FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

INFORME PROYECTO DE GRADO
INGENIERÍA EN COMPUTACIÓN

ALGORITMOS PARA EL PROBLEMA DE DIMENSIONAMIENTO DEL LOTE ECONÓMICO CON MÚLTIPLES NIVELES Y RESTRICCIONES DE CAPACIDAD

5 de marzo de 2017

AUTORES
Andrés Tipoldi Nimo
Emiliano Vázquez Troitiño

TUTORES
Pedro Piñeyro Cabral
Carlos Testuri Cedrés

"La recompensa está en el camino, no en el resultado"
Agradezco profundamente alcanzar este punto del camino a todos mis seres queridos, en especial a mi esposa Adriana e hija Cami. A ellas gracias por la infinita paciencia y comprensión por mi ausencia durante el estudio, el ánimo fundamental para avanzar siempre.
También a mis amigos por el apoyo sostenido e incondicional durante tanto tiempo.
Por último, a nuestros tutores Pedro Piñeyro y Carlos Testuri por hacer este Proyecto de Grado tan disfrutable, motivadores, dedicados y con una disponibilidad inmejorable.

Emiliano Vázquez Troitiño

Gracias a mis papás que me dieron las herramientas y la tenacidad para no rendirme, y a mi hermanita que estuvo y estará en todo momento para darme fuerza.
Agradezco el apoyo incondicional de Estefa quien me alentó a retomar la carrera y me apaciguó en mis momentos de frustración y dudas, sin ella no hubiese sido posible.
Gracias a Carlos y Pedro (nuestros tutores) por la paciencia que nos tuvieron y la dedicación desde el primer momento. Y por último, agradezco al destino por haberme dado un gran compañero como Emiliano.

Andrés Tipoldi Nimo

Resumen

En una red de distribución típica de un producto, existen múltiples niveles, desde el productor, pasando por varios distribuidores intermedios hasta llegar al minorista. Estos niveles con sus características específicas conforman la red de distribución necesaria para satisfacer a tiempo la cantidad de producto demandado por el consumidor final.

Seleccionar una adecuada política de distribución significa un complejo desafío para las organizaciones, teniendo que decidir cuándo y cuánto producir o almacenar para minimizar los costos implicados. Por esta razón, surge El Problema de Dimensionamiento de Lote; un modelo de planificación de producción que procura establecer los tamaños de lotes a producir en cada período, considerando las características de los niveles y la demanda futura para minimizar los costos de inventario y producción.

En este proyecto de grado se abordó en particular, un problema de Dimensionamiento de Lotes con múltiples niveles y capacidades de producción variables (en cada período y nivel el límite máximo de producción cambia). Éste es una extensión del problema de un solo nivel con restricciones de capacidades, el cual se ha demostrado pertenecer a la clase de problemas NP-Hard.

El objetivo del proyecto fue el desarrollo de un procedimiento heurístico que elabore una planificación de producción conveniente en tiempos computacionales razonables. La heurística implementada se basó en el algoritmo Dijkstra para el problema del camino más corto en un grafo.

Para analizar los resultados obtenidos se realizó un estudio comparativo con un método de resolución exacta y otros procedimientos heurísticos existentes (Lote por Lote, Lote de Tamaño Fijo y Silver-Meal). Para el método de resolución exacta se utilizo el solver de GLPK (herramienta de uso académico).

El tiempo de búsqueda de una solución óptima se acotó explícitamente por la naturaleza del problema, respondiendo a este hecho, a medida que crece el tamaño del problema la desviación al óptimo aumenta por parte de GLPK y los métodos de resolución heurísticos van tomando mayor relevancia.

Todos los métodos de resolución propuestos se evaluaron con un gran número casos de prueba, las instancias fueron generadas teniendo en cuenta una amplia variedad de configuraciones (costos, capacidades, etcétera). De los resultados obtenidos se puede concluir que el objetivo del inicio del proyecto se logró concretar. Para todos los casos de prueba en el cual el método de resolución exacto logró encontrar una solución óptima, la heurística propuesta basada en Dijkstra fue capaz de obtener una solución con un valor muy cercano a éste en tiempos computacionales razonables.

Palabras claves: Control de Inventario, Capacitated Lot Sizing Problem, Multi-echelon, Optimización

Tabla de Contenido

1. Introducción

Mantener un inventario para su uso futuro es una práctica común para una gran cantidad de empresas. La decisión de cuando fabricar o cuando almacenar para su uso posterior resulta una decisión muy importante para la disminución de costos y por consiguiente la optimización de los procesos de producción, almacenamiento y distribución.

En empresas pequeñas los administradores pueden tomar este tipo de decisiones haciendo un recuento del inventario, y estableciendo cuando es mejor producir. Pero en empresas de mayor porte donde pueden existir varios niveles de producción distribuidos en diversos establecimientos o locaciones, se vuelve una necesidad contar con un enfoque más científico con respecto a sus políticas de inventario.

Es así que surge la Teoría de Inventarios, la cual establece modelos matemáticos con el fin de minimizar los costos de las empresas, buscando el mejor momento para producir o inventariar según la realidad de la compañía frente a varios períodos de demanda a lo largo del tiempo. Obviamente cada empresa está determinada por una realidad distinta dependiendo entre otras cosas de: el producto en cuestión, niveles de producción, horizonte de planificación, etc.

Es dentro de la teoría de inventarios donde parte el estudio de los problemas de dimensionamiento de lote. Donde en cada nivel de producción, desde el primero (fuente) hasta el último (minorista) el producto va sufriendo modificaciones que le agregan valor. En estos niveles es donde se toma la decisión de re abastecer el inventario o producir (enviándolo hacia el siguiente nivel). Esta decisión no solo se basa tomando en cuenta los costos de producción e inventario sino también considerando las capacidades de producción para cada nivel y período.

De esta forma, aunque pueda ser conveniente producir en el primer período para satisfacer todas las demandas, quizás resulte imposible si su capacidad de producción es insuficiente, la producción entonces tendrá que ser separada en distintos períodos.

Con el fin de poder ilustrar este tipo de problemas de forma sencilla es que surgen representaciones por medio de un diagrama de red. Estas permiten plantear de forma gráfica una realidad particular que se desea describir. Como ejemplo se puede ver en la Figura 1 la representación en red de un problema de diez períodos con tres niveles. El vértice en el nivel l en el período t se representa como (l, t) y tiene un arco entrante de producción de cantidad $x_{l,t}$. Dada esta representación del problema, un plan de producción y distribución está dado por la distribución de las $d[1, 10]$ unidades desde los vértices productores $(1, t)$ pasando por sus vértices intermedios (l, t) hacia los vértices finales (L, t), con $1 < l < L$.

1.1. Clasificación de Problemas

El estudio de la teoría de inventarios se inicia a comienzos del siglo XX y continua hasta la fecha. Con Harris en 1913 [8] presentando la cantidad económica de pedido, pasando por Wagner y Whitin en 1958[24] quienes comienzan con el estudio de dimensionamiento de lote, hasta llegar a la fecha con autores contemporáneos como ser Wolsey, Pochet, Van Vyve y Van Hoesel, entre otros.

Dentro de la gran cantidad de artículos concernientes a esta área de investigación, se hacen varios acercamientos diferentes al problema. La clasificación más general junto con los principales referentes en cada una de ellas se muestra a continuación:

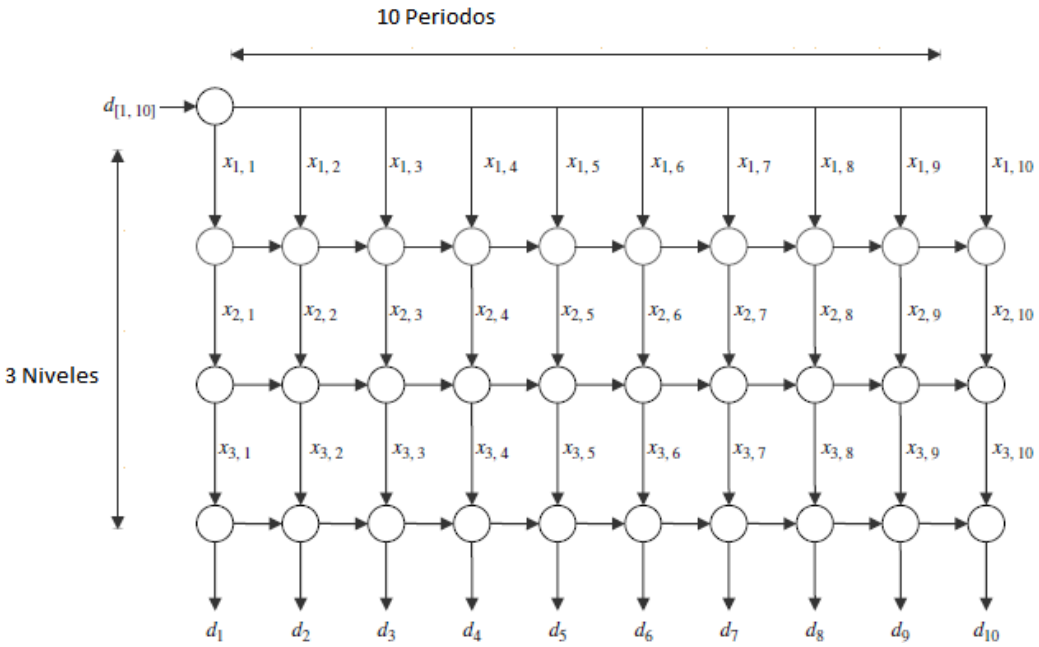

Figura 1: La cadena de producción esta conformada por varios niveles. Cada uno de ellos agrega valor al producto y lo envía al siguiente nivel o lo almacena para ser utilizado en otro período.

a. **ELSP** - Problema de Dimensionamiento de Lote Económico
Artículos: Wagner y Whitin en 1958[24], Veinott 1963 [23], Zabel y Eppen 1964 [25]

b. **CLSP** - Problema de Dimensionamiento de Lote con Capacidades
Artículos: Florian and Klein en 1971[6], Florian et al 1980[7], Bitran Yanasse en 1982[3], Chung y Lin 1988[5], Van hoesel y Waelmans 1996[20], Van Heuvel y Wagelmans 2006[19]

c. **MLSP** - Problema de Dimensionamiento de Lote con Múltiples Niveles
Artículos: Manne y Veinott 1966 [12], Zangwill 1966 [26], Zangwill 1969[27], Kaminsky y Simchi-Levi 2003 [9], Sargut y Romeijn 2007 [16], Melo y Wolsey 2012 [13]

d. **MLSP-PC** - Problema de Dimensionamiento de Lote con Múltiples Niveles con Capacidades de Producción.
Artículos: Bitran 1982[3], Van Hoesel 2002[22], van Hoesel et al 2005, Arntzen et al. 1995, Chandra and Fisher 1994, Geoffrion and Powers 1995, Thomas and Griffin 1996

Otra clasificaciones posible es según si se conoce la demanda para cada período o si el valor de la misma esta modelado por una variable aleatoria. Se denominan modelos determinísticos o de demanda aleatoria respectivamente.

1.2. Descripción del Problema

El alcance de este proyecto de grado se definió sobre el marco de esta teoría de inventarios. El objetivo fue proveer una política óptima que permitiera minimizar los costos totales satisfaciendo la demanda para cada período respetando las capacidades de producción establecidas. Siendo que existen diversos problemas dentro de esta área de investigación,

se construyó un modelo matemático propio basado en modelos anteriormente definidos por otros autores.

En particular el proyecto abordó los problemas de inventarios con demanda determinista, con un horizonte de producción finito de T períodos y N niveles de producción con capacidades variables en cada uno de ellos.

Por otro lado, la estructura de costos que se eligió fue la siguiente, el costo de inventario es directamente proporcional a la cantidad almacenada y el costo de producir se divide en dos componentes, una parte proporcional a la cantidad ordenada y otro constante llamado costo de preparación o fijo. Éste último incluye los costos administrativos de colocar una orden, el trabajo preliminar y otros gastos de poner un lote a producirse. Todas las funciones de costos responden a un comportamiento lineal.

Se ha demostrado que este problema pertenece a la clase NP-Hard, y por lo tanto la resolución mediante un método heurístico se hace necesario para obtener resultados próximos al óptimo en tiempos razonables. Según la clasificación realizada por Bitran y Yanasse en 1982 para varios problemas con un solo nivel de producción en [3], en presencia de capacidades de producción variable el problema es NP-Hard. Puesto que el modelo definido es una extensión a varios niveles del mismo, se puede asegurar que no es resoluble en tiempo polinomial.

Dicho lo anterior, el objetivo del proyecto fue el de proponer un procedimiento de resolución del problema para obtener soluciones de buena calidad en un tiempo de computo razonable. Con el fin de lograr estos objetivos se establecieron tres enfoques de resolución para su comparación y análisis:

- Se definió un modelo matemático, para resolver de forma exacta el problema con el uso de una herramienta de resolución por programación entera mixta de uso académico, como es GNU Linear Programming Kit (GLPK).
- Se seleccionaron algunos procedimientos heurísticos conocidos explicados dentro del Estado del Arte [18]. Se implementaron y adaptaron al problema definido para su posterior análisis.
- El estudio del comportamiento de estos métodos ayudó a la creación de una heurística propia, la cual proveyó un resultado cercano a la solución provista por el algoritmo de resolución de GLPK, pero con tiempos de ejecución menores en comparación.

Asimismo, para posibilitar este estudio se generó una gran cantidad de Instancias para ser procesadas tanto por el algoritmo de GLPK como por todas las heurísticas implementadas. Este conjunto de instancias fue clasificado según varios factores, por ejemplo la cantidad de niveles, la cantidad de períodos, la capacidad de producción, etcétera. El resultado de la ejecución de las heurísticas y el método de programación lineal entera mixta sobre este conjunto de instancias determinó la calidad de la heurística propuesta en comparación con las demás. Las características de estos conjuntos de instancias se detallan en el Capítulo de Casos de Prueba (Capítulo 3).

1.3. Formulación del Modelo Matemático

El modelo matemático fue definido para un problema de dimensionamiento de lote de un único artículo con demanda determinista. Es decir se conoce de antemano la demanda para cada uno de los períodos del horizonte de producción finito.

Los períodos representan una partición del horizonte de producción (por ejemplo: semanas, meses, etc). En cada uno de ellos se debe satisfacer la demanda, lo cual podrá realizarse produciendo en ese mismo período o tomando artículos del inventario (los cuales fueron producidos anteriormente). Las producciones en cada período están acotadas por una capacidad de producción, la cual podrá ser variable.

El problema también está formado por múltiples niveles. Es decir en cada periodo de tiempo, existen varios niveles de producción. Cada uno de estos niveles puede representar una fábrica, un depósito, o cualquier establecimiento que mantenga el producto o lo modifique para agregarle valor. Cada período contendrá la misma cantidad de niveles y en cada uno de estos niveles se podrá almacenar en inventario.

La formulación del problema a resolver propone minimizar la suma total de los costos asociados a la producción e inventario para un problema de n niveles (con $1 \leq n \leq N$) y t períodos (con $0 \leq t \leq T$). Estos costos tienen un comportamiento lineal, teniendo la función de costos de producción la particularidad de presentar una estructura con cargo de preparación (costo fijo de efectuar una orden).

$$\min \sum_{n=1}^{N} \sum_{t=1}^{T} (z_t^n k_t^n + c_t^n x_t^n + h_t^n I_t^n)$$

Sujeto a:

$$x_t^n - x_t^{n-1} = I_{t-1}^n - I_t^n, \;\; t = 1, ..., T \;\; y \;\; n = 2, ..., N \tag{1}$$

$$d_t = x_t^L, \;\; t = 1, ..., T \tag{2}$$

$$I_0^n = 0, \;\; n = 1, ..., N \tag{3}$$

$$x_t^n \leq b_t^n, \;\; t = 1, ..., T \;\; y \;\; n = 1, ..., N \tag{4}$$

$$z_t^n (\sum_{i=t}^{T} d_i) \geq x_t^n, \;\; t = 1, ..., T \;\; y \;\; n = 1, ..., N \tag{5}$$

$$z_t^n \in \{0, 1\}, \;\; x_t^n \geq 0, \;\; I_t^n \geq 0, \;\; t = 0, ..., T \;\; y \;\; n = 1, ..., N \tag{6}$$

Siendo los parámetros:

b_t^n capacidad de producción, $b_t^n \geq 0$

c_t^n costo por unidad producida, $c_t^n > 0$

k_t^n costo de preparación de producción, $k_t^n > 0$

h_t^n costo por unidad almacenada, $h_t^n > 0$

d_t demanda, $d_t \geq 0$

Siendo las variables:

x_t^n producción, $x_t^n \geq 0$

z_t^n vale 1 en caso de incurrir en producción, sino 0

I_t^n cantidad almacenada, $I_t^n \geq 0$

Restricciónes del modelo:

- (1) Conservación del flujo general - La cantidad de producto que ingresa ya sea por inventario o por produccion del nivel anterior es igual a la cantidad que se dispone para producir o inventariar para el periodo próximo.

- (2) Demanda igual a última producción - Esta restricción es la principal, ya que satisfacer la demanda es el objetivo del problema.

- (3) Inventario inicial vacío - De no estar la restricción se satisfaría la demanda solo con costos de inventario (ya que podría existir infinito inventario en el último nivel), esta restricción no implica pérdida de generalidad [21].
- (4) Capacidad de producción acotada - Cualquiera sea el nivel y el período, la producción no puede superar la capacidad correspondiente.
- (5, 6) Se activan costos de preparación si se produce - la sumatoria de las demandas entre t y $maxT$ resulta en la mínimo valor que garantiza ser suficientemente grande para obligar a que se cumpla $z[t, n] = 1$ cuando $x[t, n]$ es mayor que cero. Z es una variable de decisión binaria que activa o no el costo de preparación (K).

Nótese que no hace falta una restricción para inventario final cero ya que la expresión a minimizar no producirá más producto que el justo necesario.
El modelo elegido (ME-CLSP) es una extensión del utilizado en 1980 por Florian et al. [7] (CLSP), quienes demostraron que la complejidad de resolución es NP-hard para modelos de un nivel de producción y capacidades de producción variables. Justamente por ser una extensión del mismo modelo pero con múltiples niveles se asegura que no se puede determinar una solución óptima en tiempo polinomial.

Para la implementación del modelo se utilizó GLPK, un software de resolución de problemas lineales. Su programación es en el lenguaje matemático MathProg y su método de resolución está basado en *Branch and Cut*, el cual busca determinar la solución óptima.

A continuación, este documento se organiza de la siguiente manera, en el Capítulo 2 se presentan las heurísticas conocidas adaptadas y diseñadas. En el Capítulo 3 se describe los parámetros de creación y clasificación de casos de Prueba (Instancias). En el Capítulo 4 se describe la solución de software para gestión y análisis de resultados. En el Capítulo 5 se presentan los resultado obtenidos. Finalmente en el Capítulo 6 se describen las conclusiones del análisis y posibles modificaciones a futuro.

2. Heurísticas

A continuación se detallan las heurísticas que fueron implementadas. Entre las mismas se encuentra un conjunto de tres heurísticas conocidas las cuales fueron adaptadas al problema definido. Estas heurísticas son:

- Lote por Lote [14]: Produce en cada período exactamente el lote necesario para satisfacer la demanda correspondiente.
- Lote de Tamaño Fijo [14]: El lote de producción es el mismo para todos los períodos dentro de un mismo nivel.
- Silver-Meal [17]:Produce en un mismo período lo necesario para satisfacer futuras demandas. Toma la decisión de cuantas demandas satisfacer considerando cuando el costo de inventario sobrepasa el costo de activar la producción en un período futuro.

También se describe la heurística propuesta y sus variantes. Esta heurística está basada en el algoritmo de Dijkstra de camino mas corto para grafos dirigidos. Para bajar los tiempos de ejecución se implementaron tres variantes de esta heurísitca:

- Triángulos: Descarta caminos con el supuesto de maximizar los costos de preparación ya activados.
- Franjas: Separa el problema en varios sub problemas para bajar los tiempos de ejecución.
- Diagonal: Aprovecha la forma general de las soluciones, descartando aquellos vértices del grafo que en lo general no participan de soluciones óptimas.

Puede consultarse el pseudocódigo correspondiente a cada una en el Anexo B.

2.1. Heurísticas Conocidas Adaptadas

Las heurísticas fueron implementadas de manera iterativa por nivel, comenzando desde el último (aquel minorista que cumple la demanda) subiendo hasta la fuente (manufacturador). Con este enfoque el problema se puede ir resolviendo nivel a nivel, debiendo definir la producción para todos los períodos de un nivel dado. Así pues cuando el minorista resuelva según las cantidades de producto que requiere para poder cumplir su producción, se pasará a considerar dichas cantidades como las nuevas demandas y será entonces que se aplicará el mismo procedimiento desde el nivel superior.

Esta sección incluye: la propuesta de cada heurística, como han sido adaptadas, referencia al pseudocódigo y un ejemplo de resolución sobre una instancia común con sus respectivos tiempos y costos resultado, dicho ejemplo corresponde a una instancia de 15 niveles, 15 períodos, con capacidades bajas y bajo decrecimiento entre niveles, costos de inventario y preparación altos.

2.1.1. Lote por Lote

Las cantidades producidas para cumplir con las demandas coincidirán nivel a nivel. La propuesta de la heurística Lote por Lote es prescindir del uso de inventario y producir exactamente lo demandado en cada período. Cada lote se transfiere a través de todos los niveles hasta llegar al cliente.

Una vez determinada la planificación de producción del último nivel, se establecerá las producciones del nivel superior al mismo (las cuales serán exactamente iguales). Estas

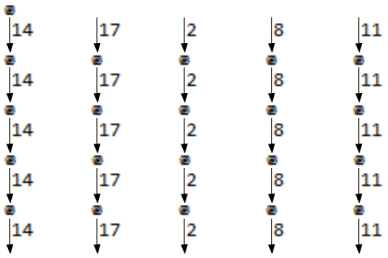

Figura 2: Grafo solución de Lote por Lote

producciones condicionaran con las demandas respectivas. Se itera por los niveles ascendiendo hasta llegar a la fuente, todos los niveles planificarán igual sus producciones. Esto se puede ver en la Figura 2 donde se muestra una solución provista por la heurística.

