UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

INGENIERIA EN SISTEMAS – INVESTIGACIÓN DE OPERACIONES

OPTIMIZACIÓN DE LA CADENA DE SUMINISTRO A TRAVÉS DEL PROBLEMA DE LA PROGRAMACIÓN DE LA PRODUCCIÓN Y RUTEO

TESIS QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA EN SISTEMAS – INVESTIGACIÓN DE OPERACIONES

PRESENTA:

Juan Pablo Cisneros Castañeda

TUTORA:

Dra. Idalia Flores de la Mota

MÉXICO, Ciudad de México, abril 2023

Dedicatoria

A mis padres, Ivonne y Jorge que con su arduo trabajo y ejemplo me dieron las oportunidades de vida para poder llegar al punto donde me encuentro hoy.

A Lucy, mi compañera de vida, quién me acompañó en este programa como novia, prometida y esposa, sin su apoyo y compañía no podría haber logrado esta meta y juntos seguiremos construyendo vida.

Agradecimientos

Primeramente, agradezco a la Dra. Idalia Flores de la Mota quien me llenó de conocimiento, herramientas y empoderamiento para ser capaz de encontrar las respuestas por mi propia cuenta. Ella representa el ejemplo de enseñanza que nunca debe ser forzado y más bien nace a través de la curiosidad y el conocimiento como fin último y suficiente.

Además, agradezco a Carmen, Oro y Adri que sus valiosos consejos en las sesiones de tutoría ayudaron a pulir el presente trabajo.

También a todos mis compañeros con los que compartí este programa por la bella amistad que se formó a lo largo de dos años.

Por último, a la máxima casa de estudios del país de la cual ahora soy parte y me sentiré orgulloso de representar caminando hacia adelante.

Contenido

Resumen

En este trabajo de investigación se aborda el problema de producción-ruteo (PRP) desde una visión sistemática. Así pues, se hace una revisión extensa de literatura con respecto a la coordinación de la cadena de suministro, especialmente en las de Inventario Administrado por Proveedor (VMI), donde destacan los bienes de consumo.

En la parte central se propone un modelo de optimización para la programación de la producción y la secuenciación dentro de la fábrica también llamado *scheduling*. Adicionalmente, se resuelve una instancia amplia a través de una metodología separada en dos fases que implementa en software de actualidad el modelo mencionado junto con el planteamiento del problema de ruteo de vehículos (VRP).

Con lo cual se busca determinar un modelo para resolver el problema de producción-ruteo (PRP) que busque la optimización de una cadena de suministro de tipo VMI, con una red de manufactura multiplanta y multiproducto hasta su entrega a los almacenes, a través de la minimización de costos de producción (por planta y SKU), así como los costos de transporte, con un enfoque táctico. Dicho modelo deberá resolver el flow shop scheduling por planta, es decir, considerar los niveles o etapas por los que tiene que pasar cada producto para su manufactura en las cuales se asigna a máquinas con capacidad limitada con tiempo de ciclo en función del producto, además, tomar en cuenta el tiempo de ajuste en cada máquina de cambiar de un producto a otro con una matriz variable desde-hacia con un método de resolución que sea confiable y de tiempo de cómputo razonable para realizar una comparativa de los métodos de resolución aplicando el modelo a diferentes instancias variando la razón de los costos de manufactura contra los de transporte, u otro indicador que sea clave para la comparativa.

Introducción

A lo largo de los años, las compañías han tenido una ardua búsqueda por posicionar sus bienes de consumo al alcance de la población en general. Con la evolución de la manufactura moderna vino el establecimiento y la administración de las cadenas de suministro (Mourtzis and Doukas, 2014). Usualmente, la cadena de suministro tradicional comienza con el aprovisionamiento de materias primas, para después pasar por las fábricas en donde se transforma la materia prima y tener un producto terminado, luego, este producto terminado es llevado a almacenes donde se crean las rutas para finalmente ser trasladados a las instalaciones de los clientes.

En cada uno de los pasos mencionados anteriormente se incurren en diferentes costos que se pueden agrupar a grandes rasgos en costos de producción, costos de almacenamiento y costos de transporte. Una vez que la cadena de suministro ha sido exitosamente administrada, el siguiente paso lógico en la toma de decisiones es encontrar márgenes de mejora en la optimización para minimizar los costos totales de la cadena de suministro. En la Figura 1. Estructura típica de la cadena de suministro se observa el diagrama de lo mencionado.

Figura 1. Estructura típica de la cadena de suministro.

Al tener varios eslabones que se pueden ver de manera aislada, el reto de la cadena de suministro reside en la coordinación de esta. Partiendo de una cierta demanda situada en diferentes puntos geográficos, las organizaciones usualmente sitúan sus puntos de almacenaje y distribución a través de diferentes regiones cercanas a su demanda. En las cadenas de suministro de bienes de consumo la demanda usualmente se concentra en las principales ciudades. Por otra parte, los sitios de manufactura no necesariamente son situados acordes con la demanda, ya que la ubicación de estos podría estar determinada por la disponibilidad de personal, cercanía a la fuente de la materia prima, incentivos políticos, entre otras razones. De este modo, las

organizaciones deben decidir en qué sitios producir qué cantidad de producto, a través de la programación de la producción, para luego ser transportado a los centros de distribución y finalmente llegar al punto de venta.

Por consiguiente, conforme el tamaño y la complejidad de la cadena de suministro aumentan con el crecimiento de una organización y se tienen varios sitios de producción, así como de almacenaje y distribución, el reto de coordinar la cadena de suministro se vuelve también cada vez más difícil. Numerosas decisiones se toman con el objetivo de llevar los productos adecuados a la población mientras se busca no generar desperdicios, en otras palabras, minimizar los costos a lo largo de la cadena de suministro.

Por lo que, cada vez con mayor frecuencia se ha buscado en la literatura y en la industria buscar optimizar la cadena de suministro de manera integral. La manera de trabajo de generaciones anteriores era a través de diferentes departamentos aislados en una misma compañía lo cual se le denomina trabajo por *silos*, el día de hoy se busca cambiar de paradigma y la nueva visión corporativa busca la integración de las diferentes funciones de una empresa con la finalidad de encontrar mayores beneficios en la coordinación de las áreas.

En su artículo, Chandra y Fisher (1994) demuestran que el acercamiento de optimización por *silos* contra la optimización que le denominan *coordinada* puede encontrar beneficios importantes. Esto lo hicieron a través de un experimento computacional de optimización de cadena de suministro en el cual se hacía la comparativa de optimizar las funciones de producción y distribución tanto en conjunto como por separado. De tal modo, definieron un modelo de minimización de costos totales de la cadena de suministro compuestos por costos de producción, costo fijo por ajustes, costo de inventario y costo de transporte; e hicieron la optimización por silos para después compararla contra la optimización coordinada y pudieron medir los beneficios desde un 3% hasta un 20% en ahorros. Este rango se incrementa conforme se aumentan factores como: extensión en el horizonte de planeación, número de productos, número de clientes, y según aumenta el costo de distribución a comparación del costo de producción. Puesto que, a mayor flexibilidad de un modelo de optimización se puede obtener un mayor espacio de resolución. En conclusión, a mayores grados de libertad tiene un problema, mayor puede ser la optimización.

En la misma línea, los beneficios de la resolución de este cuestionamiento se pueden ver en el trabajo de Brown et al. (2001) quienes proyectaron ahorros en la optimización táctica de la capacidad de producción por hasta $40 millones de USD en The Kellogg Company. Otras aplicaciones en ámbitos profesionales se pueden ejemplificar en empresas como Libbey-Owens-Ford (Martin et al., 1993), donde optimizaron una cadena de suministro de cuatro plantas, más

de 200 productos y más de 40 centros de demanda para obtener ahorros de $2 M USD anuales y en Pfizer (Gupta et al., 2002) donde desarrollaron un plan estratégico con ahorros anuales potenciales de hasta $5.9 M USD. De tal forma, hay una tendencia importante de la aplicación de la optimización de la cadena de suministro en las organizaciones con un enfoque tanto estratégico como táctico.

Capítulo 1. Marco Teórico

Ante todo, debemos establecer las bases teóricas para definir los conceptos básicos que serán empleados a lo largo de este trabajo. Por consiguiente, en este capítulo se hace una selección de términos y conceptos con el objetivo de contextualizar los modelos y análisis que se proponen en los capítulos posteriores. De esta manera, a continuación, se desarrolla lo encontrado a través de la literatura en conjunto con aportes puntuales del propio autor de este trabajo.

1.1 Administración de la cadena de suministro

La cadena de suministro puede definirse como un grupo de compañías interconectadas que agregan valor a una serie de entradas transformadas desde su origen hasta el producto o servicio terminado que son demandados por consumidores finales (Lu, 2011). A través de estas cadenas de suministro fluyen materiales, información, finanzas y comercio.

La definición de la administración de la cadena de suministro es variada a través de la literatura, podría definirse como el diseño, operación y mejora de los sistemas que crean y entregan los productos y servicios para la compañía (Jacobs & Chase, 2021). Por otra parte, Stanton (2018) la define como la planeación y coordinación de las personas, procesos y tecnología involucrada en crear valor para la compañía. Ultimadamente, es el manejo o administración del negocio desde una perspectiva de cadena de suministro (Lu, 2011). Así pues, corresponde al conjunto de actividades que soportan la toma de decisiones de la cadena de suministro como la negociación de los precios, la definición del ordenamiento de la manufactura, el manejo de los procesos logísticos, entre otros. Estas actividades o tareas tienen un impacto sobre las otras, por tanto, se deben de administrar de manera coordinada.

Adicionalmente, el término de *administración de la cadena de suministro* no debe de confundirse con *logística*. La logística es la planeación y la definición del marco de trabajo que busca crear un plan para el flujo de productos e información en un negocio (Christopher, 2023). La administración de la cadena de suministro va un paso más arriba y conecta los diferentes eslabones de proveedores y clientes con la propia organización. En este sentido, se podría decir que la administración de la cadena de suministro es más amplia que la logística y de este modo la logística es abarcada por la administración de la cadena de suministro.

En la misma línea, el término *operaciones* debe de distinguirse de *procesos de cadena de suministro*. Las operaciones se refieren a los procesos de manufactura y servicios para transformar ciertos recursos de una organización en productos que los clientes desean. Mientras que los procesos de cadena de suministro se encargan del flujo de los materiales e información desde y hacia los procesos de manufactura de la organización (Jacobs & Chase, 2021).

1.2 VMI

El Inventario Administrado por Proveedor (VMI por sus siglas en inglés) es un sistema de manejo de inventarios en el cual es el proveedor quien adopta la responsabilidad de definir el nivel de inventario para cada cliente. En este ambiente el proveedor toma las decisiones de las cantidades de producto a enviar a cada punto de venta, de este modo, no se tienen pedidos u órdenes del cliente.

Este tipo de modelo nació en la década de los 80's en la aplicación de la industria por parte de empresas como Wal-Mart y Procter & Gamble (P&G) (Waller et al., 1999). Una de las bondades de este sistema es la posibilidad que tienen las compañías en la administración y optimización de su propia cadena de suministro. Para finales de la década de los 90's el sistema VMI era ampliamente usado y los proveedores tenían una mejor visibilidad del consumo del producto en los puntos de venta. Adicionalmente, los clientes tienen la ventaja de deslindarse de las responsabilidades del inventario, lo cual convierte este sistema en un ganar-ganar.

Asimismo, con la estandarización del sistema VMI vino la integración vertical de las empresas. La integración vertical se define como la propiedad única de actividades consecutivas a lo largo de la cadena de suministro (Lu, 2011). Es decir, conforme las organizaciones extendían su visibilidad, distribución y manejo de inventario hacia sus clientes, se considera que aumentaba su integración vertical.

Este sistema es usado ampliamente en los bienes de consumo en el canal tradicional que en México se ejemplifica en las tiendas de abarrotes, donde el proveedor hace la recomendación al abarrotero de cuánto producto debería tener. El objetivo de este tipo de modelo de manejo de inventarios es la eficiencia para disminuir el nivel de inventario, incremento de nivel de servicio y desempeño del proveedor (Choi et al., 2004).

Muchas han sido las iniciativas para la reducción de costos de la cadena de suministro, en este trabajo nos enfocaremos en el problema de producción-ruteo (PRP), el cual es una aplicación para cadenas de suministro de tipo VMI.

1.3 Planeación de la producción

La planeación de la producción determina qué, cuándo, dónde y cuánto producir para satisfacer las necesidades de los consumidores (Sule, 2008). El proceso de la planeación de la producción comienza a partir de la demanda en los puntos de venta, esta demanda es sumada o agregada para obtener un pronóstico de venta el cual es la entrada principal para la planeación de la producción. Partiendo de este pronóstico, el trabajo de los planeadores de la producción consiste en determinar las cantidades adecuadas de producción con el objetivo de satisfacer la demanda y al mismo tiempo no mantener un inventario alto. Por consiguiente, en un escenario óptimo el

planeador de la producción balancea de manera ideal de tal modo que todos los consumidores fueron satisfechos y el inventario al final del periodo es de 0.

Por otra parte, la propia naturaleza de la organización da pie para la definición del esquema de producción que le es más benéfico. A partir de diversos factores como el volumen y la variedad de los productos a elaborar se distinguen categorías de entornos de producción para las organizaciones, Chapman (2006) las separa en cuatro categorías:

- Make-to-Stock (MTS): en este entorno de producción los productos son producidos para luego ser almacenados. Este sistema es de tipo empujar o *push* en donde es la organización quien define cuánto producto enviar al mercado.
- Assemble-to-Order (ATO): en este caso el consumidor tiene más influencia sobre el diseño del entorno. Se tienen diferentes subensambles y cuando el cliente pide cierto producto entonces se ejecuta el ensamble final.
- Make-to-Order (MTO): este entorno permite al cliente especificar el diseño exacto del producto final. Este sistema de tipo jalar o *pull* en donde es el cliente quién decide cuándo y cuánto ordenar.
- Engineer-To-Order (ETO): en este caso el cliente es propietario de todas las decisiones del producto o servicio. La organización crea el diseño personalizado desde cero para el cliente final.

Así pues, en una cadena de suministro de tipo VMI es común tener un entorno de producción MTS ya que la integración vertical de la compañía permite tomar las decisiones de producción e inventario necesarias para llevar la cantidad de producto a las localidades adecuadas.

1.4 Problema de producción-ruteo (PRP)

Dada una red donde fluyen diferentes productos se ha buscado desde hace décadas la posibilidad de hacerlo a menor costo. La primera vez que se documentó este problema fue a manera de generalización del clásico Travelling Salesperson Problem (TSP) por Dantzig y Ramser (1959) para originar lo que posteriormente se convirtió en el problema de ruteo de vehículos (Vehicule Routing Problem o VRP) que se ha continuado desarrollando por numerosas líneas de investigación en recientes décadas. Una generalización del VRP vino a través de la inclusión de la gestión de inventarios, resultando en el problema de ruteo e inventario (Inventory Routing Problem o IRP) que fue primeramente mencionado por Bell et al. (1983) en una aplicación de planificación diaria de entregas para la compañía Air Products & Chemicals, generando ahorros de operación de 6% a 10%. Luego, este problema fue formulado y generalizado por Federgruen & Zipkin (1984).

Hasta ese momento, ningún autor consideraba la función de la producción en el modelo. Siendo que en la estructura de costos promedio el costo de transformación o producción es de los más relevantes, entonces es primordial incluir este rubro en la optimización de la cadena de suministro. Así pues, el Problema de Producción Ruteo (Production Routing Problem o PRP) nace a partir de la necesidad de optimizar no sólo el ruteo y el almacenamiento de los productos, sino también de la necesidad de incluir la faceta de producción. Este modelo se crea en la unión de dos modelos históricos, toma la parte del ruteo e inventario del IRP, por otro lado, toma la faceta de producción del modelo de Lot Sizing with Direct Shipment (LSDS) (Adulyasak et al., 2015).

De esta manera lo proponen por primera vez Chandra y Fisher (1994), en su modelo integral de producción y distribución, en el cual buscaban la minimización de los costos fijos de ajustes en producción más los costos de almacenaje y distribución. Con este artículo como punto de partida para el tipo de modelos PRP, Chandra & Fisher consideran una sola planta de manufactura. En consecuencia, no se incluye en la función objetivo una minimización del costo de producción ya que no hay variación en los costos que generalmente vienen de hacerlos en diferentes fábricas. De tal modo, el primer modelo que considera una optimización multiplanta lo podemos observar en el trabajo de Lei et al. (2016). A partir de estas aportaciones, comenzó un desarrollo importante en la resolución de diferentes PRPs el cuál ha tenido un auge en la última década, una revisión extensa de literatura y algoritmos de resolución se puede encontrar en el trabajo de Adulyasaak et al. (2015).

Para ejemplificar tales aplicaciones, podemos recurrir a artículos con aplicaciones del PRP. Tal es el caso de Neves-Moreira et al. (2019), en donde se trata un problema con una planta con varias líneas de producción y entregas con ventanas de tiempo. En Qui et al. (2021) pretenden resolver un problema con dos eslabones en la cadena de suministro teniendo cross-docks satélites entre la planta y los clientes. Por otra parte, este problema también se ha usado para plantear el problema de la minimización de emisiones de CO2 cambiando el enfoque monetario (Qiu et al., 2017).

1.5 Flow-shop scheduling

El ordenamiento de la producción o *scheduling* es definido como el establecimiento de los tiempos para desempeñar una tarea (Wight, 1984). Donde los dos problemas esenciales son las prioridades y la capacidad. Así pues, el problema del ordenamiento de la producción reside en establecer qué se debe producir en qué orden. Otra definición propuesta por Sule (2008) es que el *scheduling* determina cómo obtener las metas definidas en la planeación de la producción cuando los recursos son limitados, y si las metas no se pueden obtener, cómo obtener nuevas metas que sean óptimas y prácticas con los recursos disponibles.

En términos de planeación de la producción, Sule (2008) divide los modelos de ordenamiento de la producción en las diferentes categorías, a continuación, se muestran las más relevantes:

- Máquina única: los productos o trabajos se procesan por una única máquina uno a la vez, donde cada trabajo tiene un tiempo de procesamiento.
- Máquinas en paralelo: un número de máquinas idénticas están disponibles y se pueden procesar los trabajos en cualquiera de ellas.
- Job Shop: se tienen diferentes máquinas en la planta y dependiendo del trabajo o producto se requieren ciertas o todas las máquinas en cualquier secuencia, la única restricción es que un producto no pase más de una vez por la misma máquina.
- Flow Shop: los productos se procesan en máquinas múltiples en una secuencia idéntica, es decir, siempre se va desde la primera máquina hasta la última de manera secuencial. El tiempo de procesamiento puede ser diferente para cada combinación producto-máquina.

Debido a que este trabajo se enfoca en bienes de consumo en cadenas de suministro de tipo VMI, donde generalmente un producto sigue una serie de procesos de manufactura en secuencia, para el desarrollo de este trabajo nos enfocaremos en el *scheduling* de tipo de *flow shop*. El reto en este tipo de ordenamiento de la producción es que se tienen varios niveles de producción con diferentes cantidades de máquinas por cada nivel, de tal modo que los productos o trabajos deben fluir de inicio a fin en los niveles de producción con un cierto tiempo de producción y cada máquina está limitada por una cantidad de tiempo total disponible.

Dentro de los planteamientos del *flow shop scheduling*, algunos autores han resuelto para la minimización del tiempo total (Yagmur & Kesen, 2021) y otros han estado alineados a la minimización de costos de la cadena de suministro, así nos muestran en su modelo Scholz-Reiter et al. (2011), el presente trabajo expande sobre la línea de este último artículo.

1.6 Tiempos de ajustes de producción

La complejidad del *flow shop scheduling* aumenta cuando entre cada cambio de producto en una misma máquina se detona un tiempo en el cual se deben hacer ajustes que podría corresponder de manera operativa con un cambio de herramental, configuraciones en las máquinas, cambios de empaques, lavado de la máquina, entre otras actividades. A la suma de los tiempos en este tipo de tareas donde no se está teniendo producción efectiva, pero se necesita para poder procesar el siguiente producto se le denomina *tiempo de ajuste* (*setup time*).

Al adentrarse en el *flow shop scheduling*, estos modelos usualmente consideran la variable del tiempo de ajuste entre productos, no obstante, algo que no se ha explorado es el caso particular de tiempos de ajuste dependientes del tipo de producto con una matriz desde-hacia. Un ejemplo

podría ser el siguiente: cambiar del producto A al B demora 20 minutos, pero cambiar del B al A tiene una duración de 90 minutos. Este tipo de aplicaciones es común en la transformación de alimentos, por ejemplo, al tener lavados profundos después de procesar un producto que contiene un ingrediente alérgeno.

1.7 Investigación de operaciones

Los inicios de la investigación de operaciones (IO) se remontan a la década de los 40's, teniendo la Segunda Guerra Mundial como contexto histórico, se hicieron desarrollos importantes en este campo. El aporte que podría considerarse clave como inicio formal de este campo disciplinario es la creación de *método simplex* por parte de George Dantzig en 1947 para resolver problemas de programación lineal (Hillier & Lieberman, 1997).

Hillier & Lieberman (1997) definen IO como "hacer investigación sobre las operaciones". De tal modo, este campo disciplinario es aplicado a los problemas de coordinación y toma de decisiones en las operaciones dentro de una organización. En ese sentido, la IO busca encontrar una mejor solución, llamada *solución óptima*, para el problema bajo consideración.

Los problemas son modelados matemáticamente al definir sus *variables de decisión* para las cuales se busca obtener su valor. Además, se establece una *función objetivo*, la cual representa una medida de desempeño en términos de las variables de decisión; esta función objetivo tiene la opción de buscar un valor máximo o mínimo. Posteriormente, se enlistan las *restricciones* que son expresiones matemáticas que limitan los valores que pueden tomar las variables. Por último, se tienen *parámetros* que son constantes que afectan directamente a la función objetivo y sus restricciones. Estos elementos constituyen un modelo de programación lineal donde el adjetivo lineal significa que todas las funciones son del tipo lineal. Por otra parte, el término programación hace referencia a la planeación, en contraposición con la programación computacional.

