+(7-6)^2]=[0,64+0,36]+[2,25+1]=4,25$

Si en el instante 16 se tiene un consumo acumulado de (5,7) en el instante 15 se tendrá un consumo acumulado de (4,7)

$[(5-4,8)^2+(7-6,4)^2]+ [(4-4,5)^2+(7-6)^2]=[0,04+0,36]+[0,25+1]=1,65$

El mínimo entre todos ellos es 0,45, que sería el valor que tomaría el coeficiente $c_{1,16}$ (más alto que el 0,2 original).

Estos coeficientes tienen dos problemas principales para su uso:

- Tienen en cuenta dos posiciones de la secuencia consecutiva. Esto puede arreglarse dividiendo por dos el valor de la cota obtenida por el problema de transporte (0,45/2 sigue siendo superior a 0,2).

- Requieren un mayor número de operaciones para obtener el mismo índice. Para simplificar los cálculos, pueden utilizarse dos propiedades:

  - Cada opción puede calcularse independientemente y reaprovecharse en modelos con el mismo consumo de una opción debido a la propiedad de separabilidad.
  - Es posible limitar a priori el rango de valores de consumo acumulado de una opción. El cálculo de los límites se muestra a continuación.

Primero se define una variable *Y*, que indica el uso acumulado de una opción concreta *j* hasta la etapa *t* (incluida); por tanto, la aportación a la función objetivo del problema de la etapa *t* por parte den la opción *p* puede expresarse como (26)

$$\left(Y - r_j \cdot t\right)^2 \tag{26}$$

Esta función es continua, derivable, véase (27) y tiene un único óptimo.

$$\left(\left(Y - r_j \cdot t\right)^2\right)' = 2 \cdot \left(Y - r_j \cdot t\right) \tag{27}$$

Como se observa, puede aislarse *Y* de la fórmula obtenida al derivar. Haciendo la segunda derivada se observa que el óptimo encontrado es un mínimo:

$$\left(2 \cdot \left(Y - r_j \cdot t\right)\right)' = 2 > 0 \tag{28}$$

Sea el mínimo de (26) *Y**. Como el valor de consumo acumulado debe ser un valor entero y la función (26) es continua y tiene un único mínimo, el óptimo si Y debe ser entero sería $\lfloor Y^* \rfloor$ o bien $\lceil Y^* \rceil$.

Segundo, se analiza el valor del coeficiente C_{it} dado un valor de Y, véase (29).

$$\left(Y - r_j \cdot t\right)^2 + \left(Y_{t+1} - r_j \cdot (t+1)\right)^2 \tag{29}$$

Esta función, en caso de querer optimizarse, tiene una única variable, ya que el valor de Y_{t+1} está relacionado con el valor de *Y*: o bien $Y_{t+1}=Y$ ó $Y_{t+1}=Y+c_{ij}$, donde c_{ij} es el consumo de la opción *j* por parte del modelo *i*, que ha sido secuenciado en el instante *t*+1.

Así pues, ahora tenemos una formulación para C_{it} que sólo tiene una variable (30):

$$\left(Y - r_j \cdot t\right)^2 + \left(Y - r_j \cdot (t+1) + c_{ij}\right)^2 \quad (30)$$

Derivando e igualando a cero, se obtiene el valor óptimo de Y

$$Y = r_j \cdot t - (r_j + c_{ij})/2 \quad (31)$$

Como anteriormente se ha demostrado que $\left(Y - r_j \cdot t\right)^2$ es una función convexa, y, dado que c_{ij} es una constante, $\left(Y - r_j \cdot (t+1) + c_{ij}\right)^2$ también es convexa. Por tanto, sabiendo que la suma de dos funciones convexas en **R** es también convexa, se puede afirmar que la función que calcula el coeficiente C_{it} es convexa. Como tal, también se cumplirá que el óptimo debe valer $\lfloor Y^* \rfloor$ ó $\lceil Y^* \rceil$. Por tanto, sólo será necesario estudiar las situaciones en que el consumo acumulado sea $\left\lfloor r_j \cdot t - \frac{r_j + c_{ij}}{2} \right\rfloor$ ó $\left\lceil r_j \cdot t - \frac{r_j + c_{ij}}{2} \right\rceil$. Al primero le llamaremos límite inferior y al segundo le llamaremos límite superior.

Hay un que tener en cuenta un caso específico, que es el caso de los coeficientes correspondientes a la etapa *T*, que valdrán 0 por definición.

Resumiendo, el cálculo de los componentes de la matriz de costes puede realizarse mediante el siguiente algoritmo:

Algoritmo 2: Calcular C_{it}

C_{it}=0

Para cada opción *j*

C_{it}=C_{it}+Coste$_{itj}$

Fin Para

C_{it}=C_{it}/2

Y el coste de secuenciación por opción se obtiene mediante el siguiente algoritmo (donde la c_{ij} indica el consumo de opción *j* por parte del modelo *i*, y r_j es la tasa ideal de consumo para la opción *j*):

Algoritmo 3: Calcular Coste$_{itj}$

Coste$_{itj}$=Infinito

Para cada valor *o* entre Límite inferior y Límite superior

$Coste_{itj}$=min($Coste_{itj}$; aportación(*i,t,j,o*))

Fin Para

La aportación(*i*,*t*,*j*,*o*) da como valor la aportación para la opción *j* si se secuencia el modelo *i* en la posición *t*, teniendo en cuenta que se alcanza una composición *o*.

Una vez calculada la matriz de costes, solucionando el problema de transporte asociado, se obtiene una cota para el problema de ORV. El procedimiento empleado para la resolución del problema de transporte se basa en la eliminación de circuitos negativos en un grafo auxiliar de transporte como se detalla en el apartado 6.1.4.

5.1.5. Uso de cotas inferiores por parte del resto de procedimientos

En el apartado 5.1 se han visto tres cotas inferiores que se utilizan en diferentes etapas de la resolución del problema. Con el fin de aclarar su participación en cada elemento del algoritmo final, se detallan los procedimientos en que interviene cada cota, así como alguna de sus particularidades.

Cota por simetría: La cota por simetría únicamente se utiliza en el cálculo de las cotas por separabilidad, asociadas a la resolución óptima de un subconjunto de opciones. Cabe destacar que no se trata exactamente de una cota ya que aporta el valor exacto del remanente de la solución.

Cotas por separabilidad: Esta cota es la utilizada con mayor frecuencia. Se usa al principio del algoritmo, en el procedimiento de branch-and-bound, en la programación dinámica acotada, e incluso en el cálculo de cotas por separabilidad de más de una opción, en que se usan las cotas calculadas previamente de cada opción por separado. Si se pretende utilizar esta cota durante un procedimiento enumerativo, es necesario mantener en memoria todos los estados usados para calcular su valor, por tanto es conveniente eliminar la mayor cantidad posible de estados a almacenar.

Para su utilización, y dada una composición del problema general, se calcula la composición equivalente del subproblema de forma análoga a cómo se genera el subproblema y se busca el valor de aportación acumulado asociado (dado un estado se busca el valor de aportación acumulado correspondiente a esa composición para cada uno de los subproblemas posibles). Después se combinan estos valores según las opciones usadas (por ejemplo, en un problema con cinco opciones, el valor del subproblema de opciones 1 y 2, puede combinarse con el valor obtenido para las opciones 3, 4 y 5, pero no con el valor de las opciones 2 y 3) y finalmente se escoge el mayor valor de todas estas combinaciones.

En caso que el problema tenga siete o más opciones, sólo se comprueba una combinación: la de todas las cotas de una sola opción, pero por lo demás, el procedimiento es el mismo.

También hay que tener en cuenta que puede darse el caso que el nodo correspondiente al vértice complementario no exista en el árbol, en tal caso, la cota para ese nodo era mayor que la cota superior, y por tanto, el nodo no se almacenó árbol. Si esto sucede, el nodo puede ser podado.

Cota de transporte: Esta cota se utiliza al principio del algoritmo y en el procedimiento de Branch-and-bound. No se utiliza en el programa dinámico porque requiere un mayor tiempo de cálculo que la cota por separabilidad. Para su cálculo se parte de la solución parcial ya encontrada y se calculan únicamente los costes para los instantes de secuencia aún no fijados.

5.2. Cotas superiores

5.2.1. Procedimiento *Goal-Chasing*

Para obtener una primera solución, que servirá de cota superior para ahorrar cálculos en procedimientos posteriores, se aplica una heurística *goal-chasing*, un procedimiento constructivo muy sencillo desarrollado por Monden (1983) que secuencia según una regla de prioridad por menor aportación, como indica el algoritmo 4.

Tómese como ejemplo la instancia usada en el apartado anterior, en que se tiene 1 opción, 2 modelos y el plan de producción es de [4,2].

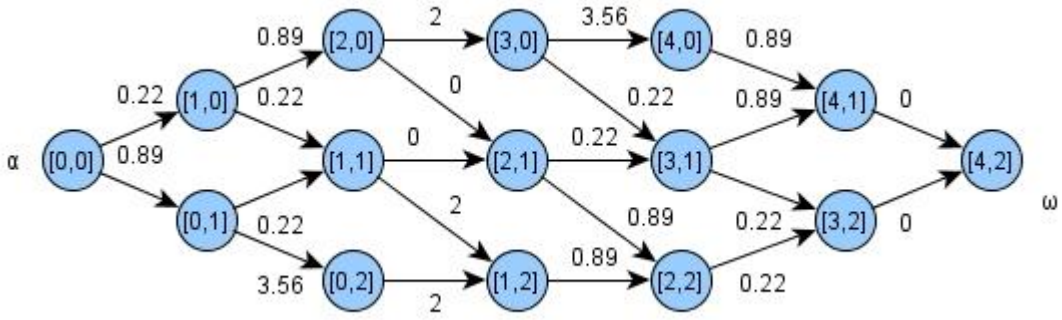

Figura 7: Ejemplo de grafo forward

Para este caso, el procedimiento *goal-chasing* empieza secuenciando una unidad de modelo 1 en la primera posición, ya que ese caso tiene una aportación de 0.22, frente a una aportación de 0.89 de secuenciar el modelo 2. En el segundo paso, se secuencia una unidad de modelo 2, con una aportación de 0.22, ya que es la más baja de las posibles. Haciendo esto sucesivamente se obtiene como solución la siguiente secuencia: (1,2,1,1,2,1), con una aportación de 0.88.

Algoritmo 4. *Goal-Chasing*

Secuencia={0}

sdq=0

Demanda remanente=Plan de producción

Para cada instante *t*

Mejor aportación=Infinito

Elegido=0

Para todo modelo *i*

Si (Demanda remanente(*i*)>0)&&(Aportación(*t,i*)<Mejor aportación)

Mejor aportación=Aportación(*t,i*)

Elegido=*i*

Fin Si

Fin Para

Secuencia(*t*)=Elegido

sdq=sdq+Mejor aportación

Demanda remanente(Elegido)= Demanda remanente(Elegido)-1

Fin Para

5.2.2. Programación dinámica acotada

Ya se ha remarcado en apartados anteriores que intentar obtener una solución al problema completo mediante programación dinámica sería inviable para problemas a partir de cierto tamaño. Esto se debe al aumento exponencial de vértices a explorar conforme aumenta el número de modelos distintos a secuenciar y la cantidad de productos total.

Se puede considerar, entonces, reducir el número de estados que se desarrollan, para disminuir así el tiempo de computación, convirtiendo el procedimiento en una heurística. Esto, por supuesto, implica que no se puede demostrar la optimalidad de las soluciones obtenidas en todos los casos.

Para ello se aplica un procedimiento de programación dinámica similar al usado para calcular las cotas por separabilidad. En este caso, sin embargo, no se desarrollarán todos los estados de una etapa, sino que se declarará un *n*: "ancho de ventana" que indicará el número máximo de estados de una etapa para los cuales se analizarán las transiciones (acción normalmente conocida como desarrollar un estado). Al no desarrollar todos los estados del programa dinámico el procedimiento deja de ser exacto en todos los casos. Además, se requiere un sistema para ordenar los estados de una etapa, escogiendo sólo los *n* primeros. En el caso del procedimiento diseñado, los estados se ordenan según la cota parcial, como se detalla en el apartado 6.2.2. El procedimiento, conocido como programa dinámico acotado o BDP, sigue los pasos descritos en el algoritmo 5.

Otra diferencia de este procedimiento con el que se ha aplicado para la resolución de los subproblemas de opciones separadas es que la BDP es un procedimiento *forward*. El procedimiento se basa igualmente en el método de las dos listas, pero en este caso se empieza por la etapa 0 y el estado α, correspondiente a un estado en que no hay nada secuenciado. A partir de este punto, se desarrolla cada estado, calculando su cota, seleccionando de cada etapa únicamente *n* estados y repitiendo el proceso hasta llegar a la etapa final, con un único estado ω, con una composición igual al plan de producción.

Algoritmo 5: BDP con ancho de ventana *n*

Vértice inicial.clave={0}

Vértice inicial.sdq=0

Lista ={∅}

Guardar el Vértice inicial en la pila

Para cada instante *t*, de 1 a *T*

Mientras la pila no esté vacía

Seleccionar y eliminar el primer elemento de la pila, que pasa a ser el vértice *v*

Para cada modelo *i*, si en el vértice *v* no ha cumplido con el plan de producción

Crear Nuevo Vértice con una unidad más de modelo *i*

Añadir a Aport.Acum(Vértice) la aportación del modelo *i* en el instante *t*

Buscar el Nuevo Vértice en la Lista

Si existe **Y** (Aport. Acum. (Nuevo Vértice) < Aport. Acum. (Vértice existente))

Cambiar Aport. Acum. (Vértice existente)

Si no existe

Si (Aport. Acum (Nuevo Vértice)+Cota inferior(Nuevo Vértice) <=UB)

Guardar el Vértice Nuevo en la Lista

Fin Si

Fin Si

Fin Para

Fin Mientras

Si se han explorado hasta *n* vértices

Guardar toda la Lista en la pila

Si no

Guardar en la pila los *n* nodos con mejor cota

Fin Si

Fin Para

Donde **Aport.Acum. (Vértice)** corresponde a la aportación acumulada del estado, y **UB** equivale al valor de la mejor solución conocida hasta el momento.

Inicialmente, se utiliza una cota inferior trivial igual a cero para cualquier estado, pero para restringir aún más el espacio de soluciones exploradas es posible aplicar de nuevo un programa dinámico acotado una vez se han calculado las cotas de los subproblemas por opciones. En tal caso, el valor de cota puede calcularse tal y como se ha indicado en el apartado 5.1.5.

Existe la posibilidad que al final de una etapa el número de estados generados para la etapa siguiente sea cero. En tal caso, podemos abortar el procedimiento ya que ha sido incapaz de mejorar la mejor solución conocida hasta el momento.

5.3. Reglas de dominancia

Para detectar soluciones parciales dominadas durante el proceso de enumeración y evitar su desarrollo, aumentando la eficiencia del procedimiento, se aplican diversas reglas de dominancia.

Existen dos principios en los que se basan las reglas de dominancia para este problema.

El primero está asociado a subsecuencias equivalentes a un mismo estado del programa dinámico y es el fenómeno observado al desarrollar el grafo asociado a un programa dinámico, cuando se llega a un mismo estado desde diversos estados. En un programa dinámico, no se pueden tener estados repetidos, por lo que solamente se guarda el mejor de éstos. En un procedimiento de enumeración será necesario detectar este fenómeno cuando suceda.