Esta heurística es totalmente factible con las instancias generadas para este proyecto, debido a que toda capacidad de producción es mayor a la demanda del período correspondiente, al plantear que todo problema de gestión de inventario en múltiples niveles tiene una configuración equivalente que cumple dicha propiedad.

2.1.2. Lote de Tamaño Fijo

La heurística Lote de Tamaño Fijo plantea que al momento de producir, se hará por una cantidad fija, dichas cantidades son independientes por nivel.

Como las capacidades de producción definidas en los casos de prueba son decrecientes a medida que se acercan al cliente, potencialmente los niveles superiores pueden decidir producir en menos períodos por más cantidad de producto, ahorrando en costos de preparación (los más significativos).

Por otro lado, esta implementación concentra las producciones de manera consecutiva en los primeros períodos de cada nivel, esta decisión favorece la factibilidad a riesgo de no elegir los períodos menos costosos.

La mayoría de las soluciones concentran sus producciones en los primeros períodos, esto se debe que no se aplicó capacidades de inventario (no hay límite de cantidad de producto para pasar de un período al siguiente dentro del mismo nivel) y no se utilizó backlogging, esto significa que el producto en el grafo se mueve libremente por los arcos horizontales hacia la derecha. Por dicha razón el producto puede transferirse con libertad hasta llegar en cualquier período futuro.

Aunque activar los niveles en los períodos más tempranos puede no ser una decisión óptima, igualmente se reutilizan costos de producción ya activados, puesto que la primera producción es estrictamente necesaria para cumplir la primera demanda. Si esta producción a su vez se reaprovecha para completar (por ejemplo) parte del segundo y/o tercer período, entonces será necesario también producir en alguno (o ambos) de esos períodos para terminar de satisfacer sus demandas. Este fenómeno se extiende para los subsiguientes períodos al volver a producir y se pudo constatar como tendencia en resoluciones de varios métodos.

Una consecuencia obvia de trabajar con un lote de tamaño fijo es la posibilidad de obtener inventario final mayor a cero. La demanda acumulada puede no ser divisible por la canti-

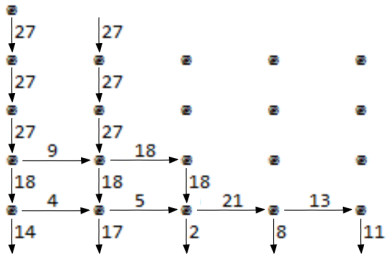

Figura 3: Grafo solución de Lote de Tamaño Fijo

dad de periodos en los que se decide producir en un nivel y consecuentemente la demanda acumulada del nivel inmediato superior puede ser aún mayor. Como la heurística resuelve iterativamente acercandose al nivel superior, en cada paso iterativo se va a tener mayores capacidades de producción y quizás más de demanda acumulada, por lo tanto es esperable que en los niveles superiores hayan menos periodos en los que se decide producir por lotes de tamaño más grandes.

De lo anterior mencionado, queda claro que al haber inventario final mayor a cero ya no se trata de una solución mínima (u óptima). Esta implementación procura reducir lo más posible el tamaño del lote para minimizar el producto sobrante. Conjuntamente se intenta reducir la cantidad de períodos en los que se produce, lo cual promueve un menor resto de una división no exacta, significando potencialmente aún menos inventario excedente.

La presente heurística es totalmente factible, puesto que si en los primeros t períodos de un nivel se decide producir, el nivel superior podrá cubrir esa nueva demanda en una cantidad de períodos menor o igual a t, dicha afirmación se basa en la característica que las capacidades de producción son mayores en los niveles superiores. Dicha característica es realista ya que las plantas productoras suelen manejar grandes volúmenes para distribuir a minoristas, para que luego estos satisfagan únicamente a su plaza local.

Las capacidades de producción por más justas que sean siempre son mayores o iguales a las demandas y por tanto, en el caso más extremo esta heurística podrá dar la misma resolución que Lote por Lote.

En el extremo opuesto, toda la demanda acumulada será producida en el primer período de cada nivel.Pero en puntos intermedios, en los cuales la holgura de las capacidades respecto a las demandas no son ni tan justas ni tan holgadas, esta implementación decidirá la cantidad de producciones y tamaño de lote validando que el producto llegue a tiempo a cada cliente, es decir la producción acumulada nunca sea menor a la demanda acumulada para todo período.

Entre las debilidades se presenta la posibilidad de un primer período con capacidad de producción muy baja (a pesar de ser este un valor aleatorio con distribución uniforme). Ésto provocaría un tamaño de lote quizas limitado por dicha capacidad extraordinaria y por consiguiente una mayor cantidad de producciones para el nivel, activando varios costos por preparación. En la Figura 3 se presenta una solución provista por la heurística.

2.1.3. Silver-Meal

Silver-Meal es una heurística que se basa en planificar la producción mediante la conveniencia de producir en un solo período para satisfacer varias demandas. Con este fin

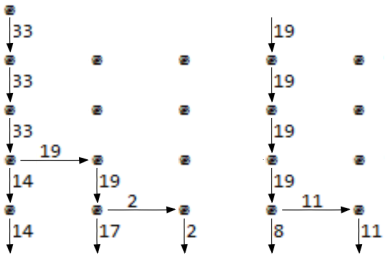

Figura 4: Grafo solución de Silver-Meal.

contrapone costos, por un lado los de producción y por otro el costo de inventario acumulado que representa dicha decisión.

Considera acumular las demandas de todos los períodos subsiguientes hasta el momento en el que incluir uno más provoca que los costos de inventario acumulado superen los costos de preparación y producción asociados. Será entonces cuando se activará en este nuevo período su costo de preparación y se empezará a acumular la producción de los períodos posteriores a él.

En comparación con las otras heurísticas utilizadas Silver-Meal es más compleja por considerar más factores para tomar sus decisiones de producción. Ésta requirió una adaptación, acumulando la producción de las demandas hasta que se alcance el límite mencionado ó bien la capacidad de producción no permita continuar acumulando. Esta pequeña adaptación la hace un poco más inteligente y adaptable al contexto de esta investigación.

El enfoque de resolución nivel a nivel comenzando desde los niveles inferiores se sigue conservando. Considerar acumular varias demandas en una producción no significa otra cosa más que acumular la materia prima requerida por el nivel previamente resuelto. Esta iteración escalará hasta el nivel fuente con el cual quedará completa la solución. En la Figura 4 se muestra un grafo solución para la heurística Silver-Meal.

Como fue mencionado en el Estado del Arte [18], se define $C(T)$ como el costo promedio de ordenar para los próximos T períodos. Numerando con uno a partir del período actual (independiente de cual sea este), queda definido:

$$C(T) = (S + \sum_{t=2}^{T-1} H.D_t)/T$$

Donde:

T Período hasta el cual se considera acumular.
S Costo de ordenar.
H Costo de mantener el inventario.
D_t Demanda del período t.

La acumulación continua hasta que se verifica $C(T) > C(T-1)$ y luego se vuelve a aplicarse el método a partir de T hasta completar los T_{max} períodos.

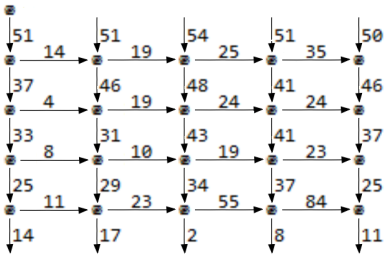

Figura 5: Grafo solución de Greedy.

2.2. Heurística Auxiliar

2.2.1. Greedy

Esta implementación del algoritmo Greedy (o también conocido por sus nombres en español algoritmo voraz), no fue incluido en este proyecto de grado para competir con las otras heurísticas, sino para dar una perspectiva de un costo próximo al máximo. En vez de minimizar, busca maximizar la siguiente expresión:

$$max \sum_{n=1}^{N} \sum_{t=1}^{T} (z_t^n k_t^n + c_t^n x_t^n + h_t^n . It^n)$$

A continuación se describe que decisiones de producción se toman en cada paso local con la esperanza de llegar a una solución general óptima.

Por medio de esta heurística auxiliar se pudo dimensionar la distancia entre el resultado "exacto" (GLPK) y cualquier otra heurística. Para ser más claro, en un ejemplo dónde el algoritmo de GLPK arroja un resultado de $100.000 por costo total óptimo y una heurística cualquiera se aproxima a dicho valor con un resultado de $120.000, no supone la misma calidad de resultado un ahorro de $200 si el costo máximo posible ronda los $250.000 o si ronda los $10.000.000.

La distancia entre los resultados del algoritmo de GLPK y Greedy representa una escala precisa del dominio de soluciones sobre la que se está trabajando instancia a instancia. En los ejemplos anteriores, el resultado de la misma heurística supone exactitudes muy diferentes según el costo total de Greedy, estos valores son 13,3 % y 0,2 % respectivamente.

La implementación resuelve activar las producciones a su capacidad (o al menos a lo que le permita la materia prima disponible facilitada por el nivel superior) para cada tupla $< periodo, nivel >$, esto significa que los costos de preparación y producción están maximizados. En cuanto al manejo de inventario, queda totalmente determinado puesto que como se produce al máximo, se está priorizando mover el producto hacia abajo en el grafo y no hacia la derecha. Pero como las capacidades de producción de las instancias son decrecientes cuanto más inferior sea el nivel, los niveles superiores acumularán saldo de producto período a período. Por esta razón se obtiene un inventario remanente enorme y con mayor concentración en los niveles superiores. Esto se puede ver en el grafo solución de la Figura 5.

Producir a capacidad provoca que la cantidad de inventario quede determinada como la resta entre la capacidad del nivel actual menos la capacidad del nivel inmediato inferior. Se elaboró el gráfico de la Figura 6 para evaluar si es posible en cierto momento producir menos y almacenar más inventario a fin de incrementar los costos totales. Notar que los

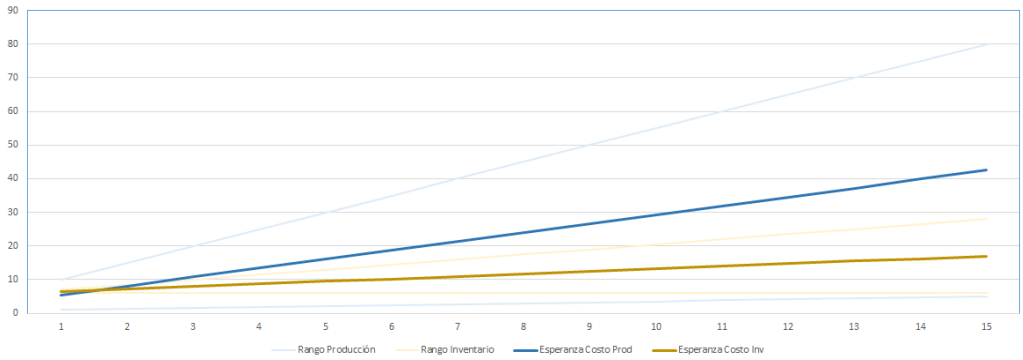

Figura 6: Comparativa de costos de inventario y producción por unidad en instancias con los más altos costos de inventario y 15 niveles.

costos de inventario son generalmente inferiores a los de producción, basados en el modelo de costos de Melo y Wolsey utilizado en el 2012 [13] y que el crecimiento lineal que sucede con los costos de producción nivel a nivel viene dado por una pendiente mayor al de los costos de inventario, por tanto los costos de inventario superan a los de producción únicamente en los primeros niveles (los menos significativos en el total). Se puede asegurar que la política de producir a capacidad practicamente maximiza el costo total.

Como se puede observar en la Figura 6, en el caso más desfavorable únicamente en el primer nivel es más probable los costos de inventario superen a los de producción.

Otra utilidad aprovechada por esta implementación de Greedy es comprobar factibilidad de las instancias. Si una instancia es factible Greedy devolverá un resultado, o lo que es equivalente, si una heurística cualquiera la puede resolver, Greedy también. Esta propiedad se deba a que la factibilidad depende únicamente de la capacidad de producción y los niveles de demanda, como fue mencionado en el Capítulo 3.

Considerar entonces que toda solución tiene un nivel de producción menor o igual a la capacidad de producción, entonces Greedy podrá cumplir en lo que refiere a cantidades. Respecto a hacer llegar dichas cantidades a tiempo para cada demanda, se demostró anteriormente que la política de mover hacia abajo el producto lo antes posible es la manera de promover que el producto llegue a cubrir la demanda, puesto que el movimiento horizontal del producto es libre por los arcos del grafo.

2.3. Heurísticas Propuestas

Como parte de los objetivos definidos para el proyecto, se implementó una heurística propia la cual provee una solución factible, tratando de minimizar la desviación al mejor resultado encontrado por el algoritmo GLPK, acercándose a la optimalidad en un tiempo de ejecución razonablemente bajo.

En principio se elaboró una única heurística y luego surgieron varias variantes de la misma. Aunque todas comparten la misma base proveen soluciones alternativas diferentes a una misma problemática. Estas modificaciones se hicieron con el fin de obtener tiempos de ejecución menores, tratando de no degradar la solución, o en caso de hacerlo que tenga el menor impacto posible.

Cabe destacar que las heurísticas variantes que se describen en las Secciones subsiguientes,

son una selección de un conjunto de opciones que fueron evaluadas durante el transcurso del proyecto. También se describirán algunas de las variantes descartadas, por no representar una mejora en la solución.

2.3.1. Heurística Base

Dada la forma del problema como un grafo dirigido, soluciones al mismo se pueden obtener por medio de algoritmos que aborden el *Problema del Camino más Corto* [1] ya conocidos en teoría de grafos, considerando la distancia entre vértices como el costo que se incurre al producir por esa arista.

Esta heurística como todas sus variantes se basan en el algoritmo de Dijkstra para encontrar el camino más corto desde los vértices "demanda" hacia el vértice "origen". Este algoritmo además de proveer el camino más corto, lo hace en orden cuadrático y por lo tanto no afecta los tiempos de ejecución en gran medida.

A continuación se describe la heurísitica, sus principales componentes, y su comportamiento paso a paso.

Representación

Se representa la estructura del problema como un grafo dirigido donde el vértice productor se encuentra como origen. Las aristas conectan vértices de un nivel al inferior de su mismo período, y de un período al siguiente del mismo nivel.

Cada arista contiene los atributos necesarios para representar el problema, a saber: costo unitario (de producción o inventario), costo de preparación, capacidad y cantidad producida. Los costos se representan con números Reales, mientras que las cantidades y capacidades con enteros.

Descripción general

Debajo se explica paso a paso el funcionamiento de la heurística y como se utiliza el algoritmo de Dijkstra. Siendo que el problema se analiza/resuelve como un grafo, con cada vértice representando un nivel de producción en un determinado período de tiempo, el termino "vértice superior", implica el nivel superior de producción en el mismo período, y el termino "hacia la izquierda" implica el mismo nivel de producción en el período anterior (cuando se utiliza el inventario).

a. **Paso 1**

   Al comienzo se genera la estructura de grafo cargando los datos del problema, los cuales se obtienen de cada uno de los archivos de instancias (Estos se explicarán en el Capitulo 3).

b. **Paso 2**

   La heurística toma de a uno los períodos, y calcula los caminos más cortos hacia el origen. Itera comenzando por el nivel de demanda de cada uno de ellos hasta llegar al último período. En estas iteraciones, se va a calcular el Dijkstra desde el origen hasta todos los vértices del grafo que pertenezcan a un período menor o igual al período en el que se está iterando (ya que los períodos futuros no podrán ser considerados para formar parte del mejor camino).

Esto va a permitir tener el camino más corto desde cada uno de los vértices hacia el vértice inicial, lo que ayudará a elegir el mejor momento para producir dado un determinado juego de datos. Es importante destacar que la selección del costo de las aristas del grafo va a determinar el camino más corto. Esto quiere decir que la heurística implementada puede variar en sus resultados dependiendo de lo que se defina como "costo arista" (costo fijo, costo producción, capacidad, combinación de los mismos, etcétera). Este detalle permite generar varios modelos distintos dentro del mismo algoritmo e incluso costos de aristas que varíen dinámicamente según vaya avanzando la ejecución del mismo, pudiendo calcularse el costo arista en función del nivel y/o período.

c. **Paso 3**

Tomado el primer período, se calcula el Dijkstra desde el origen hacia el vértice demanda. El algoritmo empieza desde abajo hacia arriba, tomando la demanda del primer período y tratando de satisfacerla con su vértice más cercano (es decir menos costoso).

Básicamente cada vértice del primer período le va a "solicitar" a su vértice adyacente la demanda que tiene que satisfacer. Una vez satisfecha, se asigna a la arista la producción realizada y se calcula el costo asociado. Cabe destacar que para el primer período la producción siempre se hará desde el nivel superior, ya que no existe posibilidad de inventario.

d. **Paso 4**

Ya completada la primer demanda, se pasa a la segunda iteración, calculándose el Dijkstra nuevamente, pero esta vez considerando el primer y segundo período. La diferencia con el caso anterior es que ahora el inventario si es relevante. Es decir el camino más corto para un vértice puede ser hacia su izquierda o hacia arriba.

Dado que cada arista tiene asignada una capacidad de producción, la heurística antes de comenzar a resolver el camino, calcula la holguras correspondientes. Tomando la capacidad del camino como la capacidad mínima de todas las aristas que lo conforman. Si esta capacidad es menor que la demanda, se separa en dos (o incluso más), y se repite el Paso 3 únicamente para la demanda que se sabe que puede producirse por ese camino.

Una vez terminada esta parte se toma la demanda restante, y se hace el mismo proceso. Se calcula el Dijkstra para ese saldo y se establece la capacidad del camino. Nuevamente se llama al proceso con la demanda restante por este nuevo camino más corto. Este paso se hace cuantas veces sea necesario para satisfacer la demanda total del período en cuestión.

Cuando en una arista se produce a capacidad, la arista se elimina del grafo. De esta forma en la próxima ejecución del Dijkstra la misma no será considerada reduciendo los tiempos de ejecución del algoritmo, y descartando caminos no factibles, mecanismo que habitualmente se denomina poda.

e. **Paso 5 al N-1**

El paso anterior es ejecutado tantas veces como períodos tenga el problema. El algoritmo de Dijkstra se llamara tantas veces por período, como divisiones en la demanda existan. Este paso, tuvo varias modificaciones desde el comienzo del proyecto, las cuales se explicaran en Secciones subsiguientes.

f. **Paso Final**

Como paso último se persiste la solución encontrada con una representación sencilla del grafo, especificando la producción e inventariado. También muestra el costo mínimo encontrado y el tiempo de ejecución necesario para llegar a la solución.

Detalles de implementación

Debajo se muestran algunos detalles particulares de la implementación de la heurística.

• Demanda
La demanda es expresada como un vértice más del grafo, el atributo capacidad de la última arista del período refiere a la demanda que se tiene que satisfacer en ese período.

• Numeración de vértices
Aunque los vértices son perfectamente identificables por su nivel y período, llegado el momento de calcular las demoras del algoritmo Dijkstra se utilizan identificadores numéricos, ya que se pudo comprobar que la ejecución es más performante.

• Costo Inventario
El costo de inventario y costo unitario de producción esta modelado por el mismo atributo de arista. Toma un valor u otro dependiendo si es una arista horizontal o vertical.

• Varios criterios de costos
Es posible la ejecución en cadena de la misma heurística considerando distintos costos de arista, mostrando los resultados y resaltando el mejor criterio para ponderar las aristas.

• Filtrado de vértices
El algoritmo de Dijkstra calcula únicamente las distancias del origen hacia todos los vértices pertenecientes a los períodos/niveles menores o iguales al período/nivel en el que se está posicionado. Esto mejora el desempeño del algoritmo, sobre todo cuando la instancia a resolver es muy grande.

• Criterios de selección de costo
El algoritmo de Dijkstra selecciona el camino más corto dependiendo del costo que se le asigne a cada arista. Como se mencionó anteriormente el criterio con el que se selecciona este costo cambia completamente el comportamiento de la heurística y por tanto el resultado. Luego de evaluar muchos criterios se seleccionó el siguiente por considerarse óptimo para la heurística implementada:

$$min^* = min(\frac{capacidad}{demandaPromedio}, periodosRestantes)$$

$$costoArista = \frac{costoFijo}{min^*} + (costoUnitario \times demandaReal)$$

Aquí se calcula un ratio entre la capacidad de la arista y el costo de preparación tomando en cuenta la demanda promedio. De esta forma, se priorizará a aquellas aristas donde la capacidad de producción sea alta en comparación con los costos de producción. Esto se explicará en mayor detalle en Secciones siguientes, junto con otros criterios que fueron descartados.

Modificaciones posibles

Se puede observar como el resultado del algoritmo varía mucho según la selección de costo de arista para el cálculo del camino más corto. En general como el Dijkstra se ejecuta

varias veces durante el transcurso del programa, el costo de las aristas puede variar de una ejecución a otra. Haciendo uso de esta propiedad se pueden tomar los criterios más diversos para generar la solución.

Definitivamente algunos criterios funcionan mejor que otros dependiendo el tipo de grafo con el cual se cuente, atributos como lo ajustadas de las capacidades de producción o la diferencia de los costos de producción unitaria respecto a los costos de preparación son ejemplos que pueden incidir.

Un posible trabajo a futuro, es la implementación de un algoritmo que analice el problema según algunos criterios previamente definidos y seleccione una u otro costo de arista según convenga. Esto haría que la heurística se adapte al contexto del problema.

2.3.2. Variante Triángulos

Una vez implementada la heurística Base surgieron varias variantes posibles a la misma con el fin de hacerla más performante obteniendo resultados semejantes. En muchos casos estas modificaciones lograron el objetivo planteado, aunque en alguno de los casos deteriorando la calidad de la solución. Las variantes que se describen en esta y en las próximas Secciones también se pueden combinar generando nuevas heurísticas.

Particularmente la variante de Triángulos toma la heurística descrita anteriormente como base. Esta heurística fomenta la utilización de caminos ya recorridos. De esta forma, se aprovechan los costos fijos de preparación, descartando otros caminos posibles.

Para lograr este objetivo la heurística construye el camino desde el vértice demanda hacia el vértice productor tomando en cuenta dos puntos:

1) Se puede tomar en todo momento el vértice inmediatamente superior (o sea de igual período) ó
2) El vértice hacia la izquierda (mismo nivel del período anterior), únicamente, si su nivel es menor que el último vértice utilizado del período anterior.

