UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

Desarrollo de un algoritmo evolutivo híbrido para la optimización de una cadena de suministro de dos empresas de ensamblaje

Trabajo de titulación previo a la obtención del título de Ingeniero en Sistemas

Autor:

Carlos Patricio Cevallos Tapia

CI: 0105817258

Correo: carloscevallosfc@gmail.com

Directora:

Ing. Lorena Catalina Sigüenza Guzmán, PhD

CI: 0102659687

Co-Director:

Ing. Mario Patricio Peña Ortega

CI: 0302168141

Cuenca, Ecuador

15-junio-2020

Resumen:

Dentro de las metas fundamentales del mundo se encuentra la de obtener un estado óptimo, o más bien, una solución exacta, precisa y perfecta para un problema específico. Estos estados óptimos pueden ser encontrados en diferentes áreas como la medicina, la ingeniería o la arquitectura. Por ejemplo, la ingeniería industrial tiene como uno de sus objetivos mejorar u optimizar los procesos de una empresa, con el fin de que éstas obtengan mayores beneficios con menores costes. Existe una gran cantidad de algoritmos de optimización como: genéticos, optimización de enjambre de partículas, micro-algoritmos o meméticos. Por lo que, óptimo sería aprovechar las bondades de cada uno y luego construir un algoritmo híbrido que incorpore las mejores características de estos; de esta manera, se pueda encontrar las soluciones al problema mucho más rápido, en términos de tiempo de ejecución y garantía de convergencia. En este contexto, esta investigación tiene como objetivo encontrar un algoritmo de optimización híbrido para una cadena de suministro. Entendiéndose por cadena de suministro (CS) a una red en la que se encuentran asociadas diferentes entidades como: fabricantes, proveedores, distribuidores, minoristas, transportistas y los clientes o usuarios finales. Como caso de estudio se tienen tres problemas los cuales han sido generados después de la obtención de información de dos empresas de manufactura y ensamblaje. La primera es una empresa que se dedica a fabricar muebles; mientras que la segunda se enfoca al ensamblaje de televisores y motocicletas. En el primer caso de estudio, a más de maximizar el beneficio de la cadena, también se desea maximizar la satisfacción del cliente o el nivel de servicio ofrecido al mismo. Para optimizar estos enunciados fue necesario construir funciones, conocidas como funciones objetivo; así mismo, se tuvo varias

restricciones como la demanda existente o la capacidad de almacenamiento de una planta. Mientras en el segundo caso de estudio se desea maximizar el beneficio de la CS que esté sujeto a variables tales como: costos de transporte, distribución y fabricación. En el último caso se desea minimizar el desperdicio producido por la materia prima durante el proceso de producción. Para poder determinar si este nuevo algoritmo desarrollado encuentra la solución óptima al problema se realizó una comparación con otros dos algoritmos. Esta comparación está basada en el tiempo de ejecución que le toma al algoritmo realizar su trabajo, así como su calidad de convergencia. Estos resultados se encontraron gracias a la implementación y ejecución de los algoritmos usando el mismo entorno de ejecución y las mismas características generales. Al final, se realiza una discusión y conclusión donde se determinan los puntos más fuertes del algoritmo desarrollado, así como, una comparativa con los otros algoritmos y con algoritmos híbridos realizados por otros autores.

Palabras claves: Optimización, cadena de suministro, algoritmos, convergencia, tiempo de ejecución, complejidad computacional, algoritmo híbrido.

Abstract:

A fundamental goal in the world is to obtain an optimal state, or rather, an accurate, precise and perfect solution for a specific problem. These optimal states can be found in different areas such as medicine, engineering or architecture. For example, Industrial Engineering has as one of its objectives to improve or optimize the processes of a company in order to obtain more benefits with lower costs. There are a lot of optimization algorithms, such as genetic, particle swarm optimization, micro-algorithms or memetic. Therefore, an optimal solution would be to take advantage of their benefits and then build a hybrid algorithm that incorporates their best features. In this manner, it is possible to find solutions to the problem much faster, in terms of runtime and convergence warranty. In this context, this research aims to find an optimization algorithm for a supply chain, which is a network that has different entities such as manufacturers, suppliers, distributors, retailers, transporters and customers or end-users. To this end, this work presents two case studies, a manufacturing, and an assembly company. The first company assembles furniture; while the second focuses on the assembly of televisions and motorcycles. In the first case study, the objective is to minimize the cost of the supply chain that is subject to several variables, such as transportation, distribution or manufacturing costs; and it is also desired to minimize the cost of product cutting. Meanwhile, in the second case, in addition to minimizing the cost of the chain, the objective is to maximize customer satisfaction. To optimize these statements, it was necessary to build functions, known as objective functions; likewise, there were several restrictions, such as the existing demand or the storage capacity of a product that has a distributor. In order to determine if this new hybrid algorithm finds the optimal solution to the problem, it was necessary to make a comparison among all the algorithms. This comparison is based on the runtime required for the algorithm to perform its work and the quality of the algorithm´s convergence. These results were

found thanks to the implementation and execution of the algorithms using the same execution environment and the same general characteristics. In the end, a discussion and conclusion are made where the strongest points of the hybrid algorithm are determined, as well as, a comparison with the other algorithms and with hybrid algorithms proposed by other authors.

Keywords: Optimization, supply chain, algorithms, convergence, runtime, computational complexity, hybrid algorithm.

Índice del Trabajo

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Carlos Patricio Cevallos Tapia en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Desarrollo de un algoritmo evolutivo híbrido para la optimización de una cadena de suministro de dos empresas de ensamblaje", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 15 de junio del 2020

Carlos Patricio Cevallos Tapia

C.I: 0105817258

Cláusula de Propiedad Intelectual

Carlos Patricio Cevallos Tapia, autor del trabajo de titulación "Desarrollo de un algoritmo evolutivo híbrido para la optimización de una cadena de suministro de dos empresas de ensamblaje", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor/a.

Cuenca, 15 de junio del 2020

Carlos Patricio Cevallos Tapia

C.I: 0105817258

Agradecimiento

A mis padres, quienes me apoyaron desde el primer momento que ingrese a la carrera universitaria. Así también por darme vivienda y alimento durante el desarrollo de mis actividades académicas.

A todos los docentes que me acompañaron durante el transcurso de mi carrera universitaria, los cuales fueron una fuente de conocimiento, así como un soporte cuando se tenía alguna duda. Así mismo, quiero agradecer a mis tutores la Ing. Lorena Sigüenza y el Ing. Mario Peña que me dieron apoyo, me guiaron y me ayudaron en cualquier inquietud en torno al trabajo de titulación.

Finalmente, agradecer al grupo de investigación IMAGINE los cuales me dieron la oportunidad de trabajar con ellos y que me supieron ayudar con información en temas que eran desconocidos para mi persona.

Carlos Cevallos

Dedicatoria

A mis padres, los cuales se han esforzado tantos años en hacerme un hombre de bien con buenos valores, dedico esta tesis para demostrarles lo grandes que sus enseñanzas han sido en mi vida y cuanto los quiero.

A mis amigos, quienes de manera indirecta han sido un gran apoyo emocional en mi vida, y con los que he compartido maravillosos momentos.

Carlos Cevallos

Capítulo I – Introducción

1.1 Justificación

Uno de los principios fundamentales del mundo es el de encontrar un estado óptimo, esto quiere decir, el encontrar la solución más adecuada para cualquier problema que se suscite. Estos problemas pueden ser detectados en áreas como la medicina o la ingeniería donde existe una cantidad considerable de variables de decisión. Para la resolución de estos problemas se ha desarrollado una técnica conocida como optimización. La optimización es la selección de la mejor solución dentro de un conjunto de soluciones posibles a un determinado problema que trabaja con un gran número de variables (Dantzig, 1998). Un problema de optimización consiste en minimizar o maximizar una función objetivo con la modificación de los valores de entrada.

En la industria, las empresas tratan de mejorar constantemente su estructura con la finalidad de obtener mejores resultados y satisfacer las necesidades de sus clientes. Es por esto que varios mecanismos y técnicas han sido desarrolladas para aumentar la eficiencia y eficacia de las organizaciones. Uno de estos mecanismos es una cadena de suministro, CS, la cual puede ser definida como una red en la que participan diferentes entidades como: proveedores, fabricantes, clientes, transportistas, almacenes, sobre la cual se da un flujo de productos y/o servicios e información desde y hacia el cliente. Dicha red tiene un gran número de variables asociadas que, al combinar valores diferentes de las mismas, pueden dar resultados completamente diferentes. Es por eso que, con el fin de encontrar los valores que maximicen o minimicen (según sea el caso) la función objetivo para esta CS es necesario optimizarla.

Tradicionalmente, para la optimización se ha usado programación lineal (con el algoritmo simplex), programación dinámica y sus variantes. En los últimos años se han desarrollado

algoritmos metaheurísticos que ayudan a encontrar soluciones óptimas o muy cercanas a lo óptimo para problemas desafiantes en los que la función objetivo no es convexa. Para estos algoritmos son necesarias las variables de entrada, una función objetivo que será minimizada o maximizada y finalmente restricciones.