1.8 Programación Lineal Entera Mixta (MILP)

Hiliier & Lieberman (1997) afirman que "el modelo matemático para programación entera es sencillamente el modelo de programación lineal con la restricción adicional de que las variables deben tener valores enteros." Consecuentemente, cuando sólo algunas de las variables de decisión toman valores enteros o binarios el modelo se le llama *programación lineal entera mixta* o MILP por sus siglas en inglés. Una variable binaria es la cual sólo puede tomar valores de 1 y 0, estas son usadas ampliamente para tomar decisiones de sí o no.

Históricamente, el algoritmo más usado para resolver los modelos de programación entera lineal es el de ramificación y acotamiento (*Branch-and-Bound*) (Taha, 2007). Este algoritmo fue desarrollado por A. Land & G. Doig (1960) y crea un árbol de soluciones explorando las diferentes ramas para encontrar subconjuntos de solución.

1.9 Problema de ruteo de vehículos (VRP)

Tradicionalmente, el objetivo del Problema de Ruteo de Vehículos, VRP por sus siglas en inglés, es el de encontrar un arreglo de rutas a menor costo tales que cada cliente se visita solamente una vez, donde cada ruta inicia y finaliza la ruta en el origen llamado depósito. En la variante más conocida, el CVRP donde la C indica vehículos capacitados, la capacidad de los vehículos no puede ser rebasada (Braekers et al., 2016).

Por la manera en que la información evoluciona se puede dividir el VRP en dos: estático y dinámico. Por otra parte, por el tipo de información el VRP se divide en otras dos clasificaciones: determinista y estático. Al combinar estos factores se obtienen 4 diferentes tipos de VRP (Pillac et al., 2011):

- Estático y determinista: el más sencillo de todos los tipos ya que no hay cambios en la información durante la ruta, además, la información se caracteriza por ser de carácter certero.
- Estático y estocástico: ciertas variables están sujetas a la aleatoriedad, pero la planeación se hace previa a la ejecución del ruteo.
- Dinámico y determinista: todas las variables son conocidas, pero durante la ejecución de las rutas se puede recalcular por lo cual se necesita tecnología para comunicarse entre vehículos.
- Dinámico y estocástico: el más complejo de todos los tipos ya que ciertas variables son reveladas durante la ejecución del ruteo y el plan de ruteo se puede modificar completamente durante el camino. Aplicaciones de este tipo de VRP se puede ver en el problema de tipo “dial-o-ride” aplicado en Wilson et al. (1977) y en Ulmer et al. (2020).

En el caso de las cadenas de suministro de tipo VMI que son altamente verticalmente integradas se tiene control sobre la información y puede hacer una planeación con antelación se puede tratar como un VRP de tipo estático y determinista, el cual no agrega complejidad adicional en términos de la evolución de la información a través del tiempo. Es decir, se tiene toda la información *a priori* con la certeza de que esta data no cambiará y el objetivo es hacer el ruteo a costo mínimo.

Asimismo, en las últimas décadas se han incursionado en diversas líneas de investigación con respecto al VRP. Algunos ejemplos de esto son el VRP con ventanas de tiempo, en el trabajo de Hou et al. (2022) se penaliza el tiempo en el que no se llega dentro de la ventana de tiempo especificada por cada cliente. Por otra parte, en el artículo de Salhi et al. (2013) se resuelve el VRP con retornos de producto hacia el depósito u origen como lo son las denominadas “rutas

lecheras". Parte del desarrollo de esta investigación buscará hacer una comparativa entre los métodos de resolución del VRP.

1.10 Métodos de Resolución VRP

Dado que el PRP contiene al VRP y siendo que el VRP es NP-duro, entonces se puede afirmar que la resolución del modelo a plantear también será NP-duro. Así pues, surge además el problema de la complejidad computacional. Con la intención solventarlo algunos autores han desarrollado técnicas heurísticas (Qiu et al., 2017) (Avci & Yildiz, 2020) (Meinecke & Scholz-Reiter, 2014) y otros también se han optado por la utilización de las metaheurísticas (Neves-Moreira et al., 2019) (Qiu et al, 2018).

El término de "metaheurísticas" fue mencionado primeramente por Glover (1986), en el que afirmaba que el algoritmo de búsqueda tabú contenía una superposición de heurísticas. Algunas de las metaheurísticas más utilizadas para la resolución del VRP son los algoritmos genéticos, recocido simulado, búsqueda tabú, entre otros (Goel & Maini, 2017). Recientemente, otro método popular para solventar el VRP es la búsqueda variable de vecindario (Qiu et al, 2018) (Absi et al., 2015). A continuación, se describen algunas técnicas metaheurísticas relevantes:

1. Algoritmo genético: iniciado por Holland (1975), este algoritmo (considerado como metaheurística años más tarde) se caracteriza por la emulación de procesos evolutivos o genéticos. Los operadores con los que cuenta este algoritmo son los de selección, en el cual se elige a ciertas soluciones potenciales, llamados individuos, que se desempeñan de mejor manera y serán los elegidos para producir la siguiente generación; por otra parte, se cuenta con el operador de cruza en el cual se combina la información genética de los individuos seleccionados; por último, este algoritmo introduce diversidad a través del operador de mutación en el cual se alteran ciertos individuos en función a un índice o porcentaje de mutación.
2. Búsqueda dispersa: este algoritmo fue introducido por Glover (1977) y consta de 5 métodos:
    - Generación de diversificación: genera una colección de soluciones de prueba, estas no son necesariamente factibles.
    - Mejora: Transforma el conjunto de soluciones de prueba para mejorar su desempeño.
    - Actualización de conjunto de referencia: se actualiza el conjunto de soluciones tomando que aportan mayor valor tanto en calidad como en diversidad, este conjunto típicamente no es mayor a 20 soluciones.
    - Generación de subconjunto: creación de nuevas soluciones a partir del conjunto de referencia.

- Combinación de solución: se combinan soluciones a partir del subconjunto en uno o más vectores combinados de solución.

3. Recocido simulado: En el trabajo de Kirkpatrick et al. (1983) se presentó esta metaheurística basada en un proceso de enfriamiento de metal. Se caracteriza por tener una temperatura inicial alta la indica que se puede explorar soluciones con alta diversidad, a lo largo de la corrida esta temperatura va disminuyendo y al "enfriarse" cada vez se aceptan cambios menos radicales en las soluciones.
4. Búsqueda tabú: En el artículo donde Glover (1986) menciona por primera vez el término de metaheurística también concibe a este algoritmo. La búsqueda tabú presenta similitudes con el recocido simulado, pero contiene las siguientes características propias:
    - Se aceptan movimientos a soluciones peores bajo ciertos criterios
    - A través de la "lista tabú" se hacen prohibiciones para no regresar a soluciones ya exploradas
    - Se establece un contador llamado "tabú-activo" para determinar la iteración en la cual ya es permitido hacer un movimiento prohibido o "tabú"
    - Los "criterios de aceptación" determinan las restricciones bajo las cuales se accede a soluciones tabú.
5. Búsqueda Variable de Vecindario: Mladenović & Hansen (1997) presentan esta metaheurística a través de la cual se exploran vecindarios con distancias incrementales, aplicando una rutina de búsqueda local para encontrar el óptimo local para después brincar hacia otros vecindarios con mejores valores hasta llegar a un óptimo global.

Capítulo 2. Desarrollo del modelo de optimización y experimentación con metaheurísticas

En este capítulo se busca construir los primeros pasos de la resolución del PRP, para que en el siguiente capítulo se haga el desarrollo de la metodología del problema. Así pues, se propone el modelo de programación lineal entera mixta (MILP) que es parte central de este trabajo de investigación. Tras presentar su formulación se aborda la verificación y validación con un caso de pequeña escala para demostrar su validez y aplicabilidad. Por último, se presenta la experimentación de metaheurísticas hecha a través del software HeuristicLab.

2.1 Descripción del problema

Se considera una red flexible de múltiples plantas donde un mismo producto puede ser procesado en diferentes fábricas. En este modelo se considera un sólo escalón que parte de las fábricas y lleva el producto a los almacenes internos de la empresa, donde se agrega y se sitúa la demanda. Este tipo de gestión de la cadena de suministro suele implementarse en la industria de productos empacados para el consumidor, que a menudo opera en un esquema VMI (Zhao, 2019). Dado que no se pretende tener almacenamiento de producto terminado dentro de las plantas, para este modelo no se considerarán inventarios ya que el inventario se contempla en los almacenes que estarán considerados como clientes, por lo tanto, el inventario está fuera del alcance de este problema y el modelo sólo considerará costos de producción y transporte.

Este modelo pretende resolver la programación o scheduling en la fase de manufactura. Cabe mencionar que esta sección el modelo de programación binaria-mixta no contempla el ruteo hacia los almacenes, esto será resuelto en una siguiente etapa. Adicionalmente, esta formulación busca resolver restricciones complejas de tiempos de ajustes entre productos, cuyo comportamiento puede explicarse mediante una matriz desde-hacia. Esto es común en la industria de alimentos por cuestiones de alérgenos, especies o cualquier factor que implique limpiezas. Por ejemplo: en una fábrica de helados se procesan productos sabor vainilla y chocolate en una misma línea, el tiempo de lavado desde vainilla hacia chocolate podría ser de media hora, pero en caso contrario el lavado se podría elevar a una hora. Así pues, no se encontró en la literatura un modelo que resolviera este tipo específico de restricciones de ajuste.

Además, dentro de la literatura estudiada la modelación de tipo flow shop scheduling se basa en trabajos u órdenes de producción, de esta manera, el contenido de cada orden se considera previamente definido con la especificación cantidad de producto a fabricar, sin embargo, en la industria se busca resolver los problemas en unidades específicas como piezas, cajas, kilogramos, litros, etc.; por lo tanto, este modelo se basa en unidades para tener la flexibilidad suficiente para ser implementado en varias industrias. Esta aportación, junto con la matriz desde-hacia, son las innovaciones principales del modelo propuesto.

Este modelo de programación lineal entera mixta (MILP) especifica la cadena de suministro a optimizar a través de la cantidad de *subíndices o conjuntos*, mismos que definen el tamaño del modelo a tratar. Adicionalmente, se describen las *variables de decisión* tanto continuas como binarias que están alineadas a las preguntas de *qué*, *cuánto*, *cuándo* y *dónde* producir y transportar. Estas variables de decisión van relacionadas con los *parámetros* que definen los coeficientes de las variables. Luego, se precisa la *función objetivo* que marca la dirección de optimización. Por último, los valores de las variables de decisión se delimitan a través de los grupos de *restricciones*. Todos estos elementos descritos constituyen el modelo de optimización que parte esencial de este trabajo y se detalla a continuación.

2.2 Subíndices / Conjuntos

- Productos j/J
- Máquinas m/M
- Niveles de producción n/N
- Órdenes de producción o/O
- Fábricas f/F
- Almacenes a/A

2.3 Parámetros

- Costos:
  - c_{jf}^{p}: Costo de producción unitario del producto *j* en la fábrica *f*
  - c_{fa}^{t}: Costo transporte unitario de la fábrica *f* al almacén *a*
- ${t^{p}}_{jmnf}$: tiempo de procesamiento por unidad del producto *j* en el nivel *n* en la máquina *m* en la fábrica *f*
- ${t^{a}}_{jj'mnf}$: tiempo de ajuste por unidad del producto *j* hacia el producto *j'* en el nivel *n* en la máquina *m* en la fábrica *f*
- HD_{mnf} : horas disponibles de la máquina *m* en el nivel *n* de la fábrica *f*
- D_{ja}: Demanda del almacén *a* para el producto *j*
- M: Constante de magnitud grande

2.4 Variables de Decisión

- x_{jmnof}^{p}: unidades a producir del producto *j* en la fábrica *f* que pase por la máquina *m* en el nivel de producción *n* en el orden de producción *o*
- x_{jfa}^{t}: unidades a transportar del producto *j* en la fábrica *f* al almacén *a*
- ${t^{f}}_{jmnf}$: tiempo de finalización producto *j* en el nivel *n* en la máquina m en la fábrica *f*

2.5 Variables Binarias

- X_{jmnof}: Si el producto *j* se procesa en el nivel *n* en la máquina *m* en el orden de producción *o* en la fábrica *f*, 0 si no
- $Y_{jj'mnf}$: Si el producto *j* se produce antes del *j'* en el nivel *n* en la máquina *m* en la fábrica *f*, 0 si no

2.6 Función Objetivo

$$Z_{min} = \sum_{j=1}^{J}\sum_{m=1}^{M}\sum_{n=1}^{1}\sum_{o=1}^{O}\sum_{f=1}^{F} c_{jmnf}^{p} * x_{jmnof}^{p} + \sum_{j=1}^{J}\sum_{f=1}^{F}\sum_{a=1}^{A} c_{fa}^{t} * x_{jfa}^{t}$$

La función objetivo buscará minimizar el total de cadena de suministro compuesto por las sumatorias de costos de producción y transporte.

2.7 Restricciones

Los siguientes grupos de restricciones son creadas para imitar el comportamiento de un flow shop scheduling, así como para la resolución del problema de producción-ruteo. Existen los casos particulares donde los subíndices al sumar o restar una unidad quedan fuera del conjunto declarado, por ejemplo: $x_{j'mno+1f}^{p}$. En estos casos los términos asociados se sustituyen por cero para no alterar las restricciones o se omiten las restricciones asociadas, según sea el caso.

1. La cantidad de producto que se traslada desde las fábricas deberá satisfacer la demanda asignada a cada almacén.

$$\sum_{f=1}^{F} x_{jfa}^{t} \geq D_{ja} \ (\forall\, a, j)$$

2. Este modelo no contempla la capacidad de almacenar inventarios, por lo tanto, la cantidad de producto que se fabrica en cada instancia deberá ser igual a la cantidad que se transporta.

$$\sum_{m=1}^{M}\sum_{n=1}^{1}\sum_{o=1}^{O} x_{jmnof}^{p} - \sum_{a=1}^{A} x_{jfa}^{t} = 0 \ (\forall\, j, f)$$

3. El tiempo de finalización de cada producto deberá ser menor al total de horas que se tienen disponibles para cada máquina en cada nivel en cada fábrica.

$$t^{f}{}_{jmnf} \leq HD_{nmf} \ (\forall\, j, m, n, f)$$

4. Se activará la variable binaria X_{jmnof} cuando se produce al menos una unidad de x_{jmnof}^{p}, estas variables binarias son necesarias para detonar los ajustes entre productos. Además, se agrega un complemento a esta restricción para que no se activen las variables binarias X_{jmnof} en caso de que no haya producción x_{jmnof}^{p} asignada.

$$MX_{jmnof} - x_{jmnof}^{p} \geq 0 \ (\forall\, j, m, n, o, f)$$

$$Mx_{jmnof}^{p} - X_{jmnof} \geq 0 \ (\forall\, j, m, n, o, f)$$

5. Se condiciona la variable binaria $Y_{jj'mnof}$ a activarse sólo si las variables binarias X_{jmnof} y $X_{j'mno+1f}$ toman el valor de 1, es decir, sólo habrá un cambio entre los productos j y j' cuando coincidan en el orden o y $o+1$ respectivamente, esto indica que va uno tras otro.

$$1 - M\left(2 - X_{jmnof} - X_{j'mno+1f}\right) - Y_{jj'mnf} \leq 0 \ (\forall\, j, j', m, n, o-1, f)$$

6. Para toda máquina m en todo nivel n se restringe a que sólo se procesa un producto a la vez, de esta manera se respeta la secuencia para cada orden o. Esta restricción sólo está activa cuando se analice el producto X_{jmnof} seguido del $X_{j'mno+1f}$.

$$t^f{}_{jmnf} + t^p{}_{j'mnf} * x^p_{j'mno+1f} + t^a{}_{jj'mnf} * Y_{jj'mnf} - t^f{}_{j'mnf} - M\left(2 - X_{jmnof} - X_{j'mno+1f}\right) \leq 0 \ (\forall\, j, j', m, n, o-1, f)$$

7. Para un mismo producto j se obliga a que sus tiempos de finalización vayan de manera ascendente conforme avanza a través de los niveles n, es decir, no puede ingresar al nivel 2 si no ha terminado el nivel 1.

$$t^f{}_{jmn-1f} + t^p{}_{jmnf} * x^p_{jmnof} + t^a{}_{j'jmnf} * Y_{j'jmnf} - t^f{}_{jmnf} - M\left(1 - X_{jmnof}\right) \leq 0 \ (\forall\, j, j', m, n, o, f)$$

8. Para que un producto pueda ser considerado como terminado, todas las unidades de este deberán procesadas a través de todos los niveles de producción.

$$\sum_{o=1}^{O}\sum_{m=1}^{M} x^p_{jmnof} - \sum_{o=1}^{O}\sum_{m=1}^{M} x^p_{jmn+1of} = 0 \ (\forall\, j, n-1, f)$$

9. Para cada máquina en cada nivel, sólo podrá ser considerado un sólo producto j para cada combinación de máquina, nivel, orden y fábrica.

$$\sum_{j=1}^{J} X_{jmnof} \leq 1 \ (\forall\, m, n, o, f)$$

10. Complementando a la restricción anterior, un producto sólo podrá ser procesado en una ocasión, es decir, en un sólo orden o.

$$\sum_{o=1}^{O} X_{jmnof} \leq 1 \ (\forall\, j, m, n, f)$$

11. Esta restricción busca restringir soluciones que no estén tomando ordenes consecutivos en una misma máquina, nivel y fábrica. Esto se hace con la intención de que se hagan los ajustes entre productos necesarios ya que de no tenerla podría darse el caso que se asigna producción a un orden 1 y después a un orden 3 sin existir una producción con orden 2. Esta restricción podrá verse alterada en función de la cantidad de elementos en el

conjunto de órdenes O, de ser necesario, se deberán agregar los términos consecutivos para conseguir el cometido descrito de esta restricción.

$$\sum_{j=1}^{J} X_{jmno-1f} + \sum_{j=1}^{J} X_{jmno+1f} - M \sum_{j=1}^{J} X_{jmnof} \leq 1 \ (\forall\ m, n, o, f)$$

12. Las variables de decisión continuas no podrán tomar valores negativos.

$$x^{p}_{jmnof}, x^{t}_{jfa}, t^{f}{}_{jmnf} \geq 0$$

13. Las variables de decisión binarias sólo podrán tomar los valores cero o uno.

$$X_{jmnof}, Y_{jj'mnf} = \{0,1\}$$

2.8 Pruebas de verificación

En la etapa de verificación se busca que el modelo planteado sea conceptualmente correcto. Este paso se puede efectuar a través de una o más pruebas como lo son las siguientes:

- Condiciones extremas: se toma uno o más parámetros del modelo y se lleva a condiciones extremas, es decir, se lleva el parámetro a un valor mínimo (0) así como a un valor máximo (infinito) y se evalúa cómo se comportará la función objetivo (verificación general) o alguna restricción (verificación parcial) para entender si hay comportamientos atípicos que no hacen sentido.
- Salidas esperadas: se busca definir una instancia pequeña en la que se tenga total certeza de qué cuál será el resultado y al hacer la resolución tendrá que salir un valor esperado.
- Animación: al trabajar con software especializado, en ciertos casos se tiene la opción de tener animaciones, la prueba de animación consiste en que los elementos animados en el software tengan sentido visualmente, en otros términos, que sus movimientos en el software sean los correctos.

Para este modelo se realiza la verificación a través de la prueba de condiciones extremas. Los parámetros del modelo por variar son el costo de producción c^{p}_{jf}, la demanda D_{ja} y el tiempo de procesamiento $t^{p}{}_{jmnf}$ y, como se mencionó previamente, se evalúa el impacto en la función objetivo de minimización de costos al asignarles un valor de 0 y de infinito. Los resultados de la prueba se pueden observar en la Tabla 1. Prueba de condiciones extremas.

Parámetro	Valor	Zmin	Comentarios para prueba de condición extrema
c_{jf}^{p}	∞	∞	Se maximiza el costo
c_{jf}^{p}	0	0	Sólo queda el componente de transporte en el costo
D_{ja}	∞	Infactible	Los tiempos de finalización tienden a infinito cuando deberían ser menor o igual a las horas disponibles de las máquinas
D_{ja}	0	0	El modelo se interrumpe y no sucede nada, de manera parcial las colas de las máquinas de producción están vacías y las máquinas están a la espera
t_{jmnf}^{p}	∞	Infactible	Los tiempos de finalización tienden a infinito cuando deberían ser menor o igual a las horas disponibles de las máquinas
t_{jmnf}^{p}	0	Cota	Se obtiene una cota de costo, pero el modelo sale del alcance para el que estaba diseñado y deja de funcionar, en este caso se sale de la verificación y la validación es necesaria

Tabla 1. Prueba de condiciones extremas.

2.9 Validación del modelo

Con la finalidad de validar el modelo de optimización previamente expuesto para el problema de producción ruteo planteado en este trabajo, se desarrolló un ejemplo base o pequeña instancia, también llamado *toy model*, la cual fue modelada en *Microsoft Excel* y resuelta a través del motor de optimización *Solver*.