Supongamos un ejemplo de solución parcial en que se han secuenciado 5 modelos (por tanto, estamos en la etapa 5), en el siguiente orden: (1,2,1,4,1). Ahora consideremos otro ejemplo en que, también en la etapa 5, se hubiera obtenido esta solución parcial: (1,4,1,2,1), y que tiene el mismo valor de sdq que la anterior. Se observa que, dado que el vector de composición de la solución es el mismo para ambos casos (y corresponde a [3,1,0,1]), por lo que estas dos secuencias corresponden a un mismo estado del PD del problema.

Si una de estas secuencias presentara mejor sdq que la otra, se dice que la primera domina la segunda, y sólo se desarrolla el árbol a partir de la primera solución parcial. El hecho de que ambas tengan el mismo valor de función objetivo significa que sólo es necesario desarrollar una de las dos, ya que con ambas se obtendrá una solución igual de buena. Para escoger entre ambas se puede usar, por ejemplo, un criterio lexicográfico de desempate. En este caso, el criterio lexicográfico empleado utiliza la siguiente regla: se busca la primera posición de la secuencia en que el modelo secuenciado difiere. Se dice que la secuencia cuyo modelo esté identificado por un valor menor domina a la otra.

El segundo está asociado a la propiedad de simetría. Dado que una secuencia es idéntica a su simétrica a todos los efectos, sólo es necesario explorar una de las dos.

Las reglas de dominancia se comprobarán en el branch-and-bound una vez hayan sido creados los vértices descendientes y sólo se comprueba para aquellos que no hayan sido eliminados por cota inferior (es decir, los vértices que cumplan que su cota parcial sea inferior a la mejor cota superior). Si se comprueba que el vértice no está dominado, se pasa a la siguiente etapa del algoritmo para generar los descendientes del vértice. En caso contrario, el vértice no se desarrolla.

5.3.1. Dominancia por *Goal-Chasing*

La primera regla de dominancia consiste en construir soluciones parciales que correspondan al mismo estado del programa dinámico mediante el procedimiento heurístico *goal-chasing*, detallado en el apartado 5.2.1.

Para ello, se crea un nuevo problema que sólo tiene en cuenta los modelos secuenciados hasta el momento, y se resuelve el problema mediante *goal-chasing*, obteniendo, por tanto, otra secuencia con la misma composición.

Si la secuencia original está dominada por la nueva, se poda del árbol. En caso contrario, se pueden intentar encontrar otras soluciones potencialmente dominantes mediante *goal-chasing*, fijando parte de la solución para que sea igual a la secuencia del vértice explorado.

Por ejemplo, supongamos que nos encontramos en la etapa 4 del Branch-and-bound con la subsecuencia (1,2,1,4) y al desarrollar los nodos descendientes, el nodo con la secuencia (1,2,1,4,1) no ha sido descartado por cota. El subproblema que deberá resolverse mediante *goal-chasing* para encontrar posibles soluciones que dominen la primera contendrá cinco unidades y tendrá el siguiente plan de producción: [3,1,0,1].

Inicialmente, se resuelve el problema siguiendo el procedimiento descrito anteriormente. Si tras la resolución, la solución original no ha sido dominada, aún es posible buscar otras combinaciones de modelos que, correspondiendo al mismo nodo en el grafo de soluciones, tengan mejor valor de sdq. Para ello, se puede volver a solucionar el subproblema pero fijando la primera posición de la secuencia, para que sea igual que la de la solución parcial que se está explorando. Esto se repite sucesivamente, fijando una posición más en cada paso, hasta que se obtiene una solución dominante o se llega al final de la secuencia sin haber dominado la solución. El algoritmo 6, presentado a continuación, muestra el pseudo-código de la dominancia indicada.

Algoritmo 5: Dominancia *Goal-Chasing*

Crear el problema a resolver con la secuencia a evaluar

Para todo instante t, de t=0 a t=T-2

Fijar las t primeras posiciones de la secuencia nueva

Solucionar el problema de *goal-chasing* asociado, según el algoritmo 4, teniendo en cuenta al parte de subsecuencia fijada

Si (sdq(Nueva Secuencia)<sdq(Secuencia Evaluación))

Vértice Dominado

FIN

Si ((sdq(Nueva Secuencia)==sdq(Secuencia Evaluación))**Y**(lexicográfico(Nueva Secuencia) < lexicográfico(Secuencia Evaluación)))

Vértice Dominado

FIN

Fin Si

Fin Para

Vértice no Dominado

FIN

Donde **sdq(secuencia)** devuelve el valor de función objetivo de la secuencia, y **lexicográfico(secuencia)** devuelve un valor que puede compararse con cualquier otro valor de la función lexicográfico para determinar qué secuencia tiene mejor valor.

Continuando con el ejemplo anterior, la primera secuencia obtenida podría ser (2,1,4,1,1), con una sdq mayor que la del vértice que se explora, con lo que la secuencia no estaría dominado. En ese caso, se fijaría la primera posición de la secuencia solución, de modo que el procedimiento por *goal-chasing* debe dar como solución una secuencia que empiece por 1. Entonces se podría obtener, por ejemplo, la secuencia (1,4,1,2,1). Si esta secuencia tiene mejor sdq que la de la secuencia original, el vértice deja de explorarse, si no, se fijarían las dos primeras posiciones y así sucesivamente.

Adicionalmente, es necesario asegurar que no se comprueban secuencias que ya coincidían con la secuencia original. Por ejemplo, si se parte de la secuencia (1,3,2,5,4,3) y el procedimiento *goal-chasing* da como secuencia (1,3,2,4,3,5), está claro que fijar la primera posición no sería suficiente para generar una nueva secuencia, y sería conveniente fijar las cuatro primeras posiciones en un único paso del algoritmo. El algoritmo implementado tiene en cuenta este detalle.

5.3.2. Árbol de nodos explorados

Esta regla de dominancia consiste en identificar soluciones parciales que contengan la misma composición de modelos que una solución previamente investigada. Si se cumple que la composición de la secuencia de la primera solución parcial es idéntica a la composición de la segunda, y la sdq de la primera es igual o inferior a la segunda, se puede afirmar que la segunda solución parcial está dominada por la primera y, por tanto, no es necesario desarrollarla.

Para realizar la comprobación de esta dominancia se necesita almacenar las composiciones de modelos secuenciados por todas las soluciones parciales exploradas hasta el momento, así como la sdq de estas subsecuencias. Esta información se almacena conforme se desarrolla el árbol de soluciones, de manera que si una cierta composición no ha sido explorada, siempre se almacenará en el árbol de nodos explorados, junto con su sdq. Si es una composición que ya existe, se comprueba si la sdq es mejor o peor que la que hay almacenada: en el primer caso, se cambia el valor de sdq almacenado y se continúa el desarrollo de la rama; en el segundo caso, se poda el vértice actual.

El funcionamiento de este procedimiento puede verse en el algoritmo 6.

Algoritmo 6: Árbol de nodos explorados (comprobación para el vértice *v*)

Calcular la composición del vértice *v*

Comprobar si el nodo que corresponde a esa composición se ha almacenado

Si existe

Si (Valor(Nodo Evaluado) ≥ Valor (Nodo Existente))

Vértice Dominado

FIN

Si (Valor(Nodo Evaluado) < Valor (Nodo Existente))

Valor (Nodo Existente)=Valor (Nodo Evaluado)

Vértice No Dominado

FIN

Fin Si

Si no existe

Almacenar Nodo Evaluado

Vértice No Dominado

FIN

Fin Si

5.3.3. Dominancia por simetrías

El árbol de vértices desarrollados permite, además, aplicar otra regla de dominancia que aprovecha la simetría del problema. Sabiendo que para un vértice cualquiera *A* la aportación corregida de su vértice complementario $\bar{A}$ corresponde a la aportación de lo que queda por explorar desde A hasta ω, se puede evitar generar estos vértices mediante dos reglas de dominancia que se exponen a continuación.

La primera regla se basa en el árbol de nodos explorados. Cada vez que se añada al árbol de vértices desarrollados un vértice *A* correspondiente a una etapa superior a *T*/2, se comprobará si el vértice complementario $\bar{A}$ ha sido desarrollado y almacenado anteriormente . Si es así, no es necesario seguir desarrollando el vértice actual, ya que la mejor secuencia desde *A* hasta la etapa *T* será la mejor secuencia desde la etapa 0 hasta el vértice $\bar{A}$, y tendrán la misma aportación acumulada. Si la etapa complementaria ($\bar{t}$) ha sido explorada por completo, entonces puede asegurarse que se ha encontrado la secuencia óptima entre el inicio y $\bar{A}$, y, por tanto, se ha obtenido una nueva solución, correspondiente a *A*+ $\bar{A}$.

Si la etapa $\bar{t}$ no ha sido explorada por completo, el vértice *A* se "congela", es decir, deja de explorarse, pero anotando que no se ha llegado aún a una solución y que el vértice no está realmente podado. Si en algún momento de la exploración, se almacena en el árbol un vértice correspondiente a una etapa anterior a *T*/2 que mejora el actual, se comprueba si existe el vértice complementario congelado, que complete la solución.

De esta manera, se reduce el número de vértices a explorar, ya que es posible que la ramificación termine dando una solución antes de explorar hasta los nodos hoja.

La segunda regla se basa en el orden de exploración de los vértices. Para reducir aún más el número de vértices por explorar, también se comprueba el valor lexicográfico de los nodos que se exploran, y se compara con el valor lexicográfico de su complementario. El orden lexicográfico se determina igual que en la heurística goal-chasing. Hasta la etapa *T*/2 sólo es necesario explorar los nodos cuyo valor lexicográfico sea mejor que el de su complementario. Eliminando así la simetría del problema.

Supongamos un ejemplo con 3 modelos, con plan de producción [3,3,2]. Si se explora el nodo de secuencia (1,2,3,1), con complementario (2,3,2,1), la solución final es exactamente igual que si se explora el nodo (1,2,3,2), con complementario (1,3,2,1). La única diferencia entre estas soluciones es el orden de exploración, por tanto no vale la pena estudiar ambas. Para evitar explorar dos veces una

solución se aplica la regla lexicográfica, con lo que sólo se exploran los nodos con mejor valor lexicográfico que su complementario. En el caso del ejemplo, (1,2,3,1) tiene mejor valor que (1,2,3,2), por lo que solamente se explorará el primero.

5.3.4. Incompatibilidades entre dominancias

Es posible que aparezcan incompatibilidades entre las reglas de dominancia de simetría y goal-chasing. Estas incompatibilidades podrían hacer que algunas soluciones no dominadas no llegaran a ser exploradas, porque las reglas de dominancia interfieren entre ellas. Un ejemplo clásico es la simetría que se expone a continuación.

Pongamos un ejemplo similar al del apartado anterior, con 3 modelos, cuyo plan de producción es [3,3,2]. Si la secuencia óptima fuera [1,2,3,2,1,2,3,1], la secuencia [1,3,2,1,2,3,2,1] tendría el mismo valor y sería interesante no explorar la segunda secuencia. Si por casualidad la subsecuencia [1,3,2,1,2] (las cinco primeras posiciones de la secuencia simétrica) tuviera mejor valor de sdq que la subsecuencia [1,2,3,2,1] (las cinco primera posiciones de la secuencia óptima) sucedería el siguiente fenómeno:

- La secuencia simétrica no se exploraría porque es preferible explorar la secuencia original y la regla de dominancia por simetría impediría su estudio.
- Si la subsecuencia original tuviera peor valor de SDQ que la subsecuencia simétrica es posible que el algoritmo de goal-chasing generara la subsecuencia simétrica y considerara que la secuencia original está dominada (Fijarse que ambas subsecuencias tienen la misma composición).

Por tanto, no se llegaría a desarrollar la secuencia óptima porque ambas se dominan entre ellas. Para solucionar este problema, durante el procedimiento de goal-chasing, es necesario comprobar en las $T/2$ primeras etapas si la subsecuencia encontrada está también dominada por simetría. Si es el caso, entonces la subsecuencia original no debe considerarse dominada. No es necesario hacer esta comprobación para las subsecuencias que dan un valor idéntico de sdq, ya que en ese caso, la regla de dominancia por goal-chasing ya elige la subsecuencia con valor lexicográfico más pequeño.

5.4. Branch-and-bound

5.4.1. Estrategia de ramificación

La estrategia de ramificación de este procedimiento se basa en múltiples búsquedas *depth-first* según una prioridad *best-first*.

Una búsqueda primero en profundidad (*depth-first*) es la que cuando se escoge un nodo para ramificar, elige el nodo de mayor profundidad en el árbol de búsqueda. Una búsqueda *best-first*

supone que el nodo que se desarrolla en cada paso del algoritmo es el que presenta un mejor valor de cota.

En el caso del procedimiento presentado, se emplea código en paralelo para desarrollar diversas búsquedas *depth-first*. De esta forma, cada uno de los procesadores o núcleos ramificará por una parte del árbol de soluciones. Habrá, entonces diversos hilos (*threads*) que trabajarán de forma paralela, excepto cuando requieran acceso a memoria, como se detallará en el apartado 6.5.

Para controlar ambos métodos de búsqueda, será necesario definir dos límites, ya sean temporales o relacionados con el número de vértices explorados. El primero dará por finalizado el procedimiento completo, mientras que el segundo detendrá cada búsqueda *depth-first*. Para evitar problemas de memoria, se deberá limitar el número de vértice intermedios que se guardan en memoria durante el proceso.

El procedimiento general se inicia con una búsqueda primero en profundidad por el árbol hasta que se llega a un nodo terminal o al límite para la búsqueda *depth-first*. En el primer caso, se almacena la nueva solución, se comprueba si es óptima y en caso contrario, se hace *backtrack* en el árbol y se sigue explorando. En el segundo caso, se crean los vértices descendientes inmediatos no explorados, se almacenan en la memoria y se hace backtrack hasta llegar al vértice inicial.

Tras esto, mientras no se supere el límite para el procedimiento general o no se llegue a una solución óptima (ya sea porque se encuentra una solución igual a la mejor cota o porque ya no quedan más vértices por explorar), cada hilo (*thread*) buscará en la pila el mejor vértice por explorar (según el criterio que se detallará a continuación) y se desarrollará el vértice seleccionado con la estrategia primero en profundidad expuesta anteriormente.

El criterio que define la búsqueda parcialmente *best-first*, véase algoritmo 8, está asociado a cómo se selecciona qué vértice debe explorarse a continuación: si la pila de vértices guardados en memoria ha llegado a tres cuartos de un límite de memoria establecido, se escogerá el vértice cuya última etapa explorada tenga un mayor valor; esto significa que quedan menos etapas por desarrollar de ese vértice, de manera que es probable que el vértice se pode (cabe recordar que un vértice es podado si su cota es superior o igual a la mejor solución conocida, o si se ha determinado que es un vértice dominado) o se desarrolle completamente su árbol de exploración, dando una nueva solución, eliminándolo así de la pila y dejando memoria libre para almacenar vértices. Si, por el contrario, no se ha llegado a tres cuartos del límite, se escoge el vértice con una mejor cota, como se detalla en el apartado correspondiente a la implementación.

5.4.2. Uso de cotas y reglas de dominancia dentro de la enumeración

El uso de cotas y reglas de dominancia se realiza antes de enumerar el siguiente vértice: para cualquier vértice no terminal (es decir, si el número de etapas desarrolladas de momento aun es menor al número de etapas total), se calcula la cota parcial correspondiente a la secuenciación de uno de los modelos de los que aun no se haya cubierto la demanda.

Para ello, a la sdq que se tendría hasta el momento si se secuenciara cada modelo se le añade una cota de lo que falta por secuenciar. Ésta se puede calcular como la mejor combinación de las cotas correspondientes a los subproblemas separados por opciones, de la misma manera que en la BDP(II) (apartado 5.2.3. Programa dinámico acotado), o bien con la cota de transporte. Para cada una de las secuencias la cota parcial es igual a la mayor de ambas.