1.2 Marco Conceptual

Poco antes de los años 50, el proceso logístico (i.e., un sistema que enlaza la producción y el mercado) estaba dado en términos militares, esto quiere decir, en base a un proceso de mantenimiento y transporte de instalaciones militares, materiales y personal (Ballou, 1978). En los años 50, se produjo un cambio conocido como la primera "Transformación", en el cual, la importancia de la logística aumentó cuando se dieron cuenta que la distribución física dentro de las fábricas o empresas era fundamental en la organización y que tenía que ser manipulado por separado (Heskett, 1964). Esta distribución física es conocida en la actualidad como CS. El concepto de administración de una CS fue forjado a inicios de los años 80. Los consultores de logísticas definieron que la CS debía ser vista como una unidad y que la decisión estratégica tomada en el nivel más alto seria usada para poder administrarla (Oliver & Webber, 1982). Después de esto, muchas revistas de fabricación, distribución, marketing, servicio al cliente, transporte y otras decidieron publicar artículos relacionados con el manejo de las CS (Tan et al., 1998).

La cadena de suministro ha sido definida por varios autores. Por ejemplo, en la publicación de Chopra y Meindl (2007) se define como una red de empresas u organizaciones que producen, venden y entregan distintos productos o servicios a diferentes clientes. Esta red incluye a

fabricantes, proveedores, transportistas, almacenes, minoristas y demás entidades relacionadas con procesos de compra/venta. Robeson (1994) define a la CS como "todas las actividades que tengan que ver en el movimiento de productos desde la materia prima hasta el usuario final". Estos procesos pueden ser la producción, el manejo del inventario, el transporte, el almacenamiento y servicio al cliente. Aunque los autores describan a la CS desde diferentes perspectivas, al final, todas estas definiciones presentan el mismo significado.

Antes de adentrarse a la optimización de una CS se debe conocer algunas técnicas y herramientas que ayudan a encontrar soluciones óptimas a algunos problemas, por ejemplo, las heurísticas y metaheurísticas. Las heurísticas son métodos que buscan encontrar las soluciones de problemas específicos que sean optimas o las más cercanas a lo óptimo, que no siempre pueden ser perfectas o eficientes (Russell & Norvig, 2009). Por otra parte, meta significa "más allá" o "alto nivel" (Lazar, 2000), por lo que se puede decir que las metaheurísticas son un conjunto de técnicas o estrategias inteligentes que ayudan a encontrar soluciones óptimas a un problema determinado.

Los algoritmos metaheurísticos han pasado por un gran proceso de desarrollo y evolución. A continuación, se describen algunos de los algoritmos más importantes y que han marcado relevancia en términos de optimización. Holland, en 1975, inventó el algoritmo genético. Este algoritmo consiste en generar una población inicial, de cual se escogerá a los mejores padres que se crucen entre sí y mute cada individuo con la finalidad de obtener una nueva población. Si esta nueva población no es la solución, se debe realizar otra vez el proceso de selección de padres, caso contrario, la solución será la elegida (Holland, 1992b). En 1986, es desarrollado el algoritmo de búsqueda Tabú, que usaba memoria adaptativa, esto quiere decir que en lugar de guardar la

solución total, solo se guardaban atributos específicos de la solución (F. Glover & Laguna, 1999). Dorigo y Di Caro (1999), desarrollan el algoritmo de colonia de hormigas, el cual es usado para solventar problemas grandes de optimización. Éste consiste en encontrar el camino más corto desde el origen hacia un determinado recurso. En 1995, Kennedy y Eberhart desarrollan el algoritmo de Optimización de Partículas, el cual consiste en imitar el comportamiento social de los humanos e insectos. A diferencia de los algoritmos genéticos, aquí las partículas viajan a través del espacio de búsqueda con el fin de determinar si esta solución es óptima o no (Kennedy & Eberhart, 1995). La optimización multi-objetivo fue desarrollada en 2002, y consiste en resolver problemas con múltiples restricciones y funciones objetivo. En 2005, el algoritmo de la colonia artificial de abejas es definido, el cual se centra en el comportamiento de las abejas al momento de encontrar la miel (Pham et al., 2005), muy similar a lo que sucede con el algoritmo de colonia de hormigas.

1.3 Estado del Arte

La optimización, dentro de las CS, es un tema bastante investigado e implementado. A continuación, se presentan algunas de estas investigaciones en donde se han usado diferentes algoritmos de optimización. En el artículo de Altipamar, Gen, Lin y Paskoy (2006) se expone el uso de un algoritmo genético dentro de una compañía que produce productos plásticos en Turquía, el cual encuentra soluciones casi óptimas para redes de cadena de suministro multi-objetivo. Jamshidi, Fatemi Ghomi, y Karimi (2012) presentan la optimización de una CS para resolver problemas ambientales, específicamente de la cantidad de NO2, CO y partículas volátiles orgánicas producidas por las instalaciones y en el transporte. Esta optimización ha sido realizada usando un algoritmo hibrido genético denominado “Taguchi”. Habib, Rahman, Alam, Zoarder y Haque (2016) usan un algoritmo genético que se encarga de obtener un conjunto de soluciones

para una ruta y costo de trasporte dentro de cualquier red de CS, con el fin de dar una solución general para el problema de ruteo.

1.4 Objetivos y Preguntas de Investigación

La investigación tiene como objetivo principal desarrollar un algoritmo metaheurístico híbrido para maximizar el beneficio generado por una CS basado en la manipulación de los indicadores o variables de decisión de la misma.

Este objetivo a su vez puede ser desglosado en objetivos más específicos que son presentados a continuación.

1. Realizar una revisión sistemática de los algoritmos metaheurísticos más utilizados y los algoritmos presentados en los trabajos de titulación de Orellana (2020) y Berrezueta (2020).
2. Analizar las variables de entrada que van a ser usadas, las funciones objetivo y las restricciones pertinentes para poder optimizar la cadena.
3. Realizar un análisis de fortalezas, oportunidades, debilidades y amenazas (FODA) de un conjunto de algoritmos revisados en el primer objetivo, con el fin de encontrar los mejores candidatos para el desarrollo del nuevo algoritmo, así como los que serán usados para la comparativa de resultados.
4. Desarrollar un algoritmo metaheurístico híbrido que sea construido usando las fortalezas de otros algoritmos; y que, comparado con otros algoritmos ya existentes obtenga una solución óptima con un menor tiempo de ejecución, mientras maximiza el beneficio, el

nivel de servicio de una CS y minimiza el desperdicio producido por el corte de materia prima dentro de las plantas.

5. Desarrollar una aplicación que optimice una CS ejecutando el algoritmo híbrido desarrollado.

Así mismo, esta investigación cuenta con la siguiente pregunta de investigación: ¿Un algoritmo metaheurístico híbrido puede realizar la optimización de una CS de una manera más eficiente en términos de tiempo y calidad de la solución? Para responder a esta pregunta es necesario analizar la literatura y encontrar los algoritmos metaheurísticos que hayan sido usados en otros estudios para solucionar este tipo de problemas. Una vez encontrados estos algoritmos, se debe seleccionar las fortalezas de cada uno y construir o desarrollar uno nuevo.

1.5 Marco Metodológico

El enfoque de este trabajo de titulación es de tipo cuantitativo debido a que, la comparación entre los diferentes algoritmos de optimización está basada en las funciones objetivo que son el valor del beneficio obtenido en dólares, el nivel de servicio en porcentaje y al tiempo en segundos, los cuales son valores numéricos cuantificables. El alcance de la investigación es de tipo correlacional debido a que se cuenta con una variable dependiente (función objetivo) que cambia según las variables independientes (indicadores de la CS). El diseño de este trabajo es de tipo no experimental longitudinal ya que se recolectan datos a través del tiempo, para poder generar un modelo que maximice el beneficio y nivel de servicio de la CS y minimice el desperdicio generado por las plantas.

Este trabajo de titulación presenta tres casos de estudio. En los tres casos son empresas de ensamblaje que cuentan con varios procesos que necesitan ser optimizados para aumentar los beneficios de los mismos. Estos casos de estudio fueron analizados por los autores Orellana (2020) y Berrezueta (2020) en sus respectivos trabajos; en donde presentan la optimización de diferentes problemas relacionados con CS usando algoritmos de optimización.

Las variables a tener en cuenta dentro de las CS son las variables de entrada de la función objetivo, que según alguna modificación del mismo pueden variar completamente el resultado de la función. Ya que este proyecto cuenta con tres casos de estudio, han sido realizadas tres funciones objetivo que se encuentran definidas en el Capítulo III.

1.6 Plan de Trabajo

El proceso para la investigación está comprendido en cuatro etapas:

1. La revisión de la teoría de CS, optimización, los algoritmos de optimización y los algoritmos usados en los trabajos de titulación de Orellana (2020) y Berrezueta (2020).
2. Se procede a revisar las variables del problema, las entradas, las funciones objetivo y las restricciones del mismo.
3. Construcción de un nuevo algoritmo metaheurístico híbrido que realice de una manera mucho más eficiente (en términos de tiempo y que garantice la convergencia) la optimización de la CS, con el fin de reducir los costos de la misma. Estos algoritmos son seleccionados en base a sus ventajas como: el tiempo de ejecución del algoritmo y la velocidad de convergencia de la función objetivo para alcanzar el valor mínimo.

4. Finalmente, la última etapa comprende la construcción de un software parametrizable para la ejecución del algoritmo metaheurístico híbrido desarrollado. Este aplicativo se encuentra en un servidor web, para que pueda ser accedido por medio del Internet.

1.7 Descripción del Contenido

El documento a continuación presenta la estructura que se visualiza en la Figura 1. Un Marco Teórico que está basado en revisión literaria de documentos y artículos académicos. Este marco teórico tiene una primera parte en donde se definen conceptos de optimización, CS y algoritmos metaheurísticos. En la siguiente sección, se realiza una descripción del Estado del Arte de los trabajos de optimización de CS; esto con el fin de reforzar la importancia del trabajo.