De esta manera las soluciones alcanzadas por esta heurística forman “triángulos” en el intento de maximizar el uso de vértices ya utilizados como se muestra en la Figura 7. La generación de estos triángulos tiene el efecto asociando de eliminar una gran cantidad de caminos. Siendo que las opciones a la hora de seleccionar un camino son mucho más limitadas, esta variante reduce los tiempos de cálculo de forma significativa.

Lamentablemente ya que esta reducción en los tiempos de ejecución se debe a la poda de soluciones posibles, la heurística provee un resultado algunas veces muy lejano al óptimo. Por consiguiente este procedimiento heurístico es únicamente recomendable para aquellos casos muy complejos, en los que se requiere un resultado en tiempo acotado.

2.3.3. Variante Franjas

Al igual que la modificación de “Triángulos” la Solución de “Franjas” busca mejorar los tiempos de ejecución en detrimento de la exactitud de la solución.

La heurística de Franjas tiene una estrategia de *Dividir y Conquistar* [15] donde se particiona el problema en varios sub-problemas. Para lograr esto, la heurística separa la instancia en conjuntos consecutivos de períodos de igual tamaño (formando una solución

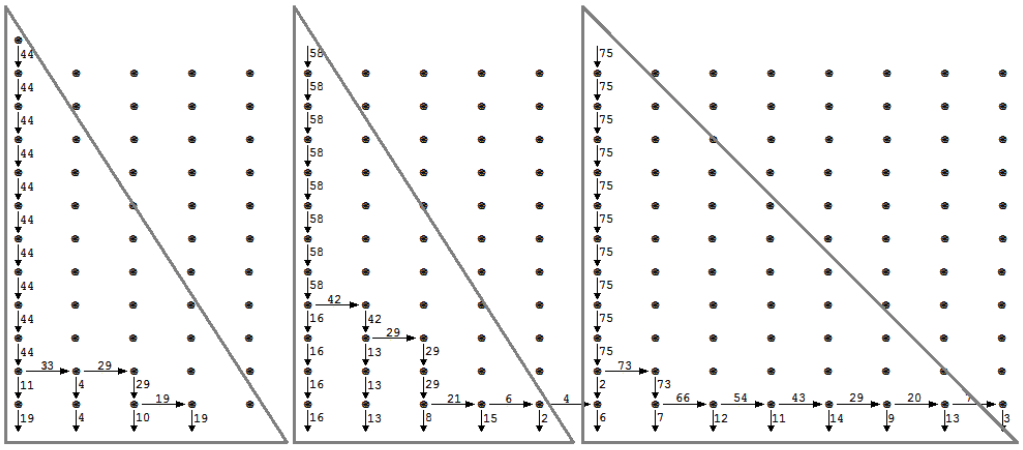

Figura 7: Variante Triángulos: Abastecimiento de demandas con forma triangular

en forma de franjas verticales como se muestra en la Figura 8). Luego para cada uno de estos conjuntos de períodos se ejecuta la heurística Base.

Este comportamiento obliga a la heurística a actuar en cada comienzo de franja como si fuese el primer período. Es decir la producción del primer período de cada franja, únicamente se podrá hacer desde el nivel superior, sin poder tomar producción del inventario del período anterior (es decir de la franja anterior).

Esta disección de la problemática en Franjas, tiene como consecuencia inmediata la reducción de los vértices que el algoritmo de Dijkstra tiene que considerar a la hora de calcular los caminos más cortos al origen, obviamente repercutiendo de forma directa en el tiempo de ejecución principalmente para aquellas instancias con gran cantidad de períodos.

Para poder calibrar la heurística, balanceando la exactitud de la solución con el tiempo de ejecución, se le puede cambiar la cantidad de períodos que tiene por ancho las franjas. Cuanto más grandes sean las franjas, más exacta será la solución y cuantos menos períodos contenga más performante será la heurística.

Para el estudio hecho dentro del alcance de este proyecto se tomaron franjas fijas de 10 períodos. Esto implica, que para aquellas instancias con menos de 10 períodos, el algoritmo se comportará simplemente como la heurística Base. Esto se debe a que con instancias pequeñas, se querrá siempre minimizar la desviación de la solución, antes que el tiempo siendo que en estos casos las ejecuciones se hacen en el orden de mili-segundos y por tanto no representan un problema.

Esta variante a la heurística Base, se puede considerar como una solución intermedia, siendo que no provee una solución tan rápidamente como la variante de “Triángulos” pero genera un resultado más próximo a la heurística Base.

El beneficio detrás de las franjas radica en forzar periódicamente las preparaciones creando caminos posibles para futuros períodos. Los cálculos para estos períodos se simplifican por disponer de una gran holgura en la capacidad de producción.

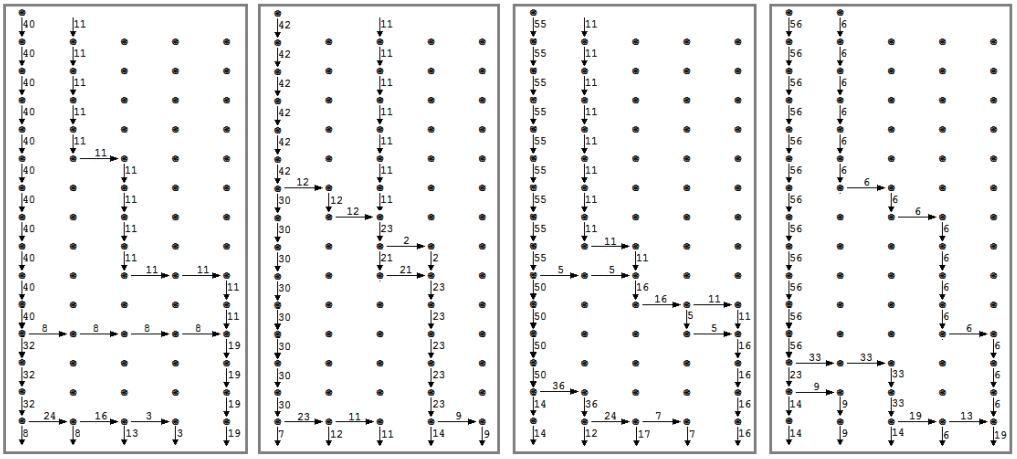

Figura 8: Variante Franjas: La solución se separa en franjas (de 5 períodos en este caso)

2.3.4. Variante Diagonal

Una vez comenzado el análisis de los resultados de las diferentes heurísticas, se identificó una forma muy particular en la que se presentaban las soluciones tanto por el algoritmo de GLPK, como por la heurística Base implementada. Siendo que ambas intentan maximizar la reutilización de los vértices en los cuales ya se produjo, la producción de toda la demanda se realiza en general en los primeros períodos del problema. Mostrando un comportamiento especulativo con respecto a la acumulación de inventario.

En particular para el caso de la heurística Base implementada, esto implica que para aquellas instancias con gran cantidad de períodos y niveles, existe una gran porción de vértices que no serán considerados a la hora de satisfacer la demanda de su mismo período. Esto se debe a que para satisfacer la demanda de los últimos períodos, la producción se genera en los períodos anteriores.

Gracias a éste comportamiento el cual parece general para la mayoría de las instancias y con el fin de mejorar los tiempos de ejecución, se generó una última modificación a la primera heurística en la cual se descartan una gran cantidad de vértices del grafo. En particular al comienzo de la ejecución, y antes de hacer ningún cálculo, se eliminan los vértices pertenecientes a los primeros niveles de los últimos períodos del grafo.

Al hacer ésto se limitan los cálculos necesarios para los caminos más cortos, ya que a la hora de ejecutar el algoritmo de Dijkstra, la cantidad de vértices se ve cuantiosamente reducida, y por lo tanto el algoritmo ejecuta en menor tiempo. Obviamente la diferencia en el tiempo de ejecución se hace evidente en aquellos problemas donde la cantidad de vértices eliminados es mayor.

Esta nueva heurística reduce el tiempo de ejecución de la Heurística Base, pero a diferencia de las dos anteriores, no impacta la calidad de la solución. Esto quiere decir que se puede obtener exactamente el mismo resultado en un tiempo menor.

De todas formas hay que tener en cuenta lo siguiente. Para aquellos grafos cuadrados (misma cantidad de niveles y períodos) es fácil identificar una gran porción de vértices a ser removidos. Sin embargo para los que la proporción entre Niveles y períodos es dispar, se debe tener más cuidado en la remoción ya que la misma puede afectar la factibilidad de la solución, haciendo que la heurística falle a la hora de resolver el problema.

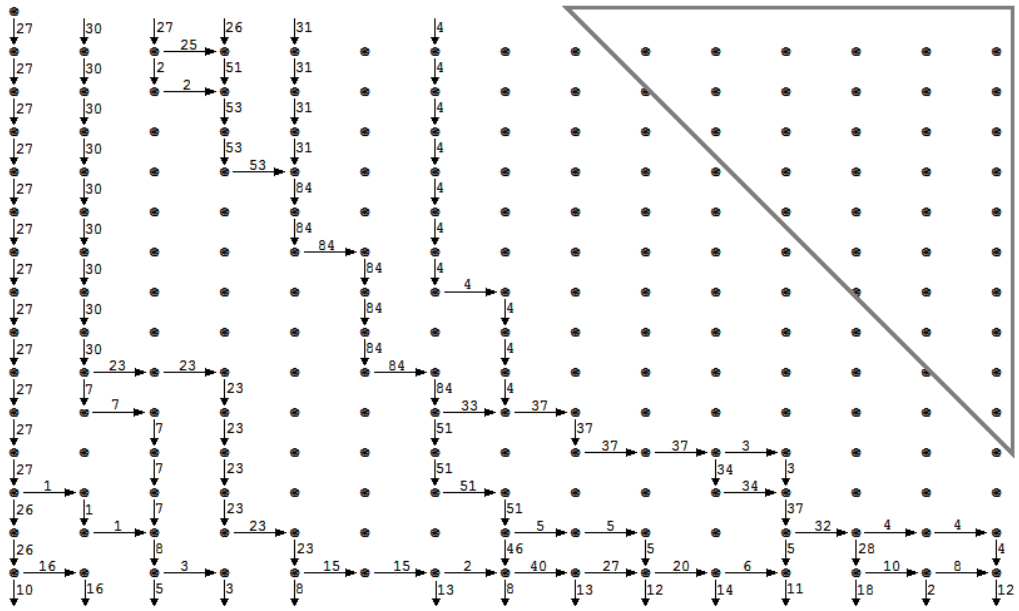

Figura 9: Se remueve el triángulo superior derecho.

Por lo tanto existe un problema inherente con esta heurística que radica en la elección de los vértices a ser quitados del grafo. Si se sigue un acercamiento conservador, y no se quitan muchos períodos, la diferencia en el tiempo de de ejecución no será de gran valor, pero por otro lado si se arriesga quitando varios vértices, el algoritmo de correrá considerablemente más rápido pero poniendo en riesgo la factibilidad de la solución.

Para el análisis de este proyecto se eliminan los vértices de un triangulo formado por los últimos períodos y primeros niveles. Para evitar el problema mencionado anteriormente, este triangulo se ajusta a partir de la cantidad de niveles y períodos que tenga la instancia. En la Figura 9 se muestra una representación del grafo solución, en el cual se marca con un un triangulo blanco, el angulo superior izquierdo en el cual existe producción alguna.

2.3.5. Evolución de la Heurística

La heurística que se usó como base sufrió un gran número de variaciones con el fin de hacerla más performante y mejorar la calidad del resultado. Debajo se listan algunos de los cambios que se realizaron en el correr del proyecto, y de las pruebas que se hicieron con un gran numero de instancias.

Algoritmo recursivo:

La heurística fue diseñada originalmente para ser un algoritmo recursivo. Una vez calculado el Dijkstra para un período, se llamaba a un método recursivo el cual se invocaba nivel a nivel hasta completar el camino desde la demanda hacia el origen. De esta forma cada vértice resolvía el problema de forma independiente.

Si la capacidad de la arista no era suficiente para la demanda, enviaba lo que la holgura permitiera por el camino más corto, y el resto se solicitaba por el otro vértice adyacente. Estos llamados recursivos conformaban una estructura arbolaría.

En el caso de que el camino más corto no fuera directamente desde el nivel superior, o si el nivel superior no tuviese capacidad suficiente para producir la demanda requerida en ese período, el método se llama a si mismo dos veces, una hacia la izquierda (pidiéndole inventario al período anterior) y otra hacia arriba (produciendo en el mismo período). Esta división de producción la realizaba tomando en cuenta las capacidades de los dos caminos hacia el origen. Siempre haciendo el pedido de inventario hacia la izquierda primero y luego de producción hacia arriba para evitar conflictos.

De esta forma la solución se iba dando paso a paso, tomando por el camino de menor costo siempre y cuando el mismo estaba limitado por la capacidad de producción, entonces se incurría en un nueva ejecución del algoritmo Dijstra, esta vez con menor cantidad de vértices. A diferencia de la heurística final, la cual resuelve los caminos más cortos siempre desde el origen hasta el destino, tantas veces como tenga que fraccionar la demanda según la capacidades de los caminos que va calculando.
Esta versión recursiva era un poco más rápida que la actual, ya que los Dijkstra intermedios se hacían considerando el período y nivel en el que la capacidad no era suficiente, aunque resultaba en descartes de mejores caminos posibles. Por lo contrario la versión iterativa tiene que calcular el Dijkstra siempre desde el inicio, haciendo el cálculo más costoso en tiempo.

La misma fue descartada, ya que generaba una desviación mucho mayor que la versión iterativa final en comparación con el algoritmo de GLPK. Esto obviamente se da por descartarse una gran cantidad de caminos.

Grafo completo:

Siendo que el costo de las aristas es dinámico, y va cambiando a medida que se vayan estableciendo las producciones, la primera versión del algoritmo trataba de hacer el cálculo de los caminos más cortos cuantas veces fuera posible. Esto permitía tener las distancias al origen actualizadas todo el tiempo.

De esta forma, el cálculo del Dijkstra se ejecutaba al comienzo de cada período, así como cada vez que una arista era borrada (por haberse producido a capacidad). En esta primer versión de la heurística el algoritmo de Dijkstra se ejecutaba de forma completa, es decir desde el vértice origen hacia el resto de los vértices del grafo.

Grafo parcial:

La primera versión de la heurística, aunque proveía resultados muy cercanos a la solución de GLPK, demostró ser muy costosa en tiempo a la hora de aplicarla a instancias de gran tamaño (ej.: 50x50).

Para poder mejorar los tiempos de ejecución, se comenzó a calcular el algoritmo de Dijkstra únicamente considerando el período en el que se está calculando y los períodos anteriores. Esto redujo considerablemente el tiempo de ejecución del algoritmo, sin afectar el resultado de la heurística. De todas formas los Dijkstra se seguían ejecutando cada vez que una arista era borrada del grafo.

Punto específico:

Con el correr de las pruebas, se vio la necesidad de acortar los tiempos de ejecución aun más. Por eso se analizó el impacto de ejecutar el algoritmo de Dijkstra únicamente para

aquellos vértices en los que el camino que se está calculando puede ser afectado.

De esta forma, conforme se va avanzando en la creación del camino si una arista es borrada, el cálculo del camino más corto para ese vértice se buscará únicamente entre los vértices del mismo nivel o superior e igual período o inferior. Esto permite que el grafo con el que Dijkstra tenga que trabajar esté acotado por nivel y período, haciendo que cada paso recursivo para el cálculo de un camino sea cada vez menos costoso.

Cálculo en bifurcación:

La última versión del algoritmo recursivo, utilizaba el algoritmo de Dijkstra únicamente cuando volvía de una bifurcación de caminos. Como se explicó antes, en el momento en el que la capacidad de un camino no permite pasar todo el producto demandado, la heurística dividía la producción en dos, pasándole el resto de la producción al otro camino.

Este tipo de bifurcaciones se pueden dar varias veces en el correr de la ejecución para satisfacer la demanda de un período. Esto quiere decir que en el cálculo de un camino, se pueden borrar varias aristas del grafo y por tanto calcularse un gran número de veces el algoritmo Dijkstra.

Para solucionar el problema, siendo que los caminos se calculaban de derecha a izquierda y de abajo a arriba, todos los algoritmos Dijkstra intermedios fueron eliminados, ya que se demostró eran innecesarios, y de esta forma se logró bajar los tiempos de ejecución de forma importante.

Como ejemplo: para instancias de 50 niveles y 50 períodos que rondaban el orden de los 700 segundos de ejecución (más de 10 minutos), con estos cambios se lograron ejecuciones de 50 segundos (menos de un minuto).

2.4. Variantes Descartadas

Dentro del alcance de este proyecto, y en la búsqueda de una heurística que minimice la desviación, garantizando tiempos bajos de ejecución, se probaron distintas variantes que fueron al final descartadas. Estas heurísticas variantes, no siempre se descartaron por proveer un mal resultado, sino muchas veces, por comportarse de forma impredecible. En muchos casos las mismas proveyeron mejores resultados que la heurística Base para algunas instancias, no siendo este resultado constante para el resto de las categorías.

2.4.1. Franjas no Arbitrarias

Observando los resultados parciales, se vio que la variante “franjas” proveía en muchos casos, un resultado más acertado que la heurística Base. Considerando que en un principio las franjas se tomaron de forma arbitraria (de a 10 períodos) se investigó la posibilidad de tomar estas franjas tomando un criterio que beneficiara al algoritmo.

En particular se hicieron tres modificaciones:

a. Se calcularon los costos fijos sumados de todo un período de producción. Es decir todos los costos fijos de producir en un mismo período pasando por todos los niveles de ese mismo período. Luego, se ordenaron los períodos, según este criterio y se generaron las franjas, tomando como primer período de cada franja, uno de los 5 en los cuales este costo fuese mínimo.

Siendo que al principio de cada franja, inevitablemente se produce en todos los niveles del primer período (activando todos los costos fijos), de esta forma se trata de que los costos de esta producción sean mínimos.

b. Se hizo un procedimiento similar al anterior pero esta vez considerando las capacidades. En este caso se ordenaron los períodos considerando el máximo de la suma de las capacidades de cada período. De esta forma se establecieron los principios de las franjas como los períodos con mayor capacidad, ayudando a que exista la mayor cantidad de holgura en esos períodos en los que la producción se activa en todos sus niveles.

c. Se definió un ratio para comparar el costo total de producir según el resultado del algoritmo Dijkstra, en comparación con producir completamente en el mismo período de producción. Este ratio se estableció en 40 %. Es decir las franjas se creaban cuando producir en un período (empezando desde el nivel inferior) era únicamente 40 % menos costoso que producir todo en ese período. De esa forma cuando esta condición se cumplía se empezaba una franja en ese período.

Aunque en algunos casos puntuales estas modificaciones dieron muy buenos resultados, las mismas fueron descartadas, ya que no mostraron una mejora real teniendo en cuenta el total de las instancias ejecutadas.

2.4.2. Costos de Aristas

Dada la naturaleza configurable de la heurística Base, se definieron algunos criterios particulares teniendo en cuenta parámetros adicionales a la hora de determinar el costo de una arista para el algoritmo Dijkstra.

En particular se destacan estos criterios.

a. El primero y más obvio fue el de simplemente tomar el costo de arista como el:

$$costoArista = costoFijo + costoUnitario \times demandaReal$$

Éste criterio aunque parece ser el indicado al principio, luego de varias ejecuciones se encontró, que al no considerar las capacidades los resultados se alejaban mucho del óptimo.

b. Otra alternativa fue tomar la demanda promedio de todos los períodos, en vez de la demanda real. La idea atrás de esto, es que se calcula el algoritmo Dijkstra con una cantidad fija en todo momento. Esto era útil para el algoritmo recursivo, el cual no se sabia a priori cual seria la demanda.

$$costoArista = costoFijo + costoUnitario \times demandaPromedio$$

c. Para aprovechar la característica de las instancias utilizadas, donde las capacidades son decrecientes para los niveles de un mismo período; se le puede dar prioridad a las aristas horizontales en los niveles más cercanos al origen y a las verticales en los niveles más cercanos a la demanda. De esta forma se fuerza a colmar las capacidades de los primeros períodos.
Si es horizontal

$$costoArista = (costoFijo + costoUnitario \times demandaReal) \times nivel$$

Si es vertical

$$costoArista = \frac{(costoFijo + costoUnitario \times demandaReal)}{nivel}$$

d. Los costos asociados a aristas pertenecientes a períodos anteriores, se incrementan artificialmente beneficiando a las aristas mas cercanas a donde se produce la demanda. Para lograr esto, se divide el costo de producción de la arista por el periodo al cual pertenece la arista. En cada nuevo periodo el costo de producir en un nivel inicial sera mas costoso.

$$costoArista = \frac{(costoFijo + costoUnitario \times demandaReal)}{periodo}$$

e. Utilizar la capacidad de la arista en comparación con los costos de preparación. Esto alternativa fue la que al final llevó a la selección última de costo arista que se mencionó En la sección anterior. En este caso lo que se tomaba era únicamente.

$$costoArista = \frac{(costoFijo + costoUnitario \times demandaReal) \times costoFijo}{capacidad}$$

f. Luego de evaluar varias estrategias se entendió que era necesario involucrar a la capacidad de la arista en el cálculo. Fue por eso que antes de sumar el costo de preparación, se lo divide por la capacidad dividida la demanda promedio. De esta forma se beneficia aquellas aristas que aunque tengan costos de preparación altos, puedan producir para muchos períodos. Es decir de alguna forma en estas aristas el costo de preparación se podrá dividir entre más períodos, y por lo tanto será menor en el largo plazo.

$$costoArista = \frac{demandaPromedio \times costoFijo}{capacidad + costoUnitario \times demandaReal}$$

Al final, luego de muchas ejecuciones se siguió con la última estrategia pero con una modificación. En vez de hacer la división por $\frac{capacidad}{demandaPromedio}$, se toma el mínimo entre ese valor y la cantidad de períodos restantes. Esto evita beneficiar grandes capacidades en los últimos períodos, ya que es probable que no se colmen esas capacidades, pues puede quedar pocos períodos de demanda.

$$min^* = min(\frac{capacidad}{demandaPromedio}, periodosRestantes)$$

$$costoArista = \frac{costoFijo}{min^*} + costoUnitario \times demandaReal$$

Esta forma de evaluar el costo de la arista no solo proveyó el resultado más acertado para la gran mayoría de las instancias, sino que también redujo los tiempos de ejecución.