Para explicar el funcionamiento del modelo se extiende el ejemplo de la fábrica de helados descrito previamente: Se plantea una cadena de suministro extremadamente simple con una sola fábrica y un almacén. Los productos por elaborar y distribuir son envases de un litro de helado sabor chocolate y vainilla. En este ejemplo, la manufactura de helados se simplifica a dos procesos: en la primera etapa o nivel se prepara la mezcla y se congela, después, en la segunda se procede a envasar los helados. Se cuenta con una sola máquina para cada proceso, así que en estas dos máquinas deberán procesarse ambos sabores de helado y por tanto se detonan los ajustes entre cada cambio de producto. Dada esta descripción, los subíndices estarían definidos de la siguiente manera:

- Productos $j = \{1,2\}$ $(1 = chocolate, 2 = vainilla)$
- Máquinas $m = \{1\}$ (Aunque haya dos máquinas, en realidad sólo hay una máquina por nivel como máximo)
- Nivel de producción $n/N = \{1,2\}$ $(1 = mezcla\ y\ congelación, 2 = envasado)$
- Orden de producción $o/O = \{1,2\}$ (orden consecutivo de procesamiento por máquina)
- Fábricas $f/F = \{1\}$ (una fábrica)
- Almacenes $a/A = \{1\}$ (un almacén)

Luego, se definen los parámetros del modelo. en esta instancia se considera una semana de producción con 160 horas disponibles en cada máquina, se tiene una demanda de 95 litros de chocolate y 90 litros de vainilla. Los costos unitarios de producción son de 6 unidades monterías (U.M.) y 5, respectivamente. Además, el costo de transporte de la planta al almacén es de 3 U.M. por litro. Adicionalmente, los tiempos de procesamiento en mezclado y congelación son de 15 minutos para el sabor chocolate y 1 hora para vainilla; y para el proceso de envasado serán de 1 hora y 15 minutos para chocolate y vainilla, respectivamente. Asimismo, debido a los colores de los productos, el tiempo de limpieza en la línea mezcladora - congeladora son de 10 horas si se planea cambiar de chocolate hacia vainilla, pero si se hace de vainilla hacia chocolate sólo es necesario limpiar 5 horas. Por otro lado, debido a la densidad de cada producto una vez que está congelado, en la línea de envasado se contemplan tiempos de lavado de 2 horas de chocolate a vainilla y de 3 horas en caso contrario. La modelación de parámetros a manera de tablas se puede encontrar en el Anexo 1. Parámetros ejemplo helados.

Una vez que se definen los subíndices y parámetros se procede a hacer la modelación de la función objetivo para la minimización de costos de manufactura y transporte.

$$Z_{min} = \sum_{j=1}^{J}\sum_{m=1}^{M}\sum_{n=1}^{1}\sum_{f=1}^{F} c_{jmnf}^{p} * x_{jmnof}^{p} + \sum_{j=1}^{J}\sum_{f=1}^{F}\sum_{a=1}^{A} c_{fa}^{t} * x_{jfa}^{t}$$

$$Z_{min} = 6x_{11111}^{p} + 6x_{11121}^{p} + 5x_{21111}^{p} + 5x_{21121}^{p} + 3(x_{111}^{t} + x_{211}^{t})$$

Posteriormente, se hace la modelación de las restricciones. Siendo que la cantidad de restricciones aumentan de manera combinatoria, con los subíndices definidos para esta instancia se obtienen en total 70 restricciones. Así pues, se procedió a la resolución de la instancia a través del optimizador Solver de Microsoft® Excel® en su versión Microsoft 365 MSO (Version 2303 Build 16.0.16227.20202) 64-bit. Por último, se introduce cada uno de los elementos del modelo a la ventana del optimizador y se seleccionó el método Simplex de programación lineal. El resultado se obtiene en 0.093 segundos, 11 iteraciones y 18 subproblemas. Las especificaciones del equipo de cómputo son las siguientes: Intel® Core™ i7-4720HQ CPU @ 2.60GHz y 16 GB de memoria RAM.

El Anexo 2. Resultados ejemplo helados contiene los valores para cada una de las variables, así como la función a minimizar que da un resultado de 1,575 U.M. Para esta instancia la función objetivo tan sólo regresa el valor de la sumatoria de los costos que siempre y cuando se obtenga una solución factible entonces será óptima bajo los parámetros descritos. La intención de este modelo es que al tener diferentes costos de manufactura a través de f fábricas, así como un costo de transporte en función de la combinación entre dos destinos, se pueda determinar la

locación óptima para fabricar cada uno de los productos, así como su ruta; es en este caso donde esta función objetivo resultará provechosa.

En adición, se puede analizar a detalle los tiempos de finalización de cada producto en cada una de las etapas, en esta instancia, se arroja una respuesta óptima al empezar a producir el producto 1 (chocolate) ya que, aunque tendrá un tiempo de ajuste, o lavado, más extenso que al producir primero el producto 2 (vainilla); el corto tiempo de producción unitario del producto 1 en el nivel 1 crea una situación en la cual se minimiza el tiempo total de producción. Es decir, aunque el modelo no contempla una minimización de tiempos en la función objetivo, el resultado será en un tiempo mínimo para cumplir con las restricciones de capacidad o de horas disponibles que tiene cada máquina.

En el Anexo 3. Restricciones ejemplo helados, se puede encontrar el detalle de las formulaciones junto a la modelación a manera de tablas para cada una de las restricciones con los resultados obtenidos por el optimizador. Esta manera de modelación nos arroja una visualización conveniente al tener los valores de los lados izquierdos y derechos para ver un análisis detallado de los valores de cada una de las restricciones. Las restricciones con la leyenda “no aplica” se refieren a combinaciones de variables que no existirían en la instancia propuesta, pero al plantear a manera de tabla resulta esa combinación, esto sucede en órdenes de producción $o = 3$.

En conclusión, primero se hizo la verificación del modelo a través de la prueba de condiciones extremas, para verificar que conceptualmente no tenemos inconsistencias en las restricciones del modelo, además, se verifica que el software no arroja errores en el planteamiento de la función objetivo o las restricciones.

Adicionalmente, también se valida el modelo, por validación se entiende que el modelo matemático abstrae correctamente a la realidad para poder resolver el problema, es decir, los valores obtenidos por el software son lógicos y corresponden a lo que se debería de obtener en un ejercicio analítico directo. No obstante, al contemplar en esta instancia simplificada tan sólo una fábrica y un almacén, algunas restricciones también fueron simplificadas. Por tanto, al expandir el modelo en el siguiente capítulo a más de una planta y almacén se obtuvieron restricciones adicionales para garantizar el funcionamiento del propio modelo. Estas restricciones ya están enunciadas en el planteamiento descrito previamente en este capítulo y corresponden con las restricciones 4 (complemento), 10 y 11.

Por lo tanto, con esta instancia se verifica y se valida el modelo de optimización planteado para el problema de producción-ruteo, resolviendo también la secuenciación de manufactura con un acercamiento de flow-shop scheduling.

2.10 Escalabilidad del modelo de optimización

Con el objetivo de entender la aplicación y escalabilidad que tiene el modelo anteriormente propuesto se puede plantear una cadena de suministro de escala nacional con diferentes plantas de producción a lo largo del país, así como otros tantos almacenes ubicados en las principales ciudades que llevarán el producto a los puntos de venta finales. En esta cadena de suministro hay una lucha entre cuáles plantas tienen costos de producción más baratos y al mismo tiempo qué plantas están más cercanas a los almacenes, donde la demanda es agregada. De tal modo, la planeación de la producción resuelve en dónde se debería de producir qué cantidad de producto para garantizar la minimización de estos costos.

Por otra parte, se añade la complejidad de la flexibilidad de la cadena de suministro que se materializa en la cantidad de ajustes que se deben de hacer al tener al pasar de un producto a otro en una máquina. Por lo tanto, ahora hay una lucha entre qué tantos productos se deberían de producir con la intención de tener una cadena de suministro flexible, contra la decisión de tener un número pequeño de productos por planta para minimizar los cambios y tener de esta manera plantas eficientes.

En conclusión, por un lado, el modelo MILP propuesto determina el punto óptimo para decidir entre optimizar los costos de producción vs los costos de transporte. Por otro lado, se decide entre el grado de flexibilidad vs eficiencia que se busca en las plantas. Todas estas decisiones buscan resolver el problema de la coordinación de la cadena de suministro con la programación de la producción que se generaliza como el problema de producción ruteo (PRP).

2.11 Experimentación con metaheurísticas en HeuristicLab

Anteriormente, se había aclarado que el modelo de programación binaria-mixta formulado previamente no resuelve la última etapa en la cadena de suministro, esto es el ruteo desde las plantas de producción hacia los almacenes, que en este caso se comportan como clientes del modelo. El modelo de optimización mencionado devuelve como resultado la asignación de cantidades de producción que cada planta deberá tener, así como el orden que deberá seguir en cada nivel y máquina de producción; así también, el resultado contempla desde qué planta hacia qué almacenes y en qué cantidades se deberá hacer el transporte final, pero no en qué orden o de qué manera.

En consecuencia, es también labor de este trabajo de investigación resolver el VRP para así finalizar la resolución de la programación de la producción y ruteo. Dado que el VRP es un problema de tipo NP-duro en términos de complejidad computacional (Lenstra & Rinnooy-Kan, 1981), es decir, los métodos exactos de resolución pueden ser deficientes ya que el tiempo computacional es exponencial. Por tal motivo, se decidió extraer la complejidad computacional

que viene del VRP del modelo de optimización MILP, para que posteriormente el VRP pueda ser resuelto con un método adecuado.

Por consiguiente, en este trabajo de investigación se optó por hacer una investigación con respecto a los métodos de resolución del VRP. En el artículo de Goel & Maini (2017) se sostiene que, si bien los métodos exactos convergen en el óptimo global, estos generalmente están limitados a un número pequeño de nodos.

Adicionalmente, ya que en este trabajo se busca una metodología que sea escalable para problemas complejos y los métodos exactos se limitarían a problemas con gran magnitud, el siguiente paso es explorar el uso de técnicas heurísticas y metaheurísticas.

Con el objetivo de encontrar un método confiable para la solución del VRP se experimentó inicialmente en el software HeuristicLab que es parte de la iniciativa del Laboratorio de Algoritmos Heurísticos y Evolutivos (HEAL por sus siglas en inglés) de la Universidad de Ciencias Superiores Aplicadas de Austria. HeuristicLab es un software preparado para la resolución de diferentes problemas clásicos de optimización, entre ellos, soporta el VRP. La ventaja que tiene este software es que nos da la opción de experimentar con diferentes algoritmos de resolución y a su vez se pueden variar los hiper-parámetros de estos mismos, asimismo, cuenta con la flexibilidad suficiente para alimentar instancias precargadas o de elección propia bajo una interfaz amigable para el usuario.

Inicialmente, se optó por experimentar con una pequeña instancia predeterminada que cuenta con 31 clientes, en una modalidad de vehículos con una capacidad homogénea fija en la cual se busca obtener la distancia mínima para el sistema completo sin restringir en una cantidad de rutas posibles.

En la Figura 2. Instancia de experimentación para VRP en HeuristicLab, se muestra la instancia mencionada con la dispersión de los clientes representados como puntos negros mientras que el punto azul representa el origen de las rutas.

Figura 2. Instancia de experimentación para VRP en HeuristicLab

Así pues, los algoritmos seleccionados para la experimentación con HeuristicLab obedecen a las opciones que recurrentemente usa la literatura, así como la disponibilidad de la metaheurística en el propio software. Se seleccionaron 5 algoritmos para solucionar la instancia de prueba. En el Anexo 4. Pseudocódigos de metaheurísticas, se muestra lo que sucede detrás de cada metaheurística. Los métodos para probar fueron los siguientes: algoritmo genético, búsqueda tabú, búsqueda dispersa, recocido simulado y búsqueda variable de vecindario.

Además de analizar el desempeño de estas cinco metaheurísticas con la instancia de experimentación, el objetivo de este procedimiento era el de explorar las capacidades de HeuristicLab para confirmarla como herramienta para escalar la experimentación y resolver las instancias más robustas y complejas. Los hiper-parámetros de los algoritmos se establecieron de manera predeterminada, solamente extendiendo la cantidad de iteraciones en algunos casos. La distancia mínima encontrada para cada algoritmo se muestra en la Figura 3. Comparativa metaheurísticas para instancia de experimentación, adicionalmente, en el Anexo 5. Rutas obtenidas en instancia de experimentación con HeuristicLab, se puede ver el resultado de las rutas obtenidas por cada una de la metaheurísticas.

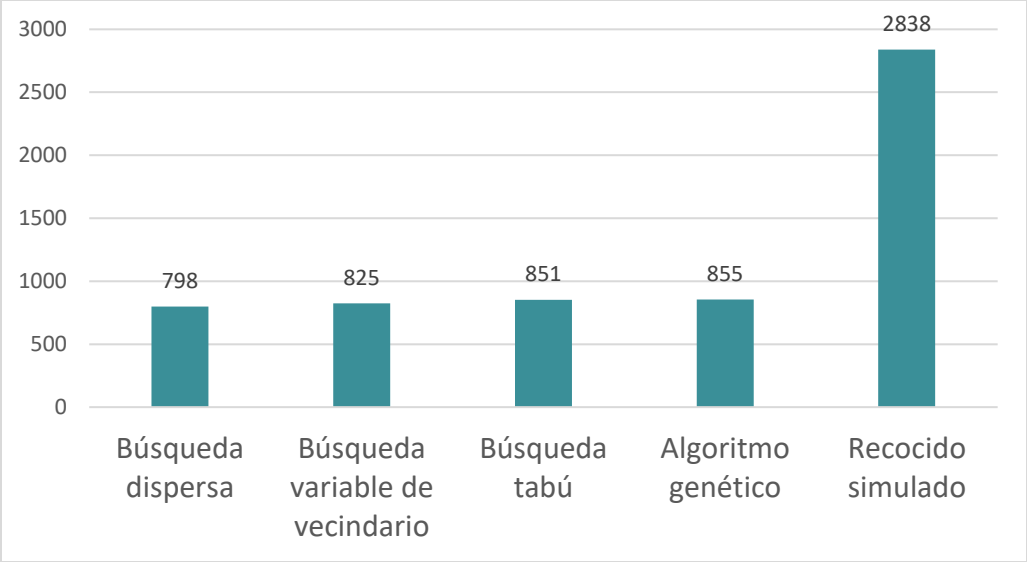


Figura 3. Comparativa metaheurísticas para instancia de experimentación

En consecuencia, por medio de esta primera corrida podemos encontrar un mejor desempeño con el algoritmo de búsqueda dispersa con 798 de valor en la función objetivo, le siguen de cerca (en orden) la búsqueda variable de vecindario, búsqueda tabú y algoritmo genético. Por último, el algoritmo de recocido simulado tuvo un desempeño pobre para esta instancia al obtener 2,838 en la función objetivo. Tras experimentar alterando la temperatura inicial y final, así como la cantidad de iteraciones no se encontró una mejora significativa.

En conclusión, esta primera experimentación hecha en el software HeuristicLab nos da el soporte necesario para que, una vez teniendo el desarrollo de las instancias robustas del problema, se pueda resolver la fase final de ruteo desde las plantas hacia los almacenes con la confiabilidad de que se podrá hacer una comparativa justa de las metaheurísticas ya mencionadas con el objetivo de encontrar los mejores métodos de resolución para el VRP planteado.

Capítulo 3. Desarrollo del Problema y Resultados

Ahora que tenemos las bases teóricas sentadas del primer capítulo y la propuesta de modelo de optimización y métodos de resolución del VRP del capítulo 2; en este capítulo se escala el problema de producción ruteo, se resuelve con técnicas y software de vanguardia y se muestran los resultados y su análisis.

3.1 Metodología de Resolución del Problema e Instancias

Una vez que se han establecido las bases de resolución con el modelo matemático de optimización y la experimentación con software para resolver el VRP a través de HeuristicLab se debe establecer una metodología con la intención de escalar las instancias del PRP y así presentar un análisis completo de este trabajo.

Como se había mencionado previamente, el PRP al tener el VRP incluido se considera como un problema NP-duro y es por esto por lo que en la metodología propuesta se separará esta complejidad para ser resuelta por las técnicas metaheurísticas. Es decir, se resolverá el modelo de optimización con un método exacto para definir las variables que deciden lo que sucede dentro de las plantas y así asignan qué cantidad de producto se ha de producir en cada planta para ser enviada a cada almacén. Una vez que se tenga esta asignación de la producción entonces será tarea de las técnicas metaheurísticas el abordar la complejidad del VRP y hacer el ruteo desde las fábricas hasta los almacenes. Estas variables son las que se deben definir en las cadenas de suministro de tipo VMI, ya que es la misma empresa la que hace la manufactura y la distribución de los productos, en este caso, de los bienes de consumo.

En el trabajo de Absi et al. (2018) se desarrolla una metodología con el objetivo de evaluar el desempeño de un PRP resolviendo con 3 métodos diferentes:

- Resolver primero la fase de producción y luego distribución
- Resolver primero la fase de distribución y luego producción
- Resolver ambas fases en un mismo modelo coordinado

Además, basados en una instancia particular desarrollaron numerosas variantes al modificar dos razones claves en su modelo:

- La razón del costo de producción contra el costo de distribución
- La razón del costo de ajustes en producción contra el costo de mantener inventario

A partir de estas variaciones pudieron analizar cadenas de suministro variadas y concluyen en su trabajo que la optimización que se puede obtener por agrupar todas las variables y parámetros en un sólo modelo coordinado no resulta tan provechoso ya que las heurísticas desaparecen las

ventajas del acercamiento integrado. No obstante, dependiendo de la instancia, en algunos casos la complejidad de las diferentes fases puede darle la ventaja a este acercamiento integrado.

Para el desarrollo de este trabajo se estableció una metodología tomando algunos conceptos del artículo mencionado, la metodología es la siguiente:

1. Generar un problema lo suficientemente robusto para establecerla como estándar en la cual se procederá a variar 2 parámetros:
    a. La razón del costo de producción contra el costo de distribución (RPD)
    b. La proporción horas de ajustes de cambios entre producto contra las horas disponibles de producción (PSD)
2. En la primera fase de resolución se abordará el modelo de programación binaria-mixta a través de Python complementando con la librería del motor de optimización de Gurobi llamada Gurobipy. Se accedió a esta librería con una licencia académica gratuita con duración de un año.
3. Una vez que se tenga la asignación de productos por planta a mandar a cada almacén del punto anterior, la segunda fase de la metodología buscará resolver el VRP para cada una de las plantas con HeuristicLab usando las metaheurísticas:
    a. Algoritmo Genético
    b. Búsqueda dispersa
    c. Recocido simulado
    d. Búsqueda tabú
    e. Búsqueda variable de vecindario
4. Análisis similar al de sensibilidad a partir de las diferentes instancias generadas con las variaciones de parámetros.
5. Comparativa de distancias obtenidas para las cinco metaheurísticas.

Así pues, con esta metodología se busca generalizar los resultados para cadenas de suministro con diferentes estructuras y comportamientos referentes al balance que puede existir entre los costos de producción y distribución, así como la negociación interna que una organización puede tener dentro de una planta en cuanto a eficiencia contra flexibilidad, que se manifiesta en la proporción de las horas de ajuste entre productos contra la cantidad de horas disponibles totales.

En la Figura 4. Diagrama de metodología de resolución del problema, se muestra un diagrama para aclarar la metodología de la resolución del problema con las diferencias de cada una de las fases que mencionaron previamente con sus entradas, variables a controlar, modelo y algoritmos utilizados, software de apoyo y salidas.

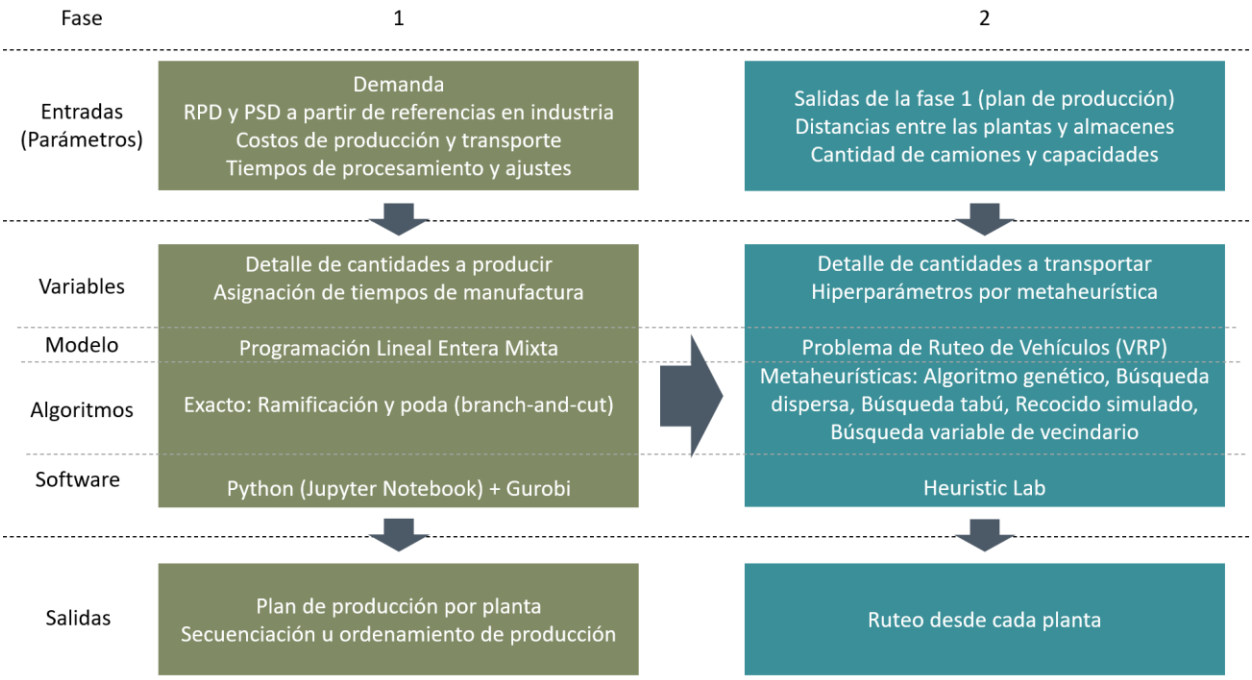


Figura 4. Diagrama de metodología de resolución del problema.

3.2 Definición parámetros RPD y PSD

Con el objetivo de generar las instancias para el modelo de optimización, primeramente, se establecieron los valores de la Razón de costos de Producción contra los costos de Distribución (RPD) y Proporción de tiempos de ajustes (Setup) contra las horas Disponibles (PSD), estos serán los parámetros que cambian a través de las instancias y definirán los resultados de este trabajo.

Para establecer los valores se hizo una investigación de casos reales en la industria de los bienes de consumo, específicamente en la industria de las bebidas y alimentos. Dado que los valores dan un vistazo a la estructura de costos y lo que sucede dentro de las fábricas de estas empresas, estos no se podrán revelar y se mantendrán de manera confidencial. No obstante, los valores de RPD y PSD para la creación de instancias fueron definidos para incluir lo que se observó a partir de estas realidades.