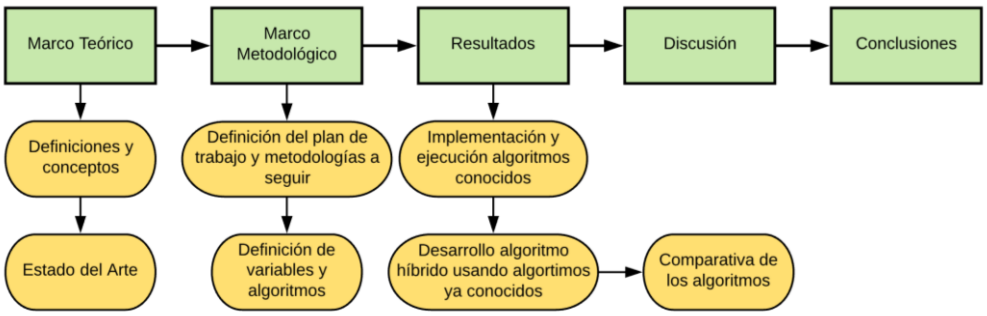

Figura 1. Contenido del documento.

Dentro del capítulo Marco Metodológico se define el tipo de investigación que se llevó acabo, el plan de trabajo seguido y las metodologías usadas. Por otra parte, más adelante se definen las diferentes variables como: función objetivo, restricciones y los algoritmos a ser implementados.

En la sección de Resultados se reporta el proceso de implementación y ejecución de los algoritmos seleccionados. Luego, se documenta el desarrollo de un nuevo algoritmo que comparte

las características de otros algoritmos conocidos y, como parte final, se comparan todos los algoritmos con el fin de analizar sí el nuevo algoritmo construido obtiene el valor óptimo de las variables de decisión.

Finalmente, se realizan las etapas de Discusión y Conclusión. En la Discusión se analizan los resultados de la comparativa y se trata sobre cómo la combinación de las fortalezas de los algoritmos puede mejorar el proceso de la generación de resultados. Mientras que, en la Conclusión se menciona sobre proyectos futuros y se responde la pregunta de investigación planteada al inicio de este documento.

Capítulo II – Marco Teórico

2.1 Cadena de Suministro

La cadena de suministro (CS) durante varios años ha sido un área muy estudiada en los aspectos estratégicos de las empresas. Sin embargo, para llegar a ser lo que es ahora, la CS ha pasado por un gran cambio a través del tiempo. A finales de los años 40, la logística estaba dada en términos militares, es decir, con todos los procesos involucrados en la obtención, mantenimiento y transporte de las instalaciones, materiales y personal militar (Ballou, 1978). Después de los años 60, se empezó con el estudio y la práctica de la distribución física y la logística vista como un conjunto de medios para satisfacer un objetivo (Heskett, 1964). El término "Administración de la Cadena de Suministro" (SCM, por las siglas en inglés de Supply Chain Management) fue acuñada en los años 80, y desde ese momento tomó un fuerte impacto dentro de las empresas y compañías. A finales de los 80s, algunos investigadores y consultores consideraban que SCM se basaba en la administración de la logística en el ámbito externo a la empresa. Este criterio se sujetó en base a la definición de logística puesta por el Consejo de Administración de Logística (Oliver & Webber, 1982). El Consejo definía a la logística como "el proceso de planear, implementar y controlar eficientemente el flujo y almacenamiento de materia prima, inventario en proceso, productos terminados y su información relacionada desde el origen hasta el cliente, de manera eficiente y con el menor costo posible" (Oliver & Webber, 1982). En los años 90, la definición de SCM cambió nuevamente y pasó al concepto de integración y manejo de procesos entre componentes o entidades participantes dentro de la cadena. De esta forma, Drucker (2007) habló sobre el cambio de paradigma dentro de la literatura de administración diciendo que: "Uno de los más grandes cambios de paradigma dentro de la administración de los negocios modernos, es que cada uno de los negocios individuales ya no compiten más como entidades autónomas, sino como cadenas de

suministro". También explicó que el manejo de los negocios había entrado a la era de la competencia de redes internas y, que el éxito de un negocio dependerá de la capacidad de la administración para integrar la red interna de la compañía con las relaciones de negocio.

La CS puede ser definida de varias maneras según los distintos autores. Cooper y Ellram (1993) definen a la CS como "una filosofía de integración que se encarga de analizar y administrar todo el flujo de distribución desde el proveedor hasta el usuario final o cliente". Según Lummus y Alber (1997), la CS es una red de entidades, por donde el material o producto viaja; esta red incluye: centros de distribución, minoristas, clientes, transportistas, fábricas, y proveedores. Robeson (1994) define a la CS, como "todas las actividades que tengan que ver en el movimiento de productos desde la materia prima hasta el usuario final". Estos procesos pueden ser la producción, el manejo del inventario, el transporte, el almacenamiento y servicio al cliente. En la publicación de Chopra y Meindl (2007) se define a la CS como una red de empresas u organizaciones que producen, venden y entregan distintos productos o servicios a diferentes clientes. Esta red incluye a fabricantes, proveedores, transportistas, almacenes, minoristas y demás entidades relacionadas con procesos de compra/venta. Según Ballou (2004), la logística y CS es un conjunto de actividades como transporte, control de inventarios que pasan por un proceso iterativo a lo largo del canal de flujo, por donde la materia prima se convierte en el producto final que será entregado a los usuarios finales o clientes. Aunque los autores manejen otras palabras, todas estas definiciones expresan conceptos similares; por lo tanto, se puede definir a una CS como una red en el cual participan varias entidades como: proveedores, fabricantes, clientes, transportistas, minoristas, centros de distribución, almacenes y, sobre todo, los productos que se moverán durante el flujo de esta red, desde su materia prima hasta llegar al consumidor final.

La importancia de las CS dentro de las organizaciones es un aspecto fundamental para la prosperidad del negocio. La logística tiene como finalidad crear valor, un valor tanto para los clientes y proveedores de la empresa como para los accionistas de la misma. Este valor se expresa en términos de tiempo y lugar. Los productos o servicios brindados no tienen ningún valor a menos que estén en manos de los clientes cuando y donde ellos deseen tenerlos; el cuándo hace referencia al tiempo y el dónde al lugar (Ballou, 2004). Por ejemplo, las entradas al estreno o prestreno de una película no tendrían valor si no están disponibles en el tiempo y lugar donde se proyecte la película; así también, no tendrían valor si ya no existen entradas disponibles debido a que los inventarios no satisfacen con la demanda de los usuarios. Por este motivo, es bueno tener una apropiada dirección logística que trate de dar los mejores beneficios hacia los usuarios finales. La CS cuenta con varias actividades que ofrecen información para los diferentes interesados del producto y/o servicio. Estas actividades son: predicción para la capacidad de reserva o el inventario inicial, órdenes de compra para iniciar la producción o la distribución del producto, administración del inventario y monitoreo del movimiento del producto, liquidaciones financieras y reportes (Singh, 1996).

Cuando la administración se da cuenta que la logística y la CS afectan de gran manera a los costos dentro de una empresa, estos pueden ser usados para poder penetrar nuevos mercados, y así aumentar los beneficios de la empresa. El valor de un producto depende de qué tan disponible esté dentro del mercado. Con esto se quiere decir que, si el producto no está disponible no tiene valor, y si este producto es trasladado desde la empresa hacia el consumidor se genera un valor. Existen cuatro tipos de valor en los productos o servicios: forma (que tipo de producto es), tiempo (cuando se encuentra en el mercado), lugar (donde se encuentra) y posesión (como se obtiene). La logística, como se había mencionado, crea dos de estos cuatro valores, tiempo y lugar. La manufactura, por

otra parte, crea el valor de forma cuando el dinero invertido o gastado se convierte en productos terminados. El valor de posesión es responsabilidad del marketing, la ingeniería y las finanzas, donde el valor se crea cuando se da publicidad y apoyo técnico al producto para que pueda ser adquirido por los clientes. La SCM incluye producción, por lo que se tiene tres de los cuatro valores. Debido a varios factores que han ido evolucionando como la comida rápida, los cajeros automáticos y el Internet, las personas (consumidores potenciales) desean obtener sus productos y servicios de una manera más rápida. Por este motivo, las compañías han estado más atentos a aplicar el concepto de respuesta rápida, con el fin de satisfacer las necesidades de ellos mismo y del cliente (Ballou, 2004) .

En la actualidad, las CS han sido un tema muy explorado dentro de las empresas. El número de artículos de investigación dentro de las revistas científicas ha incrementado. En la Figura 2 se muestra el número de artículos publicados en los últimos veinte años en las revistas AlChE Journal, Chemical Engineering Science, Computers and Chemical Engineering, e Industrial and Engineering Chemistry Research, que son consideradas revistas muy importantes debido a que en éstas han sido publicados varios artículos relacionados con los avances tecnológicos dentro de la ingeniería (Garcia & You, 2015).