Esto último se debe a que beneficiar a aquellas aristas con mayor capacidad, permite producir una gran cantidad de producto en los caminos elegidos y por lo tanto se efectúan menos bifurcaciones, reduciendo así la cantidad de veces que el algoritmo de Dijkstra es invocado.

3. Casos de Prueba

Se generó un amplio conjunto de instancias con el fin de probar las heurísticas implementadas. Para asegurarse de abarcar una gran cantidad de escenarios, éstas se clasificaron en varios grupos, considerando 486 categorías con 5 instancias cada una, totalizando 2430 casos de prueba. Esta clasificación se realizó tomando en cuenta características que resultaron de interés para la investigación. En particular se consideraron cantidades distintas de niveles y períodos, diferentes niveles de holgura de la capacidad de producción y finalmente distintas distribuciones de costos de inventario y preparación. Para establecer esta distribución y los rangos entre los que los costos fluctúan, se tomó como referencia el trabajo de Melo y Wolsey en su investigación del 2012 [13], donde se definieron una distribución de costos uniforme la cual se extrapoló a N niveles.

3.1. Generación y Clasificación

Estos conjuntos de instancias son generadas de forma aleatoria dentro de los parámetros que específica cada categoría. Una categoría esta determinada por los siguientes factores: cantidad de niveles, cantidad de períodos, capacidades de producción respecto a las demandas, decrecimiento de las capacidades de producción del primer al último nivel, costos de inventario y costos de preparación. Para cada uno de estos factores se definieron tres rangos, bajo, medio y alto. Las categorías se definieron realizando el producto cartesiano de cada rango con todos los factores resultando en 486 posibles categorías (ya que se descartaron todas aquellas cuya cantidad de niveles superan a la cantidad de períodos por no reflejar la realidad). En el Cuadro 1 se muestran los rangos en los que puede variar cada uno de los factores.

	Bajo		Medio		Alto	
	mín	máx	mín	máx	mín	máx
Cantidad de Niveles	5		15		50	
Cantidad de Períodos	5		15		50	
Cap. Prod. según Demanda	100 %	120 %	121 %	160 %	161 %	400 %
Multiplicador Cap. Prod.	1	2	3	5	6	10
Costos de Inventario	1	3	4	5	6	7
Costos de Preparación	80	200	201	400	401	700

Cuadro 1: Rangos de los valores que puede tomar cada uno de los factores que determinan una instancia.

- Las cantidades de niveles y períodos son tres valores fijos posibles como expresa el Cuadro 1. Las combinaciones presentes en este proyecto de grado expresadas como <Niveles, Periodos> son: <5,5>, <5,15>, <5,50>, <15,15>, <15,50>, <50,50>.
- El rango medio (por ejemplo) de la capacidad de producción según la demanda indica que como mínimo la capacidad de producción para el último nivel vale un 121 % y como máximo un 160 % respecto al valor de la demanda máxima.
- El valor multiplicador de capacidad de producción representa que la capacidad del primer nivel es tantas veces mayor a la del último nivel.
- El costo de inventario y preparación esta dado por un número aleatorio con distribución uniforme entre el mínimo y máximo del rango correspondiente.

	Único	
	mín	máx
Costos de Producción	1	10
Multiplicador Costos Prod.	5	8
Multiplicador Costos Inv.	1	4
Multiplicador Costos Prep.	1	1
Demandas	0	20

Cuadro 2: Parámetros comunes a todas las instancias.

Otros parámetros de creación de instancias son comunes entre las categorías y son modelados por variables aleatorias de distribución uniforme. Dichos parámetros son: costos de producción el crecimiento de los costos de producción, inventario y preparación del primer al último nivel y las demandas. El rango de los valores de estos parámetros se ilustran en el Cuadro 2.

- Nótese que los costos de producción son generalmente más altos que los de inventario y crecen de forma lineal con mayor pendiente, respecto a los costos de preparación superan ampliamente a los de producción.
- La cota inferior del rango de costos de producción se multiplica por 5 del primer al último nivel y la cota superior por 8, basados en el modelo de costos de Melo y Wolsey en su investigación del 2012 [13].
- La cota inferior del rango de costos de inventario se mantiene constante del primer al último nivel y la cota superior se multiplica por 4, basados en el modelo de costos de Melo y Wolsey.
- El rango de los costos de preparación se mantienen constantes en todos los niveles de la instancia.
- Las demandas pueden ser como mínimo 0, dejando la posibilidad que hayan períodos sin necesidad de producir y como máximo 20. Este máximo determina la escala de cantidades de producto que se transfiere a través del grafo. Las capacidades de producción se ajustarán según la demanda, por ser este el único parámetro que se ve afectado por la misma. Todo problema de dimensionamiento de lote tiene su equivalente con una formulación en la cual las capacidades de producción son mayores a las demandas [2]. Extendiendo esta propiedad se considera la demanda de un nivel al siguiente y se asume que la capacidad de producción de un nivel es mayor al siguiente sin perdida de generalidad, es decir las capacidades decrecen a medida que el nivel esta más cerca del cliente final.

Estos parámetros comunes pueden ser gestionados desde el sistema de software desarrollado para el presente proyecto a fin de crear instancias complementarias. No obstante hay otros parámetros comunes entre las categorías que no son ofrecidos para su edición por considerarse estos característicos del problema ME-CLSP objetivo.

Entre ellos están: la no estacionalidad (o ausencia de temportada zafral), por lo que la demanda al igual que demás variables aleatorias, presenta una distribución uniforme en lugar de normal. Dicha distribución representa la realidad de varios productos y ha sido utilizada en varias investigaciones (por ejemplo, Burke et al. 2008[4], Manikas et al. 2009 [11]).

Asimismo esto simplifica significativamente la interpretación de resultados, ya que los valores son más estables a lo largo de los períodos y no hay que considerarlos bajo una desviación estándar. A su vez introducir varias configuraciones de media y desviación estándar multiplicaría la cantidad de categorías.

Producto también de la distribución uniforme en las variables aleatorias, se está ante una economía no especulativa, esto significa que las decisiones de producción no se postergarán especulando un menor costo en el futuro, ni tampoco se producirá el acumulado de varias demandas creyendo estás en un período oportunidad.

Otra característica del problema objetivo es que no hay beneficios en los costos de producción por unidad al aumentar las cantidades de producción, es decir descuentos por cantidad. Algunas razones por la cual se puede no dar este fenómeno puede ser que la mayor producción conlleva un mayor consumo de energía eléctrica, el costo de la misma puede estar definido por franjas según consumo, otro ejemplo puede ser por impuestos proporcionales según el nivel de contaminación ambiental, etc.

Se encuentra disponible un ejemplo de una instancia pequeña en el Anexo A.

3.2. Factibilidad

Se procuró que el conjunto de instancias generadas fueran 100 % factible. La relevancia de dicha preocupación radica en el tiempo que significa la resolución de todas las heurísticas propuestas con la muestra de instancias disponibles.

Profundizando en este punto, la factibilidad de una instancia está dado por la competencia entre las demandas y las capacidades de producción únicamente. Al momento de generar aleatoriamente los parámetros se decidió no intervenir para garantizar la factibilidad, de manera que no estén afectadas las distribuciones de las variables aleatorias.

Al comienzo del proyecto se constató tras un estudio que muy pocas de las instancias generadas eran factibles. Generar una instancia de 50 niveles y 50 períodos requería decena de miles de intentos. El chequeo utilizado era con una implementación de la heurística Greedy, la explicación de como a través de ella se verifica factibilidad será avanzado el presente documento, Sección 2.2.1.

Luego, esté mecanismo de filtrado evolucionó a la adaptación de Greedy para consumir la información de la instancia no desde un archivo donde se almacenó sino desde la memoria RAM, ahorrando la lectura en el medio de almacenaje y reduciéndose a aproximadamente 3 horas en el computador con las características descritas en la Sección 5.2 la obtención de una instancia factible de 50 niveles y 50 períodods.

En un siguiente paso, se realizó un chequeo de factibilidad controlado período a período que la demanda acumulada no supere la capacidad acumulada. Esto fue suficiente para asegurar que produciendo potencialmente en todos los períodos y haciendo uso del inventario se pudiera satisfacer todas las demandas, para todo período. Gracias a esto, se hizo viable poder trabajar con instancias de gran tamaño.

Esto no fue más que un esbozo de la experiencia reunida en el proceso de generación de instancias, ya que finalmente se hizo un reformulación más para asegurar la factibilidad, se definieron las capacidades de producción siempre mayores o iguales que las demandas y por tanto se cumple que la demanda acumulada siempre será menor o igual a las capacidades de producción acumuladas, resultando no ser necesario ningún chequeo.

Cabe destacar que sin este último paso, las instancias podrían presentarse como parcialmente resolubles. Podría suceder que algunas heurísticas puedan arrojar resultados y otras no, un claro ejemplo es Lote por Lote [14], la cual no utiliza inventario y de haber un nivel con menor capacidad que la demanda ya no podrá solucionar la instancia (no así cualquier otra heurística propuesta que haga uso del inventario).

3.3. Muestra de Estudio

La muestra de estudio es muy basta y variable, el objetivo fue descubrir que parámetros afectan principalmente los tiempos y costos totales resueltos por las heurísticas. Las 486 variedades revelaron cuales parámetros o combinación de los mismos afectan por igual o de manera diferente a las heurísticas.

Un conjunto de 5 instancias por categoría es la máxima cantidad con la que se pudo trabajar con los plazos disponibles y se consideró suficiente para promediar los valores, atenuando cualquier caso extraordinario que pudo obtenerse. No obstante en el Capítulo 5 se hizo un estudio de la desviación estándar relevante para la elaboración de conclusiones.

El tiempo estimado de resolución del conjunto de heurísticas sobre la muestra es:

- Heurísticas Conocidas - El tiempo de las heurísticas conocidas adaptadas se supone despreciable, ya que las mismas no tienen mucha inteligencia programada en sus decisiones y carecen de cálculo recursivo.
- Heurísticas Propuestas: Se estima menos de 3 minutos para las instancias más grandes en base a ensayos previos que fueron realizados.
- Algoritmo GLPK: Este estará acotado en tiempo para su búsqueda del resultado óptimo en 15 minutos, no siendo necesario que siempre disponga de todo ese tiempo ni que el resultado sea óptimo.

Descartando las instancias más pequeñas (un tercio del total) y multiplicando por 15 (minutos), GLPK tardaría 17 días y las heurísticas propuestas 5 días, totalizando poco más de 22 días la resolución completa. Por esta razón, el tiempo de ejecución del algoritmo de GLPK fue limitado a 15 minutos. Aunque para las instancias mas grandes el algoritmo no provea el óptimo, el tiempo de ejecución se mantiene acotado asegurando la completud de la muestra en un tiempo manejable.

Los resultados se consideran sensibles a los parámetros propuestos al comienzo de esta sección, la cantidad de niveles y períodos se consideró como el candidato más natural a afectar el tiempo de resolución. Dependiendo de cuan ajustadas u holgadas sean las capacidades de producción se entiende achicará o agrandará el dominio de soluciones posibles, ya que muy holgada la solución siempre será a través del camino menos costoso y en el otro extremo, con capacidades de producción muy ajustadas las heurísticas no tendrán suficiente libertad para elegir caminos, tan solo los posibles para hacer llegar el producto.

Por el lado de los costos de inventario y preparación, se estima que no afectaran los tiempos de resolución ya que sea cual sea el camino resultado, la misma cantidad de cálculos serán necesarios. Igualmente este par de parámetros parecen interesantes para heurísticas como Silver-Meal que deciden en contraposición costos de inventario con costos de producción. Los costos de producción no fueron parte de la categorización de instancias debido a que es irrelevante, estos costos actúan fijando una escala en la que se basan los rangos de valores para costos de inventario y preparación.

4. Solución de software

El presente proyecto requirió de un desarrollo de software a medida, con el principal objetivo de generar, clasificar y analizar grandes volúmenes de información.

4.1. Metodología

Dicho desarrollo estuvo basado en metodologías ágiles por la necesidad de ponerlo en marcha desde las primeras etapas para luego añadirle más funcionalidades y mejoras. Asimismo, durante el desarrollo se fueron tomando decisiones a corto plazo, redefiniendo requisitos y soluciones a raíz de lineamientos propuestos en entrevistas.

Dicho sistema creció de manera iterativa con la integración de varios aplicativos que cumplen funciones muy específicas. Se prescindió de la utilización de base de datos para persistir la información, siendo que los datos están poco relacionados entre si y esta relación se mantiene con un esquema de nombres.

4.2. Persistencia

La información se dispone de la siguiente forma:

a. **Resultado heurística**
   Cuando una instancia es procesada por una de las heurísticas implementadas se genera un archivo de tipo output, el cual contiene el tiempo de ejecución que requirió la heurística para dicha instancia, el costo mínimo hallado, y finalmente una representación en forma de grafo con el flujo de producción por los distintos niveles y períodos.

b. **Resultado del algoritmo de GLPK**
   En el caso del algoritmo de GLPK, luego de cada ejecución se generan dos archivos. El primero de tipo output, el cual contiene una representación de la solución y el segundo de tipo log, el cual indica si llegó a un resultado, si el mismo es óptimo o su porcentaje gap, el valor del mismo y el tiempo de ejecución.

c. **Instacia**
   Cada instancias es persistida en un archivo al igual que los outputs, pero coleccionadas en un directorio aparte. Estos archivos tienen un esquema de nombres el cual codifica la categoría a la que pertenece, la información dentro comienza con un conjunto de metadatos que junto al nombre completan la especificación de parámetros con los cuales fueron generados, siguiente a los metadatos esta el input del modelo matemático GLPK y heurísticas implementadas.

d. **Log de error**
   Dicho archivo fue generado para el auxilio durante el desarrollo, no solo para encontrar defectos en el software sino también para descubrir en que casos la variante de heurística Diagonal no resultaba en una solución factible. Este archivo especifica la excepción capturada durante la ejecución, la cadena de llamadas sucedidas en el código fuente, fecha del error, método de resolución e instancia involucradas.

El esquema de nombres usado para relacionar los datos será explicado con el siguiente ejemplo:

El nombre de una instancia es $Instancia\_50N\_50P\_20D\_22C\_4Im\_5IM\_201Prem-$ $\_400PreM\_2CapM\_636134193092333093$, N hace referencia a la cantidad de niveles, P a la cantidad de períodos, D la máxima demanda posible, Im e IM son el mínimo y el

máximo costo de inventario, $Prem$ y $PreM$ son el mínimo y máximo costo de preparación, $CapM$ hace referencia a una relación de capacidad del primer respecto al último nivel y por último una marca temporal del momento en el cual se generó la instancia (expresada en cantidades de 100-nanosegundos a partir del primero de enero del año uno, 12 horas UTC). Así pues un error de resolución de la variante heurística Diagonal sobre esta instancia se llamará
$error\_Diagonal\_Instancia\_50N\_50P\_20D\_22C\_4Im\_5IM\_201Prem\_400PreM\_2CapM-$
$\_636134193092333093$, el output generado con la solución propuesta por Greedy se llamará
$out\_Greedy\_Instancia\_50N\_50P\_20D\_22C\_4Im\_5IM\_201Prem\_400PreM\_2CapM-$
$\_636134193092333093$ y el log de ejecución generado por GLPK sobre la instancia se llamará
$log\_GLPK\_Instancia\_50N\_50P\_20D\_22C\_4Im\_5IM\_201Prem\_400PreM\_2CapM-$
$\_636134193092333093$.

Esta solución de software soporta el uso concurrente de múltiples usuarios, los datos son compartidos a través de un servicio de almacenamiento archivos en línea.

4.3. Módulos

Se eligió la Descomposición Modular como la arquitectura más adecuada para el desarrollo. Cada módulo resuleve una tarea simple, completa y de valor por si misma, en su integración respeta el principio de bajo acoplamiento y alta cohesión.

El conjunto de módulos definió el sistema de software que permite la generación de instancia, clasificación, resolución, visulización y análisis de la misma.

Los módulos fueron planificados, codificados, comprobados y depurados independientemente, permitiendo la presentación de resultados parciales a lo largo de la duración del proyecto de grado.

4.3.1. GLPK

Lo primero fue la creación del modelo matemático a resolver con el paquete de programación lineal entera mixta (GLPK) (versión 4.60 liberada el 1 de abril de 2016), este paquete esta compilado en 64 bits y utiliza el lenguaje Modeling Language GNU MathProg (versión 1.2.1). El código fuente a continuación refleja exactamente lo expuesto en la Sección 1.3:

```
param maxT;
param maxN;

set T := \{0..maxT\};
set N := \{1..maxN\};
set T2 := \{1..maxT\};
set N2 := \{2..maxN\};

param c\{t in T2,n in N\};  /*costos de produccion*/
param b\{t in T2,n in N\}; /*capacidades de produccion*/
param d\{t in T2\}; /*demandas*/
param h\{t in T2,n in N\}; /*costos de inventario*/
param k\{t in T2,n in N\}; /*costos de preparacion*/

var x\{t in T,n in N\} integer $\geq 0$;
var l\{t in T,n in N\} integer $\geq 0$;
var z\{t in T,n in N\} binary $\geq 0$;
```

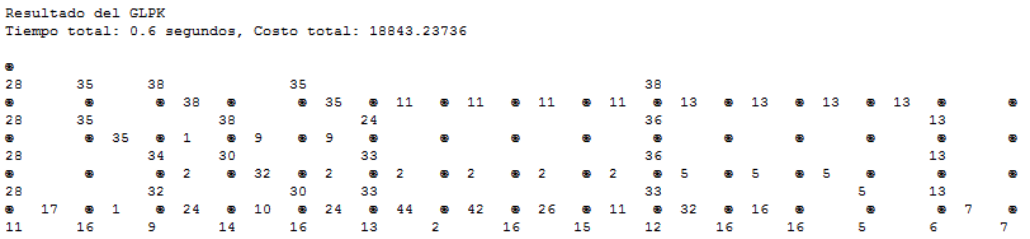

Figura 10: Extracto de salida del software desarrollado, representa un grafo solución propuesto por GLPK.

```
s.t. conservacionFlujoGeneral \{t in T2,n in N2\} : x[t,n] - x[t,n-1] = l[
    t-1,n] - l[t,n];
s.t. demandaIgualUltimaProd \{t in T2\} : d[t] = x[t,maxN];
s.t. inventarioInicialVacio \{n in N2\} : l[0,n] = 0;
s.t. produccionAcotada \{t in T2, n in N\} : b[t,n] $\geq$ x[t,n];
s.t. prepSiHayProd \{t in T2, n in N\} : z[t,n] * (sum\{t2 in t..maxT\} d[
    t2]) $\geq$ x[t,n];

minimize resultado: sum\{t in T2,n in N\} (z[t,n] * k[t,n] + x[t,n] * c[t,
    n] + h[t,n] * l[t,n]);
solve;

end;
```

A su vez se decidió no utilizar límite de búsqueda por gap (o distancia a la solución óptima), siendo posible de lo contrario interrumpir la búsqueda (por ejemplo) al alcanzar una solución garantizada del 5 % de exactitud. Esta decisión se debe a que se quiso conservar homogéneos los resultados del algoritmo de GLPK y resultados finalizados por gap y otros por tiempo, facilitando así el análisis.

Los resultados de pequeñas instancias fueron recolectados óptimos y casi de forma instantánea, de haber configurado la interrupción de búsqueda por gap, esta configuración se hubiese activado y los resultados ya no serían óptimos. Por otro lado, de haber tenido interrupción de búsqueda por gap y no por tiempo, el algoritmo de GLPK tendría un tiempo total de resolución incierto. Con el fin de determinar esto, se probo con una instancia de 50 niveles y 50 períodos con capacidades de producción ajustadas, sumando 30 horas de búsqueda sin encontrar siquiera una solución posible.

4.3.2. Representador de Grafo Solución

El segundo aplicativo implementado fue el Representador de grafos solución, el propósito fue monitorear los métodos de resolución en busca de posibles errores, ganar capacidad de interpretar las decisiones programadas, reconocer patrones de solución, experimentar con cambios en código y ver su reflejo consecuente en el flujo del producto a través del grafo. Un ejemplo de representacion se puede ver en la Figura 10.

Además de la representación del grafo solución por cada método de resolución, están presentes los valores de costo mínimo y los tiempos de resolución. En caso de no factibilidad o error no controlado en la programación, también se expone un mensaje de error detallado.

Figura 11: Interfaz de usuario correspondiente al Generador de Instancias.

4.3.3. Generador de Instancias

El tercero fue el Generador de instancias, un módulo cuya complejidad responde al alto grado de parametrización de las instancias explicado en el Capítulo 3. No se interviene a modo se asegurar factibilidad para no alterar la distribución de las variables aleatorias que modelan costos, capacidades, etcétera. El generador es altamente configurable pudiendo generar a demanda instancias con los parámetros deseados al instante y de manera ilimitada, también implementa una funcionalidad de generación automática, la cual sin requerir intervención ninguna y en cuestión de un par de segundos construye una muestra de estudio de 2430 instancias repartidas por igual en 486 categorías.

A través de la definición de una interfaz, la gestión de heurísticas es homogénea, todas ellas consumen de igual manera las instancias y sus resultados son expresados para ser interpretados por el único Representador de grafos solución. En la Figura 11 se ven los campos de la interfaz de usuario.

4.3.4. Clasificador de Instancias y Gestor Web

El siguiente desarrollo fue un módulo centralizador y una aplicación web, los mismos permiten la gestión las heurísticas, solicitar la resolución consecutiva de tantas instancias por tantas heurísticas como se desee, administra los resultados, ofrece gráficos por categoría de instancias (el promedio de duración, costos total solución y resoluciones pendientes). En la Figura 12 se puede ver una imagen de la aplicación mostrando las diferentes categorías de Instancias.

La página principal se divide en cabezal, cuerpo y pie; el cabezal contiene el panel para la utilización del generador de instancias. Desde el cuerpo de la página se enseñan dos listas ordenadas de múltiple selección, a la izquierda la de métodos de resolución y a la derecha las categorías de instancias persistidas. Por último, al pie el botón Resolver"que por cada método de resolución seleccionado ejecuta todas las instancias comprendidas por cada categoría seleccionada y el botón "Generar Reporte"que exporta toda los resultados obtenidos a una planilla de cálculo.