Para las secuencias que han pasado por cota, se comprueban las tres reglas de dominancia. Si para alguna de ellas se obtiene que es una secuencia dominada, su cota también se iguala a infinito. De esta manera, cuando al hacer la ramificación, se compruebe si la cota parcial es en algún caso mayor o igual a la mejor solución, sólo se ramificarán aquellos nodos que pueden llevar a una solución mejor, podando el resto.

Tras esto, las subsecuencias estudiadas se ordenan de menor a mayor cota y se ramifica el primer nodo siguiendo ese orden, siempre y cuando su valor de cota parcial sea menor al valor de la mejor solución. Esta rama se continua hasta que se llega a un nodo terminal (nueva solución o nodo cuyos descendientes tienen todos cota infinito), en ese momento se vuelve atrás en el procedimiento hasta que se encuentra un nodo por desarrollar con cota inferior a la solución actual y se sigue ramificando, véase algoritmo 7.

Algoritmo 7: Branch-and-bound *Depth-First* (instante **t**, vértice **v**)

Si se ha llegado a la última etapa

    Actualizar solución

    Return

Fin Si

Si se ha llegado al límite para DF

    Almacenar los vértices descendientes en pila

    Return

Fin Si

Para cada modelo *i*, tal que su plan de producción no se haya cumplido

Crear el vértice *v'* secuenciado *i* en la posición *t*

Si DominaciaLexicográfico (*v'*) = No Dominado

Si CotaTransporte(*v'*) ≥ UB **ó** CotaOpciones(*v'*) ≥ UB

Cota(*v'*)=Infinito

Si no

Si DominanciaGoalChasing(*v'*)=Dominado

Cota(*v'*)=Infinito

Si DominanciaGoalChasing(*v'*)=No Dominado

Si DominanciaÁrbol(*v'*)=Dominado

Cota(*v'*)=Infinito

Fin Si

Fin Si

Fin Si

Fin Si

Fin Para

Ordenar vértices creados de menor a mayor cota

Para cada vértice *v'* que cumpla Cota(*v'*)<UB

Si se ha llegado al límite para DF

Almacenar los vértices descendientes en pila

Return

Fin Si

Branch-and-bound (instante **t+1**, vértice **v'**)

Fin Para

Return

En el algoritmo 7, **Cota(*v*)** es un indicador del valor esperado para el vértice *v*; **CotaOpciones(*v*)** determina el valor de las cotas por separabilidad de opciones del vértice *v*; **CotaTransporte(*v*)** devuelve el valor de la cota de transporte para el vértice *v*; **DominanciaLexicográfico(*v*)** es una función para comprobar la dominancia lexicográfica; **DominanciaGoalChasing(*v*)** se encarga de la dominancia del goal-chasing; **DominanciaÁrbol(*v*)** de las dominancias por simetría y **UB** es el valor de la mejor solución conocida.

Algoritmo 8: Branch-and-bound *Best-First*

Inicializar memoria

Branch-and-bound *Depth-First* (vértice **0**, instante **0**)

Si (memoria disponible en la pila > 3/4 del total de memoria disponible en la pila)

Seleccionar el primer vértice de la pila (vértice *v*, instante *t*)

Borrar *v* de la pila

Branch-and-bound *Depth-First* (vértice **v**, instante **t**)

Si (memoria disponible en la pila <= 3/4 del total de memoria disponible en la pila)

Seleccionar el vértice de la pila con mayor profundidad en el árbol de exploración, es decir, cuya etapa t sea mayor (vértice *v*, instante *t*)

Borrar *v* de la pila

Branch-and-bound *Depth-First* (vértice **v**, instante **t**)

Fin Si

Liberar memoria

Fin procedimiento

5.5. Esquema general del algoritmo

El algoritmo general sigue el siguiente orden: Se empieza por inicializar todos los contadores de tiempo y de espacio en memoria. Tras esto, debe leerse el fichero de datos. Esto incluye el cálculo de la tasa ideal por cada opción.

Una vez se han almacenado los datos, los modelos se ordenan de manera eficiente: dando prioridad a los modelos con menor demanda, y en caso de empate, a los que tienen un menor consumo de opciones acumulado. Esto supone que, cuando se emplea el orden lexicográfico de los

modelos, en realidad no se está utilizando un orden cualquiera, y mejora la eficiencia de la regla de dominancia por simetrías.

Después, se obtiene una primera cota superior mediante *goal-chasing* y se almacena en una estructura de solución. La primera cota inferior a calcular es la cota por asimilación a un problema de transporte. Si esta cota es igual al resultado de la heurística *goal-chasing*, el procedimiento puede detenerse.

Una segunda cota superior y solución se obtiene mediante un programa dinámico acotado (I). Análogamente al punto anterior, si la solución que se obtiene verifica optimalidad, el procedimiento se detiene.

A continuación, deben generarse los subproblemas separados por opciones, para posteriormente, calcular la cota asociada solucionando cada subproblema mediante programación dinámica.

Tras la resolución, se almacena cada uno de los grafos de soluciones y se determinan las combinaciones de subproblemas que generan una cota: es decir, las combinaciones de subproblemas que incluyen todas las opciones, sin que coincida ninguna de ellas.

Con las cotas inferiores, y los grafos asociados a los subproblemas separados por opciones, puede resolverse de nuevo el problema mediante una BDP que use esta información para calcular una mejor cota parcial para lo que falta por desarrollar en cada nodo.

Si la mejor solución hasta el momento aún no ha probado ser óptima, se inicializan las estructuras de memoria necesarias para el branch-and-bound: el árbol de nodos explorados y la pila de vértices por desarrollar, ordenados según cota parcial.

Tras esto, empieza el branch-and-bound, con varios hilos (*threads*) que realizan una búsqueda depth-first en el árbol de soluciones. Cuando se llega al límite de búsqueda depth-first, se almacenan los vértices descendientes del último que se ha explorado, junto con su cota parcial, en una pila de memoria, donde se ordenan según esta cota. Mientras no se llega al límite de tiempo para el procedimiento total, cada *thread* toma el mejor vértice por desarrollar de la pila (según el criterio detallado en el apartado 5.4.1.) y se ramifica primero en profundidad hasta llegar al límite de búsqueda DF o se encuentra un nodo terminal (ya sea porque se poda o porque se encuentra una nueva solución).

Cuando se llega al límite para el procedimiento global o se verifica el óptimo, el procedimiento se detiene, se guarda o se imprime la mejor solución y se libera la memoria.

6. Implementación de los componentes

El objetivo de este apartado es dar detalles sobre cómo se han implementado los componentes presentados en el apartado anterior en el programa informático creado para la experiencia computacional. En este apartado, por tanto, se exponen las estructuras empleadas, los algoritmos usados y las estrategias que se han aplicado para solventar posibles problemas de memoria durante el cálculo de cotas y durante el branch-and-bound. También se detalla el uso que se ha hecho de la programación en paralelo en el procedimiento de enumeración.

Debido a que este apartado se centra en aspectos de implementación, en muchas ocasiones se hace uso de términos propios de C++, lenguaje en que se ha programado el algoritmo.

6.1. Cotas inferiores

6.1.1. Cota por condición de integralidad

La cota por condición de integralidad, tal y como se ha comentado en su definición, es una cota muy sencilla y muy poco restrictiva. Es por esto que no se calcula independientemente del resto de cotas para dar un valor, sino que se incluye en el procedimiento de programación dinámica acotada y en los mecanismos de poda del Branch-and-bound, para el cálculo de cotas parciales.

6.1.2. Cota por simetría

La cota por simetría se utiliza para agilizar el cálculo de las cotas por separabilidad de opciones y también aparece como regla de dominancia durante el branch-and-bound. En esta parte del algoritmo, la composición de las secuencias calculadas se expresan mediante un código unívoco, tal como se detalla en el apartado 6.1.3. Estos códigos se usan para ligar cada estado del programa dinámico con su correspondiente sdq, mediante una estructura del lenguaje C++ llamada mapa. Esta estructura y esta codificación se explican con más detalle en el apartado 6.1.3.

Por tanto, cuando se calcula la cota por simetría durante este procedimiento, es necesario obtener la composición correspondiente a la secuencia, para calcular su composición complementaria. La composición complementaria se codifica, para posteriormente buscar la sdq que corresponde a esa composición. Cabe recordar que si el nodo correspondiente no existe es porque su cota era superior al UB, y no fue explorado. Por lo tanto, dado el caso, el nodo complementario tampoco debe crearse ya que no puede llevar a una solución mejor.

6.1.3. Cota por separabilidad del problema por opciones

Para obtener la solución óptima de los subproblemas separados por opciones, se aplica un procedimiento de programación dinámica mediante el método de las dos listas. Estas listas, en la implementación, se declaran mediante una estructura llamada mapa. Esta estructura consta de dos partes: la clave y el valor de mapa. La clave se emplea para identificar de manera unívoca cada uno de los elementos del mapa. Tanto la clave como el valor de mapa pueden ser de tipos variados: por ejemplo, se puede usar como clave un número o una cadena de caracteres, y dependiendo de qué quiera almacenarse como valor de mapa, puede tratarse de una estructura propia, un número o una cadena. Las operaciones que se realizarán sobre el mapa son básicamente insertar un nuevo elemento, modificar el valor asociado a una clave, buscar una de las claves y eliminar un elemento. Todas estas operaciones tienen una complejidad logarítmica respecto al número de elementos en el mapa: O(log *n*).

En el caso de la lista que se usa en este procedimiento, el valor de clave se expresa como un solo número entero que codifica una composición de modelos de forma unívoca. Se ha optado por esta técnica en lugar de usar un vector que codifique independientemente la composición de cada modelo, ya que se usará menos espacio en memoria. A cada una de estas claves se le asocia un valor que corresponde al mejor valor de sdq encontrado para ese estado. Esta clave, representada por un único número entero (de tipo *unsigned long long int*), puede calcularse mediante la codificación en un número entero de la representación como número en base variable del vector composición.

Se entiende un número en base variable a aquél en que cada dígito tiene una base diferente (el primer dígito podría tener base 10, y como tal, tomar valores entre 0 y 9, y el segundo dígito podría tener base 7, y tomar valores entre 0 y 6). Si se asocia cada posición del vector a un dígito del número en base variable de la composición, el procedimiento para transformar un número de una base a otra permite obtener un número entero que identifica de forma unívoca cada composición.

Supongamos un plan de producción [3,5,7,9] y una composición igual a [2,3,4,2] en un instante. Para transformar la composición en un número unívoco es suficiente con:

2·1+3·(3+1)+4·(3+1)·(5+1)+2·(3+1)·(5+1)·(7+1)=494

Los índices multiplicadores se pueden precalcular y almacenar en memoria. En este caso, serían [1,4,24,192]. Estos números también resultan prácticos para recuperar la composición partiendo del código. Por ejemplo:

$\lfloor 494/192 \rfloor = 2$ y $494\ mod\ 192 = 110$

$\lfloor 110/24 \rfloor = 4$ y $110\ mod\ 24 = 14$

$\lfloor 14/4 \rfloor = 3$ y $14\ mod\ 4 = 2$

Por tanto, la composición es [2,3,4,2], que corresponde a la composición original.

Esta transformación garantiza un código es unívoco de forma parecida a Schrage y Baker (1978). Al almacenarse un solo número, no se consume excesiva memoria, mientras que si se almacenase como un vector composición, donde cada componente del vector representara el valor del número de unidades que se ha secuenciado del modelo que corresponda se consumirían 4 bytes por posición (utilizando el consumo estándar de un número entero en C++), mientras que un unsigned long long int (el tipo de número usado en este caso) consume 16 bytes, pudiendo almacenar la información de todos los modelos en una sola posición de memoria.

6.1.4. Cota por asimilación a un problema de transporte

En el apartado 5.1.4. se ha visto la adaptación que se hace del problema de transporte para calcular un cota del ORV. Una vez planteado el problema, debe resolverse de una manera eficiente: a pesar de que es posible resolver un problema de este tipo mediante programación lineal, existen procedimientos más rápidos y que requieren menos uso de memoria. En general, estos procedimientos están basados en que el problema de transporte es un caso particular del problema de flujo máximo a coste mínimo, un problema clásico de teoría de grafos. Por tanto, tratando el problema original como un problema de grafos, es posible usar técnicas de resolución para problemas de teoría de grafos con algoritmos más eficientes, como por ejemplo el cálculo de caminos extremos o el cálculo de flujos.

El método de resolución empleado en este caso es un procedimiento primal. Un procedimiento primal parte de una solución inicial factible y detecta si la solución incumbente es óptima o en caso contrario modifica la solución reduciendo su coste asociado. Este paso se repite hasta que no se mejore la solución, en cuyo caso la solución obtenida es óptima.

El principal inconveniente del procedimiento primal utilizado es que puede necesitar muchos cambios para demostrar el óptimo. Por ejemplo, trabajando con una función objetivo que dé como resultado números enteros (que no es el caso de la función objetivo con la que se trabaja), cada iteración del algoritmo sólo asegura cambiar una unidad el valor de la función objetivo. Sin embargo, un procedimiento primal aporta varias ventajas que la decisión de utilizar este procedimiento implica:

- La solución incumbente es siempre factible. Esto significa que si debiera pararse el procedimiento por cualquier motivo la solución sería válida, aunque posiblemente no óptima. Por otra parte si el problema de transporte se resuelve como cota de un problema si en algún momento la solución incumbente de la cota (incluyendo el valor de la solución del transporte) es menor que el valor de la mejor solución conocida, puede detenerse la búsqueda porque ya sabemos que la cota no va a eliminar el vértice.

- El número de iteraciones que deben hacerse es menor si se parte de una buena solución inicial. Si se dispone de una solución heurística, es posible utilizarla como punto de partida. Si esta solución está muy cerca de la solución óptima, el número de iteraciones necesario para converger en el óptimo será reducido. Obviamente esta propiedad es extremadamente interesante en el diseño de un algoritmo de enumeración implícita ya que se resolverán muchas cotas de un problema en que únicamente cambian un par de valores de la matriz de costes.

Véase ahora cómo puede plantearse el problema como un problema de grafos y solucionarse a partir de una solución inicial. Se supone el siguiente grafo asociado a un problema de transporte, figura 8. El grafo tiene dos vértices ficticios (origen, α, y destino, ω) tres vértices asociados con los emisores, en el problema ORV los modelos (I, II y III), y cuatro vértices asociados con los receptores, en el problema ORV las posiciones de la secuencia (los instantes A, B, C y D). Los arcos que van de α hasta los modelos tienen como capacidad el valor del plan de producción para el modelo en cuestión y coste cero. Los arcos que conectan cada uno de los modelos con cada uno de los instantes tienen capacidad 1 y como coste, el coste asociado a secuenciar ese modelo en ese instante. Los arcos que van de los instantes a ω tienen capacidad 1 y coste cero. Para evitar confusiones se omiten los costes y las capacidades del grafo ejemplo.

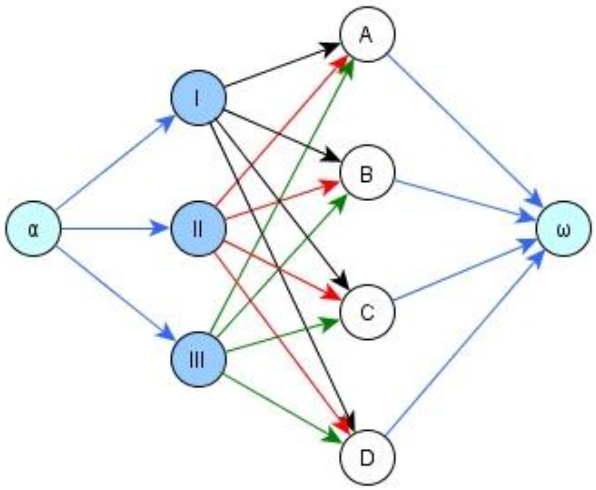

Figura 8: Ejemplo de un grafo asociado al problema de transporte

Supóngase que el plan de producción para estos modelos es [1,1,2], es decir, una unidad de los modelos I y II y dos unidades del modelo III. Obviamente cada unidad de tiempo tiene capacidad uno. En tal caso todos los arcos del grafo tendrían capacidad 1 excepto el arco entre α y III que tendría capacidad 2.