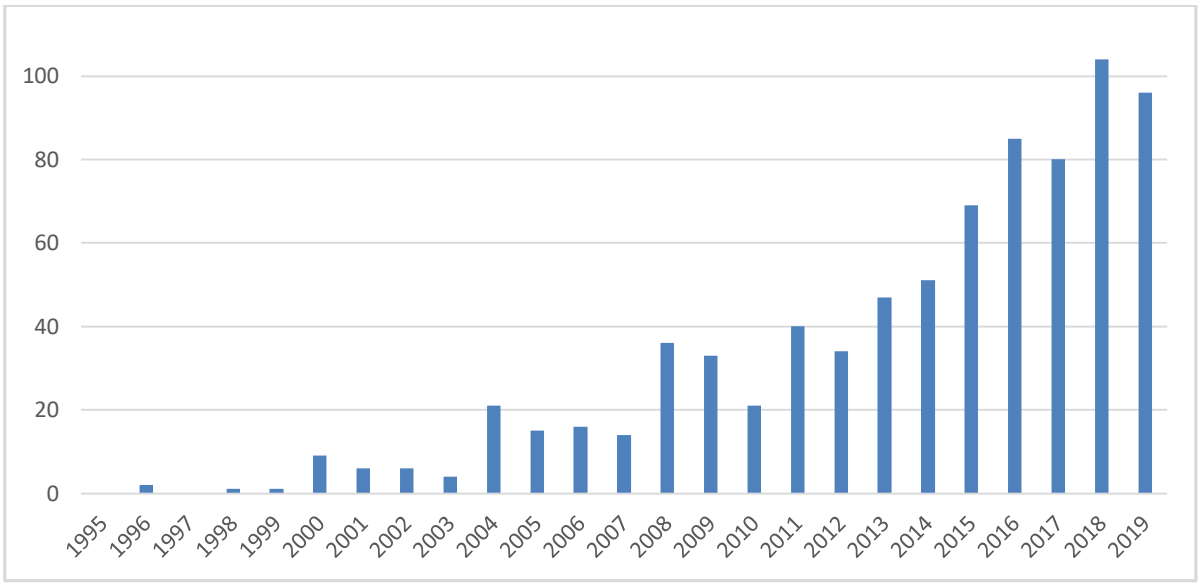

Figura 2. Artículos publicados relacionados con "cadena de suministro" desde 1995 hasta el 2019 (basado en García y You (2015)).

Entre las oportunidades y retos que se documentan en la literatura se encuentran la presentada por los autores García y You (2015). Según estos autores, existen dos áreas que pueden ser estudiadas para encontrar oportunidades: sustentabilidad de la energía y optimización de toda la empresa. En el primer punto que se refiere a los problemas de sustentabilidad de la energía, la pregunta que se genera es: ¿Cómo la sustentabilidad de la energía afecta a la CS? Este problema está muy relacionado con la CS, pero no en el ámbito interno de la misma, sino de manera externa, específicamente de parte de los clientes. Algunos de los clientes en la actualidad se ponen a pensar en los problemas ambientales que son generados por las compañías, por lo que prefieren confiar en las organizaciones que traten de reducir la contaminación en el planeta. De esta forma, las empresas tratan de enfocar en reducir la contaminación que producen sus negocios y mejorar el estatus de su compañía (Garcia & You, 2015).

El segundo punto trata sobre la gran competencia industrial que existe. En la actualidad, las compañías tratan de acaparar la mayor cantidad de mercados con el fin de obtener más ganancias a la hora de ofrecer sus productos y servicios. Por este motivo, las empresas tratan de desarrollar sus CS para poder controlar el flujo de sus productos. Las compañías deben tener en cuenta cómo administrar estas cadenas para que sean lo más eficientes y eficaces posibles (Grossmann, 2005; Varma et al., 2007), con la finalidad de obtener mayores ganancias en los negocios. Sin embargo, para realizar este proceso, las empresas se encuentran con retos que deben ser superados para cumplir los objetivos.

Los retos de las CS se dividen en cuatro factores importantes. El primero, que también es importante para el presente estudio, hace referencia a los retos multi-escala que tiene que ver con la capacidad computacional de optimizar una CS. Aquí es donde nace la terminología de complejidad temporal, espacial y computacional (Garcia & You, 2015). La complejidad temporal está relacionada con el tiempo que tome ejecutar un proceso computacional; mientras el tamaño de problema es más grande, el tiempo en resolverlo con algún método será mucho mayor. La complejidad espacial viene dada en términos de la memoria utilizada por la computadora para almacenar datos temporales mientras se realiza el proceso de resolución de problemas. Finalmente, la complejidad computacional se relaciona con el número de instrucciones que puede ejecutar un proceso computacional. Otro problema que se genera para las CS son las incertidumbres que pueden darse dentro de la compañía. Por ejemplo, los cambios de personal o de gerencia dentro de la organización pueden afectar con las políticas de la empresa. Esto a su vez generaría cambios no solo en la parte organizacional sino funcional de la empresa, por lo que la CS debe estar preparada para tolerar este tipo de cambios. Como reto también se puede encontrar el criterio de diseño de la CS. La organización debe tener en cuenta cuáles son las fortalezas, objetivos y visión de la

empresa. Una vez analizadas estas características se debe empezar a estructurar y diseñar la CS con las debidas restricciones, funciones objetivo y variables involucradas dentro del proceso.

El ultimo reto que se puede encontrar dentro de la CS es el reto multijugador. Este problema se refiere a que las decisiones tomadas dentro de la CS no están centralizadas, por lo que cada entidad puede tener un diferente objetivo y, por lo tanto, la forma de optimizar será diferente para cada uno de estos aspectos.

2.2 Optimización

Uno de los principios fundamentales de este mundo es el de buscar el estado óptimo. Esto se puede reflejar desde los átomos que intentan formar lazos con el fin de minimizar la energía de sus electrones (Pauling, 1960). Así mismo puede ser visto en el principio de biología conocido como "supervivencia del más apto" (Spencer, 1864), en donde, mediante el proceso evolutivo, las especies se adaptan a los diferentes ecosistemas. Uno de estos ejemplos es el ser humano que ha sabido evolucionar a un punto casi perfecto en donde puede adaptarse en cualquier circunstancia que se suscite. Los humanos son seres con una visión mucho mayor que el resto de criaturas en la tierra, es por eso que se trata de alcanzar los mayores beneficios realizando la mínima cantidad de esfuerzo. Cuando alguna área es muy abstracta, la matemática siempre estará involucrada con el fin de ayudar a entender de mejor manera estas situaciones. La optimización global es una rama de las matemáticas que mediante el uso de análisis numérico ayuda a resolver problemas de optimización. La optimización global trata de encontrar la mejor solución de un conjunto de elementos que satisfaga un criterio. Este criterio está expresado con valores numéricos y es conocida como "función objetivo" (Weise, 2009).

La función objetivo puede ser representada como la función matemática presentada en la Ecuación (1).

$$f: X \rightarrow Y \; con \; Y \subseteq R \qquad (1)$$

Donde, el rango Y de una función objetivo debe ser un subconjunto de números reales. El dominio X de la función es conocido como el espacio del problema, y puede ser cualquier tipo de elemento, como listas, números, entre otros. La optimización global encierra a todas aquellas técnicas que ayuden a encontrar los mejores elementos x* en X con respecto a la función objetivo (Weise, 2009).

El proceso de optimización trata de buscar un conjunto de nuevas soluciones en base a una solución ya conocida, con el fin de converger al resultado esperado. Por ende, se puede decir que, el proceso de optimización trata de encontrar el óptimo global del conjunto de soluciones, aunque conseguir esto a veces es un proceso complejo (Koziel & Yang, 2011). Este procedimiento consiste en dos componentes: el modelo y el optimizador. El modelo, el cual puede ser matemático o numérico, se encarga de representar el problema mediante el uso de ecuaciones que pueden ser resueltas numéricamente. Si no existe una relación directa entre el modelo real y el modelo matemático, quizá el problema a resolver no es el que se desee. Algo muy importante es también escoger el algoritmo u optimizador correcto, esto en base a las variables con las que se cuente. Los optimizadores son algoritmos que ayudan a resolver problemas de optimización.

Según Beheshti y Shamsuddin (2013), se puede clasificar a los algoritmos en cuatro grupos: basado en el origen del algoritmo, de acuerdo al número de soluciones, uso de la función objetivo y uso de memoria. Ejemplos de los algoritmos basados en el origen son los algoritmos inspirados

y no inspirados en la naturaleza (Koziel & Yang, 2011). Muchos de estos algoritmos son basados en eventos que suceden en la naturaleza, o comportamientos de distintas criaturas en el ecosistema. Algunos algoritmos inspirados en la naturaleza son la Optimización de Colonia de Hormigas y la Optimización de Enjambre de Partículas (OEP). Por otra parte, existen algoritmos que no se basan en la naturaleza como la Búsqueda de Tabú o la Búsqueda Local Iterada. Otros algoritmos se clasifican en base a la población y búsqueda de un solo individuo. Estos algoritmos se clasifican acorde al número de soluciones en el mismo instante. Por ejemplo, existen los métodos de trayectoria que trabajan usando una sola solución, y usualmente son algoritmos basados en la búsqueda local. Mientras que, existen otros algoritmos como el OEP en donde se tienen varias partículas moviéndose por el espacio con el fin de encontrar soluciones locales y luego interactuar entre ellos para obtener una solución global. Los algoritmos también pueden clasificarse según su función objetivo. En ocasiones, los algoritmos usan una función objetivo que no cambia durante el tiempo, como la Búsqueda Local Guiada. También se tienen las funciones multi-objetivo, los cuales tienen que cumplir dos o varios objetivos; por ejemplo, se necesita comprar un vehículo que sea muy rápido pero que no consuma mucho combustible. Finalmente, también se puede clasificar a los algoritmos según el uso de memoria. Los algoritmos que emplean poca memoria usan el último estado para realizar la siguiente acción en el proceso de búsqueda. Por otra parte, existen algoritmos que utilizan memoria a largo o a corto plazo para almacenar soluciones que ya han visitado, movimientos que ya han hecho y decisiones que ya habían tomado.