La aplicación web permite inspeccionar cada categoría de instancia, ofrece las gráficas recién mencionadas de acuerdo a los resultados acumulados hasta el momento, la posibilidad de solicitar resolución de instancias concretas y la posibilidad de visualizar los grafos solución de instancias concretas. En la Figura 13 se muestra una de las categorías, con sus instancias y gráficas asociadas.

Las gráficas explicadas de izquierda a derecha se explican a continuación:

a. *Porcentaje de Desviación:* Este gráfico muestra según el costo total promedio de las instancias de la categoría, el porcentaje relativo al resultado promedio del algoritmo

Figura 12: Extracto del cuerpo de la página web principal.

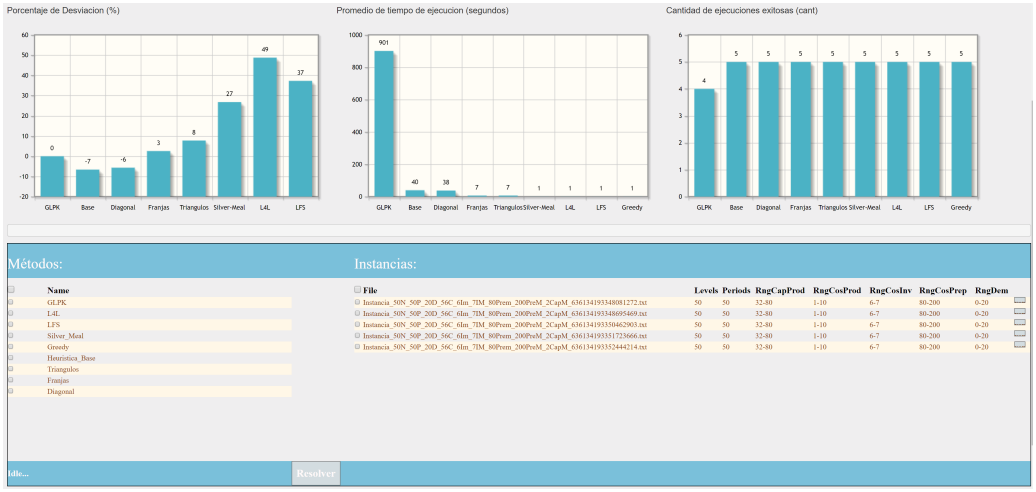

Figura 13: Página Web de una categoría.

de GLPK.

$$DP = \frac{CostoHeuristica - CostoGLPK}{CostoGLPK}$$

b. *Tiempo de Ejecución:* Gráfica el tiempo promedio en segundos que tardaron las ejecuciones para cada una de las instancias de la categoría.

c. *Ejecuciones Exitosas:* Ya que algunas combinaciones instancia/heurísticas pueden encontrarse pendientes de ejecución aún o la particularidad de dicha combinación puede no ser factible, el tercer gráfico muestra la cantidad de instancias que se ejecutaron para cada heurística de forma exitosa.

4.3.5. Generador de Reportes

La aplicación web cuenta con una funcionalidad sencilla de reporte. El botón "Generar reporte" dentro de la Web construye un archivo con los resultados obtenidos hasta el momento de la ejecución de las heurísticas.

El reporte contiene las siguientes columnas:

- ***Niveles***: Cantidad de niveles de la instancia
- ***Períodos***: Cantidad de períodos de la instancia
- ***Demanda***: Clasificación de demanda (con el criterio descrito anteriormente)
- ***Capacidad***: Holgura de Capacidad (con el criterio descrito anteriormente)
- ***Costos de Inventario***: Cota inferior y superior de dicho costo.
- ***Costos de Preparación***: Cota inferior y superior de dicho costo.
- ***Decrecimiento de capacidad de Producción***: Pendiente de las capacidades de producción entre el primer y el último nivel.
- ***Instancia***: Nombre completo de la instancia incluido su marca de tiempo
- ***Método***: Método de resolución utilizado (L4L, LFS, GLPK, etc)
- ***Tiempo***: Tiempo que llevo ejecutar la instancia con ese método
- ***Costo***: Costo mínimo hallado por la heurística
- ***Porcentaje Desviación***: Desviación del método medido en comparación con el algoritmo de GLPK.
- ***Porcentaje Optimalidad***: Cercanía al óptimo, considerando el dominio de soluciones.
- ***Diferencia de tiempo***: Diferencia de tiempo entre la ejecución del método y el algoritmo de GLPK con la misma instancia
- ***Comentarios***: Usado en el caso de que la instancia no haya podido ser procesada por el método.

Esta tabla es ingresada en una plantilla de cálculo conformada para el uso de tablas de pivot. En la misma se pueden generar reportes dinámicos nuevos de ser necesario, que ayuden a la compresión de los datos. Esta información se puede utilizar como un cubo para generar reportes instantáneos según sea de interés. Los reportes predefinidos en el plantilla son los siguientes:

- ***Cantidad satisfactoria***: Muestra la cantidad de instancias que se han ejecutado de forma satisfactoria agrupadas por cada una de las heurísticas.
- ***Desviación - Nivel Período***: Se muestra la desviación de las heurísticas implementadas en comparación con los resultados obtenidos mediante el algoritmo de GLPK.
- ***Tiempo - Nivel Período***: Grafica los tiempos de ejecución de las heurísticas implementadas según la cantidad de vértices en el grafo problema.
- ***Desviación por Capacidad***: Muestra la evolución de la desviación de las heurísticas a medida que aumenta la holgura de la capacidad de producción, y la velocidad de crecimiento de la misma.
- ***Tiempo por Capacidad***: Muestra los tiempos de ejecución de las heurísticas a medida que aumenta la holgura de la capacidad de producción, y la velocidad de crecimiento de la misma.
- ***Desviación por Costos***: Muestra la desviación de las heurísticas teniendo en cuenta las los costos de inventario y su evolución tam.
- ***Optimalidad - Nivel Período***: Muestra la optimalidad de las heurísticas teniendo en cuenta los períodos y niveles del problema.
- ***Desviación Estándar***: Muestra la desviación estándar promediada para cada una de las heurísiticas.

4.4. Arquitectura

Ya presentados los componentes, a través del diagrama establecido en la Figura 14 se pretende ilustrar como se organizan en la solución de software.

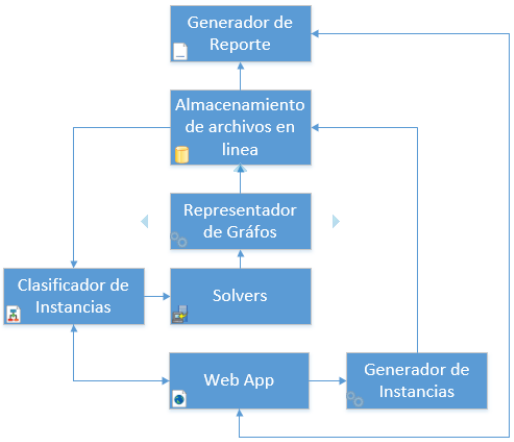

Figura 14: Comunicación entre los componentes de la solución de software.

Los casos de uso comprendidos son:

- ***Generar instancia***: El caso de uso comienza desde la solicitud de la página Web principal al Generador, el mismo recibe todos los parámetros necesarios y almacena los archivos correspondientes en el Servicio de Almacenaje de Archivos en Línea.

- ***Resolución de instancia***: Gracias a la organización por categorías del Clasificador de Instancias desde la página Web se puede seleccionar los métodos de resolución e instancias objetivos y solicitar su resolución. Así el módulo resolución va recibiendo consecutivamente todas las peticiones y a medida que va despachando cada caso utiliza el Representador para almacenar el archivo con el grafo ya dibujado.
- ***Generar reporte***: El Generador de Reporte no utiliza el Clasificador de Instancias porque abarca todas las soluciones de manera indiscriminada, una vez que consolida toda la información acumulada procede a exportarla en un formato adecuado para su manipulación.
- ***Visualizar solución***: Gracias al Clasificador de Instancias se puede acceder a la página Web de categoría objetivo, en ella se puede solicitar ver las soluciones de una instancia en particular.
- ***Visualizar gráficas de categoría***: Idem que el caso de uso anterior, una vez ubicado en la página Web de interés se puede apreciar los datos recolectados y procesados de las instancias involucradas.

5. Resultados obtenidos

Una vez ejecutadas las heurísticas y el método *Branch and Cut* (por medio de GLPK) para todas las instancias, se recolectó y analizó los resultados de las ejecuciones para determinar la calidad de la heurística propuesta en comparación con las ya existentes y con la solución provista por el algoritmo de GLPK.

Para ellos se tomaron tres métricas específicas para los distintos criterios de clasificación de instancias, para medir tanto la optimalidad como la diferencia en el tiempo de ejecución. El primero fue mediante la desviación respecto al borde inferior, considerando el porcentaje de desviación del costo de la heurística (HC) respecto al resultado obtenido por el algoritmo de GLPK (tomando este como el borde inferior o LB):

$$DP = \frac{HC - LB}{LB}$$

También se uso el borde superior como costo maximizado para satisfacer la demanda (UB). Se tomó la distancia entre UB y LB como la escala en la que se ubicará la heurística a ser evaluada. Para calcular este costo, se implementó un método "Greedy" el cual fue explicado en la Sección 2.2.1.

$$OP = \frac{HC - LB}{UB - LB}$$

Y la tercera medición se realizó basándose en el tiempo de ejecución. Particularmente para aquellos conjuntos de instancias con gran cantidad de niveles y períodos en los cuales el tiempo de de resolución del algoritmo de GLPK aumenta de forma substancial.

5.1. Ordenes de Ejecución

La heurística Base llega a la solución a través de múltiples ejecuciones del algoritmo Dijkstra por período, su orden de ejecución estará basado pues en el orden de ejecución del algoritmo Dijkstra.

El orden general del algoritmo de Dijkstra es:

$O(|V|^2 + |A|)$ [10] siendo V la cantidad de Vértices del Grafo y A la cantidad de Aristas. En particular, en los grafos utilizados para representar el problema ME-CLSP se cumple que $A = 2V - (N + T + 1)$, con N la cantidad de niveles y T la cantidad de períodos.

El algoritmo ejecuta solamente $O(|V|^2)$ operaciones de acuerdo a lo antes expresado, para minimizar el tiempo de resolución de la heurística se debe minimizar tanto el número de ejecuciones de Dijkstra como la cantidad de vértices que se considera en cada ejecución.

La heurística Base junto a sus variantes, ejecutan el algoritmo de Dijkstra al menos un vez por período y como máximo λ_t veces, resultando los λ_1 ... λ_T la cantidad máxima de particiones que tiene una demanda al momento de calcular como producirla, significando que en el peor caso la cantidad de operaciones esta dado por:

Peor caso:

$$\lambda_1(1N^2) + \lambda_2(2N)^2 + \lambda_3(3N)^2 + \ldots + \lambda_N(TN)^2$$

La ecuación de arriba se puede escribir de la siguiente forma.

$$\sum_{t=1}^{T} \lambda_t (t.N)^2$$

Para determina λ_t tener en cuenta que el producto debe bajar por el grafo N niveles y desplazarse t períodos a la derecha, suponiendo la tercera demanda de un grafo de 5 niveles y 5 períodos, un camino posible podría ser: B,B,B,B,D,D,B haciendo referencia a que baja 4 veces y luego (en último nivel) se mueve 2 veces hacia la derecha para finalmente bajar con la producción del último nivel al cliente. Así pues las cantidad de formas diferentes de combinar las Bs y Ds (con excepción de la última B que es fija) está dada por las permutaciones con repeticiones de 6 elementos agrupados en 4 y 2.

$$PR_6^{2,4} = \frac{6!}{2!4!} = 15$$

Se pueden encontrar 15 caminos posibles para satisfacer la tercera demanda en este grafo, pero considerando que la demanda máxima es solamente 20 unidades, la cantidad de ejecuciones de Dijkstra esta limitado por este número, el valor de λ_t expresados en forma general resulta:

$$\lambda_t = \min\left(\frac{(t+N-2)!}{(t-1)!(N-1)!}, Demanda_t\right)$$

Si $N = 15$, $P = 50$ y $Demanda_{max} = 20$, el conjunto $\lambda_t = \{1, 15, 20, 20, ..., 20\}$ y por lo tanto la cantidad de operaciones en el peor caso es:

$$N^2 + 15(2N)^2 + \sum_{t=3}^{T} 20(tN)^2$$

$$= 61N^2 + 20N^2 \sum_{t=3}^{T} t^2$$

$$= 61N^2 + 20N^2 \left(\frac{2T^3 + 3T^2 + T}{6} - 5\right)$$

$$= 61N^2 + \frac{10}{3}N^2 2T^3 + \frac{10}{3}N^2 3T^2 + \frac{10}{3}N^2 T - \frac{10}{3}N^2 30)$$

$$= 61N^2 + \frac{20}{3}N^2 T^3 + 10N^2 T^2 + \frac{10}{3}N^2 T - 100N^2$$

$$= (\frac{20}{3}N^2)T^3 + (10N^2)T^2 + (\frac{10}{3}N^2)T - 39N^2$$

$$= O(T^3)$$

Pero para lograr determinar los λ_t en la práctica se registró la cantidad de veces promedio que se ejecuta Dijkstra por período para cada una de las instancias.

El cálculo de ordenes de ejecución se basa en instancias de 15 niveles por ser un valor suficientemente complejo. Para estudiar la tendencia es preciso no considerar los primeros períodos, ya que la cantidad de ejecuciones del algoritmo de Dijkstra para un período esta en función de holguras remanentes de caminos ya utilizados de hasta 5 períodos posteriores aproximadamente, por esta razón la situación inicial carece de normalidad y a largo

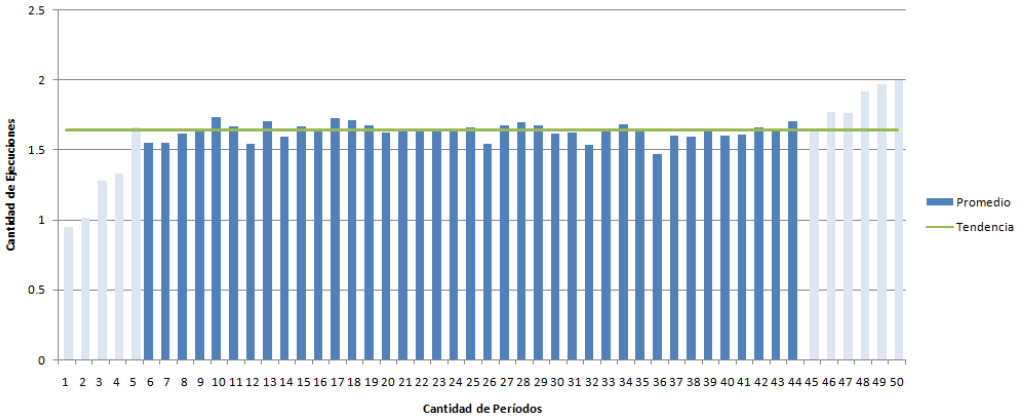

Figura 15: Línea de tendencia de los λ sobre los promedios en instancias de 15 niveles y 50 períodos

plazo no resulta relevante. De igual manera los últimos períodos tienen un comportamiento atípico, ya que el costo arista mencionado en la Sección 2.3.1 los afecta en particular logrando un menor costo total final pero mayor cantidad de ejecuciones de Dijkstra. Con estas consideraciones, se constató que esta cantidad de ejecuciones Dijkstra se mantiene constante conforme suben los períodos.

En la Figura 15 se muestra el promedio en función de la cantidad de períodos. Se puede ver que la tendencia tiene un comportamiento lineal con una pendiente nula, dicha constante fue verificada empíricamente con el subconjunto de las instancias de 15 períodos y adicionalmente con instancias elaboradas para este solo propósito de hasta 200 períodos. Por lo tanto se puede considerar la función de promedios $\lambda(t) = 1{,}642$ donde t representa el período.

Reultando:

$$\sum_{t=1}^{T} 1{,}642(15t)^2 = 369{,}45\left(\frac{T^3}{3} + \frac{T^2}{2} + \frac{T}{6}\right) = O(T^3)$$

5.1.1. Caso Variante Franjas

Por otro lado como se comentó anteriormente el principal objetivo de las variantes implementadas fue lograr resultados en menor tiempo de ejecución. Para ello lo que se pretendió fue lograr menor cantidad de cálculos. Si se toma una de las variantes más claras, como el caso de Franjas y se considera que la misma toma franjas de 10 períodos, la ecuación correspondiente sería:

$$cantFranjas \sum_{t=1}^{10} \lambda_t (tN)^2$$

Haciendo el mismo cálculo que se hizo para la heurística Base se determina los λ mediante la tendencia de los promedios. Como se puede ver en la Figura 16 la tendencia es la misma que para la Heurística Base, constante a través de todas las franjas de 10 períodos. Los últimos períodos tuvieron las mismas consideraciones que el estudio anterior pero no así los primeros, al ser estos periódicos marcan tendencia a largo plazo. La función de promedios $\lambda(t) = 1{,}413$ donde t representa el período.

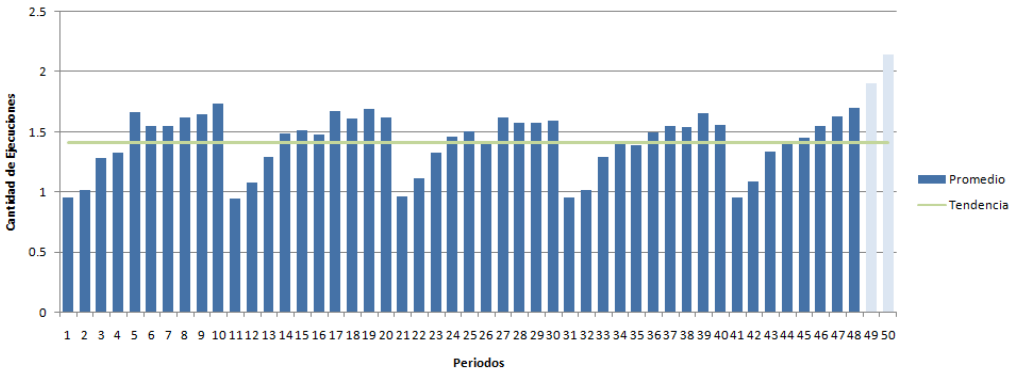

Figura 16: Línea de tendencia de los λ sobre los promedios en instancias de 15 niveles y 50 períodos para la variante Franjas

El menor promedio conjuntamente con invocarse el algoritmo de Dijkstra en grafos más pequeños, logra hacer una diferencia considerablemente menor en comparación con la Base. Esta diferencia es obviamente más notable en instancias cada vez mayores.

$$cantFranjas \sum_{t=1}^{10} \lambda_t (15t)^2 = 70{,}293T = O(T)$$

Al simplificar el cálculo de orden para esta variante, se comprueba que es lineal y que la variante cumple el propósito de reducir el orden cúbico de la Heurística Base.

5.2. Contexto de ejecución (Hardware y Software)

La integración de GLPK con el software no se realizó utilizando la API disponible, sino creando un nuevo proceso que invoca al resolvedor a través de un proceso batch. Dicho proceso si bien se estableció como de alta prioridad para el sistema, resultó indistinta esta medida, ya que una vez iniciado GLPK abre nuevos hilos de ejecución con configuraciones propias. Igual de indistinta la opción PriorityBoostEnable de la librería System.Diagnostics del framework .NET (configuración por la cual si la ventana principal del proceso tiene foco se le asignan recursos hardware extra), el tiempo de ejecución de GLPK no puede mejorarse en este sentido, no se dispuso de suficiente control sobre su ejecución.

El software construido una vez inicializado establece el hilo de ejecución principal como de alta prioridad para el sistema operativo, por lo tanto las heurísticas ejecutan con la máxima jerarquía.

El sistema operativo elegido fue Microsoft Windows® 10, el software de proyecto fue desarrollado con lenguaje de programación C# por ser este uno de los principales lenguajes de alto nivel y brindar suficiente control para el propósito de este proyecto de grado.

Las características del computador utilizado fueron:

- **Procesador** - Intel® Core™ i7-4710HQ con 4 núcleos, 8 subprocesos, 6M de Cache y frecuencia de trabajo de 3.5 GHz.
- **Memoria RAM** - 2 módulos de memoria de 8.0 GB DDR3L a 800 MHz, trabajando en doble canal.

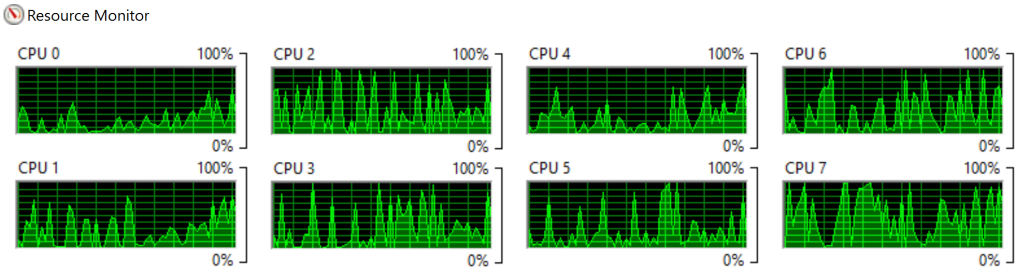

Figura 17: Estado de los procesadores durante la resolución de una instancia.

- **Sistema operativo** - Microsoft Windows® 10 Home de intrucciones de 64 bits.
- **Disco duro** - Las velocidades de la unidad de almacenaje no es relevante, ya que el tiempo de ejecución de las heurísticas no considera la lectura de las instancias ni la escritura del resultado.
- **GPU** - El potencial de procesamiento de la unidad gráfica no es relevante ya que no se aprovechó la arquitectura CUDA para obtener mejor rendimiento.

Se ha logrado sacar buen provecho de los recursos implementando el patrón filtros y tuberías con Language Integrated Query. Este componente de Microsoft .NET Framework permite utilizar una sintaxis declarativa con la cual es posible paralelizar tanto la obtención de datos (no aplica) como su procesamiento. Esto se puede apreciar en la Figura 17 donde se ve el conjunto de procesadores durante la resolución de una instancia. Este aspecto es muy relevante para lograr mejores tiempos de ejecución.