Una posible solución sería la siguiente (sólo se marcan los arcos que se utilizan):

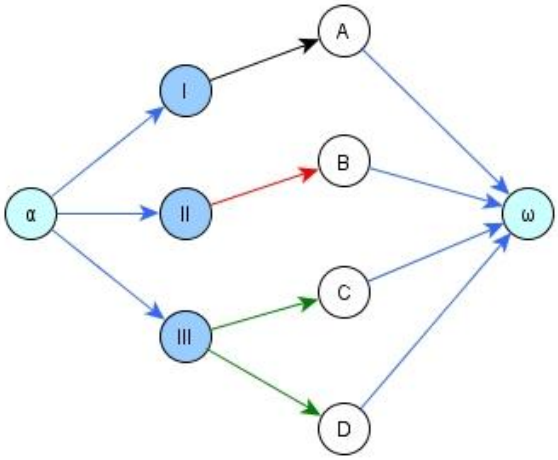

Figura 9: Grafo de una posibles solución para el ejemplo anterior

Esta solución corresponde a asignar la unidad del modelo I a la primera posición de la secuencia, la unidad de II a la segunda, y las dos unidades de III a la tercera y cuarta posición.

En una solución factible, los arcos entre α y los emisores, así como los arcos entre las posiciones y ω siempre se utilizarán, por tanto pueden suprimirse de la representación como se hará a partir de ahora.

El grafo solución mostrado en la figura 9 genera lo que se denomina grafo con capacidades remanentes o grafo de soporte. Este grafo, como ya se ha comentado, no incluye los vértices α y ω, y presenta los siguientes arcos:

- Para cada uno de los modelos que forman parte de la solución inicial, se traza un arco desde el modelo hasta el instante en el que se encuentra secuenciado en la solución inicial. El valor de coste de estos arcos es el valor negativo del coste correspondiente a esa asignación (ya que representa el ahorro de no secuenciar ese modelo en ese instante), y la capacidad (en este problema) es siempre 1.

- Para cada modelo, se traza un arco entre el nodo correspondiente al modelo y cada uno de los nodos correspondientes a instantes en los que no se ha secuenciado el modelo en la solución inicial. Estos arcos tienen el valor de coste correspondiente a secuenciar el modelo en el instante que une con el arco, esta vez en positivo, y la capacidad es 1 en todos los casos.

El grafo resultante puede verse a continuación, figura 10.

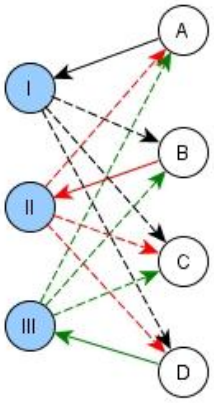

Figura 10: Grafo de soporte del ejemplo

Para simplificar la lectura se ha optado por marcar en línea discontinua los arcos no utilizados en la solución, aunque debido a la complejidad de representar el problema en un grafo, a partir de ahora se preferirá usar un formato matricial para representar el grafo (donde las filas corresponderán a los nodos emisores y las columnas a los receptores), como la siguiente.

	I	II	III	A	B	C	D
I					1	1	1
II				1		1	1
III				1	1		
A	1						
B		1					
C			1				
D			1				

Figura 11: Grafo de soporte del ejemplo en formato matricial

De la que aún se han omitido los costes, que dependen del sentido que usen. Los arcos entre vértices emisores y receptores tienen coste igual a C_{it} de la formulación matemática (de manera que si se decide asignar una unidad de I a la posición la posición B se incurrirá en el coste C_{IB}) y que los arcos entre vértices receptores y emisores tienen coste igual a $-C_{it}$ (esto es, si se decide mover una unidad de producción de I fuera de la posición B se incurre en un ahorro de c_{IB} unidades).

Utilizando esta representación y costes, que posteriormente será simplificada, puede expresarse la condición necesaria y suficiente para que la solución presentada sea óptima y en caso que no lo sea, se encuentre una nueva solución que mejore la solución anterior. Para explicar el procedimiento inicialmente se presentan dos observaciones para enunciar un lema que representará la condición necesaria y suficiente.

Observación 1. Un circuito en el grafo anterior representa una posible nueva asignación de transporte.

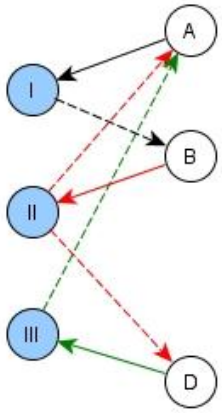

Figura 6: Simplificación del grafo de soporte, en el que se observa un circuito

Ejemplo. La figura 12 muestra un circuito posible. Partiendo del vértice I (es posible partir de cualquier otro vértice) se escoge el vértice con destino a B, de B se vuelve a II, de II se va a D, de D se vuelve a III, de III se va a A y de A se vuelve al punto de partida I.

El resultado es que la unidad de producto I va a la posición B de la secuencia (ahora mismo hay dos unidades de producto I asignadas y dos unidades en la posición B). El siguiente arco del circuito desasigna la unidad de II que estaba asignada a la posición B. El tercer arco reasigna la unidad de II a la posición D (volviendo a tener dos unidades asignadas a la posición D). Esto se corrige en el siguiente arco que desasigna la unidad de III que en el siguiente arco se asigna a la posición A. El último arco elimina la unidad de I de la posición A, generando una nueva solución factible. Si la secuencia original era [I, II, III, III], la nueva secuencia es [III, I, III, II].

Observación 2. El ahorro de un circuito es la suma de los arcos que lo componen.

Es fácil darse cuenta de esta situación, ya que cada arco usado de un emisor a un demandante representa un coste de asignación y cada arco de un demandante a un emisor elimina un coste de asignación, de ahí que tenga un coste negativo.

Con estas dos observaciones puede indicarse el siguiente lema:

Lema 1. Si el grafo residual contiene circuitos de coste negativo, la solución representada no es la solución óptima.

Este lema es ampliamente conocido, su demostración es trivial y por tanto se omite.

Dados los comentarios anteriores, y si se dispone de un procedimiento para la detección de circuitos negativos, puede plantearse un algoritmo para resolver el problema de transporte.

Algoritmo 9: Procedimiento básico de transporte

Encontrar una solución factible al problema

Hacer

Construir el grafo de soporte

Determinar un circuito de coste negativo

Si no existe FIN

En caso contrario modificar la solución de partida

Fin Hacer

Aún es necesario definir un procedimiento capaz de encontrar un circuito de coste negativo. Afortunadamente, el algoritmo de caminos extremos puede detectar fácilmente la existencia de circuitos de coste negativo.

Además, en el problema que nos incumbe el paso de construcción del grafo de soporte se modifica basándose en la observación de que sólo existe un arco que vaya de un vértice receptor a un vértice emisor, por tanto puede omitirse los vértices receptores con una reducción significativa del número de vértices a considerar en el grafo (con el resultante ahorro de tiempo en el cálculo de circuitos).

El siguiente grafo muestra el grafo reducido para la solución inicial mostrada anteriormente:

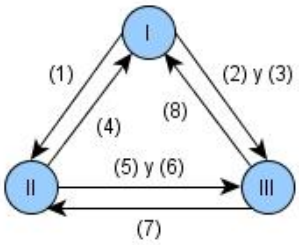

Figura 73: Grafo reducido para el ejemplo

El resultado es un multigrafo, ya que hay dos arcos que van de I a III y dos arcos que van de II a III.

Se han numerado los 8 arcos para poder interpretar el grafo reducido rápidamente, viendo a qué arcos del grafo de soporte corresponden cada uno de ellos:

(1) representa los arcos entre I y B y entre B y II

(2) representa el arco entre I y C y entre C y III

(3) representa el arco entre I y D y entre D y III

(4) representa el arco entre II y A y entre A y I

(5) representa el arco entre II y C y entre C y III

(6) representa el arco entre II y D y entre D y III

(7) representa el arco entre III y B y entre B y II

(8) representa el arco entre III y A y entre A y I

Como un problema de caminos sólo necesita tenerse en cuenta el arco de menor coste entre dos vértices, puede omitirse el arco más caro de (2) y (3) y el arco más caro de (5) y (6).

Véase ahora un ejemplo numérico de este procedimiento: supóngase el siguiente grafo (a partir de ahora se usará únicamente la forma matricial para representar el problema).

	A	B	C	D	Emite
I	5	4	3	2	1
II	6	1	4	5	1
III	3	4	5	6	2
Recibe	1	1	1	1	

Y considérese que la solución inicial es una solución idéntica a la expuesta con anterioridad. I-A, II-B, III-C y III-D con coste total 5+1+5+6=17.

El grafo de soporte, indicando costes en caso que haya un arco entre ambos puntos, es:

	I	II	III	A	B	C	D
I					4	3	2
II				6		4	5
III				3	4		
A	-5						
B		-1					
C			-5				
D			-6				

Que puede contraerse en el siguiente grafo:

	I	II	III
I		3	-4
II	1		-1
III	-2	3	

Aplicando un algoritmo de caminos mínimos puede detectarse el siguiente circuito: I-III-I, con coste -6. El arco I-III se ha generado a través de la contracción de los arcos I-D y D-III por tanto el circuito en el grafo de soporte consiste en: el arco de I a D, el de D a III, de III a A y de A a I. Recordemos que la solución inicial era I-A, II-B, III-C y III-D. Si el circuito entre A y D tiene coste negativo, la nueva solución intercambiará las posiciones de éstos, por tanto: III-A, II-B, III-C y I-D con coste total 3+1+5+2=11.

Como ha habido un cambio de solución es necesario volver a realizar otra iteración en busca de un circuito negativo.

El grafo de soporte es ahora:

	I	II	III	A	B	C	D
I				5	4	3	
II				6		4	5
III					4		6
A			-3				
B		-1					
C			-5				
D	-2						

Y el grafo reducido:

	I	II	III
I		3	-2
II	3		-1
III	4	3	

Este grafo no tiene circuitos y por tanto asegura que la solución anterior es óptima.

Esta cota se calcula de manera independiente, después del cálculo de la primera cota superior (cuya secuencia se toma como solución inicial). Además, también se utiliza este procedimiento de cálculo para la estrategia de poda del branch-and-bound, pero en ese caso, solamente se soluciona el problema de transporte asociado a los modelos que faltan por secuenciar, y se toma como solución inicial una modificación de la última secuencia encontrada por el procedimiento. Ambos cálculos se detallan a continuación.

Inicialmente, deben calcularse los componentes de la matriz de costes para cada instante *t* y modelo *m*, como se ha detallado anteriormente en el apartado 5.1.4.

Después de calcular los valores de la matriz de costes se selecciona una solución inicial. En el caso de la cota de transporte general, esta corresponderá a la mejor solución obtenida hasta el momento (la solución obtenida por el procedimiento de *goal-chasing*). En el caso del cálculo de una

cota parcial durante el proceso de poda del branch-and-bound, se toma la subsecuencia que se ha construido hasta el momento, y se completa usando como referencia la última solución obtenida a cualquier problema de transporte resuelto por el programa.

Primero se determina el plan de producción remanente teniendo en cuenta la subsecuencia construida. La solución inicial para el problema de transporte se construye desde el último instante de secuenciación hasta el primero asociando el modelo de la secuencia óptima en la secuencia inicial si es compatible con el plan de producción remanente y dejando la posición vacía en caso contrario. Tras ello, se rellenan las posiciones vacías con modelos con demanda pendiente en el plan remanente.

Posteriormente, se crea la matriz correspondiente al grafo reducido del problema, tal y como se ha expuesto con anterioridad. Las componentes de esta matriz corresponden a los distintos modelos, y la componente R_{nm} tomará valor infinito si $n=m$, y en caso contrario, tomará el menor valor que conecta los nodos n y m en el grafo completo.

Con la matriz correspondiente al grafo reducido se realiza una iteración de un algoritmo de caminos mínimos, tras la cual se detectan los circuitos existentes siguiendo los valores de la traza. El cambio de la solución modifica el grafo reducido del problema , con lo cual se aplica de nuevo el algoritmo de caminos mínimos. Estos pasos se llevan a cabo de forma iterativa mientras haya circuitos negativos. Cuando no existen circuitos negativos, se ha encontrado la mejor solución y, por tanto, la cota de transporte.

A nivel computacional, el cálculo de la cota por asimilación a un problema de transporte, tanto del procedimiento general como dentro del branch-and-bound, necesita una serie de estructuras para agilizar el cálculo.

Primero, es necesario crear una matriz inicial de costes, que almacene para cada modelo e instante el coste de su secuenciación, según se ha descrito en el apartado 5.4.1. Para facilitar aún más el cálculo de la cota, no sólo se almacena el valor del coste asociado, sino que también se almacena el valor de consumo de opciones que ha generado ese coste. Cuando se aplique la cota por asimilación a un problema de transporte para un vértice del branch-and-bound, puede comprobarse rápidamente si el consumo de alguna de las opciones no es factible para ese vértice y recalcular los costes dado el caso. La factibilidad o infactibilidad puede deducirse de los mínimos y máximos de opciones calculados en el apartado 5.4.1. Si un valor no está dentro del rango de valores posibles, teniendo en cuenta el consumo de opciones de la actual subsecuencia, será suficiente con sustituir el nuevo consumo de opciones por el límite incumplido y recalcular el mejor coeficiente.

También se crea una estructura para el cálculo de la cota en sí. Para cada hilo de cálculo, se debe almacenar la solución inicial que se está usando, la matriz correspondiente al grafo reducido, la matriz de costes correspondientes a cada modelo e instante (que puede haberse tenido que recalcular, o pueden ser iguales que los iniciales), una matriz que indique los consumos de opciones que han originado los costes, el valor de sdq acumulado, el último instante que se ha explorado, y un vector para reseguir la traza.

Algoritmo 10: Cota Transporte (General)

Para cada modelo *i*

Para cada instante *t*

C_{it}=Calcular C_{it} (ver Algoritmo 2)

Fin Para

Fin Para

Obtener Solución Inicial (la mejor solución o la solución parcial correspondiente)

Mientras haya cambios en la Matriz Reducida

Calcular Matriz Reducida

Detectar Ciclo

Si hay un ciclo

Rectificar la solución

Anotar que ha habido un cambio

Fin Si

Fin Mientras

Calcular sdq de la solución obtenida

CotaTransporte=sdq/2

6.2. Cotas superiores

6.2.1. Procedimiento *Goal-Chasing*

La primera cota superior se calcula según un procedimiento heurístico que asigna secuencialmente un modelo para cada periodo temporal, dando prioridad a aquél cuya aportación para ese instante sea menor.

Se define la demanda remanente de un modelo como la cantidad que falta por producir de ese modelo para cumplir con la demanda total y se inicializa igualándola al plan de producción. Para una etapa *t*, empezando por *t*=1, se calcula la sdq que se obtendría secuenciando cada modelo en esa etapa, sólo para los modelos con demanda remanente positiva. El modelo para el que se obtenga un valor menor de sdq se secuencia en la posición *t*, y se pasa a la etapa *t*+1, repitiendo el paso anterior.