Sin embargo, esta no es la única clasificación de los algoritmos. Según Weise (2009), se puede clasificar a los algoritmos en dos grupos adicionales: según las propiedades y según el método de operación (funcionamiento del algoritmo).

Clasificación según las propiedades.

La taxonomía, mostrada en la Figura 3, clasifica los algoritmos de optimización de una manera teórica. Sin embargo, en el caso de un ingeniero de software se está más interesado en aspectos como la velocidad y la precisión del algoritmo (Weise, 2009). La velocidad está relacionada con el tiempo de ejecución que toma el algoritmo hasta llegar a conseguir la solución deseada. Por otra parte, la precisión se refiere a qué tan cerca está la solución del óptimo, este último dependerá de la convergencia del algoritmo. También se debe tener en cuenta que con un poco más de tiempo de ejecución del algoritmo se puede ganar mejoras en su precisión. Los algoritmos de optimización en ámbitos de velocidad pueden ser distinguidos en dos tipos: en línea y fuera de línea.

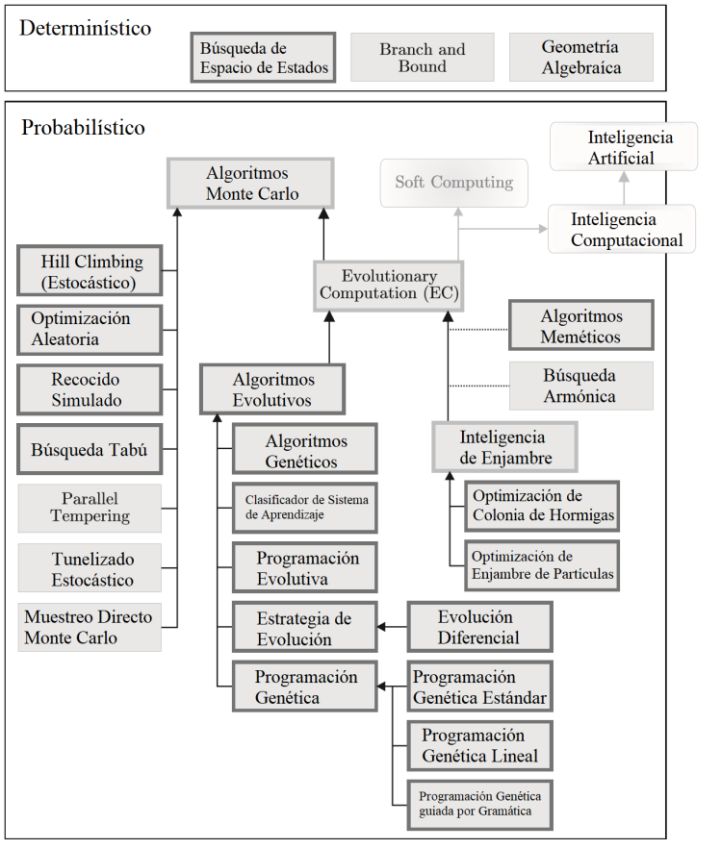

Figura 3. Taxonomía de los algoritmos de optimización global (Weise, 2009).

Optimización en línea.

Estos problemas necesitan ser resueltos de una manera muy rápida, si es posible en cuestión de milisegundos. Por este motivo, es necesario en la optimización encontrar técnicas de balanceo de carga para tener ganancias en los tiempos de ejecución (Weise, 2009).

Optimización fuera de línea.

Aquí el concepto de tiempo es relativo, esto quiere decir que, no importa cuánto tiempo tome encontrar la solución óptima, sino qué tan precisa es. Esto se puede ver reflejado en áreas de minería de datos (Weise, 2009).

Clasificación según el método de operación.

Usando la misma Figura 3, se presenta la clasificación según el método de operación de una manera teórica. Los algoritmos se dividen en dos clases básicas: algoritmos determinísticos y probabilísticos. Los algoritmos determinísticos son usados cuando existe una relación entre las características de la solución posible y su utilidad con el problema dado. De esta manera, se puede explorar el espacio de manera eficiente para encontrar las soluciones. Por otra parte, los algoritmos probabilísticos tratan de encontrar soluciones a los problemas en un tiempo más corto. Esto no implica que necesariamente se encuentre la solución óptima, pero sí una solución lo suficientemente aproximada como para ser válida (Weise, 2009). Durante todo este proceso, es necesario usar funciones heurísticas que ayuden a decidir cuáles pueden ser los nuevos candidatos a ser analizados dentro de un conjunto de soluciones. Los algoritmos determinísticos usan heurísticas para realizar el procesamiento de los candidatos. Mientras que, los algoritmos probabilísticos seleccionan solo los elementos que han sido escogidos previamente por la heurística.

Una heurística es definida como la parte de un algoritmo de optimización que, mediante la información obtenida por el algoritmo, ayuda a decidir cuál solución candidata puede ser probada después, o cómo un nuevo elemento puede ser creado (Michalewicz & Fogel, 2004; Pearl, 1984; Rayward-Smith et al., 1996). Por otra parte, una metaheurística es un método para resolver problemas generales; ésta trabaja con funciones objetivo o heurísticas de una manera eficiente (Blum & Roli, 2003; F. W. Glover & Kochenberger, 2003).

2.3 Algoritmos Metaheurísticos

Son algoritmos que usualmente suelen ser inspirados en la naturaleza (Yang, 2011). En la actualidad son de los algoritmos más usados en diferentes áreas de estudio (Hussain et al., 2019). A continuación, se presentan varios algoritmos metaheurísticos que han hecho un fuerte impacto en el área científica y de investigación.

2.3.1 Algoritmo Genético.

Inventado por J Holland (1992) y sus colaboradores entre los años 60 y 70, está basado en la teoría de la evolución de Charles Darwin. Holland fue el primero en usar operadores genéticos tales como: cruce, mutación, recombinación y selección. Este algoritmo cuenta con dos grandes ventajas. La primera es la capacidad de trabajar con problemas complejos y la segunda es trabajar de manera paralela (Hasançebi et al., 2009).

Como primer paso, el algoritmo genera una población inicial con varios individuos o elementos, después, se escoge los padres de ese conjunto. Entonces, se aplican las operaciones de cruce y mutación entre ellos con el fin de conseguir nuevos individuos o hijos. Estos hijos pueden reemplazar a los anteriores individuos y convertirse en padres y así realizar un proceso iterativo hasta encontrar la mejor solución o una de las mejores (Arifovic, 1994).

2.3.2 Optimización de Enjambre de Partículas.

Kennedy y Eberhart (1995) desarrollaron el algoritmo de optimización de enjambre de partículas. Como su nombre lo indica, este algoritmo está basado en el comportamiento de los enjambres, por ejemplo, las bandadas de pájaros y los bancos de peces. La Optimización de Enjambre de Partículas (PSO) ha sido aplicada a la mayoría de áreas donde se trabaja con optimización e inteligencia computacional. En la actualidad, existe un gran número de variantes de las PSO, y también son usados para generar algoritmos híbridos con otros algoritmos ya existentes (Lee & Geem, 2004). PSO no cuenta con los operadores de cruce o mutación (Kaur & Chhabra, 2020).

Este algoritmo consiste en encontrar dentro del espacio de búsqueda los mejores valores para la función objetivo, ajustando las trayectorias de los individuos que, en este caso, son conocidos como partículas (Lee & Geem, 2004). Cada partícula cuenta con una posición, así como su mejor posición local. Así mismo, una de estas partículas tendrá la mejor posición global conocida, por lo que, en la nueva iteración todas estas partículas tenderán a cambiar su posición en base a la posición global y su posición local.

2.3.3 Recocido Simulado.

Es un algoritmo que ha sido aplicado a casi todas las áreas de optimización, fue desarrollado por Kirkpatrick, Gelatt y Vecchi (1983). A diferencia de otros métodos basados en gradiente y métodos de búsqueda determinística, este algoritmo tiene la habilidad de evitar quedar atrapado en un mínimo local, el cual es una solución óptima pero solo dentro de un espacio específico del problema, por lo que no es el óptimo global y podría ser una solución no deseada (Yang, 2011).

Este algoritmo trata de encontrar la mejor aproximación al óptimo global en un espacio de búsqueda de grandes dimensiones (este espacio debe ser discreto). Este algoritmo ha sido usado para resolver varios problemas de optimización global y es muy conocido entre los matemáticos (Kaur & Chhabra, 2020).

2.3.4 Búsqueda de Gallinas.

Este algoritmo fue desarrollado en el 2009 por Yang (2011). Está basado en el parásito de las crías de algunas especies de gallinas. Así mismo, el algoritmo analiza que algunas especies de gallinas depositan sus huevos en un nido comunitario, después de eso sacan otros huevos con el fin de aumentar las probabilidades de que sus huevos eclosionen.

2.3.5 Optimización de Colonia de Hormigas.

Es un algoritmo de optimización propuesto por Dorigo y Di Caro (1999). Es usado para la resolución de problemas de optimización que son muy complejos. Este algoritmo es usado para encontrar soluciones a problemas relacionados con encontrar las rutas óptimas (Blum, 2005).