5.3. Análisis de Resultados

Luego de la ejecución de las 2430 instancias se efectuó un estudio detallado de los resultados obtenidos. Los mismos se obtuvieron por medio del reporte y la utilización de tablas dinámicas como se menciono anteriormente.

Cabe destacar que ya que hubieron más de 300 instancias con las cuales el algoritmo de GLPK no fue capaz de devolver una solución, para los resultados presentados donde se mide la desviación de los métodos, se excluyeron estas instancias (siendo que la desviación no se puede medir en esos casos).Por otro lado para las métricas en las que se toman los tiempos de ejecución, sí se toman estas instancias por ser los métodos independientes entre sí.

5.3.1. Porcentaje de Desviación

La forma de medir la exactitud de cada método, se hará por medio de la desviación con respecto a la solución provista por el algoritmo de GLPK. Para tener una idea global de cual es el desempeño de las 8 heurísticas implementadas se muestra debajo un primer acercamiento a los resultados obtenidos por cada una de ellas considerando ese criterio. Como se dijo antes, para muchos de los casos de 15x50 y 50x50 no se obtuvieron resultados del algoritmo de GLPK, por lo tanto no va a ser posible hacer la comparativa (esto se explicará más adelante).
En el gráfico de la Figura 18 se puede ver el desempeño de las heurísticas conocidas adaptadas que fueron implementadas para el proyecto. Se puede apreciar como la heurística de Silver-Meal, se separa del las otras dos. Esto es un comportamiento esperado ya que

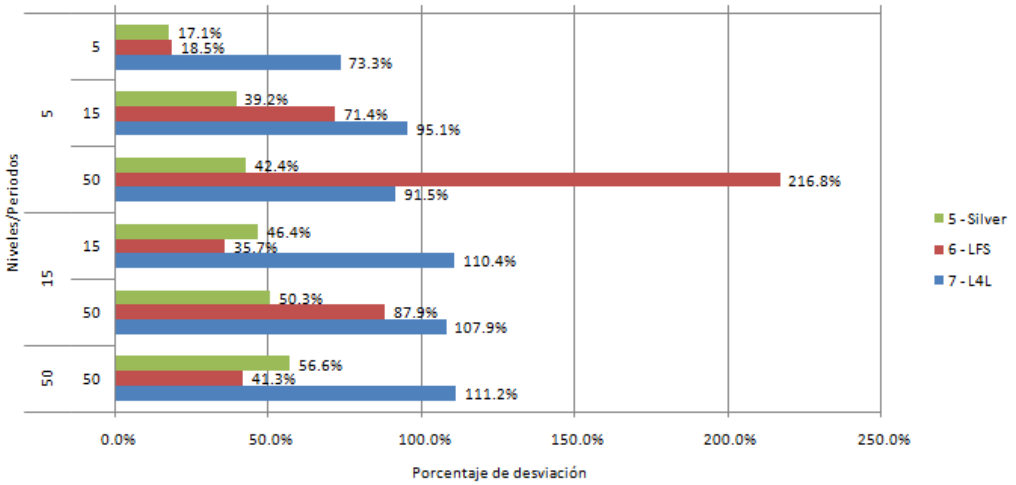

Figura 18: Desviación con respecto a solución del algoritmo de GLPK de Heurísticas Conocidas Adaptadas

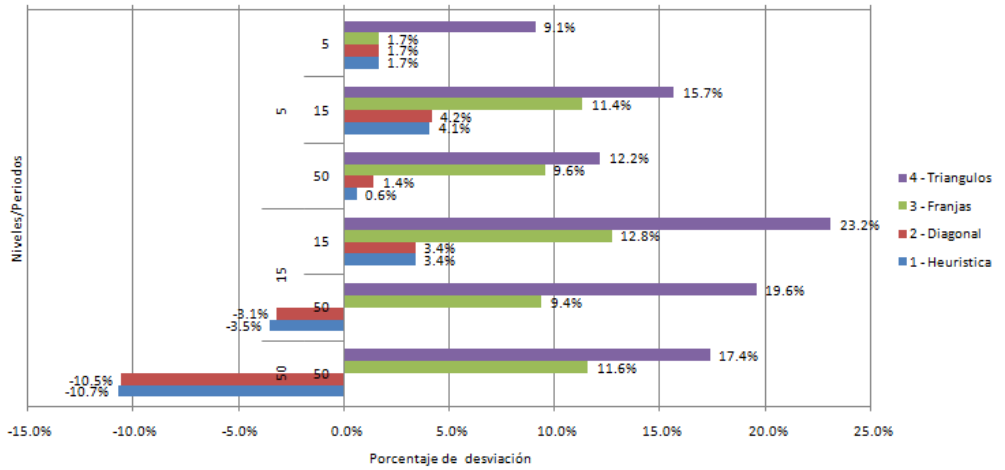

Figura 19: Desviación con respecto a la solución de GLPK de las heurísticas propuestas

la misma tiene un mayor grado de inteligencia que las heurísticas rivales.

Un poco más curioso es el resultado de las heurísticas de Lote por Lote y Lote de Tamaño Fijo. Aunque Lote por Lote se presenta como una heurística sin mayores cálculos y por tanto en la mayoría de los casos muestra los peores resultados; en aquellos donde la cantidad de períodos es muy grande (50 en este caso) se puede ver como Lote de Tamaño Fijo gana el peor lugar de los tres.

Este último comportamiento se debe a dos motivos. El primero es que tiene que definir el mejor tamaño de lote para una cantidad más grande de períodos. Ésto implica que se genera más producción innecesaria que en casos con menor cantidad de períodos. El segundo es que cuando la heurística de Lote de Tamaño Fijo genera producción de sobra, la tiene que inventariar y por lo tanto pasar por muchos más períodos de inventario. Esto genera un mayor sobrecosto.

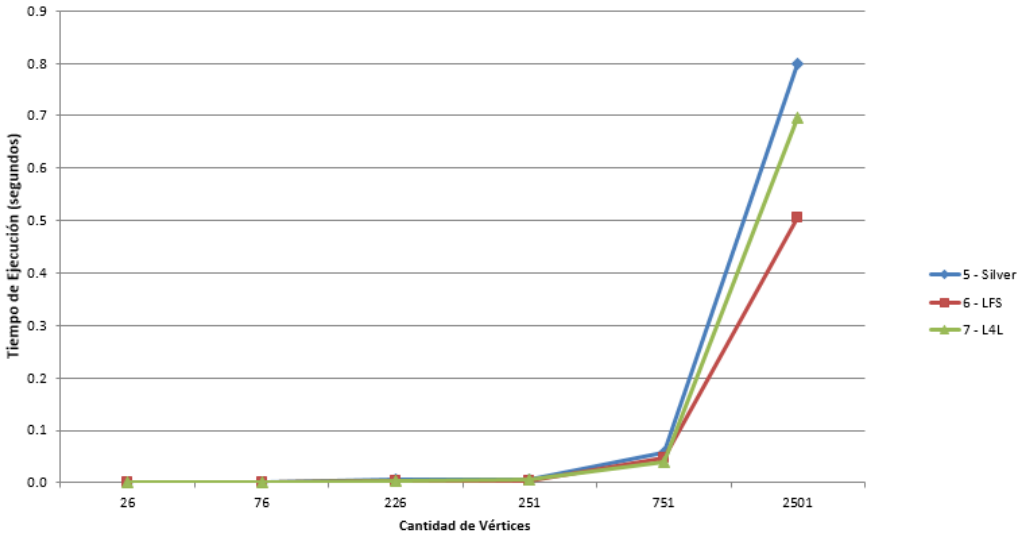

Figura 20: Tiempo de ejecución Heurísticas Conocidas Adaptadas

En la Figura 19 se aprecia el gráfico con la comparativa entre las heurísticas propuestas (nuevamente con el mismo criterio). Lo primero que sale a la vista es que el desempeño de las heurísticas propuestas es comparativamente mejor que las heurísticas conocidas adaptadas.

Dentro de las variantes desarrolladas, se constata que la mejor de ellas es consistentemente la heurística Base y la variante Diagonal en algunos casos. Esto era de esperarse ya que las variantes a la heurística Base se realizaron para bajar los tiempos de ejecución y no para mejorar la desviación.

En particular, la variante Franjas ronda el entorno al 10 % de desviación, lo cual es un porcentaje aceptable para los tiempos de ejecución (se vera más adelante).

Volviendo a la heurística Base, se puede resaltar que la misma siempre se encuentra por debajo del 5 % lo que demuestra ser bastante acertada en comparación con el óptimo. Incluso para los casos de más períodos, la misma se comporta mejor aún que el algoritmo de GLPK, siendo que este último no llega a un resultado de calidad con el tiempo de ejecución establecido. Esto se puede ver por los valores negativos de los últimos gráficos.

5.3.2. Tiempos de Ejecución

En cuanto a los tiempos de ejecución, se muestran dos gráficos con la comparativa primero entre las heurísticas conocidas adaptadas, y luego con aquellas propuestas en el proyecto. Para realizar estos gráficos en vez de tomar niveles y períodos se decidió tomar la cantidad de vértices involucrados para poder mostrar una progresión. El cálculo de vértices se realiza simplemente haciendo $V = P.N + 1$, así por ejemplo, una instancia de 15 períodos con 15 niveles, tiene 226 vértices.

En la Figura 20 se ven los tiempos de ejecución de las heurísticas conocidas adaptadas. Como fue de esperar las mismas se ejecutaron muy rápidamente. Incluso para aquellas instancias donde la cantidad de vértices es muy grande, cualquiera de las 3 terminó en menos de un segundo.

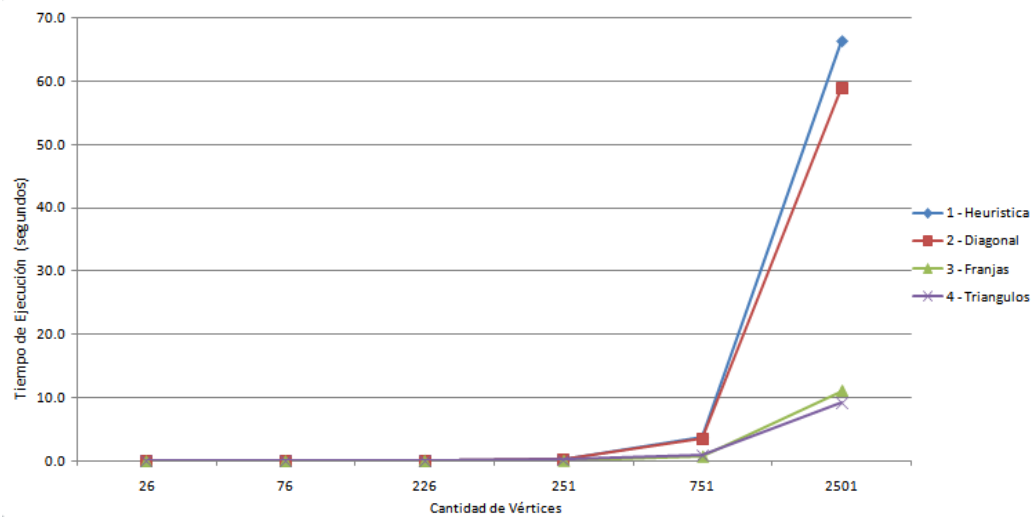

Figura 21: Tiempo de ejecución Heurísticas propuestas

Esto hace particularmente interesante al método de Silver-Meal, ya que aunque los tiempos de ejecución fueron muy rápidos, el resultado es en algunos casos bastante preciso, como se puede ver en las gráficas de desviación (Figura 18).

En el caso de las Heurísticas propuestas (Figura 21), se ejecutan en tiempos relativamente cortos, considerando la complejidad del problema. Hay que tener en cuenta que para casos donde la heurística Base demora 60 segundos en encontrar el resultado, el algoritmo de GLPK no llega a devolver un resultado en 15 minutos. De hecho, con el objetivo de estimar cuanto demoraría, se ejecutaron instancias particulares con GLPK sin limitación de tiempo, y muchas de ellas no llegaron a un resultado en menos de 30 horas.

Comparando los tiempos obtenidos con las cuatro variantes, se observa como la heurística Base es la que tiene mayores tiempos de ejecución. Como se mencionó en secciones anteriores, las variantes a esta heurística se desarrollaron con el único objetivo de obtener resultados similares a la heurística Base, pero con tiempos de ejecución más bajos.

En particular la heurísitca Franjas aunque no provee un resultado tan exacto como la heurística Base, igual devuelve una solución en el entorno al 10 % de desviación, en menos de 10 segundos. Esto la hace una opción interesante para instancias de gran tamaño. Incluso tamaños mayores de los incluidos en el alcance de este proyecto.

Por último, considerando ambos criterios de comparación, se establece que para instancias de gran tamaño, la heurística Base implementada es comparativamente mejor que el algoritmo de GLPK (en estas condiciones), siendo que provee un resultado más exacto y en un tiempo 15 veces menor.

5.3.3. Capacidad de Producción

Uno de los criterios con los que se clasificaron las instancias fue por medio de las capacidades de producción. Según el alcance se definieron 3 clasificaciones: 22, 28 y 56 (explicadas en el Capítulo 3). Estas clasificaciones determinan a grandes rasgos, las holguras entre los vértices del grafo.

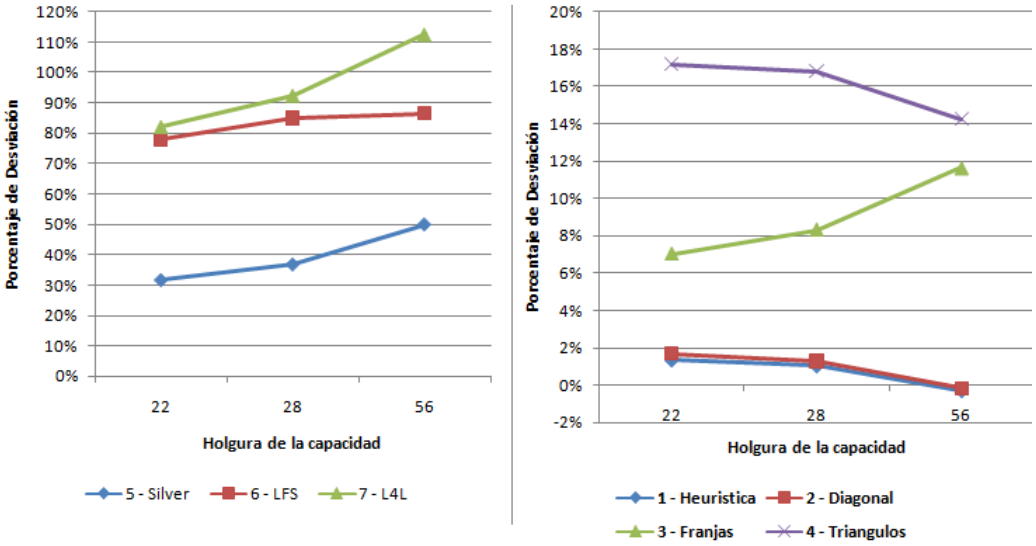

Figura 22: Desviación de las heurísticas implementadas según criterio de capacidad de producción.

Quiere decir que aquellas instancias de tipo "22" tendrán menor holgura entre la demanda promedio y la capacidad de producción de sus aristas, mientras que aquellas del tipo 56, tendrán la mayor holgura que las demás. En muchos casos esto implica que potencialmente se puede abastecer todas las demandas con solo los primeros períodos de producción.

En las Figuras 22 y 23 se ven los comportamientos de las distintas heurísticas implementadas, considerando: las "holguras" de las capacidades de producción para cada una ellas, y el incremento de la capacidad de producción respectivamente. Tanto el incremento por nivel como la holgura general de las capacidades de producción, reflejan el mismo comportamiento, descrito a continuación.

En los gráficos se ve una tendencia general de todas las heurísticas conocidas adaptadas. En los casos cuando la holgura es más ajustada, los resultados obtenidos son más exactos. Y por lo contrario en aquellos casos donde existe una mayor holgura, las heurísticas tienden a retornar peores resultados.

Esto en principio se debe a que cuando la holgura es mayor, el conjunto de soluciones se incrementa. Quiere decir que la probabilidad de elegir una solución cercana al valor óptimo disminuye en cuanto aumenta la holgura.

Se ve claramente con el caso de Lote por Lote. Esta heurística produce todo desde el nivel superior hasta el nivel de demanda, sin posibilidad de generar inventario. Cuanto menos holgura tengan las instancias, es más probable que se produzca la demanda de un período en el mismo período. Por lo tanto, Lote por Lote disminuye su desviación respecto al resultado del algoritmo de GLPK. En el caso contrario, cuando la holgura es mayor, la producción se hará en aquellos períodos con menor costo, y probablemente no esté distribuida entre todos los períodos como Lote por Lote espera, y por eso proveerá un peor resultado.

Respecto a Lote de Tamaño Fijo y Silver-Meal, son ejemplos menos extremos que Lote

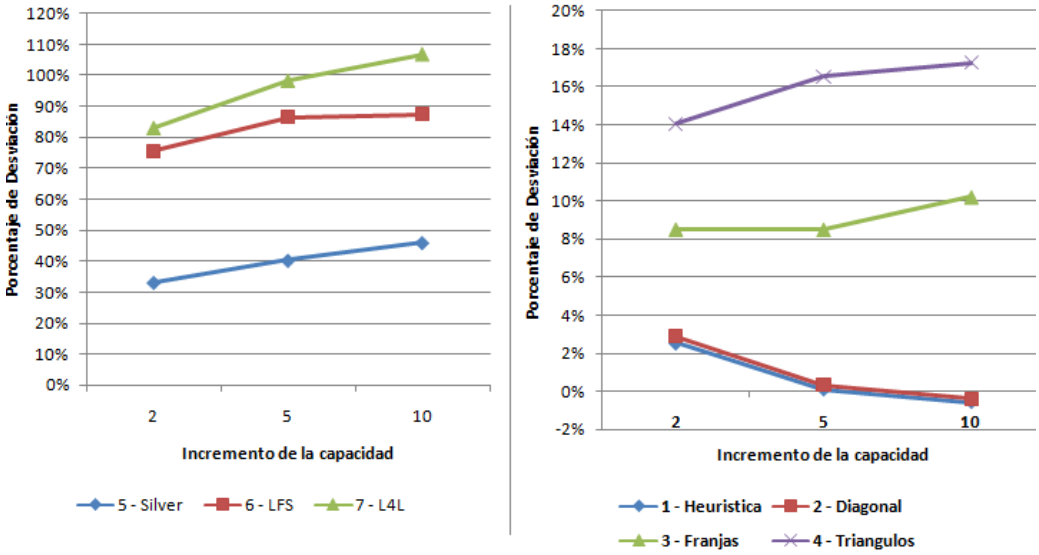

Figura 23: Desviación de las heurísticas implementadas según el incremento de capacidad de producción.

por Lote, ya que las mismas sí buscan una solución aunque el dominio de las mismas sea mayor, igualmente tienen peor desviación puesto que el algoritmo de GLPK es más eficiente en su búsqueda que ellas.

Por el lado de las heurísticas propuestas, con capacidades de producción ajustadas se obtuvo mayor desviación. Esto en un principio se atribuyó al hecho de que las instancias de 50 niveles y 50 períodos en el caso de capacidades de producción ajustadas, el algoritmo de GLPK no terminó las resoluciones y por tanto solo fueron promediados valores óptimos proveniente de instancias más pequeñas. Y en el caso contrario en grafos de 50 niveles y 50 períodos con capacidades de producción holgadas si se promediaron, pero con un resultado con alto gap; perjudicando el algoritmo de GLPK y beneficiando la desviación de las demás heurísticas.

Esta conjetura fue descartada puesto que se hizo la misma gráfica solo considerando grafos de 15 niveles y 15 períodos (tamaño para el cual casi el total de las heurísticas fueron resultas óptimamente por el algoritmo de GLPK) y el comportamiento fue el mismo que el ilustrado.

Para realmente explicar el porque de este comportamiento, considerar que a capacidades de producción más ajustadas se realizan mayor cantidad de cálculos. En el caso de las heurísticas propuestas se ejecutan varias veces el algoritmo Dijkstra por período, en tanto a GLPK le toma mayor tiempo de resolución su *Branch and Cut* aplicado. El esfuerzo de este último es más efectivo al no basarse el la optimización de abastecer demanda a demanda sino el conjunto de todas ellas. Este enfoque más global saca una ventaja a las heurísticas propuestas.

Para el comportamiento particular de Franjas, la explicación es que ante mayor holgura de capacidades de producción, más relevante se vuelve la utilización de inventario. Puesto que Franjas esta limitado en el uso de inventario y no puede trascender 10 períodos en el almacenaje, su desviación aumenta en comparación con el algoritmo de GLPK y todos los demás métodos de resolución que si lo hacen.

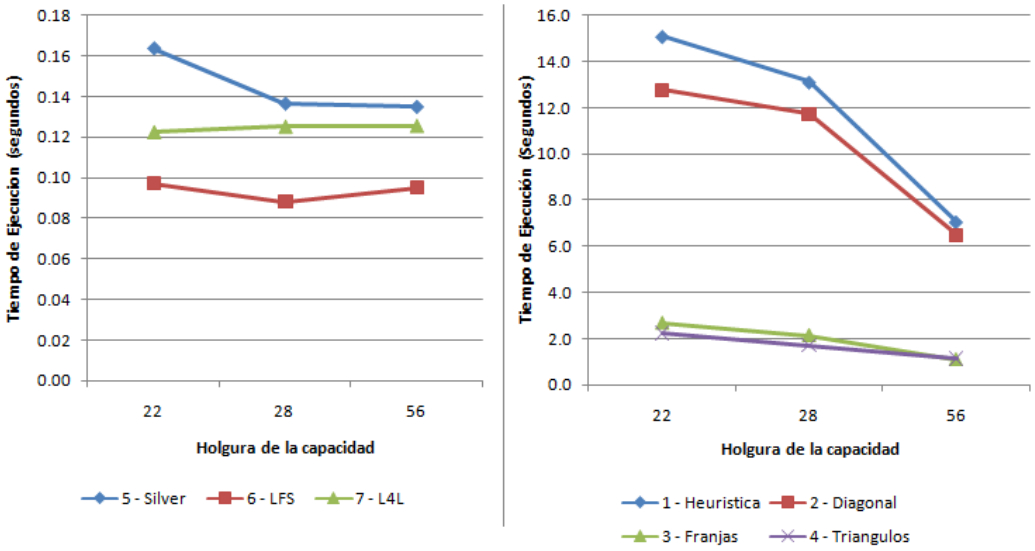

Figura 24: Tiempo de ejecución de las heurísticas implementadas según criterio de capacidad.