6.2.2. Programa dinámico acotado

Para el cálculo de la segunda cota superior, se aplica un procedimiento similar al programa dinámico utilizado para resolver las cotas por separabilidad. El programa dinámico que se aplica en este caso, sin embargo, explora el grafo de soluciones de principio a fin, es decir: usando las funciones de recurrencia forward.

Además, resolver el grafo completo sería un procedimiento costoso, tanto en memoria como en tiempo de computación. Por esa razón se aplica un procedimiento acotado: en lugar de resolver el grafo en su totalidad, al crear los nodos descendientes del nodo actual, se ordenan de menor a mayor a cota, y sólo se desarrollan los *n* primeros. Esto convierte el programa dinámico en una heurística, que será más rápida cuánto más pequeño sea el valor de *n*, aunque la calidad de la solución disminuya. El procedimiento se ha implementado de manera que el valor *n* (ancho de ventana) es un parámetro de ejecución del programa.

Para ello, se debe crear una lista de prioridad, donde se almacenará la información de los nodos por desarrollar, ordenados de mayor a menor aportación. Mientras haya nodos por desarrollar almacenados en la lista de prioridad, en cada etapa del algoritmo, se tomará el primero de esta lista para desarrollarlo, y se borrará de la lista al terminar.

Este procedimiento también trabaja con mapas para relacionar cada estado del grafo con su sdq correspondiente, aunque en este caso codificar mediante un único número entero no es posible, ya que el tamaño del número pueda superar el espacio reservado en memoria para un entero. De modo que, pese a que por lo demás el mapa usado para la BDP funciona igual que el que se utiliza para la cota por separabilidad, el mapa utilizará una string de texto en lugar de un entero como clave para codificar las composiciones.

La clave funcionará de la siguiente manera: si ninguno de los planes de producción de los modelos supera las 100 unidades, las unidades secuenciadas de cada modelo se escribirán de forma

encadenada, colocando un 0 delante si son menos de 10. Por ejemplo, si se tiene la composición siguiente: [12, 6, 21, 4, 15, 3], el código que se obtiene es: '120621041503'. En el caso de que alguno de los planes de producción fuera superior a 100 unidades, el procedimiento sería el mismo pero colocando un cero cuando el número de unidades secuenciadas sea inferior a 100 y dos ceros si es inferior a 10. Por ejemplo, la composición [13,56,143,8,52,6] se codificaría como '013056143008052006'.

A cada clave se le asocia el valor de aportaciones acumuladas, como el resto de casos, y un vector que representa la subsecuencia, para poder volver a construir la solución.

Este procedimiento se lleva a cabo después del cálculo de la cota superior por *goal-chasing* y la cota inferior por asimilación a un problema de transporte. Por tanto, un nodo puede dejar de desarrollarse si su aportación es superior a la cota por *goal-chasing*, y si la solución obtenida por el procedimiento es igual al valor de la cota de transporte, se puede detener el algoritmo general, ya que se ha demostrado la optimalidad de la solución conocida.

Después de calcular las cotas por separabilidad de opciones, se vuelve a resolver un problema dinámico acotado. La única diferencia es que, en este caso, la cota que se calcula para los nodos, tanto para comprobar si vale la pena desarrollarlos, como para ordenarlos en la lista de prioridad, y el método será más restrictivo. En el primer programa dinámico acotado la cota era exactamente igual a la aportación, pero en este caso, se añade un término correspondiente a lo que queda por secuenciar usando la información obtenida al calcular las cotas por opciones separadas.

6.3. Reglas de dominancia

Las reglas de dominancia se comprobarán en el Branch-and-bound una vez haya sido creado el vértice siguiente y se cumpla que la cota parcial del vértice sea menor a la mejor cota superior. Si se comprueba que el vértice no está dominado, se pasa a la siguiente etapa del algoritmo para generar los descendientes del vértice. En caso contrario, el vértice no se desarrolla.

6.3.1. Árbol de nodos explorados

La dominancia por árbol de nodos explorados se basa en almacenar en memoria un identificador para los nodos del grafo asociado a la instancia que han sido explorados, junto con el mejor valor de sdq que se ha obtenido para ese nodo.

Conforme se ramifica el árbol de soluciones parciales, éstas se expresan en forma de composición de modelos, para comprobar a qué estado del programa dinámico corresponde. Si el nodo no había sido explorado anteriormente, éste se almacena junto con su valor de sdq. Si, por el contrario, el nodo ya estaba almacenado, se compara el valor de sdq obtenido con el almacenado

para la solución explorada. Si el valor de sdq de la solución parcial actual es mejor (más pequeño) que el almacenado, el valor se actualiza. En caso contrario, el vértice actual deja de explorarse, ya que ese vértice nunca podrá llevar a una solución mejor que la que ya se había explorado.

La implementación de esta regla de dominancia puede presentar problemas de memoria, ya que, si no se limita la cantidad de nodos a almacenar, puede consumirse toda la memoria del ordenador. A su vez, cuántos más nodos se almacenen en el árbol de soluciones exploradas, más efectivo será el procedimiento de dominancia. Teniendo en cuenta esta dicotomía, debe limitarse el número de nodos que pueden almacenarse, de manera que cuando se llegue al número indicado, se sigan comprobando las dominancias respecto a los vértices almacenados pero no se almacenen nuevos vértices.

Además, como se detalla en el apartado de simetrías, cada vez que se almacena una secuencia, se comprobará si el vértice complementario al actual existe. En las etapas anteriores a $T/2$, se verifica si el complementario existe y está congelado, y si se mejora la solución. En las etapas posteriores, si el vértice complementario existe, y si es así se congela el vértice actual.

Para almacenar los nodos explorados de manera eficiente, se utiliza una estructura de árbol binario, que codifica la composición de modelos de una secuencia en las ramificaciones por el árbol, y que almacena los valores de sdq en los vértices terminales para cada una de ellas.

La estructura utilizada es un árbol binario en el cual la composición se transforma en una secuencia de bits tal como sigue:

- Para cada modelo, se determina el número mínimo de bits suficientes para representar el plan de producción de ese modelo (por ejemplo, si el plan contiene 12 unidades, 4 bits serían suficientes para representarlo)
- Se transforma la composición con números enteros en una composición binaria con los bits indicados anteriormente
- Se concatenan los números binarios
- La secuencia concatenada se utiliza como guía para ramificar en cada vértice del árbol binario
- Al llegar a la profundidad máxima, se guarda el valor de aportación acumulada

Claramente, a cada composición le corresponderá un único camino en este árbol binario desde el vértice raíz hasta un vértice terminal. El método utilizado para implementar este árbol binario consiste en la representación mediante vectores compactos tal como puede verse en Tarjan (1983).

La función que explora el árbol para encontrar el vértice actual, comprobar si está dominado e insertarlo en el árbol si no se había explorado (InsertarNuevo), funciona según el siguiente algoritmo:

Algoritmo 12: InsertarNuevo(vértice **v** en la etapa **t**)

Si *v* corresponde a la profundidad máxima (se ha explorado toda la etapa *t*)

Si sdq(*v*)<ValorAlmacenado

Guardar el valor de sdq del nodo

Calcular el vértice complementario, $\bar{v}$

Si $\bar{v}$ se había explorado

Si (sdq(*v*) + sdq($\bar{v}$) < UB)

Actualizar la mejor solución

Fin Si

Si $t \cdot 2 > T$

Vértice Congelado

FIN

Fin Si

Fin Si

Vértice No Dominado

FIN

Fin Si

Vértice Dominado

FIN

Fin Si

Mientras quede memoria libre para el árbol

Crear siguiente vértice *v'*

InsertarNuevo(vértice **v'** en la etapa **t+1**)

Fin Mientras

Vértice No Dominado

FIN

Obsérvese que el algoritmo sólo sigue almacenando nuevos nodos mientras no supera el límite de memoria, pero el límite de memoria no impide seguir comprobando la dominancia de los nuevos vértices respecto a los nodos ya almacenados.

6.3.2. Dominancia por simetrías

La comprobación de la primera regla de dominancia por simetría mostrada en el apartado 5.3.3 se lleva a cabo mientras se desarrolla el árbol binario de nodos explorados.

Cada vez que se almacena una secuencia, se comprueba si existe el vértice complementario al actual. Si el nodo actual es de una etapa inferior a *T*/2, y el complementario existe y está congelado, se comprueba si sumando la sdq del nodo actual y el complementario se mejora la solución. Si es así, se actualiza la cota superior (UB) y se almacena esta solución. Si la solución obtenida no puede demostrarse óptima (aún quedan nodos por explorar y el valor de sdq de la mejor solución es superior a la mejor cota inferior), se hace backtrack y se continua el branch-and-bound.

Si, en cambio, el vértice actual corresponde a una etapa superior a *T*/2 y el nodo complementario existe, pueden darse dos casos:

- La etapa complementaria se ha desarrollado por completo. En este caso, se comprueba si combinando la sdq del nodo actual y el complementario mejoran la solución. En caso positivo, se almacena, y en cualquier caso, el nodo deja de explorarse.
- La etapa complementaria no se ha desarrollado por completo. Este es el caso en que el nodo se congela: se almacena su valor de sdq, y se anota que no es un vértice podado, pero no se continúa desarrollando, sino que se hace *backtrack* en el árbol.

La segunda regla de simetría se comprueba comparando el valor lexicográfico de la composición asociada a la subsecuencia construida y la composición de su complementario.

6.4. Branch-and-bound

En el procedimiento branch-and-bound destaca el sistema de ramificación, que utiliza una búsqueda mixta primero en profundidad y primero el mejor. El subapartado 6.4.1 se dedicará a analizar esta estrategia de ramificación, mientras que el subapartado 6.4.2 detallará el uso de las estrategias de poda y reglas de dominancia dentro del procedimiento.

6.4.1. Estrategia de ramificación

El procedimiento realiza diversas búsquedas primero en profundidad en puntos diferentes del árbol de exploración. La selección de los siguientes vértices del árbol a explorar mediante el procedimiento en profundidad intenta seguir una lógica primero el mejor.

Mantener una búsqueda tipo primero el mejor obliga a almacenar en memoria las partes del árbol de exploración que aún no han sido completamente desarrolladas. Este efecto se consigue manteniendo una pila de vértices aún no desarrollados. Esta pila de vértices aún no desarrollados se mantiene en formato matricial. Como la memoria dedicada a esta matriz es finita, en ocasiones no se podrá optar por el desarrollo de subsecuencias escogidas mediante el principio best-first porque se correría el riesgo de agotar la memoria. En estos casos, se optará por iniciar una nueva búsqueda primero en profundidad.

Para determinar cuándo se abandona la búsqueda primero en profundidad se establece un límite de vértices (L) descendientes explorados según esta política. Al llegar a este límite, todos los vértices descendientes visitados pero no explorados se almacenan en la pila de vértices aún no desarrollados.

Una forma de entender el algoritmo corresponde a considerar que se realizan un máximo de L operaciones de ramificación siguiendo una política primero en profundidad por cada operación de ramificación siguiendo una política primero el mejor.

Esta forma de explorar el árbol tiene dos ventajas: primero, permite explorar diversas zonas del árbol (evitando así la búsqueda intensiva de áreas concretas del árbol que realizan la búsqueda primero en profundidad); segundo, agiliza la paralelización del algoritmo, ya que las ejecuciones primero en profundidad pueden realizarse de forma independiente, coordinándose a través de la búsqueda primero el mejor.

6.4.2. Uso de cotas y reglas de dominancia dentro de la enumeración

Durante el branch-and-bound se utilizan diversas cotas para cada uno de los vértices que descienden del vértice actual, y se aplican también a cada uno de ellos las reglas de dominancia expuestas a continuación:

Inicialmente, se comprueba la cota resultante de sumar la aportación del vértice actual más una cota de lo que queda por secuenciar obtenida de los cálculos de las cotas separadas por opciones. Este proceso lee los valores de aportación de los nodos complementarios para cada subproblema, calcula las combinaciones entre estos valores y devuelve el más restrictivo. Si esta cota es mayor a la mejor solución hasta el momento, la cota se iguala a infinito, lo cual es equivalente a podar el nodo. Para cualquier nodo con cota infinito, no deben comprobarse el resto de cotas y reglas de dominancia.

Tras esto, se comprueba la cota por asimilación con un problema de transporte: se soluciona el problema de transporte asociado a lo que queda por secuenciar según el método detallado en el apartado 6.1.4. Si, de la misma manera que con la cota anterior, el vértice se poda, se iguala la cota a infinito, para evitar que el nodo sea explorado.

La primera regla de dominancia que se comprueba es la que busca mejores soluciones con los modelos secuenciados hasta el momento, mediante una heurística *goal-chasing*. Si no se encuentre una solución mejor, se repite el cálculo fijando iterativamente una posición de la secuencia parcial actual. Si en algún momento se encuentra una secuencia con mejor valor de sdq, la cota de ese vértice pasa a ser de valor infinito.

La segunda regla de dominancia a aplicar es la que busca soluciones ya exploradas en un árbol binario. Este árbol se genera mientras se comprueba la dominancia, e incluye la comprobación de la dominancia por simetría. Esta regla poda los vértices que ya han sido explorados o cuyo valor lexicográfico es mayor que el de su complementario.

Cuando se han calculado las cotas y comprobado las reglas de dominancia para todos los vértices descendientes del actual, se crea una lista donde se ordenan por valor de cota, de menor a mayor. Obviamente, los vértices con cota infinito son los últimos de la lista y nunca deben explorarse. El resto se desarrollan según el orden establecido.

6.5. Código en paralelo

En este algoritmo se hace uso de la programación en paralelo durante el branch-and-bound, permitiendo el desarrollo de varios vértices en paralelo, y agilizando el cálculo del árbol de soluciones factibles. Sin embargo, es necesario poner especial atención en la implementación de código en paralelo a las tareas que resultan críticas en el programa y no pueden ser realizadas conjuntamente.

Esto se refiere, básicamente, a todas aquellas tareas que requieran modificar de un modo u otro la memoria de trabajo del procedimiento, ya que si se accede a la memoria con dos hilos a la vez, uno de ellos sobrescribirá la información que haya almacenado el otro. Para evitar la pérdida de información debida a este hecho, y el consecuente mal funcionamiento del programa, todos los procedimientos que requieran acceso a la memoria compartida serán marcados como críticos, de manera que nunca se realicen en paralelo: si dos hilos de trabajo llegasen a uno de estos puntos a la

vez, realizarían la tarea correspondiente en serie, y al terminar esa tarea, continuarían el resto del cálculo en paralelo.

Para evitar este tipo de accidentes al sobrescribir zonas comunes de memoria cada uno de los hilos de ejecución dispone de segmentos de memoria propios para:

- Almacenar su árbol de exploración parcial (a través de una matriz que implícitamente guarda los vértices visitados pero no explorados en el procedimiento primero en profundidad)
- Guardar la subsecuencia en curso y su valor de coste acumulado
- Calcular las cotas de transporte (que requieren una estructura de memoria con la solución actual, la matriz de costes y el consumo acumulado de opciones que se utiliza para calcular cada coeficiente de la matriz de costes)

Esto limita la interacción entre diferentes hilos de ejecución a los siguientes puntos del algoritmo:

- Todas las localizaciones de memoria que se realizan durante el procedimiento (al declarar las estructuras y su tamaño al iniciar el branch-and-bound, y también al declarar el tamaño de la pila de soluciones)
- Todas las operaciones de liberación de la memoria utilizada al final del procedimiento.
- El almacenaje de los vértices no explorados pero visitados al llegar al límite de la búsqueda primero en profundidad
- La selección del siguiente vértice a desarrollar en la parte de la búsqueda primero el mejor
- Las actualizaciones requeridas por encontrar una mejor solución
- El almacenaje de nuevos estados en el árbol de nodos explorados

En cualquiera de estos casos, sólo un hilo puede estar ejecutando cada sección de código asociada. Para ello, se utiliza un semáforo que limita el número de hilos por sección a uno.