El primer paso de este algoritmo consiste en que una hormiga viaja a través de varios caminos con el fin de encontrar el camino más corto desde su posición actual hasta la comida o la posición final. Mientras esta hormiga regresa a la fuente va dejando marcadores conocidos como feromonas en todo el camino de vuelta a su hogar. Estos marcadores ayudan a que las demás hormigas que puedan elegir cuál es el camino más corto para llegar a conseguir la comida (M. Dorigo et al., 1996).

Optimización Multi-Objetivo.

Esto más que un algoritmo es una técnica que permite simular problemas en donde se necesita trabajar con varias funciones objetivo. Varios algoritmos usan esta técnica para resolver problemas mucho más complejos. Algunos de estos son: NSGA-II (Non-dominated Sorting Genetic Algorithm), el cual es una adaptación multi-objetivo del algoritmo genético (Deb et al., 2002);y MOPSO (Mulit Objective Particle Swarm Optimization) una versión multi-objetivo del algoritmo PSO (C.A. Coello Coello & Lechuga, 2002).

2.4 Estado del arte de los algoritmos de optimización dentro de las CS

Las metaheurísticas a lo largo de los últimos años han sido muy utilizadas para los procesos de optimización de las CS. A continuación, se presentan algunos de estos proyectos, así como los algoritmos que han sido utilizados y con qué finalidad.

Algoritmos Genéticos.

En la literatura se ha encontrado que los algoritmos más usados en términos de optimización de CS han sido los algoritmos genéticos. En la publicación de Chan, Chung y Wadhwa (2005) se desarrolla un algoritmo genético híbrido para la producción y distribución en los problemas relacionados con CS en donde se encuentran involucrados varias plantas. Este artículo organiza todos los indicadores relacionados con la toma de decisiones dentro de la empresa. Luego los analiza con el fin de tomar las mejores opciones para el beneficio de la misma. Al final del artículo se presenta que la optimización usando este tipo de algoritmo es confiable y robusta. Así mismo, en el artículo escrito por Altiparmak, Gen, Lin y Paksoy (2006) se expone el uso de un algoritmo genético. Como caso de estudio se tiene una compañía que produce productos plásticos en Turquía. El objetivo del algoritmo es encontrar las soluciones cuasi-óptimas para la CS que trabaja con

varias funciones objetivo. Por otra parte, Yeh y Chuang (2011) presentan la resolución de un problema multi-objetivo en donde son usados dos algoritmos genéticos. El problema que abordan es el encontrar un modelo en donde estén involucrados cuatro objetivos: costo, tiempo, calidad de producto y la puntuación de evaluación verde. En esta publicación se comparó el promedio de soluciones Pareto-óptimas y el CPU (complejidad temporal, computacional y espacial) de los dos algoritmos. Como se mencionó en la parte de retos y oportunidades de la CS, una de las oportunidades es la sostenibilidad de la energía. Es por eso que, Jamshidi, Fatemi y Karimi (2012) presentaron, en su artículo, la optimización de una CS para resolver problemas ambientales, específicamente de la cantidad de NO2, CO y partículas volátiles orgánicas producidas por las fábricas que se encuentra dentro de la red y en el trasporte de los productos hacia los consumidores. Este proceso de optimización se realizó usando un algoritmo híbrido genético denominado "Taguchi". Finalmente, en la publicación realizada por Habib et al (2016) se usa un algoritmo genético que se encarga de obtener un conjunto de soluciones para una ruta y costo de trasporte dentro de cualquier red de CS. Este proceso también podría ser manejado mediante la optimización de colonia de hormigas.

Algoritmo de Optimización de Enjambre de Partículas.

De igual manera, se observó mediante la literatura que el algoritmo PSO ha sido aplicado con el fin de optimizar procesos de CS. En la publicación de Gao, Zhang, Lu y Wee (2011) se enuncia cómo la función objetivo en una CS puede ser vista de dos diferentes maneras. La primera en que la función toma como papel principal al consumidor sobre el proveedor; mientras que, en la segunda función se tiene como actor principal al proveedor sobre el consumidor. Este tipo de modelos son conocidos como bi-nivel, y pueden ser resueltos mediante el uso de un algoritmo de optimización de enjambre de partículas. En la investigación realizada por Mousavi, Bahreininejad,

Musa y Yusof (2017), el principal objetivo fue encontrar las ubicaciones óptimas de vendedores potenciales en relación a la cantidad ordenada por los consumidores. En este proceso se incluye el transporte y la ubicación con el fin de reducir costos dentro de una CS diseñada para la entrega de productos con múltiples vendedores y compradores. Este problema ha sido resuelto usando un algoritmo modificado de PSO. Con el fin de comprobar la mayor eficiencia del algoritmo, se ha comparado éste con un algoritmo genético, y se evidencia que, en efecto, se visualiza un mejor rendimiento del algoritmo de enjambre de partículas.

Otros algoritmos.

En la mayoría de publicaciones se habla sobre la optimización multi-objetivo; en este proceso se puede minimizar los costos de una cadena o minimizar el tiempo de entrega de algún producto. En el documento presentado por Mastrocinque, Yuce, Lambiase y Packianather (2013) se trata de minimizar el costo y el tiempo de entrega mediante el uso de un algoritmo basado en optimización de enjambre, más precisamente el algoritmo de abejas. Al final se realiza una comparativa entre éste y el de colonia de hormigas, concluyendo que el algoritmo de abejas alcanza de mejor manera la soluciones Pareto para el problema de CS. También en la publicación presentada por Yuce y Mastrocinque (2016), se describe un algoritmo híbrido que trabaja con las bondades del algoritmo de abejas, con la computación de Ángulo de bajada y con el algoritmo de escalando la cima para resolver una red de CS que trabajó con varios objetivos. En este artículo se resuelve un caso real para verificar el potencial que tiene este método para la resolución de problemas combinatorios grandes.

Mousavi, Alikar, Niaki y Bahreininejad (2015) en su publicación hablan sobre la modificación de un algoritmo de la mosca de la fruta para poder encontrar una solución óptima al problema de

ubicación de inventario. En este caso, la red de CS trabaja con el distribuidor y el comerciante. Esta red está diseñada para la ubicación del inventario de múltiples productos. Aquí se toma la distancia entre los comerciantes y distribuidores como un Euclidiano o cuadrado Euclidiano. Los productos son entregados a los distribuidores por medio de paquetes de tamaño conocido que serán guardados en los almacenes, en donde se tienen algunas restricciones de capacidad así mismo los niveles de planeación de inventario. Finalmente, lo que se desea encontrar es el óptimo número de paquetes comprados por los comerciantes a las distribuidoras usando el algoritmo modificado de la mosca de la fruta.

Elkhechafi et al presentan en su población la optimización de tamaño de lotes dentro de una cadena de suministros mediante el uso del algoritmo "firefly" (Elkhechafi et al., 2018). Este algoritmo firefly o luciérnaga en español está basado en el comportamiento de las luciérnagas. Estos insectos son atraídos por las luciérnagas más brillantes. El algoritmo trata de encontrar la luciérnaga más brillante, ya que por ende, esta es la solución óptima (Yang, 2014). En este trabajo se puede apreciar que el algoritmo no es cien por ciento preciso, ya que llega a alcanzar la mejor solución doce de las veinte ejecuciones.

Gracias a la revisión literaria realizada, se ha notado que la optimización de CS otorga beneficios para las empresas que optan por este proceso. En ellas, se ha visto un crecimiento de las economías y valor del producto; por ese motivo, este trabajo de titulación tiene como objetivo desarrollar un nuevo algoritmo híbrido que trate de utilizar los beneficios de diferentes algoritmos para optimizar una CS dentro del área de ensamblaje de una empresa.

Capítulo III – Marco Metodológico

3.1 Proyecto de Investigación

El trabajo de titulación propuesto es parte del proyecto de investigación "Modelo de optimización de costos en la CS en empresas de ensamblaje", ganador del XVII Concurso de Proyectos de Investigación de la Dirección de Investigación de la Universidad de Cuenca (DIUC). Dentro de este proyecto, se realizaron dos trabajos adicionales de titulación. El primero por parte de Orellana (2020) y el segundo de Berrezueta (2020); los dos estudiantes pertenecientes a la Carrera de Ingeniería Industrial de la Universidad de Cuenca. Estos autores fueron los encargados de acercarse a las diferentes empresas para obtener información sobre las variables necesarias para el proceso de optimización de la CS. Así mismo, se han encargado de compilar la información para obtener las constantes y variables de la cadena de suministro. También, dichos investigadores han conversado con diferentes actores de las empresas para obtener las restricciones y cuáles son los objetivos que se desean cumplir dentro de la misma. Los estudiantes han puesto estos objetivos como funciones objetivo que pueden ser maximizadas o minimizadas. Después, han seleccionado tres algoritmos de optimización para poder ejecutar en sus respectivos casos de estudio. Finalmente, los dos estudiantes entregaron toda la información encontrada para realizar este trabajo de titulación, en donde se trata de desarrollar un nuevo algoritmo, que realice este proceso de una manera más eficiente en base al tiempo de ejecución y velocidad de convergencia de la solución.

3.2 Enfoque, Alcance y Diseño

El enfoque de este trabajo de titulación es de tipo cuantitativo debido a que, la comparación entre los diferentes algoritmos de optimización está basada en las funciones objetivo (valor que pertenece al conjunto de números reales) y al tiempo (que es medido en relación milisegundos).