La RPD será calculado como el promedio de los costos de producción c_{jf}^{p} entre el promedio de los costos de transporte c_{fa}^{t}. Para este parámetro se establecieron 5 valores: 4, 2, 1, 0.5 y 0.25. Es decir, se tienen valores que incluyen desde una relación 4:1 de costos de producción vs distribución, hasta el caso contrario en una relación 1:4. Los 5 valores se muestran en la Tabla 2. Valores de RPD para la creación de instancias a continuación:

RPD ID	Valor	Relación
1	4	4:1
2	2	2:1
3	1	1:1
4	0.5	1:2
5	0.25	1:4

Tabla 2. Valores de RPD para la creación de instancias.

Por otra parte, la PSD se calcula como el promedio de todos los tiempos de ajuste $t^{a}_{jj'mnf}$, excluyendo en los casos en que $j = j'$, multiplicado por $j - 1$ para cada planta entre las horas disponibles por máquina HD_{mnf}. Es decir, se busca obtener la cantidad de ajustes por planta, multiplicarlos por el tiempo de ajuste promedio y dividirlo entre las horas disponibles totales. Para la PSD se definieron 4 valores: 0.0, 0.05, 0.10 y 0.15.

Así pues, con la combinación de estos 2 parámetros se tendrán 20 instancias en total (5 RPDs multiplicado por 4 PSDs). De tal forma, para cada una de estas instancias se definirán todos los parámetros restantes a partir de la RPD y PSD para después ser optimizados a través de Gurobipy.

3.3 Proceso de parametrización

Para la definición de cada uno de los parámetros por instancia se estableció el siguiente proceso:

1. Establecer los conjuntos que definen la cantidad de variables, parámetros y restricciones que contiene cada instancia. La cantidad de elementos por conjunto fue obtenida con la intención de poner a prueba el modelo de optimización, esta cantidad de elementos se puede observar en la Tabla 3. Cantidad de elementos por conjunto, con estos conjuntos se obtuvieron 431 parámetros, 1110 variables y 2811 restricciones:

Conjuntos	Cantidad
Productos J	5
Máquinas M	1
Niveles producción N	2
Orden producción O	5
Fábricas F	3
Almacenes A	10

Tabla 3. Cantidad de elementos por conjunto.

2. Determinar los productos que se pueden producir en cada planta, pensando en otorgarle al optimizador un balance entre restricción y flexibilidad que simule la realidad. De esta manera un producto será producido en las tres fábricas, tres productos podrán ser producidos en dos fábricas y el último producto sólo podrá ser producido en una fábrica, como lo muestra la Tabla 4. Posibilidades de producción para cada combinación fábrica-producto:

	Fábrica 1	Fábrica 2	Fábrica 3
Producto 1	✓	✓	✓
Producto 2	✓	✓	
Producto 3	✓		✓
Producto 4		✓	✓
Producto 5			✓

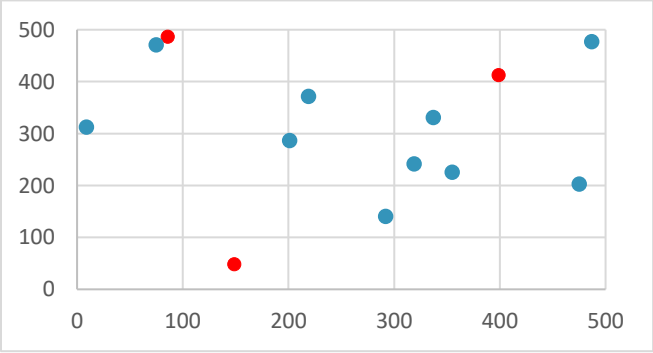
Tabla 4. Posibilidades de producción para cada combinación fábrica-producto.

3. Aleatorizar los costos de producción c_{jf}^{p} por producto con una distribución uniforme, además, para demostrar que unas plantas son más eficientes que otras se multiplicó el parámetro aleatorizado por un factor de eficiencia para cada fábrica, esto se puede observar en la Tabla 5. Factor de eficiencia por fábrica.

	Fábrica 1	Fábrica 2	Fábrica 3
Factor eficiencia	0.8	1.0	1.2

Tabla 5. Factor de eficiencia por fábrica.

4. Aleatorizar las coordenadas de las fábricas y almacenes con una distribución uniforme en un plano cartesiano, las coordenadas que se obtuvieron se pueden observar en la Gráfica 1. Plano cartesiano con coordenadas por sitio, donde las fábricas se pueden distinguir de los almacenes por su color rojo:



Gráfica 1. Plano cartesiano con coordenadas por sitio.

5. Obtener una matriz de distancias euclidianas a partir de las coordenadas y, partiendo de los costos de producción c_{jf}^{p} y la RPD en cada caso, obtener los costos de transporte c_{fa}^{t} según cada variación de las 5 diferentes RPDs. Es decir, afectar los c_{fa}^{t} por un factor tal que el promedio de los c_{jf}^{p} entre el promedio de los c_{fa}^{t} sea igual a la RPD. Se consideraron las distancias euclidianas ya que en este caso las ubicaciones son aleatorizadas, por tanto, representan un ejemplo de la realidad más no agregaría valor el posicionarlos sobre una red vial real. Además, el modelo está hecho con la flexibilidad suficiente para que, si se desean agregar distancias o tiempos particulares, estos se pueden modificar directamente en la matriz de distancias y de esta manera se podrán tomar en cuenta factores como tráfico y distancias reales.
6. Asignar un valor a las horas disponibles HD_{mnf}, en este caso se consideró un periodo semanal con 6 días hábiles y 3 turnos completos, es decir, 6x24=144 horas disponibles para cada máquina.
7. Obtener para cada una de las cuatro PSDs unos tiempos de ajuste ${t^{a}}_{jj,mnf}$. De ahí que, se obtuvieron de manera aleatoria los tiempos de ajuste distribuidos uniformemente y cada instancia se multiplicó por el factor necesario para que se cumpliera con las PSDs establecidos.
8. Aleatorizar con una distribución uniforme los tiempos de procesamiento ${t^{p}}_{jmnf}$.
9. Considerando los ${t^{p}}_{jmnf}$ aleatorios, obtener una demanda D_{ja} tal que el promedio de la utilización de las plantas, tomando el cuello de botella (máquina con la máxima utilización por planta), sea de 85% en los escenarios con PSD = 0.10. Así pues, se espera que las instancias con PSD de 0 y 0.05 tengan utilizaciones menores a 85% y en las instancias de PSD = 0.15 se tenga una utilización promedio >85%.

En suma, tras este proceso de parametrización terminado se obtienen los 431 parámetros para cada una de las 20 instancias. Todos los valores de estos parámetros se pueden ver a detalle en el Anexo 6. Valores de parámetros modelo de optimización.

3.4 Corridas modelo optimización en Gurobipy

Una vez que se tienen definidos los parámetros para correr cada instancia se buscó el software para la resolución del modelo de optimización. Ya que se tienen 1110 variables y 2811 restricciones y los softwares libres no soportan este tamaño de modelo, se buscaron soluciones más completas. La solución que se encontró fue Gurobi, al usarlo de manera académica se puede obtener una licencia por un año sin límites en el tamaño de los modelos. Esta licencia en su versión 10.0.1 fue dada de alta sobre python en su versión 3.10.4 y la creación de los códigos se corrió de manera local en la interfaz de Jupyter Notebook en su versión 6.4.8. Para combinar el motor de optimización de Gurobi sobre código python se importó la librería “gurobipy”.

Por consiguiente, con esta librería se desarrolló el código del modelo de optimización propuesto en este trabajo con la flexibilidad necesaria para sólo cambiar los parámetros necesarios y así generar las 20 instancias sin repetir una cantidad grande de tareas. El código con un ejemplo de corrida de una instancia se puede observar en el Anexo 7. Código Gurobipy para modelo de optimización.

Luego, con el código desarrollado se hicieron las corridas para las 20 instancias. El método de resolución de Gurobi para este modelo, dado que se trata de un modelo de optimización lineal con variables binarias mixtas y se cataloga como MILP (Mixed-Integer Linear Programming), es a través de una implementación avanzada y pionera del algoritmo ramificación y poda (branch-and-cut) desarrollada por Gurobi. Este algoritmo de ramificación y poda es un método exacto propuesto primeramente por Padberg & Rinaldi (1991).

Al resolver con un método exacto podemos esperar la solución óptima para cada corrida a diferencia de las metaheurísticas y aunque se tiene un modelo con un tamaño robusto el tiempo máximo de solución fue de 41.21 segundos y sólo 3 de las 20 corridas tardaron más de 10 segundos. Las corridas fueron hechas con el hardware Intel® Core™ i7-4720HQ CPU @ 2.60GHz y 16GB de memoria RAM. Los detalles por corrida se pueden observar en la Tabla 6.

RPD	PSD	Estatus de corrida
4	0	Explored 18108 nodes (252741 simplex iterations) in 3.38 seconds (2.92 work units)
2	0	Explored 12924 nodes (139550 simplex iterations) in 2.26 seconds (1.44 work units)
1	0	Explored 421 nodes (3913 simplex iterations) in 0.37 seconds (0.19 work units)
0.5	0	Explored 1141 nodes (6866 simplex iterations) in 0.46 seconds (0.27 work units)
0.3	0	Explored 390 nodes (4920 simplex iterations) in 0.50 seconds (0.25 work units)
4	0.05	Explored 25264 nodes (472750 simplex iterations) in 9.31 seconds (6.05 work units)
2	0.05	Explored 16108 nodes (254397 simplex iterations) in 6.82 seconds (4.74 work units)
1	0.05	Explored 603 nodes (7374 simplex iterations) in 0.71 seconds (0.50 work units)
0.5	0.05	Explored 433 nodes (6138 simplex iterations) in 0.54 seconds (0.39 work units)
0.3	0.05	Explored 851 nodes (12313 simplex iterations) in 0.79 seconds (0.58 work units)
4	0.1	Explored 139106 nodes (2545501 simplex iterations) in 41.21 seconds (25.00 work units)
2	0.1	Explored 14937 nodes (263902 simplex iterations) in 8.50 seconds (5.73 work units)
1	0.1	Explored 8614 nodes (109571 simplex iterations) in 3.32 seconds (2.62 work units)
0.5	0.1	Explored 528 nodes (10508 simplex iterations) in 1.49 seconds (0.86 work units)
0.3	0.1	Explored 2899 nodes (43670 simplex iterations) in 1.65 seconds (1.25 work units)
4	0.15	Explored 21164 nodes (429582 simplex iterations) in 9.64 seconds (6.04 work units)
2	0.15	Explored 25379 nodes (488909 simplex iterations) in 11.11 seconds (7.55 work units)
1	0.15	Explored 10601 nodes (157436 simplex iterations) in 6.41 seconds (4.51 work units)
0.5	0.15	Explored 17837 nodes (375433 simplex iterations) in 17.67 seconds (10.30 work units)
0.3	0.15	Explored 14473 nodes (261627 simplex iterations) in 7.38 seconds (7.19 work units)

Tabla 6. Estatus de corridas gurobipy para cada instancia.

3.5 Indicadores e hipótesis modelo de optimización

Posteriormente, una vez hechas las corridas se extrajeron los resultados para las 20 instancias y fueron procesados para obtener indicadores que ayudaron tanto a la verificación del código en gurobipy como para el análisis de los resultados. Los indicadores que se desarrollaron fueron son los siguientes:

- Costo producción: componente de producción en la función objetivo con el costo compuesto por la suma de los costos de todas las fábricas y todos los productos.
- Costo distribución: llamado también costo de transporte, indica el costo de enviar la producción desde todas las fábricas hacia todos los almacenes.
- Costo total: suma aritmética de los costos de producción y distribución, esta es la función objetivo del modelo que se busca minimizar.
- Suma de tiempos de procesamiento: considera la suma de los tiempos en que todas las máquinas estuvieron procesando productos en todas las fábricas.
- Porcentaje de procesamiento: toma la suma de tiempos de procesamiento y la divide entre la cantidad total de horas disponibles para obtener un porcentaje de tiempos de procesamiento.
- Suma tiempos de ajustes: considera la suma de los tiempos en que se tuvo que hacer un ajuste (setup) para cambiar desde un producto hacia otro en todas las máquinas y todas las fábricas.
- Porcentaje de ajustes: similar al porcentaje de procesamiento, se toma la suma de tiempos de ajustes y se divide entre el total de horas disponibles.
- Utilización promedio: se suma el porcentaje de procesamiento con el porcentaje de ajustes, luego, ya que se tienen dos máquinas en línea por planta; se toma el cuello de botella (mayor utilización) por planta y se promedian las utilizaciones de las tres fábricas.
- Suma de distancias: se obtiene al multiplicar la distancia euclidiana desde cada fábrica hacia cada almacén por la cantidad de producto a transportar para esa combinación, posteriormente, se hace una sumatoria de todas las distancias obtenidas.

De tal forma, estos indicadores fueron obtenidos para cada una de las corridas y previo al análisis de los resultados se plantearon algunas hipótesis a probar con este modelo y la metodología de instancias desarrollada. Las hipótesis son:

1. El costo de producción deberá incrementar en las instancias que se tenga una RPD baja y una PSD alta.
2. El costo de transporte deberá incrementar al tener RPD alta por definición.
3. A partir de las dos hipótesis anteriores se deduce que el costo total deberá incrementar al tener RPDs bajas.

4. Los tiempos de procesamiento deberán incrementar conforme se tiene una RPD alta.
5. Los tiempos de ajustes deberán incrementar por definición al tener PSDs altas.
6. La utilización promedio incrementa en mayor medida en instancias con PSD altas y en menor medida con RPDs bajas.
7. La suma de distancias deberá disminuir conforme la RPD disminuye. Por otra parte, la PSD también podría aportar para que a menor PSD se tenga una distancia menor.

3.6 Resultados de corridas del modelo de optimización

Con la intención de visualizar los indicadores mencionados de una mejor manera se creó una matriz con escala de colores para cada indicador de esta primera fase de la metodología. Esta matriz tiene cinco filas y cuatro columnas donde cada fila corresponde a un valor de RPD y cada columna es una PSD. De esta manera podemos evaluar las 20 instancias para cada indicador en una matriz 5x4. La escala de colores se asignó a manera de "semáforo" donde el verde tendrá el menor valor, amarillo será para el valor con percentil 50 y el rojo se asigna al mayor valor; los colores intermedios van de acuerdo con esta escala. Las matrices para cada indicador se muestran a continuación:

Costo producción		PSD			
		0	0.05	0.1	0.15
RPD	4	1,372.83	1,378.53	1,381.53	1,391.92
	2	1,387.30	1,391.45	1,390.40	1,391.92
	1	1,400.29	1,400.60	1,398.23	1,411.80
	0.5	1,426.53	1,432.30	1,438.06	1,436.67
	0.25	1,426.53	1,432.30	1,438.06	1,444.45

Tabla 7. Matriz de color para indicador de costo de producción.

Costo transporte		PSD			
		0	0.05	0.1	0.15
RPD	4	315.19	313.48	314.92	313.02
	2	611.07	609.13	617.55	626.05
	1	1,200.63	1,207.02	1,222.70	1,226.50
	0.5	2,369.13	2,380.00	2,392.54	2,412.75
	0.25	4,738.27	4,760.00	4,785.08	4,815.82

Tabla 8. Matriz de color para indicador de costo de transporte.

Costo total		PSD			
		0	0.05	0.1	0.15
RPD	4	1,688.02	1,692.02	1,696.45	1,704.95
	2	1,998.37	2,000.58	2,007.95	2,017.97
	1	2,600.92	2,607.63	2,620.94	2,638.30
	0.5	3,795.66	3,812.30	3,830.60	3,849.41
	0.25	6,164.80	6,192.30	6,223.14	6,260.26

Tabla 9. Matriz de color para indicador de costo de total.

Suma tiempos procesamiento		PSD			
		0	0.05	0.1	0.15
RPD	4	571.44	570.27	569.48	565.35
	2	565.95	563.87	564.67	565.35
	1	563.03	560.97	561.63	562.19
	0.5	559.75	560.39	561.03	561.77
	0.25	559.75	560.39	561.03	556.49

Tabla 10. Matriz de color para indicador de suma de tiempos de procesamiento.

% tiempos procesamiento		PSD			
		0	0.05	0.1	0.15
RPD	4	66.14%	66.00%	65.91%	65.43%
	2	65.50%	65.26%	65.36%	65.43%
	1	65.17%	64.93%	65.00%	65.07%
	0.5	64.79%	64.86%	64.93%	65.02%
	0.25	64.79%	64.86%	64.93%	64.41%

Tabla 11. Matriz de color para indicador de porcentaje de tiempos de procesamiento.

Suma tiempos ajustes		PSD			
		0	0.05	0.1	0.15
RPD	4	-	21.3	56.6	86.8
	2	-	26.6	60.0	87.5
	1	-	38.4	60.8	95.6
	0.5	-	35.7	70.3	77.6
	0.25	-	29.4	64.1	88.9

Tabla 12. Matriz de color para indicador de suma de tiempos de ajustes.

% tiempo usado en ajustes		PSD 0	0.05	0.1	0.15
RPD	4	0.00%	2.47%	6.55%	10.05%
	2	0.00%	3.08%	6.94%	10.13%
	1	0.00%	4.45%	7.04%	11.07%
	0.5	0.00%	4.13%	8.14%	8.99%
	0.25	0.00%	3.40%	7.42%	10.29%

Tabla 13. Matriz de color para indicador de porcentaje de tiempo usado en ajustes.

Utilización promedio		PSD 0	0.05	0.1	0.15
RPD	4	77.99%	80.53%	83.53%	85.75%
	2	78.54%	80.71%	83.92%	85.75%
	1	78.19%	81.77%	85.19%	86.24%
	0.5	78.60%	82.01%	84.95%	87.45%
	0.25	78.60%	82.01%	84.70%	87.18%

Tabla 14. Matriz de color para indicador de utilización promedio.

Suma de distancias		PSD 0	0.05	0.1	0.15
RPD	4	130,223	129,516	130,108	129,327
	2	126,233	125,833	127,572	129,327
	1	124,011	124,672	126,291	126,684
	0.5	122,352	122,913	123,561	124,604
	0.25	122,352	122,913	123,561	124,354

Tabla 15. Matriz de color para indicador de suma de distancias.

3.7 Análisis de resultados modelo de optimización

Para hacer el análisis de resultados se desarrolló una matriz de correlaciones de Pearson que nos dará un valor objetivo que relacione los parámetros RPD y PSD contra los indicadores desarrollados. El coeficiente de correlación de Pearson (PCC) (Pearson, 1895) es un valor continuo que va desde -1 hasta 1 y determina si dos variables están relacionadas linealmente. Una correlación con valor de 1 indica una relación positiva o directa perfecta, un valor de -1 indica que se tiene una correlación negativa o inversa perfecta, un valor de 0 indica que no hay correlación alguna. En este caso se desprecian las correlaciones de porcentajes de procesamiento y ajustes ya que se contiene la misma información de correlación a partir de la suma de tiempos de procesamiento y ajustes.

Para esta matriz se desarrolló un código de colores donde el color verde se asigna a una correlación directa con valor de 1, el color neutro se asigna al 0 que tiene correlación nula y por último el color rojo se asigna a la correlación inversa de -1. Ya que queremos evaluar el impacto de las variaciones de RPD y PSD en los indicadores, sólo nos enfocaremos en las primeras dos columnas de la Tabla 16. Matriz de correlación para corridas de modelo de optimización:

	RPD	*PSD*	*Costo producción*	*Costo transporte*	*Costo total*	*Tiempos procesa-miento*	*Tiempos ajuste*	*Utilización promedio*	*Suma distancias*
RPD	1.00								
PSD	0.00	1.00							
Costo producción	-0.86	0.20	1.00						
Costo transporte	-0.75	0.01	0.87	1.00					
Costo total	-0.76	0.01	0.87	1.00	1.00				
Tiempos procesamiento	0.92	-0.15	-0.87	-0.75	-0.76	1.00			
Tiempos ajuste	-0.06	0.99	0.24	0.04	0.04	-0.20	1.00		
Utilización promedio	-0.15	0.98	0.34	0.13	0.14	-0.28	0.98	1.00	
Suma distancias	0.91	0.28	-0.83	-0.76	-0.76	0.86	0.22	0.12	1.00

Tabla 16. Matriz de correlación para corridas de modelo de optimización.

A través de esta matriz de correlaciones ponemos a prueba las hipótesis antes descritas y podemos obtener conclusiones. La prueba de hipótesis es de la siguiente manera:

1. Con una correlación inversa de -0.86 se demuestra que el costo de producción aumenta conforme se tienen valores de RPD bajas ya que se prioriza la distribución en estos casos. De igual manera, con una correlación muy baja de 0.20 se puede observar que el costo de producción aumenta conforme aumenta la PSD, aunque en menor medida, esto se explica por la saturación de las máquinas teniendo menor flexibilidad para buscar mejores soluciones en términos de producción. Estos resultados se ejemplifican en la Tabla 7. Matriz de color para indicador de costo de producción.
2. También podemos observar que se tiene una correlación negativa de -0.75 de RPD vs costo de transporte, esto sabíamos que por definición al disminuir la RPD los costos de transporte deberían de subir. Para demostrar que el modelo prioriza el transporte y no sólo de manera artificial tendremos que evaluarlo con las corridas posteriores de escenarios de VRP, adicionalmente, el indicador de distancia total nos dará más información. Los costos de transporte se pueden ver en la Tabla 8. Matriz de color para indicador de costo de transporte.
3. También afirmamos que la RPD seguirá la misma relación inversa contra el costo total ya que se compone de los costos de producción y transporte, en esta ocasión la correlación inversa es de -0.76. Los resultados relacionados al costo total están disponibles en la Tabla 9. Matriz de color para indicador de costo de total.