En las Figuras 24 y 25 se muestran los gráficos asociados al tiempo de ejecución según los criterios de holgura de la capacidad e incremento de la capacidad respectivamente.
La primer conclusión que se desprende de estos gráficos es que el tiempo de ejecución para las heurísticas conocidas adaptadas no se ve afectado por la capacidad. En realidad existe una diferencia mínima, que varía entre unas milésimas de segundo, por se descartó esta diferencia para el análisis realizado. Esto se da en parte porque los tiempos son tan cortos, que las diferencias son imperceptibles.

Por el contrario, para las heurísticas propuestas, los tiempos de ejecución son menores cuando la holgura de las capacidades de producción incrementan. Esto se debe a que, cuando la capacidad de las aristas es baja en comparación con la demanda, se van a producir más bifurcaciones. Cada una de esas bifurcaciones implicará un nuevo cálculo de camino mas corto (algoritmo de Dijkstra), sumándole el tiempo que esto significa.

En el caso en que la holgura sea mayor, se pueden satisfacer varias demandas con un único período. Y por lo tanto se precisará en lo general únicamente un cálculo de camino por período, haciendo que los tiempos de ejecución sean sensiblemente más cortos.

En cuanto a las distintas variantes, las que más se ven afectadas son la heurística Base y Diagonal. Esto se debe a que estas son las que tratan de buscar la solución más exacta. Por otro lado la variante de Franjas hace los cálculos del algoritmo Dijkstra únicamente comprendido en la franja el la que se esta considerando, obviando todos los períodos previos y de esta forma el sobre trabajo es menor.

5.3.4. Costos de Inventario

Otro de los criterios que se seleccionó para clasificar las instancias fueron los costos de inventario y su evolución. En la Figura 26 se grafica la desviación de todas las heurísticas implementadas según el incremento de inventario. Como fue explicado en el Capitulo 3 existen 3 categorías: i) 1 a 3, ii) 4 a 5 y iii) 6 a 7. Siendo la primera categoría donde los

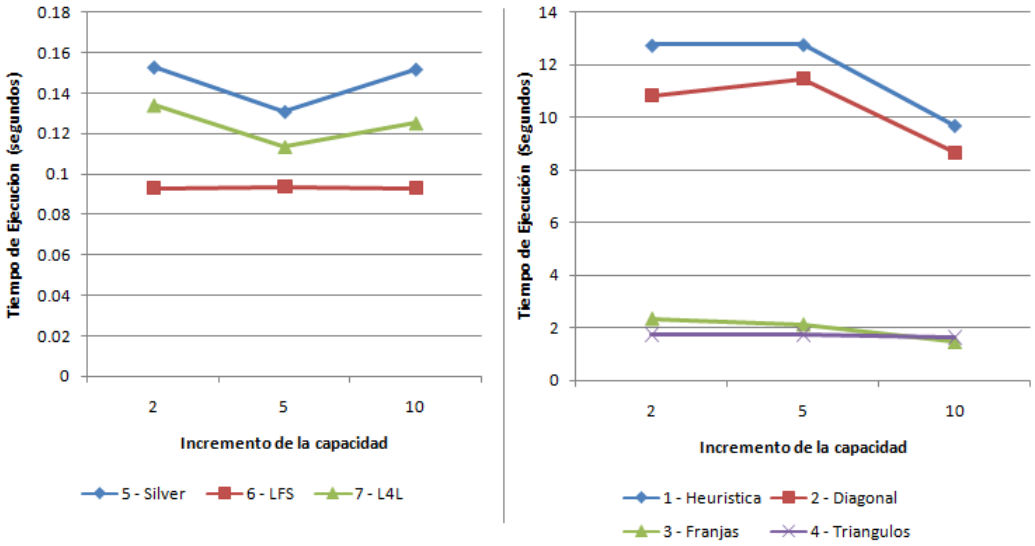

Figura 25: Tiempo de ejecución de las heurísticas implementadas según incremento de capacidad.

costos de inventario son menores y la última donde los costos de inventario son mayores.

Se observa que para todas las heurísticas implementadas (menos para Lote de Tamaño Fijo) cuando el costo de inventario aumenta los resultados que se obtienen son más exactos en comparación con lo retornado por el algoritmo de GLPK. Esto se debe a que cuanto más barato es el inventario, más opciones cercanas al óptimo existen, y por lo tanto la probabilidad de aumentar la desviación es mayor. Por el contrario si almacenar producto insume un gran costo, la producción, tendrá que dividirse en varios períodos para evitarlos y por lo tanto se acotan las soluciones posibles, aumentando la probabilidad de acercarse al óptimo.

Nuevamente un ejemplo claro para ver esto es con la heurística de Lote por Lote. Cuando el precio del inventario es muy alto, el óptimo tenderá a producir el producto para satisfacer la demanda en el mismo período. Esto quiere decir que Lote por Lote se acercará más a la solución exacta. Por otro lado, cuando el inventario es barato, producir siempre en el mismo período no es la única solución posible, degradándose el resultado de Lote por Lote.

La única heurística que se ve perjudicada con el aumento del inventario es la de Lote de Tamaño Fijo. Esto se debe a que acumula inventario innecesario, es decir cuando la heurística termina, generalmente lo hace con inventario remanente. Cuando inventariar se vuelve un costo relevante, esta acumulación de producto tiene un gran impacto en el resultado final.

5.3.5. Costos de Preparación

Los costos de preparación juegan un papel muy importante a la hora de hallar el óptimo. Éstos normalmente son mayores que los costos unitarios de producción e inventario. Por esto se definieron como uno de los criterios para separar las instancias. En particular se definieron 3 rangos distintos, donde el primero establece costos de preparación entre 80 y 200, el segundo entre 201 y 400 y el tercero entre 401 y 700. En el gráfico de la Figura 27 se ve el desempeño de las heurísticas considerando su desviación, según las tres clasifica-

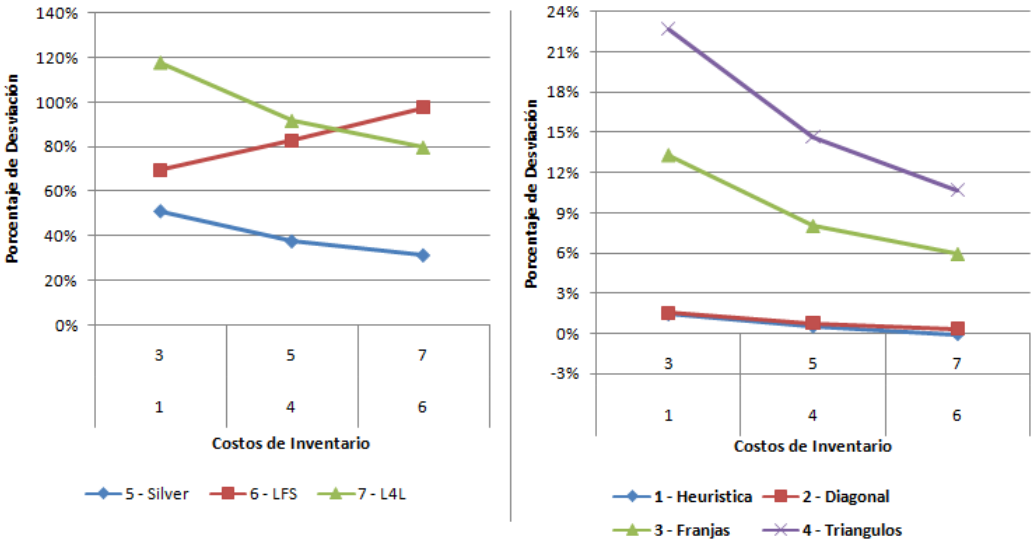

Figura 26: Desviación de las heurísticas implementadas según criterio de costo de inventario.

ciones mencionadas.

El costo de preparación tiene un efecto directo sobre el porcentaje de desviación para todas las heurísticas. Cuando los costos de preparación son elevados las heurísticas implementadas (menos la de Lote de Tamaño Fijo) muestran un deterioro en la calidad de su solución. Esto se debe a que desviarse de la solución óptima implica asumir grandes costos. Por lo contrario si los costos de preparación son bajos, el error implica menor desviación.

Nuevamente tomando el caso de Lote por Lote se ve que los costos de preparación tienen un resultado devastador, duplicándose la desviación cuando aumentan los costos. Obviamente esto es producto de la decisión de producir en todo momento, sin inventariar.

El único caso en que la tendencia es la contraria es en el caso de Lote de Tamaño Fijo, donde al contrario que en el caso de los costos de inventario, al tratar maximizar las producciones, cuando los costos de preparación son altos, los costos de inventario se vuelven mínimos, y por tanto el sobre costo de inventariar en exceso no implica una gran diferencia de costo.

5.3.6. Desviación Estándar

Otra medición interesante es la dispersión de los resultados según la heurística. En la Figura 28 se muestra un gráfico con la desviación estándar general para cada uno de los métodos utilizados. El resultado de este análisis indica que la heurística base al igual que la variante Diagonal, son métodos estables conservándose debajo del 5 %. Esto indica que estas heurísticas a parte de brindar resultados cercanos al óptimo lo hacen de forma consistente. No así las otras dos variantes propuestas que muestran una dispersión cercana al 10 %.

Es importante destacar que se trabajó en este punto para poder mejorar esta medición. En las primeras versiones de la heurística, aunque los resultados eran en lo general cercanos

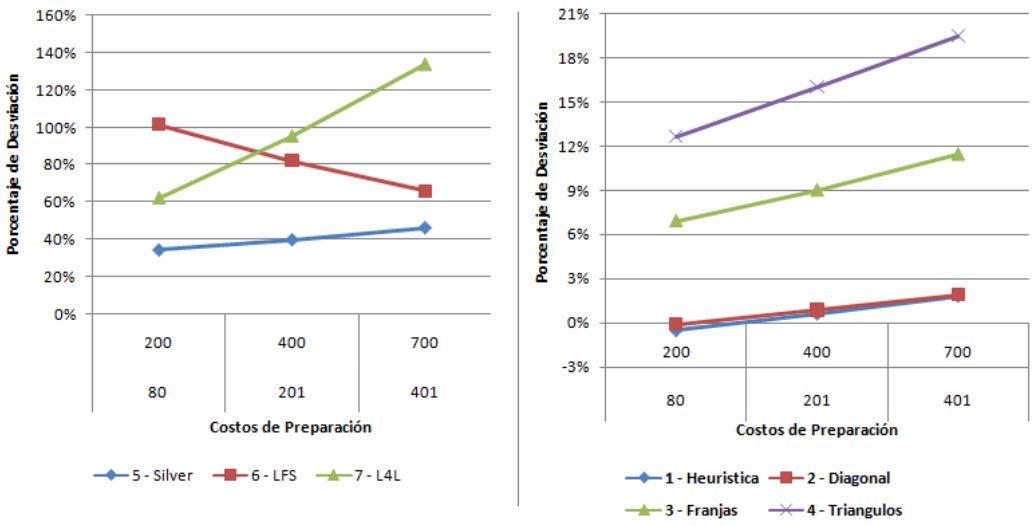

Figura 27: Desviación de las heurísticas implementadas según criterio de costo preparación.

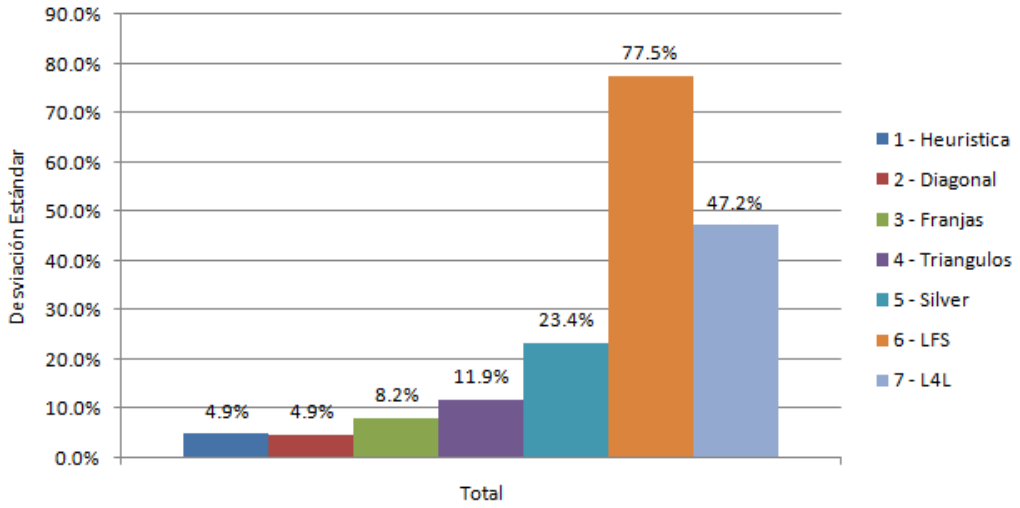

Figura 28: Desviación Estándar medido para todas las heurísticas.

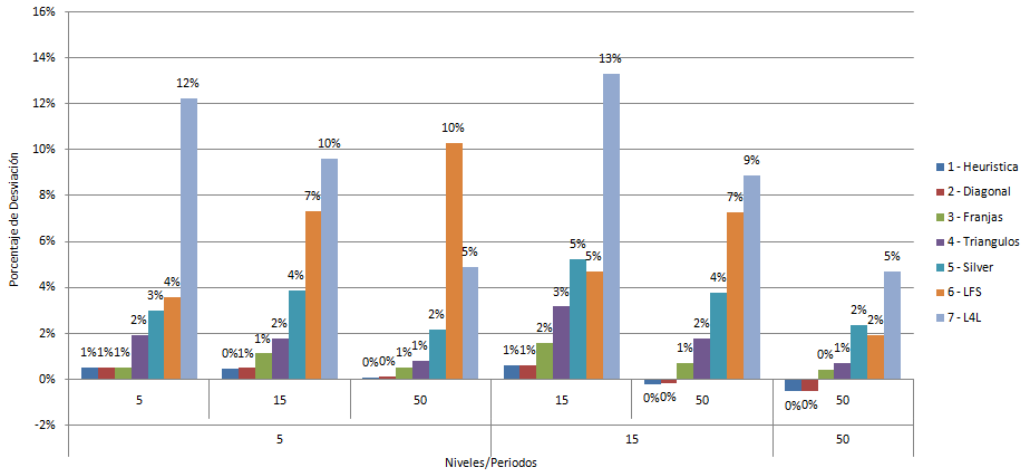

Figura 29: Optimalidad de las heurísticas comparados con los resultados de Greedy y el algoritmo de GLPK

al óptimo, habían casos en que llegaban a estar cerca del 40 % de desviación. Esto hacia que el promedio se mantuviera bajo, cercano al 5 % pero la desviación estándar superará las dos cifras.

Analizados estos casos especiales se tomaron medidas para evitar esta desviación. Ejecutando siempre el algoritmo Dijkstra desde los vértices de demanda, se asegura que los caminos más cortos se calculen desde el inicio para cada uno de los períodos. Esto evita que se tomen decisiones intermedias que puedan descartar caminos óptimos generando grandes cambios en los costos finales.

Por último, sobre la estabilidad de las heurísticas conocidas adaptadas, se ve que es muy variable. Particularmente la dispersión de los datos obtenidos para el caso de lote fijo supera el 77 % lo que lo deja muy detrás de los otros métodos en competencia.

5.4. Porcentaje de Optimalidad

Como se mencionó en la introducción, se considerará una medida adicional para establecer el desempeño de las heurísticas. El porcentaje de optimalidad nos permite saber donde se encuentran las soluciones propuestas dentro del rango de costos posibles para una instancia. Considerando el resultado del algoritmo de GLPK como limite inferior(LB), y Greedy como limite superior(UB), la misma esta dada por:

$$OP = \frac{HC - LB}{UB - LB}$$

En el gráfico de la Figura 29 se ve cuan cerca del óptimo se encuentran las heurísticas, considerando la amplitud del dominio de soluciones posibles.
En términos generales se observa que los porcentajes de optimalidad son reducidos en comparación con los de desviación, lo que significa claramente que todas las heurísticas buscan con menor o mayor éxito el resultado óptimo.

Encontrar en el contexto de 50 niveles y 50 períodos a Lote por Lote bajo el 1 %, da crédito de lo significativo que es producir solo lo exacto, ya que el porcentaje de desviación de

esta heurística es de 32.1 %. Esta conclusión no va en perjuicio del resultado sobresaliente de Lote de Tamaño Fijo, heurística que consigue una optimalidad al menos 2 veces mejor y si presenta inventario final. Lote de Tamaño Fijo logra pasar de 11.9 % a menos de 0.5 % por activar la menor cantidad posible de preparaciones.

Por el lado de las heurísticas propuestas, se puede asegurar que encontrarse a una desviación de ±3 % equivale a una optimalidad de ±0.2 %, lo que refuerza la calidad de los resultados. O sea, es posible tener una desviación de 1 % (por ejemplo), pero no necesariamente significa contar con buen resultado. Si el dominio de soluciones va de $10000 a $10100, entonces incluso el peor resultado tiene una desviación de 1 %.

Justamente, la situación es opuesta con los resultados obtenidos en 50 niveles y 50 períodos, ya que es cuando mayor es el dominio de soluciones, yendo del resultado promedio de Greedy $8802464 a el resultado promedio del algoritmo de GLPK $553149 y por esto cobra más importancia los gráficos de desviación anteriormente analizados en esta sección.

Siendo que las heurísticas son más requeridas a mayor tamaño las instancias, se centra el análisis en las instancias de 50 niveles y 50 períodos, pero igualmente se aprecia estabilidad en las optimalidades de las instancias más chicas por parte de las heurísticas propuestas. No así los resultados de Lote por Lote y Lote de Tamaño fijo las cuales para instancias menores tan solo se aproximan al óptimo un 10 %.

5.5. Evolución del MIP-GAP

El MIP-GAP es parte de la salida de la ejecución de GLPK, expresada como porcentaje de la solución encontrada. Representa el menor rango donde se asegura que se encuentra la solución óptima, un MIP-GAP igual a cero equivale a la solución óptima. Cuando GLPK se ve obligado a interrumpir la resolución del problema por la condición impuesta de tiempo límite, la solución deja de ser óptima y por tanto el GAP deja de ser 0 %.

Las siguientes gráficas complementan el análisis realizado, ya que todos ellos se basan en el algoritmo de GLPK. Y dicho análisis no pueden desprenderse de un estudio del GAP, (indicador de la calidad del resultado por el método de programación lineal entera mixta).

El gráfico de la Figura 30 ilustra el comportamiento del tiempo de ejecución en segundos a medida que aumentan la cantidad de vértices de la instancia. En el gráfico de la Figura 31 se aprecia como en instancias con 251 vértices el GAP ya es superior al 15 %, quizás aún mejor que la heurística Base que se desvía solo 0.6 % del algoritmo de GLPK. Pero ya en instancias con 751 vértices el gap supera 20 % y la heurística Base mejora su valor 3.4 %.

Por esta razón, la porción de datos recogidos más sólida para sacar conclusiones son todas las instancias menos las de 2501 vértices, ya que las mismas presentan un gap excesivo de casi 80 %. Desestimar la mejora en costo de las heurísticas propuestas en este conjunto de instancias no es significativo ya que una mejora semejante fue obtenida ya con instancias de 751 vértices.

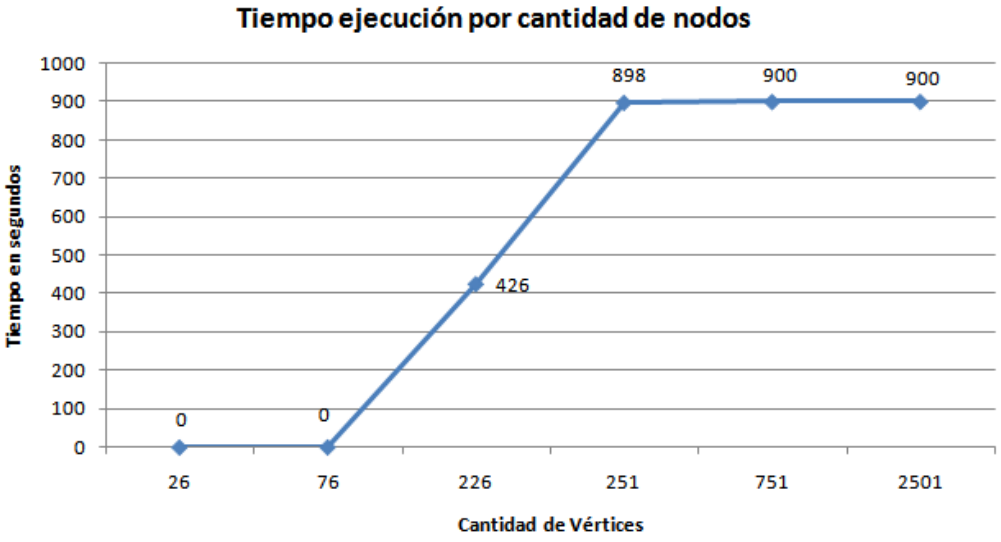

Figura 30: Promedio de tiempos de ejecución de GLPK

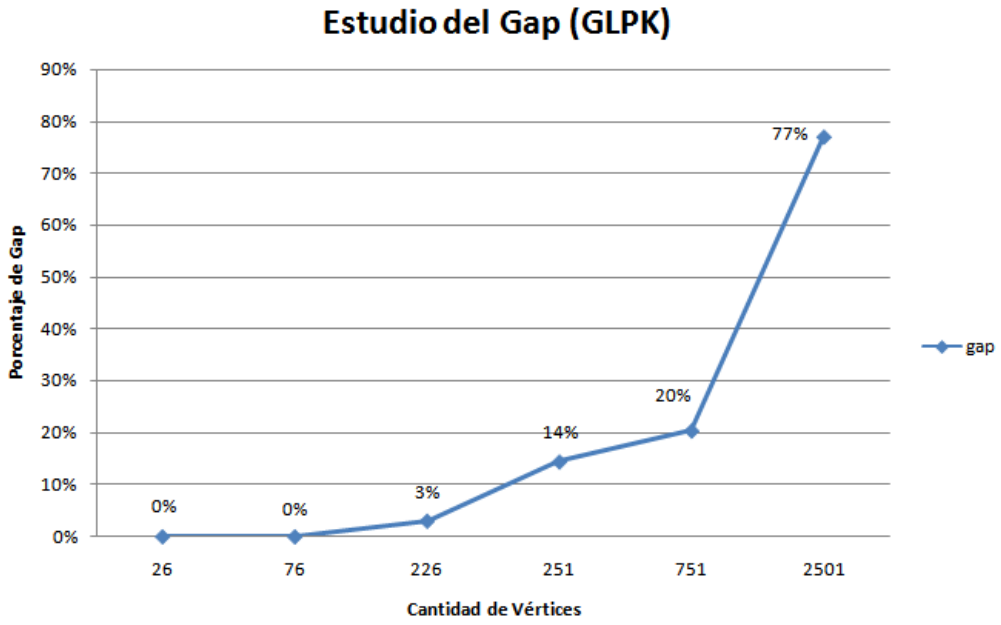

Figura 31: Promedio de tiempos de ejecución de GLPK

6. Conclusiones

El proyecto abordó un problema de planificación de la producción conocido como problema de dimensionamiento de lote con restricciones de capacidad y múltiples niveles. El mismo fue planteado con un modelo matemático para el cual se tomo como base el de otros autores en la materia. Siendo este un problema NP-hard, se implementaron heurísticas que pudieran brindar resultados cercanos al óptimo en tiempos de ejecución bajos. Dentro del grupo de heurísticas implementadas se desarrollaron tres heurísticas conocidas (Lote por Lote, Lote de Tamaño Fijo y Silver-Meal) las cuales fueron adaptadas al problema en cuestión, y cuatro heurísticas propias basadas en el algoritmo de Dijkstra (Heurística Base, Triángulos, Franjas y Diagonal).