El alcance de la investigación es de tipo correlacional debido a que se cuenta con una variable dependiente. Esta variable dependiente es la función objetivo que se había definido en el marco teórico. Esta función puede cambiar su valor inicial (o de partida) dependiendo de qué variables independientes sean usadas.

Además, el diseño de este trabajo es de tipo no experimental longitudinal debido a que se recolectan datos a través del tiempo, específicamente de los años 2017, y 2018, con el fin de crear un modelo que devuelva los valores óptimos para cumplir las necesidades de la empresa.

3.3 Caso de Estudio

Este trabajo de titulación presenta tres casos de estudio que fueron analizados por Orellana (2020) y Berrezueta (2020) en sus respectivos trabajos. Por acuerdos de confidencialidad, no se mencionan los nombres de las empresas.

Para el desarrollo del trabajo de Orellana (2020) el caso de estudio seleccionado ha sido una empresa de ensamblaje de televisores, ubicada en la región Austral. La planta se encuentra ubicada en Cuenca, y se dedica al ensamblaje de motocicletas y televisores. Su producción es de aproximadamente 500 unidades al día de distintos modelos, entre ellos televisores inteligentes, 4K y curve. Orellana decidió trabajar en la producción de televisores, con el fin de reducir costos dentro de la CS de la empresa. Para la ejecución de los algoritmos Jimena utilizó un programa implementando en MATLAB. Sin embargo, tuvo que modificarlo para adaptarlo a su problema.

Berrezueta (2020), por su parte, trabajó como caso de estudio, una empresa de ensamblaje especializada en la fabricación de muebles. Esta empresa de ensamblaje se encuentra también ubicada en la ciudad de Cuenca. Los productos son fabricados en Ecuador, cuentan con la última

tecnología, y tratan de ayudar al medio ambiente con su forma de producción. Entre la variedad de productos que se ensamblan, resaltan: camas y muebles. Así mismo, Berrezueta realizó una minimización de costos en la producción de muebles dentro de la CS. Berrezueta se encargó de implementar todos los algoritmos de optimización en el lenguaje de programación R con la finalidad de ejecutarlos.

3.4 Hipótesis

- Se puede mejorar la eficiencia del algoritmo en base al menor tiempo de ejecución y mayor convergencia de la solución a la hora de maximizar el beneficio y el nivel de servicio ofrecido por una CS mediante un algoritmo metaheurístico híbrido desarrollado.

- Se puede mejorar la eficiencia del algoritmo en base al menor tiempo de ejecución y mayor convergencia de la solución a la hora de maximizar el beneficio obtenido por un producto en un periodo especifico de una CS mediante un algoritmo metaheurístico híbrido desarrollado.

- Se puede mejorar la eficiencia del algoritmo en base al menor tiempo de ejecución y mayor convergencia de la solución a la hora de minimizar el desperdicio generado por una planta mediante un algoritmo metaheurístico híbrido desarrollado.

3.5 Recolección de Datos

Para la recolección de datos se ha pedido a las empresas varios documentos para recopilar la información correspondiente a las variables necesarias.

Los documentos recibidos por Berrezueta (2020) fueron:

- Bill of Material (BOM) de los productos manufacturados en el 2018.
- Chatarra generada por el corte de tuberías del año 2018 (SCRAP).
- Costo de transporte hacia los Centros de Distribución e Hipermercados.
- Ingresos y egresos de tubería (materia prima) durante el año 2018.
- Portafolio de productos de la empresa en 2018.
- Precio de Venta al público de los productos en los Hipermercados.
- Producción del año 2018.
- Ventas del año 2018.

Los documentos recibidos por Orellana (2020) fueron:

- Ventas televisores periodo 2017.
- Estudio realizado por Guamán, Cárdenas, Siguenza-Guzman y Segarra (2020).
- Tiempos estándar de modelos periodo 2017.
- Documento de titulación de Guerrero (2018).

3.6 Ejecución de Algoritmos

Los algoritmos han sido implementados en lenguaje de programación C, que cuenta con características para optimización de procesos como el paralelismo de tareas. Con respecto al

tiempo de ejecución, se debe tener en cuenta las características de la máquina en donde se ejecuta el algoritmo. Para este caso, la máquina usada contó con las siguientes características:

- Memoria RAM: 12 GB
- Disco Duro: 1 TB
- Sistema Operativo: Windows 10 Home
- Procesador: Intel Core I7 de sexta generación (2.59 GHz)
- Tipo de Sistema: 64 bits

Para la implementación de los algoritmos en lenguaje C, se usó el programa Dev C++, el cual tiene un ambiente muy intuitivo para poder trabajar con lenguajes como C o C++. Este software es de libre distribución. En el caso de este trabajo de titulación se ha usado la versión 5.11.

3.7 Plan de Trabajo

El proceso para la investigación estuvo comprendido en cuatro etapas presentadas en la Figura 4. La primera etapa consistió en la revisión literaria para encontrar información sobre los algoritmos de optimización. Así también como la revisión de los algoritmos presentados e implementados en los trabajos de titulación de Orellana (2020) y Berrezueta (2020). En la segunda etapa se procedió al análisis y definición de las variables que fueron usadas para este problema de CS. Como tercera etapa se tuvo la construcción de un nuevo algoritmo metaheurístico híbrido que realice de una manera mucho más eficiente (en términos de tiempo y calidad de solución) la optimización de la CS. Estos algoritmos fueron seleccionados en base a sus ventajas como: el tiempo de ejecución del algoritmo y la velocidad de convergencia de la función objetivo para

alcanzar el valor mínimo. Finalmente, la última etapa fue la construcción de un software parametrizable para la ejecución del algoritmo metaheurístico híbrido desarrollado. En los siguientes párrafos se describe de manera más detallada las etapas de este proyecto.

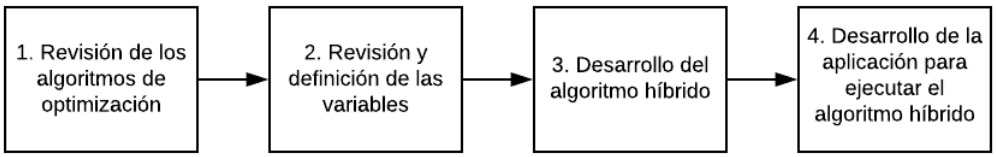

Figura 4. Etapas del Trabajo de Titulación.

3.7.1 Primera etapa.

En la primera etapa del trabajo de titulación se realizó una revisión sistemática, esto con el fin de encontrar información acerca de los diferentes algoritmos metaheurísticos que han sido utilizados para la optimización de las funciones objetivos de las CS. Para este proceso fue usada la metodología de Fink (2010). Esta metodología consiste en una serie de pasos: realizar las preguntas de investigación, seleccionar las bases de datos, seleccionar las palabras claves, establecer criterios de inclusión y exclusión de resultados y analizar los resultados obtenidos para escoger la información pertinente. Para este caso, se han formulado las siguientes preguntas: ¿Cuál es la historia, definición y clasificación de los algoritmos de optimización? ¿Cómo se puede optimizar una CS? ¿Cuáles han sido los algoritmos de optimización más usados en las CS? Luego se seleccionó la base de datos a consultar, en este caso Google Académico. Para el tercer paso se tuvieron que definir las palabras clave afines al tema que ayudaron en la búsqueda de nuestros resultados. En el caso de este proyecto se eligieron las siguientes frases: optimization, supply chain, algorithms, hybrid, classification, optimización, cadena de suministro, algoritmos e híbrido.

En el paso número 4, fue necesario definir los criterios de exclusión e inclusión de búsqueda. En la parte práctica se eligieron publicaciones que se encontraban escritas en el idioma inglés o español y que hayan sido publicados en los diez últimos años; mientras que, por el lado metodológico se encontraron documentos que contaban con definiciones, aplicaciones y pseudocódigos de calidad para esto se tuvo en cuenta el número de citaciones del documento. En la Tabla 1 se presentan los resultados obtenidos. Sin embargo, de todo este conjunto, solo se han escogido 50 documentos, los cuales han pasado por un filtro como el número de citaciones o el año en el que fueron publicados.

Tabla 1. Resultados obtenidos usando la metodología Fink (2010).

Palabras Claves	**Base de datos**	**Resultados**	**Año**	**Citaciones**
"supply chain" "optimization algorithms"	Google Académico	3.760	Desde 2015	Cualquiera
classification "optimization algorithms"	Google Académico	81.500	Cualquiera	Mínimo 200
"history" "optimization algorithms"	Google Académico	56.900	Cualquiera	Mínimo 200
"cadena de suministro" "algoritmo de optimización"	Google Académico	130	Desde 2000	Cualquiera
"cadena de suministro" "algoritmo hibrido" optimización	Google Académico	59	Desde 2000	Cualquiera
"supply chain" "optimization hybrid algorithm"	Google Académico	27	Desde 2000	Cualquiera
"cadena de suministro" and "definición"	Google Académico	8.490	Cualquiera	Mínimo 100

"historia" "algoritmos de optimización"	Google Académico	872	Cualquiera	Mínimo 50
"clasificación" "algoritmos de optimización"	Google Académico	1.760	Cualquiera	Mínimo 50
"optimización" "cadena de suministro"	Google Académico	6.570	Desde 2000	Cualquiera

Como se mencionó anteriormente, los algoritmos a ser implementados ya fueron previamente seleccionados por Orellana (2020) y Berrezueta (2020). Por parte de Orellana (2020), fueron seleccionados los algoritmos: memético para la optimización multi-objetivo (M-PAES, por sus siglas en inglés), optimización de enjambre de partículas multi-objetivo (MOPSO, por sus siglas en ingles) y Non-dominated sorting Genetic Algorithm 2 (NSGA-II). En el caso de Berrezueta (2020) se eligieron los algoritmos: NSGA-II, micro-algoritmos y un nuevo algoritmo desarrollado combinando NSGA-II con la teoría de k-means, el cual trata de dividir en grupos a los diferentes a la población total. Para la adquisición de datos, los dos autores fueron a cada una de las empresas (casos de estudio) en donde obtuvieron la información necesaria para probar los algoritmos. Estas variables son: la función objetivo (variable dependiente), las restricciones y las variables de entrada (variables independientes). El proyecto de investigación comprendió, además, un proceso de revisión literaria de distintas fuentes con el fin de encontrar los indicadores de rendimiento de las empresas con mayor frecuencia de aparición y que son más representativas dentro de las CS. Sin embargo, como ya se observó en la función objetivo, de este gran conjunto de variables, alrededor de 149, se han seleccionado un número menor, debido a que éstas han sido las que más

impacto tienen sobre la CS, específicamente estas variables están relacionadas con el área de costos. A continuación, se presenta una descripción de los algoritmos seleccionados.