7. Experimentos computacionales y resultados

Se entiende como instancia de un problema al conjunto completo de datos que define un caso de éste. Para el problema ORV la instancias queda definida por el número total de productos a secuenciar (que es igual al número de instantes **T**), el número de modelos **M** y su plan de producción **D**, el número de opciones **P**, y la matriz completa de consumos **C** que tiene cada modelo para cada una de las opciones. Cabe recordar que el algoritmo es capaz de resolver problemas que incluyan consumos no binarios.

Para la experiencia computacional se han usado un total de cinco juegos de instancias, procedentes de la literatura, que se describen brevemente a continuación:

- Instancias no binarias (originalmente presentadas en Bautista et al. (1996), ampliadas posteriormente en Jin y Wu (2002)) : Estas instancias son las únicas que incluyen consumos no binarios. Las instancias originales de Bautista et al. contaban con T=20, P={4,5,8} y M=4. Jin y Wu las amplían multiplicando los planes de producción por 10, obteniendo instancias con T=200. En este trabajo, siguiendo el método propuesto por Jin y Wu para ampliar las instancias, se han construido instancias con T={40, 60, 80, 100, 120, 140, 160, 180}, que se incorporan a las instancias anteriores. El total de instancias de este juego es de 2250 instancias, 225 instancias para cada tamaño T.
- Instancias binarias pequeñas de Boysen, Fliedner y Scholl (2008): Estas instancias se derivan de un problema con los mismos datos iniciales y objetivo diferente. El juego contiene 486 instancias cuyo número de unidades a secuenciar toma estos valores {10, 15, 20}, y tanto el número de opciones como el de modelos toman los siguientes valores: {5, 7, 9}.
- Instancias binarias medianas de Boysen, Fliedner y Scholl (2008): Estas instancias se derivan de un problema con los mismos datos iniciales y objetivo diferente. El juego contiene 486 instancias, cuyo número de unidades a secuenciar varía entre los valores {25, 30, 35}, y el número de opciones y modelos toman los siguientes valores: {5, 7, 9}.
- Instancias binarias derivadas del Car Sequencing Problem (origen: www.csplib.org): Estas instancias se obtienen eliminando las restricciones asociadas a las opciones y manteniendo el plan de producción y de consumo. Del total de instancias disponibles se utilizan las 9 instancias originales, a las que se añade la instancia charme, procedente de un panfleto informativo de uno de los primeros solvers de propagación de restricciones. Estas instancias tienen 100 unidades a secuenciar, 5 opciones y el número de modelos se encuentra en el rango entre 18 y 26 modelos. En total se incluyen 10 instancias en este grupo. A partir de ellas, se han generado un total de 150 instancias reducidas de {25,50,75} productos a secuenciar. La generación se ha realizado eliminando productos aleatoriamente (con la consecuente eliminación de modelos, en caso que el plan de producción llegue a ser 0), hasta que el total de productos a secuenciar es el deseado. Para cada instancia y tamaño se han generado 5 variantes, para un total de 160 instancias.
- Instancias binarias generadas siguiendo Drexl y Kimms (2001): Estas instancias se han creado según se indica en el artículo anterior. El método parte de un número total de productos a secuenciar (que en el caso presente son los números de diez en diez entre 10 y

100), un número de opciones {3,5,7} y una serie de ratios de uso de las opciones (que se dividen en dos grupos: fáciles y difíciles, según criterios basados en el cumplimiento de las restricciones por opciones) y crea instancias que saturan los ratios de uso de opciones dados.

Con el objetivo de probar la calidad del algoritmo es necesario implementarlo en un programa informático capaz de resolver los conjuntos de instancias propuestos.

El procedimiento se ha programado en C++, y el código resultante (de unas 3200 líneas, aproximadamente) se ha compilado con gcc v. 4.2.1.

Este programa se ha ejecutado en un ordenador que utiliza Linux con un kernel 3.4.4 de 64 bits, con 4 procesadores Intel Core i5 650 a 3,2 GHz y 4 Gb de memoria RAM. Al contar con cuatro núcleos, éste será el número total de *threads* que podrán trabajar en paralelo.

Durante la descripción de los componentes del algoritmo y la implementación se han presentado diversos parámetros que deben limitarse, por ejemplo el ancho de ventana para la BDP o el número máximo de vértices que pueden almacenarse en el árbol de vértices explorados. Tras unas pruebas iniciales, se han escogido los siguientes parámetros:

- Ancho de ventana para el programa dinámico acotado: 10^3
- Número de iteraciones máximo en *depth-first*: 10^4
- Límite de vértices a almacenar en el árbol: $4 \cdot 10^8$
- Límite de tiempo para el proceso (general): 3600 s

A continuación se presenta un resumen de los resultados obtenidos para las pruebas de 1 hora con los conjuntos de instancias presentados en el apartado anterior.

Los resultados que se muestran a continuación se han separado según el conjunto de instancias a las que se ha aplicado el algoritmo. Para cada juego de instancias se presentan tres tablas:

En la primera se exponen los resultados para las tres heurísticas: para el programa dinámico acotado sin cota por opciones separadas (BDP-1), el programa dinámico acotado con cota por opciones separadas (BDP-2) y para el método goal-chasing (GC). Para cada uno, se presenta:

- El número de instancias en que la heurística encuentra la solución óptima (solución óptima demostrada por el branch-and-bound)
- El número de instancias en que la heurística encuentra la mejor solución conocida (solución generada por alguno de los métodos heurísticos o por el procedimiento de branch-and-bound)

En la segunda tabla de cada juego se muestran los índices de calidad relativa de las heurísticas y de los métodos de acotación utilizando los índices (32) y (33)

- El gap de optimalidad medio para las cotas de separabilidad por opciones y por asimilación a un problema de transporte
- El gap de optimalidad medio para cada uno de los métodos heurísticos

Por último, la última tabla para cada juego de instancias muestra los resultados relativos al branch-and-bound. Los resultados mostrados son:

- El gap de optimalidad medio
- El número de óptimos demostrados
- El tiempo de computación medio
- El tiempo de computación máximo.

El método branch-and-bound no reportará el valor de la mejor solución conocida, puesto que se inicializa con la mejor solución procedente de los métodos heurísticos. Por tanto, siempre termina con el mejor valor conocido antes de su ejecución o con uno mejor.

La forma de obtener el gap de optimalidad varia según se calcule para evaluar una cota inferior o una cota superior.

Para una cota inferior, se calcula teniendo en cuenta UB_{best} (mejor solución conocida) y LB (la cota que se pretende evaluar) y se obtiene mediante la siguiente fórmula:

$$Opt.Gap\ LB\ (\%) = \frac{UB_{best} - LB}{UB_{best}} \cdot 100 \qquad (32)$$

Para una cota superior, se calcula teniendo en cuenta LB_{best} (la mejor cota conocida) y UB (la solución obtenida por el método que se evalúa) y se obtiene mediante la siguiente fórmula

$$Opt.Gap\ (\%) = \frac{UB - LB_{best}}{UB} \cdot 100 \qquad (33)$$

Finalmente, cabe tener en cuenta que el tiempo de computación no corresponde a tiempo real, ya que hay un máximo de 4 procesadores trabajando en paralelo. Esto implica que, a pesar de que el tiempo real esté limitado a 3600 s, el tiempo de computación total puede ser de hasta 14400 s (nunca llegará a este máximo, ya que éste se daría si absolutamente todas las operaciones pudieran realizarse en paralelo, y no es el caso, como se ha expuesto en el apartado 6.5.).

7.1. Resultados para instancias no binarias:

La tabla 7 muestra los resultados de las heurísticas.

Se observa que para el primer conjunto de instancias, el programa dinámico acotado más sencillo ya es capaz de resolver todas las instancias del set de manera óptima. Nótese que la heurística goal-chasing tiene una calidad muy inferior a ésta. Cabe decir que el ancho de ventana utilizado (1000) es mayor que en la mayoría de experimentos previos con programación dinámica realizados para este tipo de problemas. Aun así, los tiempos de computación son inferiores a 1 minuto.

Tamaño	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
20	225	225	225	225	225	14	14
40	225	225	225	225	225	14	14
60	225	225	225	225	225	13	13
80	225	225	225	225	225	12	12
100	225	225	225	225	225	12	12
120	225	225	225	225	225	11	11
140	225	225	225	225	225	11	11
160	225	225	225	225	225	11	11
180	225	225	225	225	225	11	11
200	225	225	225	225	225	11	11

Tabla 7: Resultados de las heurísticas para las instancias no binarias. Para cada tamaño de instancias se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

En la tabla 8 se muestran las medidas de calidad relativa para las heurísticas y las cotas.

Obviamente, las heurísticas basadas en programación dinámica consiguen un optimality gap de 0, puesto que han encontrado la solución óptima para todas las instancias. Por el contrario, la heurística del goal-chasing da soluciones bastante alejadas del óptimo, según los gaps de optimalidad observados, incrementando la desviación respecto a la solución óptima según incrementa el tamaño de la instancia.

En cuanto a las cotas, se observa: (1) que la cota por separabilidad es significativamente mejor que la cota por asimilación a un problema de transporte; (2) que la calidad de la cota de transporte se mantiene estable respecto al tamaño de las instancias, mientras que la cota por separabilidad resulta una mejor aproximación conforme incrementa el número de unidades a secuenciar.

Finalmente la tabla 9 muestra los resultados del algoritmo de branch-and-bound. Puede observarse que para esta colección de instancias, el algoritmo es capaz de demostrar la optimalidad de las soluciones obtenidas por programación dinámica en tiempos de computación muy pequeños (son casi negligibles en promedio y siempre inferiores a 2,5 segundos).

Tamaño	Opt. Gap LB Cota opc.	Opt. Gap LB Cota tran.	Opt. Gap UB BDP-1	Opt. Gap UB BDP-2	Opt. Gap GC
20	18,294	29,186	0,000	0,000	27,211
40	14,687	29,118	0,000	0,000	35,514
60	13,089	29,093	0,000	0,000	40,492
80	13,079	29,081	0,000	0,000	43,948
100	11,980	29,073	0,000	0,000	46,361
120	11,887	29,068	0,000	0,000	47,931
140	11,195	29,065	0,000	0,000	49,162
160	9,423	29,062	0,000	0,000	50,068
180	8,818	29,060	0,000	0,000	50,805
200	8,313	29,058	0,000	0,000	51,383

Tabla 8: Calidad de las cotas y las heurísticas para las instancias no binarias. Para cada tamaño de instancias se muestra el gap de optimalidad de la cota por opciones separadas (Opt. Gap Cota Opc., expresado en %), el gap de optimalidad para la cota de transporte (Opt. Gap Cota tran., expresado en %), y los gaps de optimalidad para las heurísticas (Opt. Gap BDP-1, Opt. Gap BDP-2 y Opt. Gap GC, respectivamente)

Tamaño	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)	T. máx. comp. (s)
20	0	225	0,00062	0,00500
40	0	225	0,00120	0,01700
60	0	225	0,00232	0,04399
80	0	225	0,00435	0,10298
100	0	225	0,00815	0,19197
120	0	225	0,01352	0,32495
140	0	225	0,02126	0,57391
160	0	225	0,03284	0,99285
180	0	225	0,04828	1,37779
200	0	225	0,07090	2,41763

Tabla 9: Resultados del branch-and-bound para las instancias no binarias. Para cada tamaño de instancias se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T- máx. comp.) también en segundos.

7.2. Resultados para instancias binarias pequeñas de Fliedner et al. (2010)

La tabla 10 muestra los resultados de las heurísticas para este juego de instancias.

Se observa que, como sucedía con el primer set de instancias, los procedimientos de programación dinámica acotada son capaces de obtener la solución óptima para todas las instancias del conjunto. Se puede observar, además, que el procedimiento goal-chasing no es capaz de obtener la solución óptima de manera consistente para instancias de tamaño reducido y sólo ocasionalmente (aproximadamente en un 10% de las ocasiones) para instancias de tamaño 20.

Tamaño	# Opc.	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
10	5	54	54	54	54	54	22	22
	7	54	54	54	54	54	21	21
	9	54	54	54	54	54	22	22
15	5	54	54	54	54	54	15	15
	7	54	54	54	54	54	14	14
	9	54	54	54	54	54	8	8
20	5	54	54	54	54	54	5	5
	7	54	54	54	54	54	4	4
	9	54	54	54	54	54	5	5

Tabla 10: Resultados de las heurísticas para las instancias binarias pequeñas. Para cada tamaño y número de opciones se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

La tabla 11 muestra los resultados en cuanto a calidad relativa de las cotas y los procedimientos heurísticos. Tanto en esta tabla como en la tabla 14, se observa que la calidad de la cota de transporte es dependiente del número de opciones que tenga la instancia, mientras que la cota de separabilidad se comporta de forma más o menos regular para instancias de tamaño reducido y crece según el número de opciones para instancias de mayor tamaño.

Por otra parte, puede apreciarse comparando la tabla 8 con la tabla 11 que la heurística goal-chasing obtiene mejores valores en instancias binarias que en instancias no binarias.

En cuanto al procedimiento exacto se observa en la tabla 12 que pueden resolverse todas las instancias de este conjunto sin superar las 2 décimas de segundo para ninguna de ellas.

Tamaño	# Opc.	Opt. Gap LB Cota opc.	Opt. Gap LB Cota tran.	Opt. Gap UB BDP-1	Opt. Gap UB BDP-2	Opt. Gap UB GC
10	5	19,277	4,979	0,000	0,000	7,175
	7	10,443	10,882	0,000	0,000	6,885
	9	12,149	15,933	0,000	0,000	6,127
15	5	12,334	6,944	0,000	0,000	8,209
	7	10,410	11,099	0,000	0,000	8,322
	9	11,790	13,160	0,000	0,000	9,403
20	5	16,691	6,810	0,000	0,000	11,913
	7	11,360	10,480	0,000	0,000	12,360
	9	11,501	14,225	0,000	0,000	10,810

Tabla 11: Calidad de las cotas y las heurísticas para las instancias binarias pequeñas. Para cada tamaño de instancias y número de opciones se muestra el gap de optimalidad de la cota por opciones separadas (Opt. Gap Cota Opc., expresado en %), el gap de optimalidad para la cota de transporte (Opt. Gap Cota tran., expresado en %), y los gaps de optimalidad para las heurísticas (Opt. Gap BDP-1, Opt. Gap BDP-2 y Opt. Gap GC, respectivamente)

Tamaño	# Opc.	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)	T. máx. comp. (s)
10	5	0,000	54	0,0005	0,0010
	7	0,000	54	0,0011	0,0060
	9	0,000	54	0,0029	0,0140
15	5	0,000	54	0,0010	0,0040
	7	0,000	54	0,0032	0,0100
	9	0,000	54	0,008	0,036
20	5	0,000	54	0,0023	0,0190
	7	0,000	54	0,0094	0,0510
	9	0,000	54	0,0278	0,1620

Tabla 12: Resultados del branch-and-bound para las instancias binarias pequeñas. Para cada tamaño de instancias y número de opciones se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T- máx. comp.) también en segundos.

7.3. Resultados para instancias binarias medianas de Fliedner et al. (2010)

Como se observa en la tabla 13, en el caso del set de instancias de tamaño mediano, la dificultad para encontrar la solución óptima de las instancias es superior para la heurística de goal-chasing, ya que el método goal-chasing no es capaz de resolver muchas de ellas. Sin embargo, el procedimiento por programación dinámica acotada sigue siendo capaz de obtener la mejor solución para todas las instancias planteadas.