4. Por añadidura, podemos comprobar que los tiempos de procesamiento van altamente relacionados al RPD con una correlación de 0.92, es decir, a mayor RPD se tienen mayores tiempos de procesamiento. Esto es debido a que cuando se tienen costos altos de producción unitarios el optimizador elige los costos más bajos sin importar el tiempo que demore. Adicionalmente, con una correlación muy baja e inversa de -0.15 se puede decir que el tener tiempos de ajuste altos (PSD) hace que disminuya, aunque en factor bajo, los tiempos de procesamiento ya que se necesita tiempo adicional para los ajustes y se prefiere reducir el procesamiento. Esto se muestra en la Tabla 10. Matriz de color para indicador de suma de tiempos de procesamiento y en la Tabla 11. Matriz de color para indicador de porcentaje de tiempos de procesamiento.
5. Por otra parte, también por definición observamos cómo a mayor PSD los tiempos de ajuste aumentan casi con una relación perfecta, en este caso con una correlación de 0.99. Lo que es destacable es los porcentajes de tiempos de ajuste siempre son menores al PSD, por ejemplo, cuando se tiene un PSD de 0.15 el escenario con mayor % de ajuste es de 11%. Esto se da por el comportamiento estocástico asignado a los tiempos de ajuste que se distribuye de manera uniforme, es en esta variabilidad que el optimizador encuentra oportunidades para reducir los tiempos de ajuste en cada instancia. Esto se observa en la Tabla 12. Matriz de color para indicador de suma de tiempos de ajustes y en la Tabla 13. Matriz de color para indicador de porcentaje
6. Además, con respecto a la utilización promedio se verifica que la PSD influye en mayor medida con una correlación de 0.98 en la utilización promedio ya que agrega horas de tiempos de ajuste, saturando las máquinas. De otro modo, en menor medida la RPD tiene una correlación baja e inversa de -0.15 que indica que a menor RPD la utilización promedio aumenta ya que la prioridad cambia de la producción al transporte. Por añadidura, se resalta que conforme se había establecido previamente, las utilizaciones promedio para el caso de PSD=0.10 están alrededor (un poco por debajo por la labor del optimizador) del 85%. Agregado a lo anterior, las instancias con menores PSDs baja este factor de utilización, mientras que en las instancias con PSD=0.15 la utilización sí es mayor a 85%. Los resultados relacionados a las utilizaciones están disponibles en la Tabla 14. Matriz de color para indicador de utilización promedio.
7. Por último, para la suma de distancias tenemos una fuerte correlación positiva de 0.91 contra la RPD, lo que indica que en las instancias donde el costo de transporte es elevado el optimizador busca reducir la distancia total de los productos a transportar desde cada fábrica hacia cada almacén. Asimismo, al tener una correlación baja de 0.28 se puede observar que en una proporción pequeña al tener tiempos de ajustes altos se aumenta la distancia ya que se saturan más las máquinas dejando menor oportunidad de

optimización en transporte. Esto se encuentra en la Tabla 15. Matriz de color para indicador de suma de distancias.

3.8 Resolución VRP: Proceso de corridas en HeuristicLab

Al término del proceso de corridas del modelo de optimización con gurobipy una de las variables de decisión que se obtienen como resultado es la cantidad de producto a transportar de cada fábrica hacia cada almacén. Así pues, como se mencionó previamente, se tiene ya asignada la producción por planta y ahora el problema de esta segunda fase reside en la manera que se hará el transporte hacia cada almacén llamado también ruteo. Es en este punto en el que el software HeuristicLab en su versión 3.3.16.17186 fue usado, a través de este software se pueden plantear instancias de VRP (en este caso particular se utilizó el CVRP) para después ser resueltas por diferentes metaheurísticas.

La información que se alimentó como entrada al software fueron las unidades a transportar desde cada planta hacia cada almacén x_{jfa}^{t}, así como las coordenadas de los sitios generados de manera aleatoria. Además, se ingresan las cantidades de vehículos por fábrica, así como su capacidad en unidades. Para obtener estos parámetros se buscó que las corridas en promedio tuvieran una utilización de vehículos del 80% y de este modo se definieron 5 vehículos por planta con capacidad de 50 unidades. Estos supuestos académicos están basados en que la industria generalmente maneja una utilización menor en vehículos comparado a la utilización de las plantas, que para el modelo de optimización se definieron en 85%. No obstante, la cantidad de vehículos y su capacidad pudiese cambiar para acoplarse a cualquier organización y esta metodología no se vería afectada.

Asimismo, una vez que se definieron los parámetros para cada corrida en HeuristicLab se separaron las corridas de acuerdo con cada fábrica ya que de ellas se crean y parten las rutas, por tanto, el proceso de una no afecta a las demás y se pueden considerar de manera aislada. En consecuencia, para cada una de las 20 instancias de gurobipy se deberán de hacer 3 corridas para obtener así 60 instancias diferentes del VRP. Más aún, como se había mencionado previamente, uno de los objetivos de este trabajo es establecer una comparativa entre métodos de resolución; en este caso, se resolverá cada una de las instancias por cada uno de las cinco metaheurísticas seleccionadas y explicadas con anterioridad. En suma, fueron 20x3x5=300 corridas las que se hicieron a través de este software para resolver la sección del VRP. Para recapitular, las cinco metaheurísticas con las que se resolvieron las 60 instancias, y sus siglas en inglés que serán usadas como identificación, se muestran a continuación:

i. Algoritmo Genético (GA)
ii. Búsqueda Dispersa (SS)

iii. Búsqueda Tabú (TS)
iv. Recocido Simulado (SA)
v. Búsqueda Variable de Vecindario (VNS)

En este caso, se enfocan los resultados en sólo dos indicadores que nos ayudarán a evaluar las corridas:

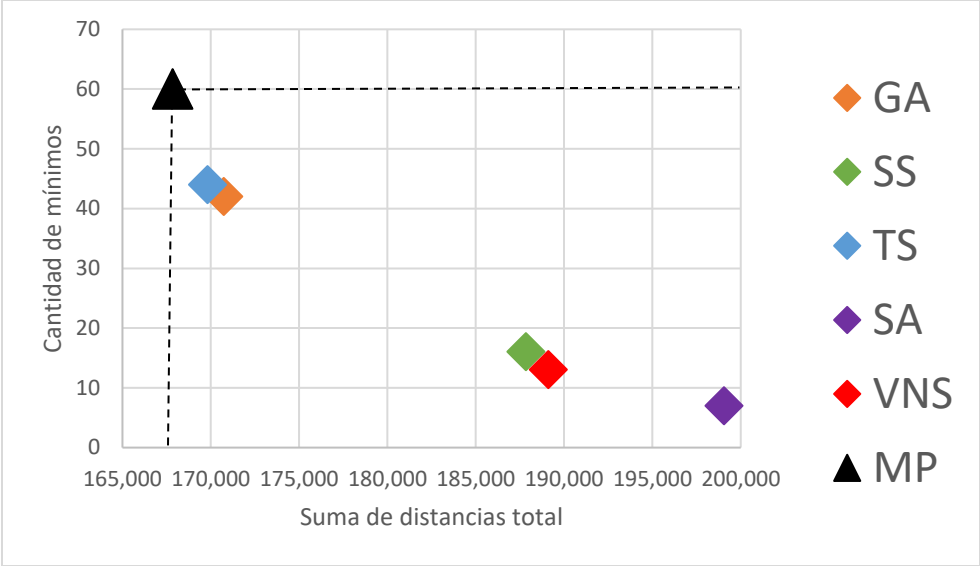
1. Distancia total: suma de las distancias de los recorridos de todos los vehículos tras hacer el ruteo.
2. Cantidad de mínimos: tomará valor de 1 si la solución fue la de menor distancia contra los otros métodos, tendrá valor de 0 en caso contrario.

Adicionalmente, para cada instancia se tomó el valor mínimo de los cinco métodos y se catalogó ese dato como el mejor posible (MP). Con estos resultados se hicieron las evaluaciones de los métodos de resolución.

3.9 Resultados de corridas HeuristicLab

Tras la ejecución de las 300 corridas en HeuristicLab, adicionalmente a los dos indicadores ya mencionados se despliega como resultado un plano cartesiano con las rutas hechas para cada fábrica para los 5 vehículos disponibles. Un ejemplo de este resultado se puede observar en el Anexo 8. Ejemplo de ruteo en HeuristicLab.

A partir de los dos indicadores mencionados se graficó por cada uno de los cinco métodos, más la mejor opción posible (MP), los resultados se pueden ver en la Gráfica 2. Comparativa de resultados del VRP por metaheurística. En el eje “X” tenemos la suma de las distancias obtenidas por cada metaheurística para las 60 instancias (3 fábricas por cada instancia del modelo de optimización), en el eje “Y” tenemos la cantidad de veces que cada método fue el mejor para cada corrida. Así pues, el triángulo negro en la gráfica establece una asíntota vertical (línea punteada) con la menor distancia posible, así como una asíntota horizontal con la mayor cantidad de mejores soluciones posible. De esta manera, los rombos que estén más cercanos al triángulo representarán mejores métodos de resolución.



Gráfica 2. Comparativa de resultados del VRP por metaheurística.

Adicionalmente, se construyó para cada solución la matriz con escala de colores con la suma de las distancias de las tres fábricas para cada una de las 20 instancias, las matrices se obtienen a partir de los 5 valores para la RPD (filas) y los 4 valores para la PSD (columnas). Al igual que las escalas de colores descritas con anterioridad, podemos observar en color verde los valores menores y en rojo los mayores para la suma de distancias.

		PSD			
MEJOR POSIBLE		0	0.05	0.1	0.15
RPD	4	8,819	8,719	8,717	8,454
	2	8,150	8,371	8,296	8,454
	1	8,306	8,276	8,464	8,715
	0.5	8,117	8,601	8,081	8,518
	0.25	8,117	8,601	8,081	7,989

Tabla 17. Suma de distancias para la mejor opción posible entre las metaheurísticas.

		PSD			
GA		0	0.05	0.1	0.15
RPD	4	9,102	8,719	8,717	8,630
	2	8,655	8,521	8,666	8,630
	1	8,364	8,308	8,464	8,715
	0.5	8,166	8,601	8,599	8,518
	0.25	8,166	8,601	8,599	7,989

Tabla 18. Suma de distancias para metaheurística de algoritmo genético.

		PSD			
	SS	0	0.05	0.1	0.15
RPD	4	9,625	9,211	9,284	9,501
	2	9,108	9,106	9,501	9,501
	1	9,453	9,240	9,398	9,403
	0.5	9,204	9,534	9,534	9,534
	0.25	9,204	9,534	9,534	9,436

Tabla 19. Suma de distancias para metaheurística de búsqueda dispersa.

		PSD			
	TS	0	0.05	0.1	0.15
RPD	4	8,819	8,719	8,717	8,630
	2	8,561	8,521	8,635	8,630
	1	8,364	8,308	8,464	8,715
	0.5	8,166	8,601	8,109	8,518
	0.25	8,166	8,601	8,109	8,480

Tabla 20. Suma de distancias para metaheurística de búsqueda tabú.

		PSD			
	SA	0	0.05	0.1	0.15
RPD	4	10,750	10,702	9,988	9,033
	2	10,248	10,831	8,866	9,033
	1	9,661	10,121	9,938	10,527
	0.5	11,453	10,186	8,845	9,109
	0.25	11,453	10,186	8,845	9,282

Tabla 21. Suma de distancias para metaheurística de recocido simulado.

		PSD			
	VNS	0	0.05	0.1	0.15
RPD	4	10,435	9,058	9,010	10,210
	2	9,394	9,593	9,529	10,210
	1	8,999	9,448	10,360	9,086
	0.5	8,414	10,269	9,030	9,964
	0.25	8,414	10,269	9,030	8,398

Tabla 22. Suma de distancias para metaheurística de búsqueda variable de vecindario.

3.10 Análisis de resultados VRP

Para comenzar a analizar los resultados de esta segunda fase de la metodología del problema, se deben distinguir los resultados obtenidos por los métodos exactos de las heurísticas o metaheurísticas. Con un método exacto se garantiza obtener un óptimo global, en este caso sería

una distancia mínima global, el inconveniente de los métodos exactos son su tiempo de ejecución para problemas complejos como lo es el VRP. En cambio, las heurísticas y metaheurísticas buscan acelerar el tiempo de ejecución sacrificando una solución que podría resultar ser óptima. Teniendo este contexto de los métodos de resolución se procede a hacer el análisis de los resultados obtenidos.

Primeramente, haciendo la comparativa entre métodos de resolución con la gráfica mostrada previamente podemos hacer un ranking entre las metaheurísticas. Donde la mejor solución tiene la menor suma de distancias, así pues, el ranking quedaría de la siguiente manera:

1. Búsqueda Tabú (TS)
2. Algoritmo Genético (GA)
3. Búsqueda dispersa (SS)
4. Búsqueda Variable de Vecindario (VNS)
5. Recocido Simulado (SA)

En la misma línea, durante el proceso de las corridas en HeuristicLab se observó que las soluciones obtenidas por la búsqueda tabú y el algoritmo genético eran muy cercanas, como lo muestra la Gráfica 2. Comparativa de resultados del VRP por metaheurística. En 14 de las 20 instancias estos dos métodos llegaron exactamente a la misma solución. Por consiguiente, se puede afirmar que es marginal la ventaja de la búsqueda tabú sobre el algoritmo genético y por lo tanto ambas metaheurísticas son ampliamente recomendadas para la solución de problemas de VRP.

En contraposición, los otros tres métodos: la búsqueda dispersa, búsqueda variable de vecindario y recocido simulado; encontraron en general peores soluciones. No obstante, en ciertos casos estos métodos encontraron también soluciones mejores a los demás métodos por lo que no se puede despreciar el valor que aportan en lo individual. Incluso, se hizo una pequeña experimentación para contestar la pregunta: si sólo se tuviesen dos métodos de solución, ¿qué combinación sería la mejor? Se pensaría que al tomar las dos mejores soluciones (búsqueda tabú y algoritmo genético) se obtendría el mejor par de métodos, pero en este caso el mejor par viene de la búsqueda tabú junto al recocido simulado, es decir, el mejor y el peor método en lo individual. Esto se da porque el método de recocido simulado obtiene soluciones tan alejadas de la búsqueda tabú que cuando se acierta y regresa la menor distancia contra sus contrapartes esto lo hacía por un amplio margen. En conclusión, se recomienda en lo particular la búsqueda tabú y el algoritmo genético, sin embargo, las cinco metaheurísticas tienen valor agregado en lo individual.

Además de obtener una comparativa de métodos de resolución, el siguiente objetivo del proceso de corridas en HeuristicLab, recordando los resultados de la optimización en Gurobipy; es el de determinar que en los casos donde se tienen costos de transporte alto entonces la distancia total debería de ser menor. Esto debido a que el modelo de optimización debería de asignar producción a las fábricas que tienen más cercanía a los almacenes independientemente de su costo de producción.

Agregado a lo anterior, se corrió una matriz de correlación en este caso de los dos parámetros variantes en el modelo de optimización, RPD y PSD, contra las distancias obtenidas tanto en el optimizador, que representa sólo un indicador de suma de distancias sin aplicación real de distribución; como las distancias obtenidas por HeuristicLab que vienen tras el proceso de ruteo y nos dan una suma de distancias de todos los vehículos. Estas correlaciones se pueden observar en la Tabla 23. Matriz de correlación para parámetros vs distancias.

	RPD	*PSD*	*Distancias Hlab*	*Distancias Gurobipy*
RPD	1.00			
PSD	-	1.00		
Distancias Hlab	0.57	0.08	1.00	
Distancias Gurobipy	0.91	0.28	0.60	1.00

Tabla 23. Matriz de correlación para parámetros vs distancias.

Como se había analizado previamente, las distancias obtenidas en Gurobipy tienen una correlación fuerte con la RPD y PSD. En la misma línea y poniendo en comparativa, al tener una correlación de 0.57 podemos afirmar que hay una correlación positiva entre la RPD y las distancias del ruteo en HeuristicLab. De tal modo, al tener RPDs menores (costos de transporte altos) se obtendrán distancias menores también. Por otra parte, para el caso de la PSD, al tener una correlación de tan sólo 0.08 no podemos afirmar una correlación significativa.

Por último, podemos afirmar que al usar métodos exactos se correlaciona mejor el factor del costo de transporte, reflejado en la RPD, contra la distancia con un coeficiente de Pearson de 0.91. En contraste, las metaheurísticas al no garantizar la solución óptima global, por ganar en tiempo de ejecución, pierden un poco la fuerza en la correlación, lo cual hace sentido con la propia naturaleza de cada tipo de método de solución. En conclusión, esto demuestra que la metodología desarrollada en la que se dividió el problema de coordinación de la cadena de suministro en dos fases para abordar cada una con los métodos de resolución adecuados fueron consistentes con lo esperado.

Conclusiones

Aplicación de la metodología en la industria

Tras demostrar la validez y aplicabilidad de la metodología y modelos propuestos se deben obtener conclusiones con respecto a lo que nos dicen los resultados de las corridas hechas. En estos resultados no se busca obtener que un escenario o una instancia sea mejor que otra, más bien, se busca entender el comportamiento de los costos y los tiempos de ajuste con respecto a las decisiones tomadas en la cadena de suministro.

En conclusión, cuando las organizaciones tienen una estructura de costes tal que los costos de producción son más altos con respecto al transporte entonces al hacer la programación de la producción se debe velar por obtener las mejores combinaciones de costos de manufactura para las combinaciones fábrica vs producto, en otras palabras, se debe producir en donde sea más barato. En contraposición, cuando el costo predominante es el de transporte entonces toma menor relevancia las restricciones que se puedan tener dentro de las plantas y más bien se buscará producir en los sitios que estén más cercanos al punto de demanda o centro de almacenamiento.

Por otra parte, se muestra cómo los tiempos de ajuste agregan complejidad dentro de las plantas de producción. En consecuencia, el minimizar estos tiempos y seleccionar la mejor opción de ordenamiento de producción es de alto valor para las organizaciones para hacer más eficiente la operación y así minimizar el costo de manufactura.

Del cumplimiento de los objetivos

Inicialmente se plantearon 4 objetivos principales que han sido resueltos a lo largo del trabajo presentado.

1. En el capítulo 2 se presentó un modelo de programación lineal entera mixta para la resolución del PRP para una cadena de suministro de tipo VMI lo suficientemente flexible para considerar *f* fábricas y *j* productos. En dicho modelo se hace la minimización de los costos de la cadena de suministro compuestos por el costo de producción y transporte.
2. En este mismo modelo se tiene un acercamiento de *flow shop scheduling* en el que se consideran *n* niveles de producción por las cuales los productos pasan de manera secuencial, compuestas de *m* máquinas limitadas por la cantidad de horas disponibles. Adicionalmente, se resuelve la restricción de tiempos de ajuste detonados por un cambio de producto que se comporta bajo un esquema desde-hacia.
3. Se presentó una comparativa de metaheurísticas para la resolución del problema de ruteo de vehículos (VRP) destacando los métodos con mejores resultados como lo fueron el algoritmo genético y la búsqueda dispersa.

4. En el capítulo 3 se generó una instancia robusta la cual fue expandida a 20 instancias al variar la RPD definida como la razón de los costos de producción contra los costos de distribución, asimismo, se agregó un segundo parámetro que denota la proporción de las horas de ajuste contra las horas disponibles llamado PSD, con los resultados obtenidos se demostró la validez del modelo y la metodología propuesta.

En suma, los objetivos planteados al inicio de este trabajo fueron cumplidos en su totalidad, haciendo capaz la programación de la producción a través de un modelo para la resolución del problema de producción-ruteo (PRP) resuelto con métodos exactos para después plantear un VRP que fue resuelto con técnicas metaheurísticas y comprobado con 20 instancias con un grado importante de complejidad.

De la metodología del problema

En adición a los cumplimientos de los objetivos, uno de los aportes esenciales de este trabajo de investigación es la metodología desarrollada para abordar el PRP. En la primera fase, se resuelve la programación y asignación de la producción con el modelo propuesto de programación lineal entera mixta (MILP), el cual al ser resuelto con el paquete de Gurobipy se resuelve de manera exacta, obteniendo soluciones óptimas a cada instancia.

Por otra parte, la segunda fase de esta metodología fue deliberadamente apartada del modelo de optimización porque es ahí donde radicaba la complejidad computacional en el ruteo de vehículos (VRP). Por tanto, por medio de la resolución con técnicas metaheurísticas se obtienen mejores tiempos de corrida de lo que se hubiese obtenido si se hubiera incluido el VRP en el modelo de optimización, asegurando así la escalabilidad y aplicación de este modelo para las organizaciones de tipo VMI que en su mayoría se compone de bienes de consumo.

Por consiguiente, con esta metodología de dos fases usando software y paquetes de vanguardia como lo son Gurobipy y HeuristicLab se hace una coordinación tanto de la fase de manufactura o producción junto con la fase de transporte, para remover los silos que usualmente se construyen en las organizaciones y entregar una solución unificada.

Líneas de investigación

Por último, se estableció previamente que el modelo de optimización propuesto pretende resolver el ordenamiento de producción para productos con pasos consecutivos que no pueden regresar en el proceso, llamado *flow shop scheduling*, acoplándose a los bienes de consumo que son enfoque de este trabajo. Por tanto, queda abierta la línea de investigación para establecer un modelo de optimización que se acople a los procesos productivos de tipo *job shop scheduling*.

Referencias Bibliográficas

Absi, N., Archetti, C., Dauzère-Pérès, S., Feillet, D. (2015). A Two-Phase Iterative Heuristic Approach for the Production Routing Problem. Transportation Science, Maritime Transportation and Port Logistics, pp. 721-1005.

Absi, N., Archetti, C., Dauzère-Pérès, S., Feillet, D., Speranza, M. G. (2018). Comparing sequential and integrated approaches for the production routing problem. European Journal of Operational Research 269, 633–646.

Adulyasak, Y., Cordeau, J. F., Jans R. (2015). The production routing problem: A review of formulations and solution algorithms. Computers & Operations Research 55, 141–152.

Avci, M. and Yildiz, S. T. (2020). A mathematical programming-based heuristic for the production routing problem with transshipments. Computers and Operations Research 123, 105042.

Bell, W. J., Dalberto, L. M., Fisher, M. L., Greenfield, A. J., Jaikumar, R., Kedia, P., Mack, R. G., Prutzman, P. J. (1983). Improving the Distribution of Industrial Gases with an On-Line Computerized Routing and Scheduling Optimizer. Interface, Dec. 1983, Vol. 13, No. 6, CPMS/TIMS Prize Papers, pp. 4-23.