Para probar el desempeño de las mismas se implementó un sistema capaz de manipular un gran conjunto de instancias y la ejecución de ellas. Tomando como referencia el solver de GLPK se realizó la comparativa entre las heurísticas. El resultado obtenido por éste se utiliza para calcular la desviación de las heurísticas implementadas. A su vez, los tiempos de ejecución fueron también utilizados como medición comparativa.

Obtenidos los resultados, se observaron tiempos de ejecución bajos en las heurísticas. Únicamente en el caso de la Heurística Base y Diagonal, para aquellas instancias de tamaños significativos (50 niveles y 50 períodos) se ven tiempos de resolución mayores a los 10 segundos. Considerando que para muchas de estas instancias GLPK no le bastan 30 horas para retornar una solución, se entiende que los tiempos de ejecución logrados fueron satisfactorios.

En lo referente al costo de solución, tanto la Heurística Base como Diagonal proporcionaron soluciones siempre con una desviación menor al 5 %. Incluso para las instancias de mayor tamaño en las cuales GLPK se tomo 15 minutos para devolver una solución, estas dos heurísticas devolvieron resultados aún mejores que las provistas por la herramienta. Considerando además, que demostraron estabilidad numérica presentando una desviación estándar debajo del 5 %, se concluye que éstas devuelven resultados de buena calidad. En el caso de las heurísticas Franjas y Triángulos, aunque sus desviaciones sean superiores, demostraron ser alternativas aceptables con tiempos de ejecución muy bajos, tal cual fueron diseñadas.

Tomando en cuenta la complejidad del problema, la heurística Base y sus tres variantes devolvieron resultados que se distinguen de las heurísticas conocidas adaptadas y además lograron una desviación baja respecto al algoritmo de GLPK. Las heurísticas Base y Diagonal superaron al algoritmo de GLPK en las instancias más grandes y compitieron con buena precisión y tiempo en las demás. Específicamente la Heurística Base se destaca sobre Diagonal, ya que a pesar de que esta última solucione las instancias grandes en casi 10 segundos menos, su desviación empeora 0.4 %.

En cuanto a la selección del algoritmo de Dijkstra, se concluye que fue una alternativa buena para la resolución de los problemas de dimensionamiento de lote por su bajo orden de ejecución. Como a su vez la resolución por las heurísticas propuestas requirieron una cantidad acotada de ejecuciones del algoritmo Dijkstra, los tiempos de ejecución fueron bajos teniendo en cuenta la dificultad del problema. Igualmente se intentó bajar aún más los tiempos de ejecución, implementando alternativas como Triangulo y Franjas los cuales se enfocaron en trabajar sobre grafos de menor tamaño y/o reducir la cantidad de veces que el algoritmo se ejecuta, presentando este último un orden

Existen varios factores que afectan tanto los tiempos de resolución como la calidad de los

resultados. La holgura de las capacidades de producción es uno que tiene gran impacto sobre los tiempos. La resolución de las instancias con menor holgura toma más tiempo que aquellas que disponen de mayor capacidad. A su vez, cuanto mayor es esta holgura, mejores serán los resultados devueltos (en particular de la heurística Base). También se mostró como los costos de inventario y preparación juegan un papel importante en la calidad de los resultados de las heurísticas, cuanto mayor sean los costos de inventario y menor los costos de preparación mejor es el resultado.

Como trabajo a futuro, se puede mejorar la flexibilidad del software, logrando que cargue heurísticas a través de librerías externas. Alineado con esto, sería interesante implementar más heurísticas, conocidas y otras propuestas basadas en otros algoritmos de caminos más cortos. El uso de post-optimización también resulta una alternativa interesante, aunque requiere un amplio estudio para entender el beneficio potencial contra el tiempo extra de resolución que significa.

Bibliografía

[1] R. K. Ahuja, K. Mehlhorn, J. Orlin, and R. E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM (JACM)*, 37(2):213–223, 1990.

[2] E. Berk, A. Ö. Toy, and Ö. Hazır. Single item lot-sizing problem for a warm/-cold process with immediate lost sales. *European Journal of Operational Research*, 187(3):1251–1267, 2008.

[3] G. R. Bitran and H. H. Yanasse. Computational complexity of the capacitated lot size problem. *Management Science*, 28(10):1174–1186, 1982.

[4] G. J. Burke, J. Carrillo, and A. J. Vakharia. Heuristics for sourcing from multiple suppliers with alternative quantity discounts. *European Journal of Operational Research*, 186(1):317–329, 2008.

[5] C.-S. Chung and C.-H. M. Lin. An o (t 2) algorithm for the ni/g/ni/nd capacitated lot size problem. *Management Science*, 34(3):420–426, 1988.

[6] M. Florian and M. Klein. Deterministic production planning with concave costs and capacity constraints. *Management Science*, 18(1):12–20, 1971.

[7] M. Florian, J. K. Lenstra, and A. Rinnooy Kan. Deterministic production planning: Algorithms and complexity. *Management science*, 26(7):669–679, 1980.

[8] F. W. Harris. How many parts to make at once. 1913.

[9] P. Kaminsky and D. Simchi-Levi. Production and distribution lot sizing in a two stage supply chain. *IIE Transactions*, 35(11):1065–1075, 2003.

[10] M. Leyzorek, R. Gray, A. Johnson, W. Ladew, S. Meaker Jr, R. Petry, and R. Seitz. Investigation of model techniques–first annual report–6 june 1956–1 july 1957–a study of model techniques for communication systems. *Case Institute of Technology, Cleveland, Ohio*, 1957.

[11] A. Manikas, Y.-L. Chang, and M. Ferguson. Bluelinx can benefit from innovative inventory management methods for commodity forward buys. *Omega*, 37(3):545–554, 2009.

[12] A. Manne. *Investments for Capacity Expansion: Size, Location, and Time- Phasing*. Studies in the economic development of India. M.I.T. Press, 1966.

[13] R. A. Melo and L. A. Wolsey. Mip formulations and heuristics for two-level production-transportation problems. *Computers & Operations Research*, 39(11):2776–2786, 2012.

[14] Y. Pochet and L. A. Wolsey. *Production planning by mixed integer programming*. Springer Science & Business Media, 2006.

[15] R. Rugina and M. Rinard. Recursion unrolling for divide and conquer programs. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 34–48. Springer, 2000.

[16] F. Z. Sargut and H. E. Romeijn. Capacitated production and subcontracting in a serial supply chain. *IIE Transactions*, 39(11):1031–1043, 2007.

[17] E. A. Silver and H. Meal. A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. *Production and inventory management*, 14(2):64–74, 1973.

[18] A. Tipoldi and E. Vázquez. Estado del arte - tesis de grado - gestión de inventario en múltiples niveles. 2017.

[19] W. Van den Heuvel and A. P. Wagelmans. An efficient dynamic programming algorithm for a special case of the capacitated lot-sizing problem. *Computers & operations research*, 33(12):3583–3599, 2006.

[20] C. Van Hoesel and A. P. M. Wagelmans. An o (t 3) algorithm for the economic lot-sizing problem with constant capacities. *Management Science*, 42(1):142–150, 1996.

[21] S. Van Hoesel, H. E. Romeijn, D. R. Morales, and A. P. Wagelmans. Integrated lot sizing in serial supply chains with production capacities. *Management Science*, 51(11):1706–1719, 2005.

[22] S. Van Hoesel, H. E. Romeijn, D. Romero Morales, and A. P. Wagelmans. Polynomial time algorithms for some multi-level lot-sizing problems with production capacities. 2002.

[23] A. Veinott Jr. Production planning with concave costs. *unpublished class notes, Stanford University, Stanford, California*, 1963.

[24] H. M. Wagner and T. M. Whitin. Dynamic version of the economic lot size model. *Management science*, 5(1):89–96, 1958.

[25] E. Zabel. Some generalizations of an inventory planning horizon theorem. *Management Science*, 10(3):465–471, 1964.

[26] W. I. Zangwill. A deterministic multi-period production scheduling model with backlogging. *Management Science*, 13(1):105–119, 1966.

[27] W. I. Zangwill. A backlogging model and a multi-echelon model of a dynamic economic lot size production system-a network approach. *Management Science*, 15(9):506–527, 1969.

Anexos

Anexo A. Ejemplo de Instancia

El siguiente ejemplo corresponde a una instancia de cinco niveles y cinco períodos, sus parámetros en orden son: cantidad de períodos ($maxT$) y niveles ($maxN$), demandas (d), capacidades de producción (b), costos de producción (c), costos de inventario (h) y costos de preparación (k).

```
param maxT := 5;
param maxN := 5;

param: d:=
1 7
2 9
3 19
4 19
5 12
;

param b :1 2 3 4 5 :=
1 34 28 22 21 7
2 35 32 28 24 9
3 32 31 25 23 19
4 30 27 27 21 19
5 26 26 26 25 12
;

param c :1 2 3 4 5 :=
1 8.26660530328127 12.2563475084443 41.526896684055 62.9921353994832 0
2 2.77572104044991 18.0987336008959 36.9296268437041 67.6098393986979 0
3 4.28748793261474 20.0532078875166 36.4822106897593 31.8356304275038 0
4 2.02515156801099 4.97684258423288 31.9767599332349 72.1945123082001 0
5 5.14241529309303 22.7723713270881 18.2067734980366 60.1306639821877 0
;

param h :1 2 3 4 5 :=
1 0 2.61787829809723 4.51542942389633 7.71472489773982 8.06408745286245
2 0 2.14686095022916 5.95222106573741 8.50555866933687 10.1683565057667
3 0 2.70046190717279 5.87407432630382 7.30956474612912 8.36166808072555
4 0 1.82800494359248 5.45092968151482 5.96012892805046 2.34079138438254
5 0 0 0 0 0
;

param k :1 2 3 4 5 :=
1 428.287399876065 463.919462365061 475.208756903749 440.689999283613 0
2 687.563620505651 639.636503636668 506.626513666299 411.039543548617 0
3 418.767098181773 478.802871480958 686.527782497242 663.08917863578 0
4 540.139900879068 507.46858107367 406.983773911364 650.953237186164 0
5 418.393327653126 456.254095471582 616.47638179384 432.190096637788 0
;
```

Esta instancia es producto de todos los parámetros descritos anteriormente (en la repre-

sentación anterior no se incluyen los metadatos diseñados para facilitar su gestión en el software). Como se puede observar, naturalmente, las demandas y capacidades de producción son números enteros, no así los costos.

Anexo B. Pseudocódigo de Heurísitcas

En este anexo se muestra el pseudocódigo más significativo de cada heurística.

B.1. Lote por Lote

Muy sencillo en su implementación, Lote por Lote establece la producción de un nivel como la colección ordenada de demandas.

```
function
CalcularProduccionesL4L(demandas, nivel) {
  return demandas
}
```

B.2. Lote de Tamaño Fijo

Lote de Tamaño Fijo busca iterativamente satisfacer la demanda con la menor cantidad de producción posible.

```
function
CalcularProduccionesLFS(demandas, nivel) {
    integer set capacidades      <- GetCapacidades(nivel)
    integer     maxT             <- Contar(demadas)
    integer     capacidadTotal   <- SumaPeriodos(maxT, capacidades)
    integer     demandaTotal     <- SumaPeriodos(maxT, demandas)
    integer     cantProducciones <- 1
    integer set produccion
    integer set resultado
    resultado[1]                  <- demandaTotal
  while not EsFactible(capacidades, demandas, resultado) {
    cantProducciones <- cantProducciones + 1
    if cantProducciones > maxT {
      cantProducciones <- 1
      demandaTotal     <- demandaTotal + 1
      if demandaTotal > capacidadesTotal {
              break //Las capacidades no permiten satisfacer las demandas."
      }
    }
    produccion <- ParteEntera(demandaTotal / cantProducciones)
    for t = 1 to maxT + 1 {
          if t > cantProducciones
              resultado[t] <- 0
            else
              resultado[t] <- produccion
    }
  }
  return resultado
}

function
EsFactible(capacidades, demandas, resultado) {
  integer maxT <- Largo(demadas)
  for t = 1 to maxT + 1 {
      if capacidades[t] < resultado[t]
          return False  //Capacidades insuficientes
        if SumaPeriodos(t, demada) > SumaPeriodos(t, resultado)
          return False  //Produccion no satisface en tiempo y cantidad
        t <- t + 1
    }
    return True
}
```

B.3. Silver-Meal

Iterativamente Silver-Meal acumula producciones en un período hasta que se cumple la condición que el costo de inventario supera el costo de activar un nueva producción o se alcanzó la capacidad de producción.

```
function
CalcularProduccionesSM(demandas, nivel) {
  integer set periodosDeProduccion <- GetPeriodosAProducir(demandas, nivel)
  integer set cuantoProducir
  integer     maxT                 <- Largo(demandas)
  integer     acumuladoDemanda     <- 0
  integer     proximaProduccion
  integer set periodosMayoresOrdenados
  for t = 0 to maxT + 1 {
    if periodosDeProduccion contiene t {
      periodosMayoresOrdenados <- FiltrarMayores(t, periodosDeProduccion)
      if periodos vacio
        periodoProximaProduccion <- maxT + 1
      else
        periodoProximaProduccion <- periodosMayoresOrdenados[0]

      for tt = t to periodoProximaProduccion
        acumuladoDemanda <- acumuladoDemanda + demandas[tt]

      cuantoProducir[t] <- acumuladoDemanda
      acumuladoDemanda  <- 0
    }
    else
      cuantoProducir[t] <- 0
  }
  return cuantoProducir
}

function
GetPeriodosAProducir(demandas, nivel) {
  double  set costosDePreparacion   <- GetCostosPreparacion(nivel)
  double  set costosDeInventario    <- GetCostosInventario(nivel)
  double  set costosSM
  integer set periodosDeProduccion
  integer set capacidadesProduccion <- GetCapacidadesProduccion(nivel)
  costosSM[1]                       <- costosDePreparacion[1]
  periodosDeProduccion.Add(1)
  double  valorAnterior
  double  numero
  integer capacidadActual
  integer requerido
  integer ultimaProduccion
  for t = 2 to maxT + 1 {
    if periodosDeProduccion contiene t - 1
      valorAnterior <- costosSM[t - 1]
    else
      valorAnterior <- costosSM[t - 1] * t
    numero          <- (
      valorAnterior + (demandas[t] * t * costosDeInventario[t])
    ) / (t + 1)
    ultimaProduccion <- periodosDeProduccion[Largo(periodosDeProduccion) -
    1]
    capacidadActual  <- capacidadesProduccion[ultimaProduccion]
    requerido        <- 0
    for tt = ultimaProduccion to t
      requerido <- requerido + demandas[tt]
    if numero >= costosSM[t - 1] o capacidadActual < requerido {
      costosSM[t] <- costosDePreparacion[t]
      periodosDeProduccion.Add(t)
    }
    else
      costosSM[t] <- numero
```

```
  }
  return periodosDeProduccion
}
```

B.4. Greedy

La producción en cada período es establecida por esta implementación de Greedy como el mínimo entre la cantidad de producto recibido por el nivel superior y la actual capacidad de producción.

```
function
CalcularProduccionesGreedy(demandasNivelSuperior, nivel)
{
  integer set capacidadesProduccion <- GetCapacidadesProduccion()
  integer set resultado
  integer     maxT                  <- Largo(demandasNivelSuperior)
  for t = 1 to maxT + 1
    resultado[t] <- Minimo(demandaNivelSuperior[t], capacidadesProduccion[t
    ])
  return resultado
}
```

B.5. Heurística Base

El método itera por todos los períodos calculando los caminos óptimos para satisfacer la demanda de cada uno de los períodos. Posicionado en un período, busca el camino más corto desde el vértice fuente al vértice de demanda de ese período por medio del algoritmo de dijkstra.

Una vez calculado el camino más corto para satisfacer la demanda, se llama al método ActualizarHolgurasCamino, el cual verificara que la capacidad del camino encontrado sea suficiente para producir lo requerido. En el caso contrario se producirá el máximo producto posible por el camino seleccionado, y se procederá a buscar el próximo mejor camino para producir el resto de la demanda.

Este proceso se realiza para cada uno de los períodos de forma iterativa hasta completar la demanda.

```
function
CacluloPorDemanda(grafo, costoFinal, nextNode)
{
  foreach (verticeDemanda in grafo) {
    boolean termino <- false
    integer demanda <- verticeDemanda.Demanda
      while (!termino) {
        integer set caminoDijkstra <- grafo.EjecutarDijkstra(verticeDemanda
    , demanda)
        //calcula el mejor camino, y lo almacena en array camino
        nextNode <- caminoDijkstra[verticeDemanda]
        termino  <- grafo.ActualizarHolgurasCamino(nextNode, verticeDemanda
    , demanda, costoFinal, caminoDijkstra)
              //si termino=false, el metodo devuelve en demanda el remanente
    a ser producido.
      }
  }
}
```

El método siguiente establece la capacidad del camino, y calcula los costos de satisfacer la demanda del período. En caso de que la capacidad del camino no sea suficiente para satisfacer la demanda, retorna un false, indicándole al método invocador que calcule otro

camino para el producto restante. Cuando la capacidad de la arista queda colmada, se borra del grafo para que no se la considere en el próximo cálculo de camino más corto.

```
function
ActualizarHolgurasCamino(origen, destino, demanda, costoTotal, camino)
{
  boolean ok              <- true
  integer seProduce       <- 0
  integer capacidadMaxima <- capacidadMaximaDeCaminoAlOrigen(destino,
    camino)
  if (capacidadMaxima >= demanda) {
    ok        <- true
    seProduce <- demanda
  }
  else {
    demanda    <- demanda - capacidadMaxima
    seProduce  <- capacidadMaxima
    ok         <- false
  }

  integer next
  integer current <- destino
  while (current != origen) {
    next                          <- camino[current]
    Arista  aristaAux             <- ObtenerArista(next, current)
    boolean primeravez            <- aristaAux.CantidadProducida == 0
    aristaAux.CantidadProducida   <- aristaAux.CantidadProducida +
    seProduce
    aristaAux.CapacidadProduccion <- aristaAux.CapacidadProduccion -
    seProduce
    if (aristaAux.capacidadProduccion <= 0) {
      BorrarArista(next, current)
    }
    if (primeravez) {
      costoTotal <- costoTotal + (seproduce * aristaAux.Costo) + aristaAux.
    CostoSetUP
    }
    else {
      costoTotal <- costoTotal + seProduce * aristaAux.Costo
    }
    current <- next
  }
   return ok
}
```

B.6. Variante Triángulos

El método siguiente es invocado por la variante triángulos para eliminar las aristas de los períodos anteriores en los que no haya habido producción. Este obtiene todas las aristas que tienen como origen o destino el período en el que se esta calculando el camino, y borra aquellas que no han sido utilizadas.

```
function
BorrarAristasTriangulo(vDemanda)
{
  Aristas set aristas         <- ObtenerAristasOrigenODestinoSinProduccion(
    vDemanda.Periodo);
  Vertice set verticesABorrar <- SeleccionarVerticesOrigen(aristas)
  foreach (vertice in verticesABorrar) {
    BorrarArista(BuscarAristasOrigenODestino(vertice))
  }
}
```

B.7. Variante Franjas

El método mostrado abajo borra las aristas previas al período pasado por parámetro para generar una nueva franja. Éste es invocado cada 10 períodos, o cuando se desee.

```
function
BorrarAristasFranjas(periodo)
{
  Arista set aristas <- obtenerAristasDePeriodosPrevios(periodo)
  foreach (arista in aristas) {
    BorrarArista(arista)
  }
}
```

B.8. Variante Diagonal

Debajo se puede ver el código que elimina el triangulo superior. En el mismo define un ïncremento"que se usa como multiplicador para que el triangulo varíe su tamaño según la matriz sea cuadrada o no. La proporción del triangulo en comparación con el grafo aumenta si el problema tiene gran cantidad de períodos o niveles.

```
function
BorrarDiagonalFinal(maxPeriodo, maxNivel, grafo)
{
  double incremento <- maxPeriodo / maxNivel
  double capPeriodo <- maxPeriodo / 3
  // Para aquellas instancias con mas de 30 niveles el triangulo se toma
    mas grande
  if (maxNivel > 30) {
    capPeriodo <- capPeriodo * 2
  }
  //El metodo itera sobre el conjunto de aristas y aplica la condicion,
    donde a es canda arista del conjunto.
  Arista set aristas <- grafo.ObtenerAristasCondicion(maxPeriodo - a.
    Destino.Periodo + (incremento * a.Destino.Nivel) < capPeriodo)
  foreach (arista in aristas) {
    BorrarArista(arista)
  }
}
```