Tamaño	# Opc.	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
25	5	54	54	54	54	54	0	0
	7	54	54	54	54	54	1	1
	9	54	54	54	54	54	2	2
30	5	54	54	54	54	54	3	3
	7	54	54	54	54	54	0	0
	9	54	54	54	54	54	4	4
35	5	54	54	54	54	54	1	1
	7	54	54	54	54	54	0	0
	9	54	54	54	54	54	1	1

Tabla 13: Resultados de las heurísticas para las instancias binarias medianas. Para cada tamaño de instancias y número de opciones se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

La tabla 14 confirma las conclusiones alcanzadas con anterioridad (véanse comentarios asociados a la tabla 11). El gap de optimalidad de la heurística de goal-chasing indica que este método obtiene mejores resultados para instancias binarias que para no binarias.

Y en cuanto al procedimiento exacto, cuyos resultados se muestran en la tabla 15, los tiempos de ejecución siguen siendo muy pequeños (siendo el mayor de ellos 2,5 segundos) y es posible demostrar la optimalidad para todas las instancias del grupo.

Tamaño	# Opc.	Opt. Gap LB Cota opc.	Opt. Gap LB Cota tran.	Opt. Gap UB BDP-1	Opt. Gap UB BDP-2	Opt. Gap UB GC
25	5	8,8613	7,8353	0,0000	0,0000	10,8121
	7	10,035	10,607	0,000	0,000	13,024
	9	14,104	15,415	0,000	0,000	11,465
30	5	9,9887	8,0595	0,0000	0,0000	13,0548
	7	10,1788	11,0673	0,0000	0,0000	14,1852
	9	13,345	14,316	0,000	0,000	12,939
35	5	8,642	8,952	0,000	0,000	10,692
	7	11,508	12,914	0,000	0,000	12,112
	9	14,971	16,402	0,000	0,000	12,418

Tabla 14: Calidad de las cotas y las heurísticas para las instancias binarias medianas. Para cada tamaño de instancias y número de opciones se muestra el gap de optimalidad de la cota por opciones separadas (Opt. Gap Cota Opc., expresado en %), el gap de optimalidad para la cota de transporte (Opt. Gap Cota tran., expresado en %), y los gaps de optimalidad para las heurísticas (Opt. Gap BDP-1, Opt. Gap BDP-2 y Opt. Gap GC, respectivamente)

Tamaño	# Opc.	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)	T. máx. comp. (s)
25	5	0,000	54	0,0094	0,0820
	7	0,000	54	0,0208	0,1090
	9	0,000	54	0,0729	0,4529
30	5	0,000	54	0,0290	0,4599
	7	0,000	54	0,0484	0,2940
	9	0,000	54	0,1313	0,7409
35	5	0,000	54	0,0619	0,4019
	7	0,000	54	0,1408	1,2958
	9	0,000	54	0,3853	2,5016

Tabla 15: Resultados del branch-and-bound para las instancias binarias medianas. Para cada tamaño de instancias y número de opciones se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T. máx. comp.) también en segundos

7.4. Resultados para instancias binarias derivadas del Car Sequencing Problem

Este conjunto de instancias resulta más difícil de resolver por las heurísticas implementadas. Como se observa en la tabla 16, el procedimiento de programación dinámica acotado más potente (BDP que incluye cotas por opciones separadas) es capaz de resolver de forma óptima las instancias más pequeñas del conjunto. Sin embargo, conforme el tamaño de las instancias aumenta, los procedimientos heurísticos resultan menos efectivos, y en el caso de las instancias más grandes, de 75 o más unidades, no acostumbra a encontrar la mejor solución conocida. Por su parte, el algoritmo de goal-chasing demuestra sus limitaciones al no encontrar ninguna de las mejores soluciones, ni siquiera para las instancias de 25 unidades.

En la tabla 17 se observan los optimality gap para las heurísticas y las cotas.

Tamaño	Familia	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
25	charme	5	5	5	5	5	0	0
	p_4_72	5	5	5	5	5	0	0
	p_6_76	5	5	5	5	5	0	0
	p_10_93	5	5	5	5	5	0	0
	p_16_81	5	5	5	5	5	0	0
	p_19_71	5	5	5	5	5	0	0
	p_21_90	5	5	5	5	5	0	0
	p_26_82	5	5	5	5	5	0	0
	p_36_92	5	5	5	5	5	0	0
	p_41_66	5	5	5	5	5	0	0
50	charme	5	2	2	4	4	0	0
	p_4_72	5	2	2	5	5	0	0
	p_6_76	5	3	3	4	4	0	0
	p_10_93	5	4	4	5	5	0	0
	p_16_81	5	0	0	4	4	0	0

Tabla 16: Resultados de las heurísticas para las instancias binarias. Para cada tamaño y familia de instancias se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

Tamaño	Familia	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
50	p_19_71	5	3	3	5	5	0	0
	p_21_90	5	2	2	5	5	0	0
	p_26_82	5	3	3	5	5	0	0
	p_36_92	5	4	4	5	5	0	0
	p_41_66	5	5	5	5	5	0	0
75	charme	5	1	1	2	2	0	0
	p_4_72	5	0	0	0	1	0	0
	p_6_76	5	1	1	3	4	0	0
	p_10_93	5	1	1	3	3	0	0
	p_16_81	5	0	0	2	3	0	0
	p_19_71	5	0	1	2	3	0	0
	p_21_90	5	1	1	1	1	0	0
	p_26_82	5	0	0	0	0	0	0
	p_36_92	5	1	1	2	2	0	0
	p_41_66	5	2	2	4	4	0	0
100	charme	1	0	0	0	0	0	0
	p_4_72	1	0	1	0	1	0	0
	p_6_76	1	0	0	0	0	0	0
	p_10_93	1	0	0	0	0	0	0
	p_16_81	1	0	0	0	1	0	0
	p_19_71	1	0	0	0	0	0	0
	p_21_90	1	0	0	0	0	0	0
	p_26_82	1	0	0	0	0	0	0
	p_36_92	1	0	0	0	0	0	0
	p_41_66	1	0	0	0	1	0	0

Tabla 16 (cont.): Resultados de las heurísticas para las instancias binarias. Para cada tamaño y familia de instancias se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

Tamaño	Familia	Opt. Gap LB Cota opc.	Opt. Gap LB Cota tran.	Opt. Gap UB BDP-1	Opt. Gap UB BDP-2	Opt. Gap UB GC
25	charme	6,3029	5,3119	0,0000	0,0000	11,8457
	p_4_72	7,1674	7,5866	0,0000	0,0000	13,9950
	p_6_76	8,1469	4,9642	0,0000	0,0000	13,1127
	p_10_93	7,6543	5,3953	0,0000	0,0000	12,4922
	p_16_81	5,7193	2,8877	0,0000	0,0000	16,0420
	p_19_71	7,5276	5,0777	0,0000	0,0000	9,5307
	p_21_90	6,2281	3,9203	0,0000	0,0000	12,6204
	p_26_82	5,1824	5,8118	0,0000	0,0000	11,4418
	p_36_92	5,4224	4,4493	0,0000	0,0000	8,0098
	p_41_66	5,7502	4,3601	0,0000	0,0000	6,2336
50	charme	4,4643	3,6000	1,4405	0,0287	14,4043
	p_4_72	4,7459	3,5610	0,3267	0,0000	13,4125
	p_6_76	5,7355	4,7227	0,2248	0,1597	13,7881
	p_10_93	5,4042	4,1401	0,1665	0,0000	10,5582
	p_16_81	4,6701	3,6411	0,8220	0,0352	13,7695
	p_19_71	4,0597	4,1367	0,3502	0,0000	16,9746
	p_21_90	7,6549	5,7977	0,3664	0,0000	10,6963
	p_26_82	6,0233	4,4540	0,2323	0,0000	11,3207
	p_36_92	6,4811	5,0013	0,0657	0,0000	13,3374
	p_41_66	3,7971	2,9831	0,0000	0,0000	14,4005
75	charme	4,2260	3,3464	0,9760	0,1740	9,5114
	p_4_72	5,9094	5,2249	6,2601	4,6365	19,0839
	p_6_76	6,3700	4,6390	2,0993	1,7149	15,0700
	p_10_93	3,5328	3,0189	1,9610	1,1950	13,4831
	p_16_81	4,0228	2,6854	2,9867	1,4478	11,9211
	p_19_71	4,2206	3,4208	1,5223	0,9392	15,1971
	p_21_90	3,2984	2,9997	0,8928	0,6784	11,4116
	p_26_82	6,0049	4,7326	3,7686	3,4148	14,9367
	p_36_92	4,4516	3,6814	2,4419	1,8369	16,5377
	p_41_66	4,7301	3,2085	0,3378	0,0442	9,8820

Tabla 17: Calidad de las cotas y las heurísticas para las instancias binarias. Para cada tamaño y familia de instancias se muestra el gap de optimalidad de la cota por opciones separadas (Opt. Gap Cota Opc., expresado en %), el gap de optimalidad para la cota de transporte (Opt. Gap Cota tran., expresado en %), y los gaps de optimalidad para las heurísticas (Opt. Gap BDP-1, Opt. Gap BDP-2 y Opt. Gap GC, respectivamente)

Tamaño	Familia	Opt. Gap LB Cota opc.	Opt. Gap LB Cota tran.	Opt. Gap UB BDP-1	Opt. Gap UB BDP-2	Opt. Gap UB GC
100	charme	3,2831	2,0241	1,0238	0,0450	15,8562
	p_4_72	4,5883	3,6462	3,6462	3,6462	16,9786
	p_6_76	0,9800	1,1293	1,0187	1,0187	17,8005
	p_10_93	0,8604	0,4949	6,3903	0,8567	15,4829
	p_16_81	3,1201	2,4873	3,2670	2,2609	16,0419
	p_19_71	6,3395	4,3942	4,8315	3,8359	14,4949
	p_21_90	4,2065	3,4985	4,7265	3,4223	22,5407
	p_26_82	1,5245	1,4611	2,4274	1,8111	15,2029
	p_36_92	3,4837	3,1937	3,4943	2,7198	17,7847
	p_41_66	4,5510	4,0652	3,1909	2,8059	12,3346

Tabla 17 (cont.): Calidad de las cotas y las heurísticas para las instancias binarias. Para cada tamaño y familia de instancias se muestra el gap de optimalidad de la cota por opciones separadas (Opt. Gap Cota Opc., expresado en %), el gap de optimalidad para la cota de transporte (Opt. Gap Cota tran., expresado en %), y los gaps de optimalidad para las heurísticas (Opt. Gap BDP-1, Opt. Gap BDP-2 y Opt. Gap GC, respectivamente)

Tal como se observa en la tabla 17, la cota de transporte tiene un valor generalmente mejor que la cota por separabilidad de opciones, aunque estas diferencias no son muy elevadas, y ambas tienen optimality gaps pequeños, especialmente para las instancias más grandes.

Tamaño	Familia	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)	T. máx. comp. (s)
25	charme	0,000	5	0,0112	0,0400
	p_4_72	0,000	5	0,0508	0,1320
	p_6_76	0,000	5	0,0058	0,0200
	p_10_93	0,000	5	0,0532	0,0980
	p_16_81	0,000	5	0,0152	0,0540
	p_19_71	0,000	5	0,0248	0,0420
	p_21_90	0,000	5	0,0230	0,0950
	p_26_82	0,000	5	0,0156	0,0420
	p_36_92	0,000	5	0,0144	0,0430
	p_41_66	0,000	5	0,0012	0,0050

Tabla 18: Resultados del branch-and-bound para las instancias binarias medianas. Para cada tamaño y familia de instancias se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T. máx. comp.) también en segundos

Tamaño	Familia	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)	T. máx. comp. (s)
50	**charme**	0,000	5	10,582	18,579
	p_4_72	0,000	5	85,449	245,826
	p_6_76	0,000	5	168,286	633,339
	p_10_93	0,000	5	157,142	248,688
50	**p_16_81**	0,000	5	125,551	314,746
	p_19_71	0,000	5	40,635	88,807
	p_21_90	0,000	5	203,343	368,566
	p_26_82	0,000	5	95,633	228,138
	p_36_92	0,000	5	128,147	500,856
	p_41_66	0,000	5	2,834	7,228
75	**charme**	0,000	5	1240,780	3917,505
	p_4_72	4,0911	0	13830,29	14112,19
	p_6_76	1,6713	3	9919,46	14159,57
	p_10_93	1,1331	3	7737,49	13886,62
	p_16_81	1,1148	3	7759,58	13443,37
	p_19_71	0,9005	3	9871,88	13871,63
	p_21_90	0,4635	4	4563,46	14060,12
	p_26_82	3,0370	1	12249,85	14049,28
	p_36_92	1,6709	3	9610,05	14119,89
	p_41_66	0,0000	5	3880,75	8523,47
100	**charme**	0,0000	1	7148,86	-
	p_4_72	3,6462	0	6838,82	-
	p_6_76	0,6391	0	12032,80	-
	p_10_93	0,4302	0	10139,48	-
	p_16_81	2,2609	0	499,27	-
	p_19_71	3,5416	0	6053,95	-
	p_21_90	3,0907	0	11802,79	-
	p_26_82	1,2705	0	8178,17	-
	p_36_92	2,5460	0	12144,09	-
	p_41_66	2,8059	0	13316,20	-

Tabla 18 (cont.): Resultados del branch-and-bound para las instancias binarias medianas. Para cada tamaño y familia de instancias se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T. máx. comp.) también en segundos

También se observa una diferencia apreciable en los resultados obtenidos para las heurísticas: a pesar de no ser capaz de obtener la mejor solución para muchas instancias, se observan optimality gaps muy pequeños para la BDP con cotas por opciones separadas.

En la tabla 18, se observan los resultados obtenidos con el procedimiento de branch-and -bound.

Se observa que el branch-and-bound es capaz de resolver todas las instancias de 50 o menos productos a secuenciar. Para las instancias de 75 productos, se han resuelto el 60% de las instancias, con 2 horas y media de tiempo medio de computación. Sin embargo, el algoritmo es capaz de resolver solamente una de las instancias de 100 productos en una hora de tiempo real. A pesar de ello, mirando con detenimiento la tabla 16, puede observarse que, en el tiempo límite de una hora, el procedimiento de branch-and-bound mejora la solución de entrada, justificando su uso.

7.5. Resultados para instancias binarias generadas según Drexl y Kimms (2001)

La tabla 19 muestra los resultados de los procedimientos heurísticos para los juegos de datos generados según Drexl y Kimms (2001).

Se observa el programa dinámico acotado (II) es capaz de resolver todas las instancias para menos de 7 opciones, y todas las instancias (incluyendo las de 7 opciones) de tamaño inferior a 60. Las conclusiones de estos resultados se realizan de forma conjunta con el análisis de los optimality gaps mostrados en la tabla 20.