Braekers, K., Ramaekers, K., Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering, Volume 99, Pages 300-313. ISSN 0360-8352.

Brown, G., Keegan, J., Vigus, B., Wood, K. (2001). The Kellogg company optimizes production, inventory, and distribution. Interfaces, 31(6):1–15.

Chandra P., Fisher M. (1994). Coordination of production and distribution planning. European Journal of Operational Research, 72(3):503–17.

Chapman, S N. (2006). The fundamentals of production planning and control. ISBN 0-13-017615-X.

Choi, K., Dai, J., Song, J. (2004). On measuring supplier performance under vendor-managed inventory programs in capacitated supply chains, Manufacturing & Service Operations Management 6 (1), 53–72.

Christopher, M. (2023). Logistics and supply chain management. Description: Sixth Edition, Hoboken, NJ: Pearson, Revised. Identifiers: LCCN 2022042551, ISBN 9781292416182.

Dantzig, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. Management Science, Vol. 6, No. 1, pp. 80-91.

Federgruenand, A. and Zipkin P. (1984). A Combined Vehicle Routing and Inventory Allocation Problem. Operations Research Vol. 32, No. 5.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. Decision Sciences, 8(1):156–166, doi: 10.1111/j.1540-5915.1977.tb01074.x.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers & Operations Research, Vol. 13, No. 5, pp.533–549.

Goel, R. and Maini, R. (2017). Vehicle routing problem and its solution methodologies: a survey. Int. J. Logistics Systems and Management, Vol. 28, No. 4, pp.419-435.

Gupta, V., Peters, E., Miller, T., Blyden, K. (2002). Implementing a Distribution-Network Decision-Support System at Pfizer/Warner-Lambert. Interfaces, INFORMS Vol. 32, No. 4, pp. 28–45.

Hillier F. S. & Lieberman G. J. (1997). Introducción a la investigación de operaciones. 6ta ed. McGraw-Hill. ISBN: 970-10-1022-1.

Holland, J. (1975). Adaptation In Natural and Artificial Systems. University of Michigan Press, Ann Arbor, USA.

Hou Y., Fu Y., Gao K., Zhang H., Sadollah A. (2022). Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows. Expert Systems with Applications, Volume 187, 115827, ISSN 0957-4174.

Jacobs, F. and Chase, R. (2021). Operations and Supply Chain Management. Published by McGraw-Hill Education, 2 Penn Plaza, New York, NY 10121. Copyright © 2021 by McGraw-Hill Education. ISBN 978-1-260-57594-1.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. (1983). Optimization by simulated annealing. Science, 220(4598):671–680, doi: 10.1126/science.220.4598.671.

Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. Econometrica. Vol. 28, no. 3. pp. 497–520. doi:10.2307/1910129.

Lei L., Liu S., Ruszczynski A., Park S. (2006). On the integrated production, inventory, and distribution routing problem. IIE Trans, 38(11):955–70.

Lenstra, J. K., & Rinnooy-Kan, A. H. G. (1981). Complexity of vehicle routing and scheduling problems. Networks, 11(2), 221–227.

Lu, D. (2011). Fundamentals of supply chain management. Frederikesberg, Denmark: Ventus Publishing Aps. ISBN 978-87-7681-798-5.

Martin, C. H., Dent, D. C., Eckhart, J. C. (1993). Integrated Production, Distribution, and Inventory Planning at Libbey-Owens-Ford. INTERFACES 23: 3, pp. 68-78.

Meinecke, C. and Scholz-Reiter B. (2014). A heuristic for the integrated production and distribution scheduling problem. International Science Index, 8(2), 290–297.

Mladenović, N. & Hansen, P. (1997). Variable neighborhood search. Computers & Operations Research, 24(11):1097–1100, doi: 10.1016/s0305-0548(97)00031-2.

Mourtzis, D. and Doukas, M. (2014). Design and planning of manufacturing networks for mass customisation and personalisation: Challenges and Outlook, Procedia CIRP 19, 1 – 13.

Neves-Moreira, F., Almada-Lobo, B., Cordeau, J.F., Guimarães, L. (2019). Solving a large multi-product production-routing problem with delivery time windows. Omega 86, 154–172.

Padberg, M. & Rinaldi, G. (1991). A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. SIAM Review. 33 (1): 60–100, doi:10.1137/1033004. ISSN 0036-1445.

Pearson, K. (1895). Notes on regression and inheritance in the case of two parents. Proceedings of the Royal Society of London. 58: 240–242. Bibcode:1895RSPS...58..240P.

Pillac, V., Guéret, C., Medaglia, A. (2011). Dynamic Vehicle Routing Problems: State of the art and Prospects, ffhal-00623474f.

Qiu, Y., Qiao, J., Pardalos, P. M. (2017). A branch-and-price algorithm for production routing problems with carbon cap-and-trade. Omega 68, 49–61.

Qiu, Y., Wang, L., Xu, X., Fang, X., Pardalos, P.M. (2018). A variable neighborhood search heuristic algorithm for production routing problems. Applied Soft Computing 66, 311–318.

Qiu, Y., Zhou, D., Du, Y., Liu, J., Pardalos, P. M., Qiao, J. (2021). The two-echelon production routing problem with cross-docking satellites. Transportation Research Part E 147, 102210.

Salhi, S., Wassan, N., Hajarat, M. (2013). The fleet size and mix vehicle routing problem with backhauls: formulation and set partitioning-based heuristics. Transportation Research Part E: Logistics and Transportation Review, Vol. 56, No. 1, pp.22–35.

Scholz-Reiter, B., Schwindt, C., Makuschewitz, T., Frazzon, E. M. (2011). An approach for the integration of production scheduling and inter-facility transportation within global supply chains. International Journal of Logistics Systems and Management, 10(2), 158–179.

Stanton, D. (2018). Supply Chain Management For Dummies®. Published by: John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ. Copyright © 2018 by John Wiley & Sons, Inc., Hoboken, New Jersey. ISBN: 978-1-119-41019-5.

Sule, D. R. (2008). Production planning and industrial scheduling: examples, case studies and applications, second ed, ISBN: 978-1-4200-4420-1.

Taha, H. (2007). Operations research: an introduction. 8th ed. Pearson Prentice Gall. ISBN: 0-13-188923-0.

Ulmer, M., & Goodson, J., Mattfeld, D., Thomas, B. (2020). On modeling stochastic dynamic vehicle routing problems. EURO Journal on Transportation and Logistics. 9. 100008. 10.1016/j.ejtl.2020.100008.

Waller, M., Johnson, E., Davis, T. (1999). Vendor-Managed Inventory in the retail supply chain, Journal of Business Logistics 20 (1), 183-203.

Wight, O.W., (1984). Production and Inventory Management in the Computer Age, Van Nostrand Reinhold Company, Inc., New York.

Wilson, N., and Colvin, N. (1977). Computer Control of the Rochester Dial-A-Ride System. Cambridge: Massachusetts Institute of Technology, Center for Transportation Studies.

Yagmur, E. and Kesen, S. E. (2021). Multi-trip heterogeneous vehicle routing problem coordinated with production scheduling: Memetic algorithm and simulated annealing approaches. Computers & Industrial Engineering 161, 107649.

Zhao, R. (2019) A Review on Theoretical Development of Vendor-Managed Inventory in Supply Chain. American Journal of Industrial and Business Management, 9, 999-1010.

Anexos

Anexo 1. Parámetros ejemplo helados

c^P_{jf} Costos de producción

j	f	Valor
1	1	10
2	1	5

c^T_{fa} Costos de transporte

f	a	Valor
1	1	5

t^P_{jmnf} Tiempo de procesamiento

j	m	n	f	Valor
1	1	1	1	0.25
2	1	1	1	1
1	1	2	1	1
2	1	2	1	0.25

$t^a_{jj'mnf}$ Tiempo de ajuste

j	j'	m	n	f	Valor
1	1	1	1	1	999
1	2	1	1	1	10
2	1	1	1	1	5
2	2	1	1	1	999
1	1	1	2	1	999
1	2	1	2	1	2
2	1	1	2	1	3
2	2	1	2	1	999

HD_{mnf} Horas disponibles

m	n	f	Valor
1	1	1	160
1	2	1	160

D_{ja} Demanda

j	a	Valor
1	1	95
2	1	90

M
999

Anexo 2. Resultados ejemplo helados

Función Objetivo

x^P_{jmnof}	c^P_{jf}	$c^P_{jmnf} * x^P_{jmnof}$	x^T_{jfa}	c^T_{fa}	$c^T_{fa} * x^T_{jfa}$	Total
95	6	570	95	3	285	855
90	5	450	90	3	270	720
		1020			555	**1575**

$$Z_{min} = \sum_{j=1}^{J}\sum_{m=1}^{M}\sum_{n=1}^{1}\sum_{f=1}^{F} c^P_{jmnf} * x^P_{jmnof} + \sum_{j=1}^{J}\sum_{f=1}^{F}\sum_{a=1}^{A} c^T_{fa} * x^T_{jfa};$$

X^P_{jmnof} Cantidades a producir

j	m	n	o	f	Valor
1	1	1	1	1	95
2	1	1	1	1	0
1	1	1	2	1	0
2	1	1	2	1	90
1	1	2	1	1	95
2	1	2	1	1	0
1	1	2	2	1	0
2	1	2	2	1	90

X_{jmnof} Binaria de producción

j	m	n	o	f	Valor
1	1	1	1	1	1
2	1	1	1	1	0
1	1	1	2	1	0
2	1	1	2	1	1
1	1	2	1	1	1
2	1	2	1	1	0
1	1	2	2	1	0
2	1	2	2	1	1

X^T_{jfa} Cantidades a transportar

j	f	a	Valor
1	1	1	95
2	1	1	90

$Y_{jj'mnf}$ Binaria de ajustes

j	j'	m	n	f	Valor
1	1	1	1	1	0
1	2	1	1	1	1
2	1	1	1	1	0
2	2	1	1	1	0
1	1	1	2	1	0
1	2	1	2	1	1
2	1	1	2	1	0
2	2	1	2	1	0

tf_{jmnf} Tiempo de finalización

j	m	n	f	Valor
1	1	1	1	23.75
2	1	1	1	123.75
1	1	2	1	118.75
2	1	2	1	148.25

Anexo 3. Restricciones ejemplo helados

R1

$$\sum_{f=1}^{F} x^{t}_{jfa} \geq D_{ja} \; \forall \, a, j$$

x^{t}_{jfa}	D_{ja}	L.I.	Restr	L.D.
95	95	0	>=	0
90	90	0	>=	0

R2

$$\sum_{m=1}^{M} \sum_{n=1}^{1} \sum_{o=1}^{O} x^{p}_{jmnof} - x^{t}_{jfa} = 0 \; \forall \, j, f, a$$

x^{p}_{jmnof}	x^{t}_{jfa}	L.I.	Restr	L.D.
95	95	0	=	0
90	90	0	=	0

R3

$$t^{t}_{jmnf} \leq HD_{nmf} \; (\forall \, j, m, n, f)$$

t^{t}_{jmnf}	HD_{nmf}	L.I.	Restr	L.D.
23.75	160	-136.25	<=	0
123.75	160	-36.25	<=	0
118.75	160	-41.25	<=	0
148.25	160	-11.75	<=	0

R4

$$MX_{jmnof} \geq x^{p}_{jmnof} \; \forall \, j, m, n, o, f$$

MX_{jmnof}	x^{p}_{jmnof}	L.I.	Restr	L.D.
999	95	904	>=	0
0	0	0	>=	0
0	0	0	>=	0
999	90	909	>=	0
999	95	904	>=	0
0	0	0	>=	0
0	0	0	>=	0
999	90	909	>=	0

R5

$$1 - M(2 - X_{jmnof} - X_{j'mno+1f}) \leq Y_{j'mnf} \; \forall j, m, n, o, f$$

X_{jmnof}	$X_{j'mno+1f}$	$Y_{j'mnf}$	L.I.	Restr	L.D.	
1	0	0	-998	<=	0	
1	1	1	0	<=	0	
0	0	0	-1997	<=	0	
0	1	0	-998	<=	0	
0			-1997	<=	0	no aplica
0			-1997	<=	0	no aplica
1			-998	<=	0	no aplica
1			-998	<=	0	no aplica
1	0	0	-998	<=	0	
1	1	1	0	<=	0	
0	0	0	-1997	<=	0	
0	1	0	-998	<=	0	
0			-1997	<=	0	no aplica
0			-1997	<=	0	no aplica
1			-998	<=	0	no aplica
1			-998	<=	0	no aplica

R6

$$t^{t}_{jmnf} + t^{p}_{j'mnf} + x^{p}_{jmno+1f} + t^{o}_{jj'mnf} + Y_{j'mnf} \leq t^{t}_{j'mnf} + M(2 - X_{jmnof} - X_{j'mno+1f}) + M(1 - Y_{j'mnf}) \; \forall j, m, n, o, f$$

t^{t}_{jmnf}	$t^{p}_{j'mnf}$	$x^{p}_{jmno+1f}$	$t^{o}_{jj'mnf}$	$Y_{j'mnf}$	$t^{t}_{j'mnf}$	X_{jmnof}	$X_{j'mno+1f}$	$Y_{j'mnf}$	L.I.	Restr	L.D.	
23.75	0.25	0	999	0	23.75	1	0	0	-1998	<=	0	
23.75	1	90	10	1	123.75	1	1	1	0	<=	0	
123.75	0.25	0	5	0	23.75	0	0	0	-2897	<=	0	
123.75	1	90	999	0	123.75	0	1	0	-1908	<=	0	
23.75	0.25		999			0			-2973.25	<=	0	no aplica
23.75	1		10			0			-2973.25	<=	0	no aplica
123.75	0.25		5			1			-1874.25	<=	0	no aplica
123.75	1		999			1			-1874.25	<=	0	no aplica
118.75	1	0	999	0	118.75	1	0	0	-1998	<=	0	
118.75	0.25	90	2	1	148.25	1	1	1	-5	<=	0	
148.25	1	0	3	0	118.75	0	0	0	-2967.5	<=	0	
148.25	0.25	90	999	0	148.25	0	1	0	-1975.5	<=	0	
118.75	1		999			0			-2878.25	<=	0	no aplica
118.75	0.25		2			0			-2878.25	<=	0	no aplica
148.25	1		3			1			-1849.75	<=	0	no aplica
148.25	0.25		999			1			-1849.75	<=	0	no aplica

R7

$$t^{t}_{jmn-1f} + t^{p}_{jmnf} + x^{p}_{jmnof} + t^{o}_{j'jmnf} + Y_{j'jmnf} \leq t^{t}_{j'jmnf} + M(1 - X_{jmnof}) \; \forall j, m, n, o, f$$

t^{t}_{jmn-1f}	t^{p}_{jmnf}	x^{p}_{jmnof}	$t^{o}_{j'jmnf}$	$Y_{j'jmnf}$	$t^{t}_{j'jmnf}$	X_{jmnof}	HD	L.I.	Restr	L.D.
	0.25	95	999	0	23.75	1	160		0 <=	0
	0.25	95	5	0	23.75	1	160		0 <=	0
	1	0	10	1	123.75	0	160	-1112.75	<=	0
	1	0	999	0	123.75	0	160	-1122.75	<=	0
	0.25	0	999	0	23.75	0	160	-1022.75	<=	0
	0.25	0	5	0	23.75	0	160	-1022.75	<=	0
	1	90	10	1	123.75	1	160	-23.75	<=	0
	1	90	999	0	123.75	1	160	-33.75	<=	0
23.75	1	95	999	0	118.75	1	160	0	<=	0
23.75	1	95	3	0	118.75	1	160	0	<=	0
123.75	0.25	0	2	1	148.25	0	160	-1021.5	<=	0
123.75	0.25	0	999	0	148.25	0	160	-1023.5	<=	0
23.75	1	0	999	0	118.75	0	160	-1094	<=	0
23.75	1	0	3	0	118.75	0	160	-1094	<=	0
123.75	0.25	90	2	1	148.25	1	160	0	<=	0
123.75	0.25	90	999	0	148.25	1	160	-2	<=	0

R8

$$\sum_{o=1}^{O} \sum_{m=1}^{M} x^{p}_{jmnof} - \sum_{o=1}^{O} \sum_{m=1}^{M} x^{p}_{jmn+1of} = 0 \; \forall j, n, f$$

x^{p}_{jmnof}	$x^{p}_{jmn+1of}$	L.I.	Restr	L.D.	
95	95	0	=	0	
90	90	0	=	0	
95	0				no aplica
90	90				no aplica

R9

$$\sum_{m=1}^{M} X_{jmnof} \leq 1 \; \forall \, m, n, o, f$$

L.I.	Restr	L.D.
0	<=	0
0	<=	0
0	<=	0
0	<=	0

Anexo 4. Pseudocódigos de metaheurísticas

Algoritmo genético:

```
Genetic_Algorithm() {
                Initialize random population;
                Evaluate the population;
                Generation = 0;
     While termination criterion is not
     satisfied {
             Generation = Generation + 1;
             Select good chromosomes by
             reproduction procedure;
             Perform crossover with probability
             crossover (Pc);
             Perform mutation with probability
             of mutation (Pm);
             Evaluate the population;
     }
}
```

Búsqueda dispersa:

```
Procedure ScatterSearch ;
      Begin
      repeat
         Generate_Reference_Set(Ref_Set, Ref_Set_Size);
         Repeat
            Select_Subset(SubSet, SubSetSize);
            Combine_Solutions(SubSet, Current_Sol);
            Improve_Solution(Current_Sol, Improved_Sol);
         until (Inner_Stop_Criterion);
         Update_Reference_Set(Ref_Set);
      until (Outer_Stop_Criterion);
      end.
```

Recocido simulado:

Simulated annealing algorithm

```
1  Select the best solution vector x0 to be optimized
2  Initialize the parameters: temperature T, Boltzmann's constant k, reduction factor c
3  while termination criterion is not satisfied do
4      for number of new solution
6          Select a new solution: x0+Δx
7          if f(x0+Δx) > f(x0) then
8              f_new = f(x0 + Δx);  x0 = x0 + Δx
9              else
10                 Δf = f(x0 + Δx) − f(x0)
11                 random r(0, 1)
12                 if r > exp(−Δf/kT) then
13                     f_new = f(x0 + Δx),  x0 = x0 + Δx
14                     else
15                     f_new = f(x0),
16                 end if
17         end if
18         f = f_new
19         Decrease the temperature periodically: T = c x T
20     end for
21 end while
```

Búsqueda tabú:

Algorithm 1 Tabu Search Algorithm Pseudocode.

```
1: procedure TABU SEARCH
2:     Generate initial solution X.
3:     Initialize the Tabu List.
4:     while (set of candidate solution X" is not complete) do
5:         Generate candidate solution X" from current solution x.
6:         Add X" to X" only if X" is not Tabu.
7:     end while
8:     Select the best candidate solution x* in X.
9:     if(fitness(x*) > fitness(x))
10:    x = x*
11:    end if
12:    if(termination condition met) then finish, otherwise go to step 4
13: end procedure
```

Búsqueda variable de vecindario:

Procedure ***Variable Neighborhood Search***

Initialization: *Select a set of neighborhood structures* N_k, $k = 1,2,\ldots,k_{\max}$, *and random distributions for the Shaking step that is used in the search. Find an initial solution* x.

While (the stopping condition is not met)

$k \leftarrow 1$;

Repeat for $k = 1 \, to \, k_{\max}$

Shaking: *Generate a point* y *randomly from the k-th neighborhood of* x *denoted by* $N_k(x)$

Local search: *Apply some local search method with* y *as initial solution to obtain a local optimum denoted by* y'.

Neighborhood change: *if this local optimum is better than the incumbent, move there (* $x \leftarrow y'$ *), and set* $k \leftarrow 1$;

Endrepeat

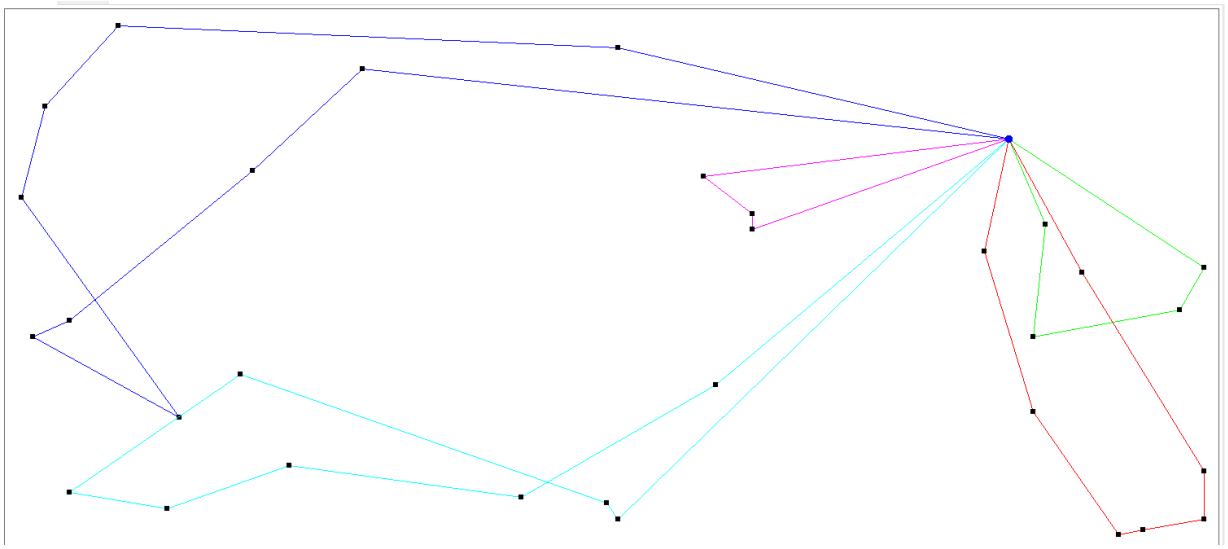
Endwhile

Return x

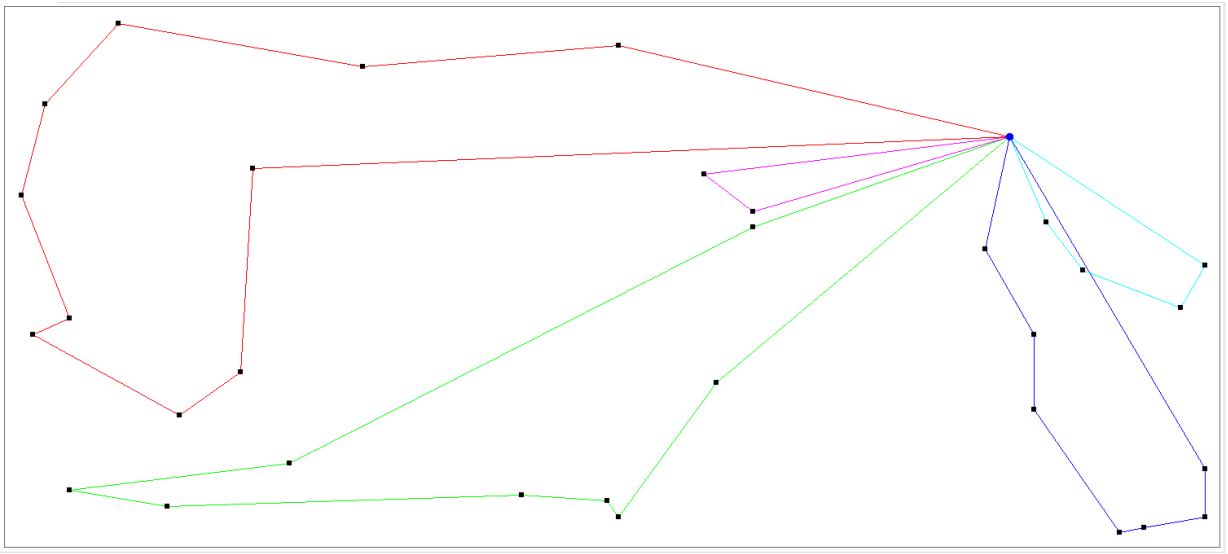
Endprocedure

Anexo 5. Rutas obtenidas en instancia de experimentación con HeuristicLab

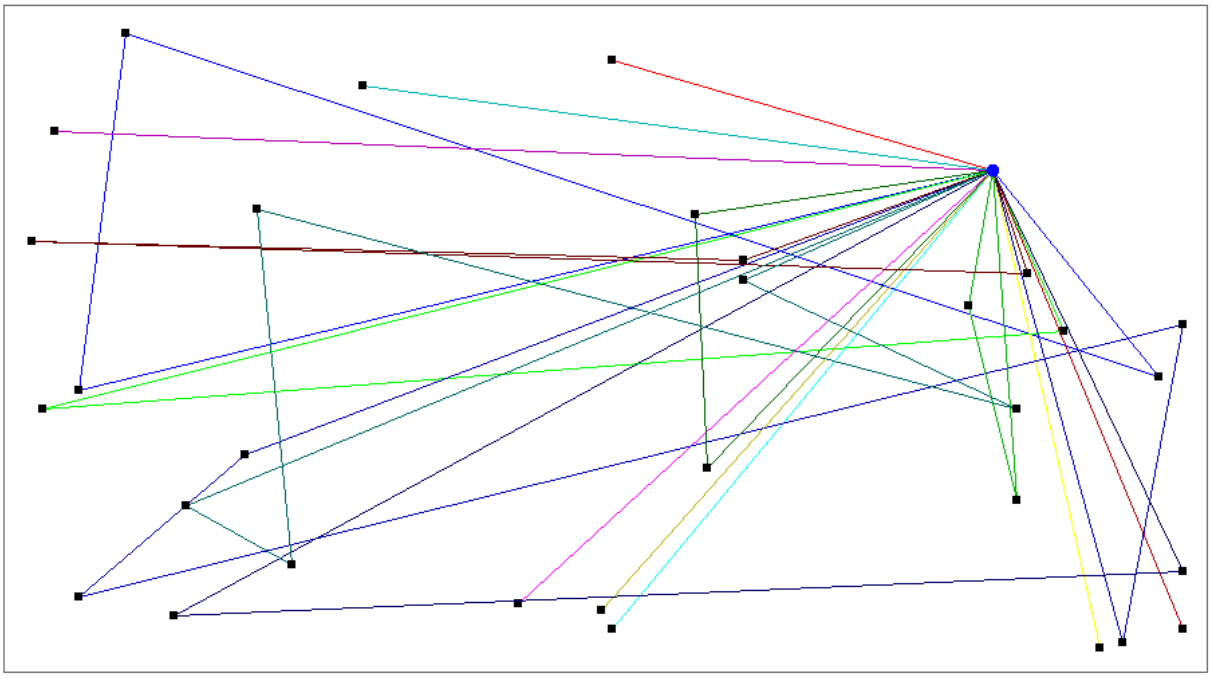
Algoritmo genético:



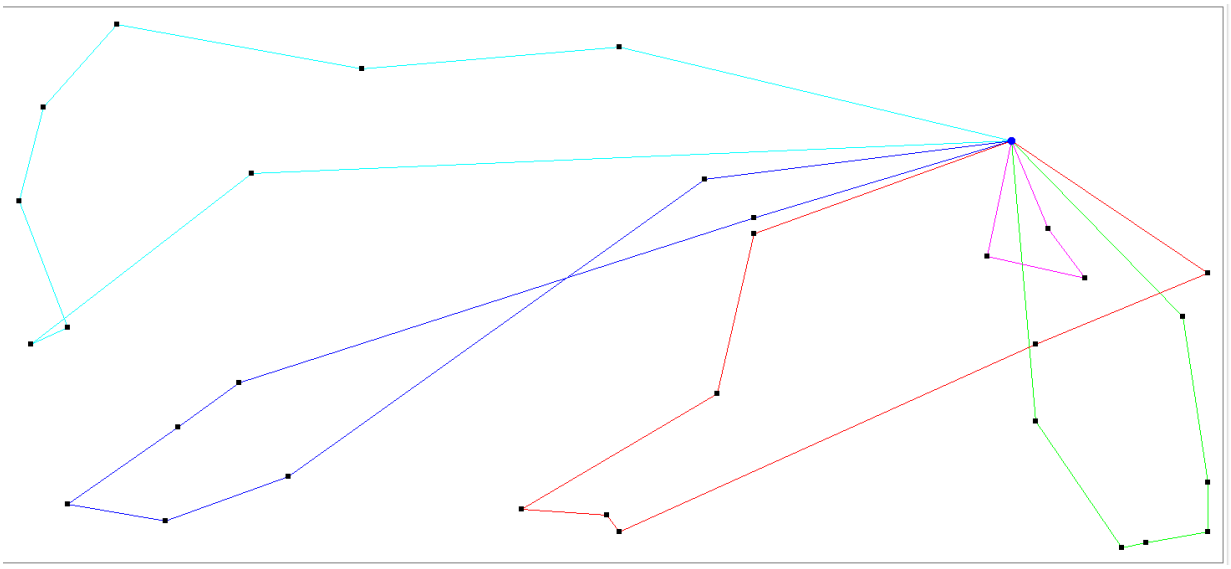
Búsqueda dispersa:



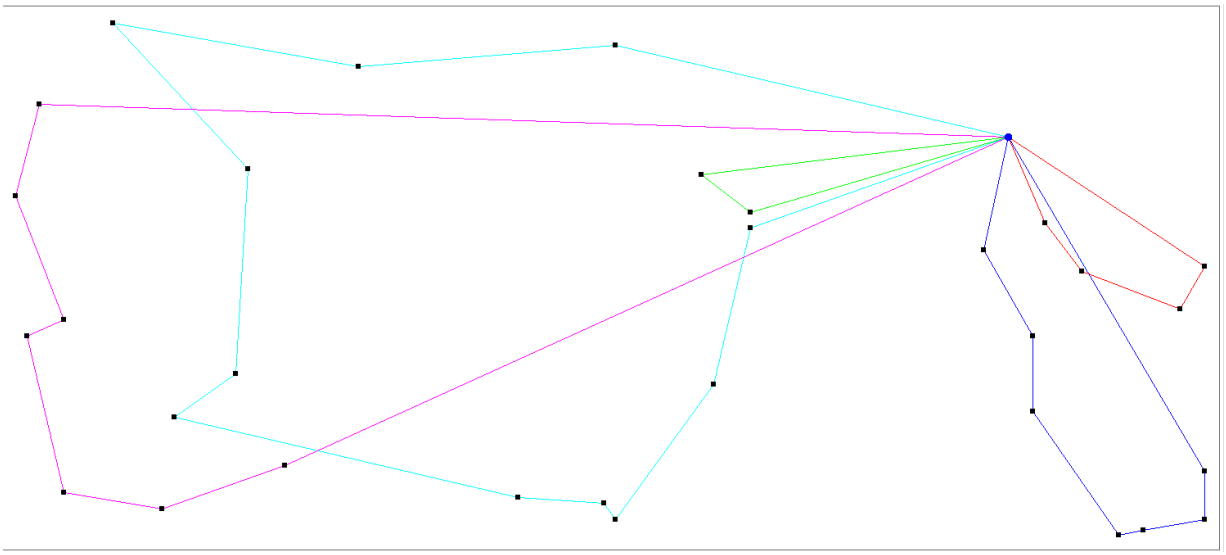
Recocido Simulado:



Búsqueda tabú:



Búsqueda variable de vecindario:



Anexo 6. Valores de parámetros modelo de optimización

Como se describió en el proceso de parametrización del modelo de optimización los valores de ciertos parámetros son iguales para todas las instancias y para otros grupos de parámetros estos van relacionados al RPD y PSD serán los que se varían, de este modo los parámetros que no cambian serán: c_{jf}^{p} , $t^{p}{}_{jmnf}$, HD_{mnf} y D_{ja}. Los cuales se muestran a continuación:

Producto *j*	Fábrica *f*	c_{jf}^{p}
1	1	2.88
1	2	3.6
1	3	4.32
2	1	1.36
2	2	1.7
2	3	99999
3	1	1.12
3	2	99999
3	3	1.68
4	1	99999
4	2	2.8
4	3	3.36
5	1	99999
5	2	99999
5	3	2.76

Máquina *m*	Nivel *n*	Fábrica *f*	HD_{mnf}
1	1	1	144
1	2	1	144
1	1	2	144
1	2	2	144
1	1	3	144
1	2	3	144

Producto *j*	Máquina *m*	Nivel *n*	Fábrica *f*	t^p_{jmnf}
1	1	1	1	0.48
2	1	1	1	0.51
3	1	1	1	0.71
4	1	1	1	0.79
5	1	1	1	0.33
1	1	2	1	0.48
2	1	2	1	0.43
3	1	2	1	0.46
4	1	2	1	0.17
5	1	2	1	0.04
1	1	1	2	0.72
2	1	1	2	0.47
3	1	1	2	0.73
4	1	1	2	0.6
5	1	1	2	0.62
1	1	2	2	0.14
2	1	2	2	0.28
3	1	2	2	0.41
4	1	2	2	0.6
5	1	2	2	0.76
1	1	1	3	0.24
2	1	1	3	0.39
3	1	1	3	0.56
4	1	1	3	0.55
5	1	1	3	0.13
1	1	2	3	0.7
2	1	2	3	0.35
3	1	2	3	0.61
4	1	2	3	0.22
5	1	2	3	0.67

Producto *j*	Almacén *a*	D_{ja}
1	1	20
2	1	8
3	1	20
4	1	4
5	1	11
1	2	17
2	2	15

3	2	2
4	2	6
5	2	8
1	3	20
2	3	13
3	3	9
4	3	7
5	3	18
1	4	22
2	4	18
3	4	13
4	4	15
5	4	6
1	5	5
2	5	16
3	5	7
4	5	10
5	5	14
1	6	2
2	6	9
3	6	5
4	6	23
5	6	20
1	7	7
2	7	16
3	7	10
4	7	18
5	7	4
1	8	0
2	8	21
3	8	8
4	8	21
5	8	1
1	9	15
2	9	16
3	9	19
4	9	10
5	9	3
1	10	21
2	10	12
3	10	15
4	10	8
5	10	4

Por otra parte, los c_{fa}^{t} cambian según cada RPD, así como los $t^{a}{}_{jj'mnf}$ cambian según las PSD. Sus valores para cada instancia se pueden ver a continuación:

$\boldsymbol{c_{fa}^{t}}$

Fábrica f	Almacén a	RPD = 4	RPD = 2	RPD = 1	RPD = 0.5	RPD = 0.25
1	1	0.0450	0.0900	0.1801	0.3602	0.7204
1	2	0.5563	1.1126	2.2252	4.4504	8.9009
1	3	0.9055	1.8110	3.6220	7.2441	14.4881
1	4	0.8166	1.6332	3.2664	6.5328	13.0655
1	5	0.7140	1.4280	2.8561	5.7122	11.4244
1	6	0.9726	1.9451	3.8903	7.7806	15.5612
1	7	0.4583	0.9167	1.8333	3.6667	7.3333
1	8	0.4240	0.8480	1.6959	3.3919	6.7838
1	9	0.9708	1.9417	3.8833	7.7666	15.5333
1	10	1.1643	2.3287	4.6574	9.3147	18.6294
2	1	0.7971	1.5942	3.1884	6.3769	12.7537
2	2	0.5668	1.1335	2.2670	4.5340	9.0681
2	3	0.4626	0.9252	1.8505	3.7010	7.4019
2	4	0.4548	0.9095	1.8190	3.6380	7.2761
2	5	0.2469	0.4938	0.9876	1.9751	3.9503
2	6	0.7052	1.4104	2.8208	5.6417	11.2833
2	7	0.9739	1.9478	3.8956	7.7912	15.5824
2	8	0.4463	0.8926	1.7852	3.5704	7.1408
2	9	0.2648	0.5296	1.0592	2.1184	4.2368
2	10	0.5383	1.0765	2.1531	4.3062	8.6124

3	1	1.0394	2.0788	4.1575	8.3151	16.6301
3	2	0.5920	1.1840	2.3680	4.7361	9.4722
3	3	0.6590	1.3179	2.6358	5.2716	10.5433
3	4	0.6243	1.2487	2.4973	4.9947	9.9893
3	5	0.8223	1.6447	3.2894	6.5788	13.1575
3	6	0.4129	0.8258	1.6515	3.3030	6.6060
3	7	0.7254	1.4508	2.9017	5.8033	11.6067
3	8	0.8023	1.6046	3.2092	6.4185	12.8369
3	9	1.3219	2.6438	5.2877	10.5753	21.1507
3	10	0.8737	1.7474	3.4948	6.9896	13.9792

$t^{a}_{jj'mnf}$

Producto *j*	Producto *j'*	Máquina *m*	Nivel *n*	Fábrica *f*	PSD = 0	PSD = 0.05	PSD = 0.10	PSD = 0.15
1	1	1	1	1	-	-	-	-
2	1	1	1	1	-	1.28	2.56	3.84
3	1	1	1	1	-	0.68	1.36	2.04
4	1	1	1	1	-	0.73	1.46	2.19
5	1	1	1	1	-	1.47	2.94	4.41
1	2	1	1	1	-	2.77	5.54	8.31
2	2	1	1	1	-	-	-	-
3	2	1	1	1	-	0.75	1.50	2.25
4	2	1	1	1	-	1.74	3.48	5.22
5	2	1	1	1	-	2.27	4.54	6.81
1	3	1	1	1	-	1.82	3.64	5.46
2	3	1	1	1	-	0.70	1.40	2.10

3	3	1	1	1	-	-	-	-	
4	3	1	1	1	-	0.75	1.50	2.25	
5	3	1	1	1	-	2.27	4.54	6.81	
1	4	1	1	1	-	2.12	4.24	6.36	
2	4	1	1	1	-	3.05	6.10	9.15	
3	4	1	1	1	-	0.38	0.76	1.14	
4	4	1	1	1	-	-	-	-	
5	4	1	1	1	-	4.29	8.58	12.87	
1	5	1	1	1	-	4.21	8.42	12.63	
2	5	1	1	1	-	3.26	6.52	9.78	
3	5	1	1	1	-	3.69	7.38	11.07	
4	5	1	1	1	-	4.17	8.34	12.51	
5	5	1	1	1	-	-	-	-	
1	1	1	2	1	-	-	-	-	
2	1	1	2	1	-	3.70	7.40	11.10	
3	1	1	2	1	-	1.51	3.02	4.53	
4	1	1	2	1	-	2.59	5.18	7.77	
5	1	1	2	1	-	2.22	4.44	6.66	
1	2	1	2	1	-	3.66	7.32	10.98	
2	2	1	2	1	-	-	-	-	
3	2	1	2	1	-	0.96	1.92	2.88	
4	2	1	2	1	-	0.51	1.02	1.53	
5	2	1	2	1	-	1.06	2.12	3.18	
1	3	1	2	1	-	3.23	6.46	9.69	
2	3	1	2	1	-	2.57	5.14	7.71	
3	3	1	2	1	-	-	-	-	

4	3	1	2	1	-	0.67	1.34	2.01	
5	3	1	2	1	-	4.10	8.20	12.30	
1	4	1	2	1	-	1.41	2.82	4.23	
2	4	1	2	1	-	2.28	4.56	6.84	
3	4	1	2	1	-	2.93	5.86	8.79	
4	4	1	2	1	-	-	-	-	-
5	4	1	2	1	-	3.79	7.58	11.37	
1	5	1	2	1	-	2.53	5.06	7.59	
2	5	1	2	1	-	0.15	0.30	0.45	
3	5	1	2	1	-	2.62	5.24	7.86	
4	5	1	2	1	-	2.50	5.00	7.50	
5	5	1	2	1	-	-	-	-	-
1	1	1	1	2	-	-	-	-	-
2	1	1	1	2	-	2.46	4.92	7.38	
3	1	1	1	2	-	3.53	7.06	10.59	
4	1	1	1	2	-	3.88	7.76	11.64	
5	1	1	1	2	-	0.86	1.72	2.58	
1	2	1	1	2	-	1.51	3.02	4.53	
2	2	1	1	2	-	-	-	-	-
3	2	1	1	2	-	1.33	2.66	3.99	
4	2	1	1	2	-	3.58	7.16	10.74	
5	2	1	1	2	-	3.45	6.90	10.35	
1	3	1	1	2	-	1.87	3.74	5.61	
2	3	1	1	2	-	2.79	5.58	8.37	
3	3	1	1	2	-	-	-	-	-
4	3	1	1	2	-	2.79	5.58	8.37	

5	3	1	1	2	-	2.85	5.70	8.55	
1	4	1	1	2	-	1.32	2.64	3.96	
2	4	1	1	2	-	2.70	5.40	8.10	
3	4	1	1	2	-	4.00	8.00	12.00	
4	4	1	1	2	-	-	-	-	-
5	4	1	1	2	-	1.92	3.84	5.76	
1	5	1	1	2	-	0.09	0.18	0.27	
2	5	1	1	2	-	1.40	2.80	4.20	
3	5	1	1	2	-	3.18	6.36	9.54	
4	5	1	1	2	-	0.62	1.24	1.86	
5	5	1	1	2	-	-	-	-	-
1	1	1	2	2	-	-	-	-	-
2	1	1	2	2	-	0.85	1.70	2.55	
3	1	1	2	2	-	2.82	5.64	8.46	
4	1	1	2	2	-	2.57	5.14	7.71	
5	1	1	2	2	-	3.53	7.06	10.59	
1	2	1	2	2	-	1.35	2.70	4.05	
2	2	1	2	2	-	-	-	-	-
3	2	1	2	2	-	0.35	0.70	1.05	
4	2	1	2	2	-	1.93	3.86	5.79	
5	2	1	2	2	-	0.52	1.04	1.56	
1	3	1	2	2	-	3.47	6.94	10.41	
2	3	1	2	2	-	2.45	4.90	7.35	
3	3	1	2	2	-	-	-	-	-
4	3	1	2	2	-	0.22	0.44	0.66	
5	3	1	2	2	-	1.40	2.80	4.20	

1	4	1	2	2	-	3.82	7.64	11.46	
2	4	1	2	2	-	0.87	1.74	2.61	
3	4	1	2	2	-	2.41	4.82	7.23	
4	4	1	2	2	-	-	-	-	-
5	4	1	2	2	-	4.00	8.00	12.00	
1	5	1	2	2	-	0.99	1.98	2.97	
2	5	1	2	2	-	0.79	1.58	2.37	
3	5	1	2	2	-	0.86	1.72	2.58	
4	5	1	2	2	-	4.25	8.50	12.75	
5	5	1	2	2	-	-	-	-	-
1	1	1	1	3	-	-	-	-	-
2	1	1	1	3	-	2.61	5.22	7.83	
3	1	1	1	3	-	4.20	8.40	12.60	
4	1	1	1	3	-	3.47	6.94	10.41	
5	1	1	1	3	-	1.32	2.64	3.96	
1	2	1	1	3	-	0.41	0.82	1.23	
2	2	1	1	3	-	-	-	-	-
3	2	1	1	3	-	0.13	0.26	0.39	
4	2	1	1	3	-	0.58	1.16	1.74	
5	2	1	1	3	-	0.51	1.02	1.53	
1	3	1	1	3	-	3.72	7.44	11.16	
2	3	1	1	3	-	2.70	5.40	8.10	
3	3	1	1	3	-	-	-	-	-
4	3	1	1	3	-	0.41	0.82	1.23	
5	3	1	1	3	-	3.80	7.60	11.40	
1	4	1	1	3	-	3.75	7.50	11.25	