Grupo	Tamaño	# Opc.	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
Easy	10	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	20	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	30	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0

Tabla 19: Resultados de las heurísticas para las instancias según Drexl y Kimms (2001). Para cada tipo de instancias, tamaño y número de opciones se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

Grupo	Tamaño	# Opc.	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
Easy	40	**3**	1	1	1	1	1	1	1
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	50	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	60	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	70	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	0	0	0	1	1	0	0
	80	**3**	1	1	1	1	1	1	1
		5	1	1	1	1	1	0	0
		7	0	0	0	0	0	0	0
	90	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	0	0	0	0	0	0	0
	100	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	0	0	0	1	0	0
Hard	10	**3**	1	1	1	1	1	1	1
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	20	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0

Tabla 19 (cont.): Resultados de las heurísticas para las instancias según Drexl y Kimms (2001). Para cada tipo de instancias, tamaño y número de opciones se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

Grupo	Tamaño	# Opc.	# Inst.	# Ópt. BDP-1	# Best BDP-1	# Ópt. BDP-2	# Best BDP-2	# Ópt. GC	# Best GC
Hard	30	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	40	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	50	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	1	1	1	1	0	0
	60	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	0	1	0	1	0	0
	70	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	0	0	0	0	0	0
	80	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	0	1	0	1	0	0
	90	**3**	1	1	1	1	1	0	0
		5	1	1	1	1	1	0	0
		7	1	0	0	0	0	0	0
	100	**3**	1	1	1	1	1	0	0
		5	1	0	0	1	1	0	0
		7	1	0	0	0	1	0	0

Tabla 19 (cont.): Resultados de las heurísticas para las instancias según Drexl y Kimms (2001). Para cada tipo de instancias, tamaño y número de opciones se muestra el número de instancias resueltas (# Inst.), el número de óptimos y mejor soluciones encontradas por la BDP sin cotas (# Ópt. BDP-1, #Best BDP-1, respectivamente), el número de óptimos y mejores soluciones encontradas por la BDP con cota por separabilidad de opciones (# Ópt. BDP-2, #Best BDP-2, respectivamente), y el número de óptimos y mejores soluciones encontradas por la heurísticas de goal-chasing (# Ópt. GC, #Best GC, respectivamente)

Grupo	Tamaño	# Opc.	Opt. Gap LB Cota opc.	Opt. Gap LB Cota tran.	Opt. Gap UB BDP-1	Opt. Gap UB BDP-2	Opt. Gap UB GC
Easy	10	**3**	0,000	0,000	0,000	0,000	12,121
		5	11,650	1,942	0,000	0,000	3,738
		7	5,517	6,897	0,000	0,000	18,079
	20	**3**	0,000	0,000	0,000	0,000	9,375
		5	36,554	2,089	0,000	0,000	12,756
		7	3,484	6,272	0,000	0,000	4,013
	30	**3**	0,000	0,000	0,000	0,000	10,309
		5	3,280	2,343	0,000	0,000	10,109
		7	0,957	5,107	0,000	0,000	9,003
	40	**3**	0,000	0,000	0,000	0,000	0,000
		5	6,150	3,690	0,000	0,000	10,953
		7	2,115	5,729	0,000	0,000	12,360
	50	**3**	0,000	0,275	0,000	0,000	1,625
		5	7,698	3,448	0,000	0,000	5,745
		7	2,001	5,559	0,000	0,000	19,508
	60	**3**	0,000	0,096	0,000	0,000	4,235
		5	1,837	1,837	0,000	0,000	10,297
		7	0,949	4,823	0,000	0,000	10,348
	70	**3**	0,000	0,000	0,000	0,000	2,077
		5	3,881	2,156	0,000	0,000	8,952
		7	3,513	3,846	0,302	0,000	10,418
	80	**3**	0,000	0,000	0,000	0,000	0,000
		5	6,259	3,129	0,000	0,000	14,911
		7	1,093	4,877	1,173	0,568	17,486
	90	**3**	0,000	0,000	0,000	0,000	2,096
		5	0,247	4,097	0,000	0,000	6,333
		7	0,446	6,363	0,514	0,377	15,304
	100	**3**	0,000	0,275	0,000	0,000	1,625
		5	1,013	0,760	0,000	0,000	8,780
		7	3,141	3,704	3,869	2,445	19,932

Tabla 20: Calidad de las cotas y las heurísticas para las instancias según Drexl y Kimms (2001). Para cada familia de instancias, tamaño y número de opciones se muestra el gap de optimalidad de la cota por opciones separadas (Opt. Gap Cota Opc., expresado en %), el gap de optimalidad para la cota de transporte (Opt. Gap Cota tran., expresado en %), y los gaps de optimalidad para las heurísticas (Opt. Gap BDP-1, Opt. Gap BDP-2 y Opt. Gap GC, respectivamente)

Grupo	Tamaño	# Opc.	Opt. Gap LB Cota opc.	Opt. Gap LB Cota tran.	Opt. Gap UB BDP-1	Opt. Gap UB BDP-2	Opt. Gap UB GC
Hard	10	**3**	0,000	0,000	0,000	0,000	0,000
		5	3,846	0,000	0,000	0,000	23,529
		7	7,273	9,697	0,000	0,000	23,963
	20	**3**	0,000	4,274	0,000	0,000	1,681
		5	4,831	7,730	0,000	0,000	9,607
		7	2,024	7,083	0,000	0,000	13,431
	30	**3**	0,000	3,071	0,000	0,000	6,463
		5	0,869	6,517	0,000	0,000	9,085
		7	3,902	6,904	0,000	0,000	9,752
	40	**3**	0,000	5,302	0,000	0,000	0,841
		5	0,994	6,087	0,000	0,000	12,784
		7	3,423	8,900	0,000	0,000	4,261
	50	**3**	0,000	4,845	0,000	0,000	2,167
		5	0,836	4,685	0,000	0,000	14,577
		7	7,764	7,145	0,000	0,000	21,053
	60	**3**	0,000	4,236	0,000	0,000	2,442
		5	1,770	5,699	0,000	0,000	18,120
		7	3,045	6,661	2,131	2,131	15,619
	70	**3**	0,000	5,198	0,000	0,000	1,818
		5	0,331	3,305	0,000	0,000	3,662
		7	3,300	6,999	3,108	2,723	17,908
	80	**3**	0,000	5,302	0,000	0,000	3,282
		5	0,747	5,853	0,000	0,000	14,026
		7	2,341	7,976	2,081	2,081	9,964
	90	**3**	0,000	3,372	0,000	0,000	3,019
		5	0,945	5,022	0,000	0,000	14,823
		7	3,868	6,664	4,053	3,683	19,094
	100	**3**	0,000	5,130	0,000	0,000	1,569
		5	0,566	5,683	0,040	0,000	5,612
		7	3,362	8,859	3,361	3,334	12,234

Tabla 20 (cont.): Calidad de las cotas y las heurísticas para las instancias según Drexl y Kimms (2001). Para cada familia de instancias, tamaño y número de opciones se muestra el gap de optimalidad de la cota por opciones separadas (Opt. Gap Cota Opc., expresado en %), el gap de optimalidad para la cota de transporte (Opt. Gap Cota tran., expresado en %), y los gaps de optimalidad para las heurísticas (Opt. Gap BDP-1, Opt. Gap BDP-2 y Opt. Gap GC, respectivamente)

Puede observarse que, al igual que en el problema CSP, las instancias hard (sobre todo para número de unidades grande) son más difíciles que las instancias easy. Pero las diferencias son mínimas, especialmente para las heurísticas y las cotas (aunque existen instancias individuales en que este fenómeno se aprecia en sentido contrario).

De la misma manera, en general, la cota por opciones es mejor que la cota de transporte para este juego de instancias.

Grupo	Tamaño	# Opc.	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)
Easy	10	**3**	0,000	1	0,001
		5	0,000	1	0,000
		7	0,000	1	0,001
	20	**3**	0,000	1	0,000
		5	0,000	1	0,001
		7	0,000	1	0,004
	30	**3**	0,000	1	0,001
		5	0,000	1	0,001
		7	0,000	1	0,001
	40	**3**	0,000	1	0,000
		5	0,000	1	0,000
		7	0,000	1	2,706
	50	**3**	0,000	1	0,001
		5	0,000	1	0,000
		7	0,000	1	82,274
	60	**3**	0,000	1	0,000
		5	0,000	1	0,001
		7	0,000	1	138,529
	70	**3**	0,000	1	0,000
		5	0,000	1	0,001
		7	0,000	1	3551,352
	80	**3**	0,000	1	0,000
		5	0,000	1	0,001
		7	0,525	0	14363,111

Tabla 21: Resultados del branch-and-bound para las instancias según Drexl y Kimms (2001). Para cada familia de instancias, tamaño y número de opciones se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T. máx. comp.) también en segundos

Grupo	Tamaño	# Opc.	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)
Easy	90	**3**	0,000	1	0,000
		5	0,000	1	0,000
		7	0,343	0	14364,616
	100	**3**	0,000	1	0,000
		5	0,000	1	0,000
		7	2,445	0	14353,335
Hard	10	**3**	0,000	1	0,000
		5	0,000	1	0,001
		7	0,000	1	0,002
	20	**3**	0,000	1	0,000
		5	0,000	1	0,003
		7	0,000	1	0,012
	30	**3**	0,000	1	0,001
		5	0,000	1	0,003
		7	0,000	1	0,503
	40	**3**	0,000	1	0,000
		5	0,000	1	0,010
		7	0,000	1	16,707
	50	**3**	0,000	1	0,001
		5	0,000	1	0,100
		7	0,000	1	966,825
	60	**3**	0,000	1	0,001
		5	0,000	1	2,893
		7	2,131	0	14359,020
	70	**3**	0,000	1	0,000
		5	0,000	1	0,635
		7	2,447	0	14354,142
	80	**3**	0,000	1	0,000
		5	0,000	1	1,446
		7	2,081	0	14363,316

Tabla 21 (cont.): Resultados del branch-and-bound para las instancias según Drexl y Kimms (2001). Para cada familia de instancias, tamaño y número de opciones se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T. máx. comp.) también en segundos

Grupo	Tamaño	# Opc.	Opt. Gap BB (%)	# Ópt. BB	T. medio comp. (s)
Hard	90	**3**	0,000	1	0,001
		5	0,000	1	16,539
		7	3,446	0	14355,829
	100	**3**	0,000	1	0,001
		5	0,000	1	6,215
		7	3,334	0	14352,582

Tabla 21(cont.): Resultados del branch-and-bound para las instancias según Drexl y Kimms (2001). Para cada familia de instancias, tamaño y número de opciones se muestra el gap de optimalidad del procedimiento exacto (Opt. Gap BB), el número de óptimos demostrados por el procedimiento (# Ópt. BB), el tiempo medio de computación (T. medio comp.) en segundos y el tiempo de computación máximo (T. máx. comp.) también en segundos

En la tabla 21 pueden apreciarse los resultados para el procedimiento de branch-and-bound. En general, este procedimiento es capaz de resolver instancias de hasta 5 opciones y del tamaño máximo considerado (100 unidades). Sin embargo, presenta problemas para resolver instancias con 7 opciones de 60 o más unidades, probablemente debido a que la cota por opciones es menos efectiva al incrementar el número de opciones.

Conclusiones

Tras el análisis de los resultados presentados, se concluye que el procedimiento exacto presentado para la resolución de líneas de montaje mixed-model con consumos no binarios es capaz de resolver instancias de hasta 60 unidades de cualquier juego de datos, siendo capaz de resolver instancias de mayor tamaño dependiendo de las características. A pesar de que para las instancias de mayor tamaño es necesario limitar el tiempo de ejecución del algoritmo exacto, habitualmente es capaz de mejorar las soluciones de partida. Por otra parte, hay que destacar la calidad de las soluciones obtenidas por los métodos BDP, que en la mayoría de ocasiones obtienen la solución óptima del problema. En caso contrario, la solución que ofrecen es comparable con la solución obtenida por el procedimiento exacto.

Para poder comparar la calidad del método exacto, hay que observar que:

- El tamaño de las instancias resueltas de forma óptima para el problema que origina el juego de datos de Fliedner et al. (2010) es de 25 unidades
- El tamaño de las instancias resueltas de forma óptima para el juego de datos derivado de Bautista et al. (1996) era de 20 unidades
- El tamaño de las instancias resueltas de forma óptima para el problema que origina el juego de datos de Drexl y Kimms (2001) es de 30 unidades

Por tanto, aunque la dificultad del problema ORV es comparable con la de los otros problemas, este algoritmo es más eficiente y es capaz de resolver instancias el doble de grandes que procedimientos anteriores.

El procedimiento presenta otra limitación relacionada con las cotas por opciones separadas: esta cota se calcula mediante la resolución de un programa dinámico, en el que se deben generar y almacenar todos los estados. Se ha observado que esto es factible para tres opciones o menos, pero incluso sin tener en cuenta todas las opciones, si se aplicara este procedimiento a instancias de mayor tamaño que las que se han empleado en las experiencias computacionales, el número de estados a almacenar podría ser demasiado grande y generar problemas de memoria. Por tanto, a pesar de que se ha comprobado que la cota que explota la separabilidad del problema por opciones es eficiente, especialmente para las instancias de consumo no binario, esta cota no podría usarse en ordenadores de sobremesa para resolver instancias de gran tamaño.

Bibliografía

Referencias bibliográficas

Bautista, J.; Companys, R.; Corominas, A., Heuristics and exact algorithms for solving the Monden problem. *European Journal of Operational Research*, 88, 101-113 (1996)

Boysen, N. Bock, S.; Schedulling just-in-time part supply for mixed-model assembly lines. *European Journal of Operational Research*, 211, 15-25 (2011)

Boysen, N.; Fliedner, M.; Scholl, A., Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192, 349-373 (2007)

Boysen, N.; Fliedner, M.; Scholl, A., Sequencing mixed-model assembly lines to minimize part inventory cost, *Operations Research Spectrum*, 30, 611-633 (2008)

Drexl, A., Kimms, A., Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints, *Management Science,* 12, 480–491 (2001)

Erel, E.; Gocgun, Y.; Sabuncouğlu, I., Mixed-model assembly line sequencing using beam search, *International Journal of Production*, 45 (22), 5265-5284 (2007)

Fliedner, M.; Boysen, N.; Scholl, A., Solving symmetric mixed-model multi-level just-in-time scheduling problems. *Discrete Applied Mathematics*, 158, 222-231 (2010)

Ghosh, S.; Gagnon, R.J., A comprehensible literature review and analysis of the design, balancing and scheduling of assembly systems. *International Journal of Production Research*, 27 (4), 637-670 (1989)

Jin, M.; Wu, D.S., A new heuristic method for mixed model assembly line balancing problem. *Computers & Industrial Engineering*, 44, 159-169 (2002)

Kubiak, W.; Sethi, S., A note on schedules for mixed-model assembly lines in just-in-time production systems. *Management Science*, 37, 121-122 (1991)

Kubiak, W., Minimizing variation of production rates in just-in-time systems: A survey. *European Journal of Operational Research*, 66, 259-271 (1993)

Leu, Y. –Y.; Huang, P. Y.; Russell, R. S.; Using beam search techniques for sequencing mixed-model assembly lines, *Annals of Operational Research*, 70, 379-397 (1997)

Miltenburg, J., Level schedules for mixed-model assembly lines in just-in-time production systems. *Management Science*, 35, 192–207 (1989).

Monden, Y., Toyota production system (2nd ed) Institute of Industrial Engineering, Norcross, GA (1983)

Parrello, B. D.; Kabat, W. C.; Wos, L., Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem, *Journal of Automated Reasoning*, 2 (1), 1-42 (1986)

Sumichrast R. T ; Russell, R. S., Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems. *Journal of Operations Management*, 9 (3), 371-390 (1990)

Tarjan, R. E., Data Structures and Network Algorithms. *Society for Industrial and Applied Mathematics* (1983)

Thomopoulos, N. T., Line balancing-sequencing for mixed-model assembly. *Management Science*, 14 (2), 59-75 (1967)

Yano, C. A; Bolat, A., Survey, development, and applications of algorithms for sequencing paced assembly lines. *Technical Report*, 88-13 (1989)

Bibliografía adicional:

Chapman, B.; Jost, G.; Van Der Pas, V., Using OpenMP: Portable shared memory parallel programming, *The MIT Pres (Scientific and Engineering Computation series)*, 2008

Quinn, M. J., Parallel programming in C with MPI and OpenMP, *McGraw-Hill*, 2003