2	4	1	1	3	-	2.80	5.60	8.40	
3	4	1	1	3	-	4.29	8.58	12.87	
4	4	1	1	3	-	-	-	-	-
5	4	1	1	3	-	4.00	8.00	12.00	
1	5	1	1	3	-	0.75	1.50	2.25	
2	5	1	1	3	-	3.90	7.80	11.70	
3	5	1	1	3	-	2.44	4.88	7.32	
4	5	1	1	3	-	3.86	7.72	11.58	
5	5	1	1	3	-	-	-	-	-
1	1	1	2	3	-	-	-	-	-
2	1	1	2	3	-	4.26	8.52	12.78	
3	1	1	2	3	-	2.96	5.92	8.88	
4	1	1	2	3	-	0.13	0.26	0.39	
5	1	1	2	3	-	0.67	1.34	2.01	
1	2	1	2	3	-	3.29	6.58	9.87	
2	2	1	2	3	-	-	-	-	-
3	2	1	2	3	-	2.99	5.98	8.97	
4	2	1	2	3	-	2.43	4.86	7.29	
5	2	1	2	3	-	3.99	7.98	11.97	
1	3	1	2	3	-	4.11	8.22	12.33	
2	3	1	2	3	-	1.47	2.94	4.41	
3	3	1	2	3	-	-	-	-	-
4	3	1	2	3	-	2.45	4.90	7.35	
5	3	1	2	3	-	4.01	8.02	12.03	
1	4	1	2	3	-	2.37	4.74	7.11	
2	4	1	2	3	-	1.12	2.24	3.36	

3	4	1	2	3	-	0.60	1.20	1.80	
4	4	1	2	3	-	-	-	-	-
5	4	1	2	3	-	3.87	7.74	11.61	
1	5	1	2	3	-	0.60	1.20	1.80	
2	5	1	2	3	-	0.06	0.12	0.18	
3	5	1	2	3	-	0.43	0.86	1.29	
4	5	1	2	3	-	2.17	4.34	6.51	
5	5	1	2	3	-	-	-	-	-

Anexo 7. Código Gurobipy para modelo de optimización.

jupyter modelo_optimizacion_tesis_v8_instancias Last Checkpoint: 02/21/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
from gurobipy import *
```

In [2]:
```
# Sets
productos = [1,2,3,4,5] #p
prodprima = productos #p
maquinas = [1] #m disminución para comprobar ejercicio
niveles = [1,2] #n
ordenes = [1,2,3,4,5] #o
fabricas = [1,2,3] #f
almacenes = [1,2,3,4,5,6,7,8,9,10] #a
```

In [3]:
```
# Aux Sets
niveles_noult = niveles[:-1]
ordenes_noult = ordenes[:-1]
niveles_nopri = niveles[1:]
ordenes_nopri = ordenes[1:]
ordenes_medios = ordenes[1:-1]
```

In [4]:
```
# Subíndices variables
jmnof = [(j,m,n,o,f) for j in productos for m in maquinas for n in niveles for o in ordenes for f in fabricas]
jmnf = [(j,m,n,f) for j in productos for m in maquinas for n in niveles for f in fabricas]
jpmnf = [(j,p,m,n,f) for j in productos for p in prodprima for m in maquinas for n in niveles for f in fabricas]
pmnof = [(p,m,n,o,f) for p in prodprima for m in maquinas for n in niveles for o in ordenes for f in fabricas]
pmnf = [(p,m,n,f) for p in prodprima for m in maquinas for n in niveles for f in fabricas]
pjmnf = [(p,j,m,n,f) for p in prodprima for j in productos for m in maquinas for n in niveles for f in fabricas]
jfa = [(j,f,a) for j in productos for f in fabricas for a in almacenes]

# Subíndices parámetros
jf = [(j,f) for j in productos for f in fabricas]
fa = [(f,a) for f in fabricas for a in almacenes]
mnf = [(m,n,f) for m in maquinas for n in niveles for f in fabricas]
ja = [(j,a) for j in productos for a in almacenes]
```

In [5]:
```
# Verificación subíndices
print('jmnof ',len(jmnof))
print('jmnf ',len(jmnf))
print('jpmnf ',len(jpmnf))
print('pmnof ',len(pmnof))
print('pmnf ',len(pmnf))
print('pjmnf ',len(pjmnf))
print('jfa ',len(jfa))

print('')
print('jf ',len(jf))
print('fa ',len(fa))
print('mnf ',len(mnf))
print('ja ',len(ja))
```

```
jmnof  150
jmnf  30
jpmnf  150
pmnof  150
pmnf  30
pjmnf  150
jfa  150

jf  15
fa  30
mnf  6
ja  50
```

In [6]:
```
# Parámetros
cp={(1,1):2.88,(1,2):3.6,(1,3):4.32,(2,1):1.36,(2,2):1.7,(2,3):99999,(3,1):1.12,(3,2):99999,(3,3):1.68,(4,1):99999,(4,2):2.8,(4,3
#
ct={(1,1):0.7204,(1,2):8.9009,(1,3):14.4881,(1,4):13.0655,(1,5):11.4244,(1,6):15.5612,(1,7):7.3333,(1,8):6.7838,(1,9):15.5333,(1,
tp={(1,1,1,1):0.48,(2,1,1,1):0.51,(3,1,1,1):0.71,(4,1,1,1):0.79,(5,1,1,1):0.33,(1,1,2,1):0.48,(2,1,2,1):0.43,(3,1,2,1):0.46,(4,1,
#
ta1={(1,1,1,1,1):0,(2,1,1,1,1):0,(3,1,1,1,1):0,(4,1,1,1,1):0,(5,1,1,1,1):0,(1,2,1,1,1):0,(2,2,1,1,1):0,(3,2,1,1,1):0,(4,2,1,1,1)
ta2={(1,1,1,1,1):0,(2,1,1,1,1):0,(3,1,1,1,1):0,(4,1,1,1,1):0,(5,1,1,1,1):0,(1,2,1,1,1):0,(2,2,1,1,1):0,(3,2,1,1,1):0,(4,2,1,1,1)
hd={(1,1,1):144,(1,2,1):144,(1,1,2):144,(1,2,2):144,(1,1,3):144,(1,2,3):144}
#D
d={(1,1):20,(2,1):8,(3,1):20,(4,1):4,(5,1):11,(1,2):17,(2,2):15,(3,2):2,(4,2):6,(5,2):8,(1,3):20,(2,3):13,(3,3):9,(4,3):7,(5,3):
M=99999
```

In [7]:
```
m=Model("Modelo_tesis_i1")
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-01-29
```

In [8]:
```
# Declaración variables
xp=m.addVars(jmnof,vtype=GRB.CONTINUOUS,name='xp')
xpprima=m.addVars(pmnof,vtype=GRB.CONTINUOUS,name='xpprima')
xt=m.addVars(jfa,vtype=GRB.CONTINUOUS,name='xt')
tf=m.addVars(jmnf,vtype=GRB.CONTINUOUS,name='tf')
tfprima=m.addVars(pmnf,vtype=GRB.CONTINUOUS,name='tfprima')
X=m.addVars(jmnof,vtype=GRB.BINARY,name='X')
Xprima=m.addVars(pmnof,vtype=GRB.BINARY,name='Xprima')
Y1=m.addVars(jpmnf,vtype=GRB.BINARY,name='Y1')
Y2=m.addVars(pjmnf,vtype=GRB.BINARY,name='Y2')
```

In [9]:
```
# Función objetivo
m.setObjective(quicksum(cp[j,f]*xp[j,m,1,o,f] for j in productos for m in maquinas for o in ordenes for f in fabricas) + quicksur
```

In [10]:
```
# Restricciones 1
m.addConstrs(quicksum(xt[j,f,a] for f in fabricas) >= d[j,a] for j in productos for a in almacenes)
```

Out[10]:
```
{(1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 4): <gurobi.Constr *Awaiting Model Update*>,
 (1, 5): <gurobi.Constr *Awaiting Model Update*>,
 (1, 6): <gurobi.Constr *Awaiting Model Update*>,
 (1, 7): <gurobi.Constr *Awaiting Model Update*>,
 (1, 8): <gurobi.Constr *Awaiting Model Update*>,
 (1, 9): <gurobi.Constr *Awaiting Model Update*>,
 (1, 10): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (2, 4): <gurobi.Constr *Awaiting Model Update*>,
 (2, 5): <gurobi.Constr *Awaiting Model Update*>,
 (2, 6): <gurobi.Constr *Awaiting Model Update*>,
 (2, 7): <gurobi.Constr *Awaiting Model Update*>,
 (2, 8): <gurobi.Constr *Awaiting Model Update*>,
 (2, 9): <gurobi.Constr *Awaiting Model Update*>,
 (2, 10): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1): <gurobi.Constr *Awaiting Model Update*>,
 (3, 2): <gurobi.Constr *Awaiting Model Update*>,
 (3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (3, 4): <gurobi.Constr *Awaiting Model Update*>,
 (3, 5): <gurobi.Constr *Awaiting Model Update*>,
 (3, 6): <gurobi.Constr *Awaiting Model Update*>,
 (3, 7): <gurobi.Constr *Awaiting Model Update*>,
 (3, 8): <gurobi.Constr *Awaiting Model Update*>,
 (3, 9): <gurobi.Constr *Awaiting Model Update*>,
 (3, 10): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (4, 3): <gurobi.Constr *Awaiting Model Update*>,
 (4, 4): <gurobi.Constr *Awaiting Model Update*>,
 (4, 5): <gurobi.Constr *Awaiting Model Update*>,
 (4, 6): <gurobi.Constr *Awaiting Model Update*>,
 (4, 7): <gurobi.Constr *Awaiting Model Update*>,
 (4, 8): <gurobi.Constr *Awaiting Model Update*>,
 (4, 9): <gurobi.Constr *Awaiting Model Update*>,
 (4, 10): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1): <gurobi.Constr *Awaiting Model Update*>,
 (5, 2): <gurobi.Constr *Awaiting Model Update*>,
 (5, 3): <gurobi.Constr *Awaiting Model Update*>,
 (5, 4): <gurobi.Constr *Awaiting Model Update*>,
 (5, 5): <gurobi.Constr *Awaiting Model Update*>,
 (5, 6): <gurobi.Constr *Awaiting Model Update*>,
 (5, 7): <gurobi.Constr *Awaiting Model Update*>,
 (5, 8): <gurobi.Constr *Awaiting Model Update*>,
 (5, 9): <gurobi.Constr *Awaiting Model Update*>,
 (5, 10): <gurobi.Constr *Awaiting Model Update*>}
```

In [11]:
```
# Restricciones 2
m.addConstrs(quicksum(xp[j,m,1,o,f] for m in maquinas for o in ordenes) - quicksum(xt[j,f,a] for a in almacenes) == 0 for j in pr
```

Out[11]:
```
{(1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1): <gurobi.Constr *Awaiting Model Update*>,
 (3, 2): <gurobi.Constr *Awaiting Model Update*>,
 (3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (4, 3): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1): <gurobi.Constr *Awaiting Model Update*>,
 (5, 2): <gurobi.Constr *Awaiting Model Update*>,
 (5, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [12]:
```
# Restricciones 3
m.addConstrs(tf[j,m,n,f] <= hd[m,n,f] for j in productos for m in maquinas for n in niveles for f in fabricas)
```

```
Out[12]: {(1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>}
```

```
In [13]: # Restricciones 4
         m.addConstrs(M*X[j,m,n,o,f]-xp[j,m,n,o,f] >=0 for j in productos for m in maquinas for n in niveles for o in ordenes for f in fa
```

```
Out[13]: {(1, 1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 5, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 5, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 5, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
```

```
In [14]: # Restricciones 4b
         m.addConstrs(xp[j,m,n,o,f] -X[j,m,n,o,f] >=0 for j in productos for m in maquinas for n in niveles for o in ordenes for f in fab
```

```
Out[14]: {(1, 1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 5, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 5, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 5, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
```

```
In [15]: # Restricciones 5
         m.addConstrs(1-M*(2-X[j,m,n,o,f]-Xprima[p,m,n,o+1,f])-Y1[j,p,m,n,f] <= 0 for j in productos for p in prodprima for m in maquinas
```

```
Out[15]: {(1, 1, 1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
```

```
In [16]: # Restricciones 6
         m.addConstrs(tf[j,m,n,f]+tp[p,m,n,f]*xpprima[p,m,n,o+1,f]+ta1[j,p,m,n,f]*Y1[j,p,m,n,f]-tfprima[p,m,n,f]-M*(2-X[j,m,n,o,f]-Xprima
```

```
Out[16]: {(1, 1, 1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
```

```
In [17]: # Restricciones 7a
         m.addConstrs(tp[j,m,n,f]*xp[j,m,n,o,f] + ta2[j,p,m,n,f]*Y1[p,j,m,n,f] - tf[j,m,n,f] - M*(1-X[j,m,n,o,f]) <= 0 for j in productos
```

```
Out[17]: {(1, 1, 1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 5, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 5, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 1, 5, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
```

```
In [18]: # Restricciones 7b
         m.addConstrs(tf[j,m,n-1,f] + tp[j,m,n,f]*xp[j,m,n,o,f] + ta2[j,p,m,n,f]*Y1[p,j,m,n,f] - tf[j,m,n,f] - M*(1-X[j,m,n,o,f]) <= 0 fo
```

```
Out[18]: {(1, 1, 1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 5, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 5, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2, 5, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 2, 1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
```

```
In [19]: # Restricciones 8
         m.addConstrs(quicksum(xp[j,m,n,o,f] for m in maquinas for o in ordenes) - quicksum(xp[j,m,n+1,o,f] for m in maquinas for o in or
```

```
Out[19]: {(1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (3, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (4, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (5, 1, 3): <gurobi.Constr *Awaiting Model Update*>}
```

```
In [20]: #Restricciones 9
         m.addConstrs(quicksum(X[j,m,n,o,f] for j in productos) <= 1 for m in maquinas for n in niveles for o in ordenes for f in fabrica
```

```
Out[20]: {(1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
          (1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
```

```
 (1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 5, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 5, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 5, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 5, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 5, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 5, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [21]:
```
# Restricciones 10
m.addConstrs(quicksum(X[j,m,n,o,f] for o in ordenes) <= 1 for j in productos for m in maquinas for n in niveles for f in fabrica
```

Out[21]:
```
{(1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
```

```
 (5, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [22]:
```
# Restricciones 11a
m.addConstrs(quicksum(X[j,m,n,o-1,f] for j in productos) + quicksum(X[j,m,n,o+1,f] for j in productos) -M*quicksum(X[j,m,n,o,f] +
```

Out[22]:
```
{(1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 4, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [23]:
```
# Restricciones 11b
m.addConstrs(quicksum(X[j,m,n,2,f] for j in productos) + quicksum(X[j,m,n,5,f] for j in productos) -M*(quicksum(X[j,m,n,3,f] for
m.addConstrs(quicksum(X[j,m,n,1,f] for j in productos) + quicksum(X[j,m,n,4,f] for j in productos) -M*(quicksum(X[j,m,n,2,f] for
```

Out[23]:
```
{(1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [24]:
```
# Restricciones 11c
m.addConstrs(quicksum(X[j,m,n,1,f] for j in productos) + quicksum(X[j,m,n,5,f] for j in productos) -M*(quicksum(X[j,m,n,2,f] for
```

Out[24]:
```
{(1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [25]:
```
# Restricciones 12
m.addConstrs(xp[j,m,n,o,f] >=0 for j in productos for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(xt[j,f,a] >=0 for j in productos for f in fabricas for a in almacenes)
m.addConstrs(tf[j,m,n,f] >=0 for j in productos for m in maquinas for n in niveles for f in fabricas)
```

Out[25]:
```
{(1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (5, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [26]:
```
# Restricciones 13 (de 5) igualdades de las X-Xprima
m.addConstrs(X[1,m,n,o,f]-Xprima[1,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(X[2,m,n,o,f]-Xprima[2,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(X[3,m,n,o,f]-Xprima[3,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(X[4,m,n,o,f]-Xprima[4,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(X[5,m,n,o,f]-Xprima[5,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
```

Out[26]:
```
{(1, 1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 4, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 5, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 5, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 5, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
```

In [27]:
```
# Restricciones 14 (de 6) igualdades de xp-xpprima y tf-tfprima
m.addConstrs(xp[1,m,n,o,f]-xpprima[1,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(xp[2,m,n,o,f]-xpprima[2,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(xp[3,m,n,o,f]-xpprima[3,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(xp[4,m,n,o,f]-xpprima[4,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)
m.addConstrs(xp[5,m,n,o,f]-xpprima[5,m,n,o,f] == 0 for m in maquinas for n in niveles for o in ordenes for f in fabricas)

m.addConstrs(tf[1,m,n,f]-tfprima[1,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(tf[2,m,n,f]-tfprima[2,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(tf[3,m,n,f]-tfprima[3,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(tf[4,m,n,f]-tfprima[4,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(tf[5,m,n,f]-tfprima[5,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
```

Out[27]:
```
{(1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [28]:
```
# Restricciones 15 igualdades entre las Yjj' con las Yj'j
m.addConstrs(Y1[1,1,m,n,f]-Y2[1,1,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[1,2,m,n,f]-Y2[2,1,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[1,3,m,n,f]-Y2[3,1,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[1,4,m,n,f]-Y2[4,1,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[1,5,m,n,f]-Y2[5,1,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
```

```
m.addConstrs(Y1[2,1,m,n,f]-Y2[1,2,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[2,2,m,n,f]-Y2[2,2,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[2,3,m,n,f]-Y2[3,2,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[2,4,m,n,f]-Y2[4,2,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[2,5,m,n,f]-Y2[5,2,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)

m.addConstrs(Y1[3,1,m,n,f]-Y2[1,3,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[3,2,m,n,f]-Y2[2,3,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[3,3,m,n,f]-Y2[3,3,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[3,4,m,n,f]-Y2[4,3,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[3,5,m,n,f]-Y2[5,3,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)

m.addConstrs(Y1[4,1,m,n,f]-Y2[1,4,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[4,2,m,n,f]-Y2[2,4,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[4,3,m,n,f]-Y2[3,4,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[4,4,m,n,f]-Y2[4,4,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[4,5,m,n,f]-Y2[5,4,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)

m.addConstrs(Y1[5,1,m,n,f]-Y2[1,5,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[5,2,m,n,f]-Y2[2,5,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[5,3,m,n,f]-Y2[3,5,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[5,4,m,n,f]-Y2[4,5,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
m.addConstrs(Y1[5,5,m,n,f]-Y2[5,5,m,n,f] == 0 for m in maquinas for n in niveles for f in fabricas)
```

Out[28]:
```
{(1, 1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [29]:
```
m.setParam('TimeLimit', 2*60)
```

```
Set parameter TimeLimit to value 120
```

In [30]:
```
m.display()
```

```
Minimize
  0.0
Subject To
```

In [31]:
```
m.optimize()
```

```
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (win64)

CPU model: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 3266 rows, 1110 columns and 10830 nonzeros
Model fingerprint: 0x283f8665
Variable types: 510 continuous, 600 integer (600 binary)
Coefficient statistics:
  Matrix range     [4e-02, 1e+05]
  Objective range  [7e-01, 1e+05]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+05]
Presolve removed 2161 rows and 633 columns
Presolve time: 0.02s
Presolved: 1105 rows, 477 columns, 7381 nonzeros
Variable types: 327 continuous, 150 integer (150 binary)

Root relaxation: objective 6.111671e+03, 259 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 6111.67100    0    6          - 6111.67100      -     -    0s
H    0     0                    8350.4441096 6111.67100  26.8%     -    0s
     0     0 6111.67100    0   16 8350.44411 6111.67100  26.8%     -    0s
     0     0 6111.67100    0    6 8350.44411 6111.67100  26.8%     -    0s
H    0     0                    6167.8536075 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    5 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    8 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    8 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    6 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    6 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    6 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    6 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    8 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    4 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    4 6167.85361 6111.67100  0.91%     -    0s
     0     0 6111.67100    0    4 6167.85361 6111.67100  0.91%     -    0s
     0     2 6111.67100    0    4 6167.85361 6111.67100  0.91%     -    0s
*   93    63              23    6164.8018233 6111.67100  0.86%   7.4    0s

Cutting planes:
  Implied bound: 14
  MIR: 17
  Flow cover: 15
  GUB cover: 2
  Zero half: 1
  RLT: 2
  Relax-and-lift: 1
```

```
Explored 390 nodes (4920 simplex iterations) in 0.50 seconds (0.25 work units)
Thread count was 8 (of 8 available processors)

Solution count 3: 6164.8 6167.85 8350.44

Optimal solution found (tolerance 1.00e-04)
Best objective 6.164801823256e+03, best bound 6.164801823256e+03, gap 0.0000%
```

In [32]:
```
# Imprimir valor F.O. y valores de variables xp, xt, tf, X, Xprima, Y1, Y2
print('F.O.:',str(round(m.ObjVal,2)))
for v in m.getVars():
    print(str(v.VarName)+'='+str(round(v.x,2)))
```

```
F.O.: 6164.8
xp[1,1,1,1,1]=0.0
xp[1,1,1,1,2]=0.0
xp[1,1,1,1,3]=0.0
xp[1,1,1,2,1]=44.0
xp[1,1,1,2,2]=0.0
xp[1,1,1,2,3]=0.0
xp[1,1,1,3,1]=0.0
xp[1,1,1,3,2]=0.0
xp[1,1,1,3,3]=0.0
xp[1,1,1,4,1]=0.0
xp[1,1,1,4,2]=69.21
xp[1,1,1,4,3]=15.79
xp[1,1,1,5,1]=0.0
xp[1,1,1,5,2]=0.0
xp[1,1,1,5,3]=0.0
xp[1,1,2,1,1]=0.0
xp[1,1,2,1,2]=0.0
xp[1,1,2,1,3]=0.0
```

In [33]:
```
m.write("tesis_i1.lp")
```

Anexo 8. Ejemplo de ruteo en HeuristicLab.

A continuación, se muestra un ejemplo de la interfaz de resultados en HeuristicLab, así como el ruteo obtenido en la solución. En este caso se observan los resultados obtenidos con el algoritmo genético (GA) en la instancia con RPD = 4 y PSD = 0.0 para la fábrica 1 que se representa como el punto azul, los cuadros negros representan almacenes a donde se lleva la producción y los colores de las líneas representan un vehículo diferente. Así pues, en este ejemplo tenemos 3 rutas con una sola entrega y rutas con hasta 4 entregas.

Se decidió colocar sólo un ejemplo del ruteo ya que las 300 soluciones no aportarían el valor suficiente para el volumen que tomarían en esta sección.