3.7.1.1 Algoritmo M-PAES.

Es un algoritmo memético usado para la resolución de problemas multi-objetivo. Está basado en la estrategia de evolución de archivos de Pareto (PAES, por sus siglas en ingles de Pareto Archived Evolution Strategy). La población pasa cada vez por un proceso de cruce para recombinar los locales óptimos encontrados por el proceso de PAES. Aquí se tienen dos archivos, uno global y uno local (Knowles & Corne, 2000).

Pseudocódigo

1. Generar una población inicial *P*
2. Colocar cada miembro no-dominado de *P* en un archivo global *G*
3. Mientras no se cumpla el criterio de parada, hacer:
    a. Por cada solución c en *P*, hacer:
        i. Crear un archivo local *H*
        ii. Llenar el archivo *H* con soluciones de *G* que no dominen *c*
        iii. Copiar la solución *c* desde *P* a *H*
        iv. Realizar el procedimiento PAES de búsqueda local
        v. Reemplazar la solución mejorada *c* en *P*
    b. Crear una población intermedia y un valor n_i
    c. Mientras no se cumpla criterio de parada, hacer:
        i. Mientras no se cumpla criterio de parada, hacer:
            1. Escoger dos padres aleatoriamente y recombinar con *c*

2. Comparar *c* con *G*
3. Cambiar *G* con *c* si es necesario

ii. Si *c* es dominado por *G*, descartar *c* y usar torneo binario para encontrar un nuevo valor para *c*

iii. Colocar *c* en la población intermedia

d. Actualizar la población con la población intermedia

3.7.1.2 Algoritmo de enjambre de partículas multi-objetivo.

El algoritmo de enjambre de partículas multi-objetivo (MOPSO) trata de encontrar óptimos locales y uno global de un conjunto de partículas o individuos (C.A. Coello Coello & Lechuga, 2002). Estos individuos cambian en base a una tasa que será una velocidad; esta velocidad cambia según la Ecuación (2).

$$v_i = \omega v_i + c1 * r1(p_i - x_i) + c2 * r2(g_i - x_i), donde \qquad (2)$$

$w, c1\ y\ c2\ son\ constantes,$

$r1\ y\ r2\ son\ enteros\ aleatorios\ que\ pueden\ tomar\ el\ valor\ de\ 0\ o\ 1,$

$v_i\ es\ la\ velocidad\ de\ la\ particula\ i,$

$x_i\ es\ la\ particula\ i,$

$p_i\ es\ la\ mejor\ particula\ local\ i,$

$g_i\ es\ la\ mejor\ particula\ global\ i$

Los valores de *w, c1* y *c2* pueden ser tomados de diferentes formas. La primera es realizando un ejercicio de prueba y error hasta encontrar valores que se ajusten al problema. La otra forma es encontrar valores generalizados para cualquier tipo de problema. Para este trabajo se ha optado por usar la segunda opción. De esta manera, *w* toma el valor de uno, mientras que *c1* y *c2* toman el valor de dos (Kennedy & Eberhart, 1995). Por otra parte, se determinó que debe existir una velocidad máxima para evitar que las variables tomen valores fuera del rango.

Pseudocódigo

1. Por cada partícula hacer:
    a. Inicializar la posición de la partícula mediante un vector.
    b. Inicializar la mejor posición de la partícula conocida.
    c. Actualizar el valor de la mejor posición conocida.
    d. Inicializar la velocidad de la partícula.
2. Mientras no se cumpla el criterio de parada (número de iteraciones o variación entre la solución actual y la anterior) repetir:
    a. Por cada partícula hacer:
        i. Por cada dimensión hacer:
            1. Elegir números aleatorios.
            2. Modificar la velocidad de la partícula.
        ii. Modificar la posición de la partícula.
        iii. Si la función objetivo en la posición es menor a la función objetivo en la mejor posición local entonces:
            1. Modificar la mejor posición de la partícula.

a. Si la función objetivo en la mejor posición local es menor a la función objetivo en la posición global entonces:
b. Modificar la mejor posición global.

3. Retornar la mejor posición global.

3.7.1.3 Micro-algoritmo NSGA-II (uNSGA-II).

El algoritmo NSGA-II cuenta con dos funciones y tres operaciones importantes. Las funciones son el ordenamiento no dominado y la distancia entre la población. Las operaciones importantes son: selección, cruce y mutación (Deb et al., 2002). Para la etapa de selección se ha usado la función de ordenamiento no dominado y la distancia entre la población. El cruce está basado en probabilidad; si la probabilidad es menor a 0.5 entonces se toma el gen del primer padre, si la probabilidad esta entre 0.5 y (1/Número de Individuos) se toma el gen del segundo padre, o caso contrario, el gen realiza el proceso de mutación y toma un valor aleatorio del espacio del problema. Hay que tener en cuenta que para que un algoritmo sea micro, lo único que se debe hacer es reducir su población inicial; esto con el fin de reducir tiempo de ejecución y reducir memoria utilizada por el computador (Coello Coello & Toscano Pulido, 2001).

Pseudocódigo

1. Inicializar Población (Máximo 10 Individuos)
2. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
3. Realizar el Ordenamiento no dominado
    a. Buscar las distancias entre la población

b. Seleccionar los mejores individuos que serán los nuevos padres

4. Mientras no se cumpla el criterio de parada, hacer:
    a. Generar los hijos de la población
        i. Recombinación y Mutación
    b. Evaluar las Funciones Objetivo
        i. Dar un peso a cada una de las funciones
    c. Realizar el Ordenamiento no dominado
        i. Buscar las distancias entre la población
        ii. Seleccionar los mejores individuos que serán los nuevos padres

3.7.1.4 Algoritmo NSGA-II k-means.

Este algoritmo fue desarrollado por Berrezueta (2020). Básicamente es una combinación entre el algoritmo NSGA-II y la teoría de la agrupación por k-means (k grupos o clusters). Lo que el algoritmo trata de hacer es probar los individuos con NSGA-II en un número fijo de iteraciones y luego de eso, empezar a agrupar los individuos mediante el proceso de k grupos o k-means. Por cada uno de estos grupos volver a ejecutar NSGA-II; sin embargo, en este caso va a ser mucho más rápido debido a que cada grupo tendrá pocos individuos, y además determinarán soluciones en diferentes posiciones del espacio de búsqueda.

Pseudocódigo

1. Inicializar Población
2. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
3. Realizar el Ordenamiento no dominado

a. Buscar las distancias entre la población

b. Seleccionar los mejores individuos que serán los nuevos padres

4. Realizar un número específico de operaciones:

    a. Generar los hijos de la población

        i. Recombinación y Mutación

    b. Evaluar las Funciones Objetivo

        i. Dar un peso a cada una de las funciones

    c. Realizar el Ordenamiento no dominado

        i. Buscar las distancias entre la población

        ii. Seleccionar los mejores individuos que serán los nuevos padres

5. Crear 10 grupos mediante el proceso k-means y colocar individuos

6. Mientras no se cumpla el criterio de parada:

    a. Por cada grupo, hacer:

        i. Generar los hijos de la población

            1. Recombinación y Mutación

        ii. Evaluar las Funciones Objetivo

            1. Dar un peso a cada una de las funciones

        iii. Realizar el Ordenamiento no dominado

            1. Buscar las distancias entre la población

            2. Seleccionar los mejores individuos que serán los nuevos padres

3.7.2 Segunda etapa.

Esta etapa consistió en analizar los datos y definir las variables. Gracias a los datos compilados y analizados por Berrezueta (2020) y Orellana (2020), se ha llegado a obtener funciones objetivo y restricciones para los diferentes casos de estudio. En la siguiente sección se presentan estos datos.

3.7.2.1 Definición de Variables.

Debido a que los algoritmos de optimización son usados para poder encontrar las mejores soluciones de la CS, se tiene que modelar una función objetivo en base a distintos indicadores de la cadena. En el caso de esta investigación se cuenta con tres casos de estudio y tres funciones objetivo.

Para el primer caso de estudio se tiene una optimización multi-objetivo en donde se tienen dos funciones objetivo que serán maximizadas al mismo tiempo. Para el segundo caso se tiene la maximización de beneficio generado por un producto dentro de una CS. Finalmente, el último caso presenta la minimización de desperdicio generado por materia prima generado por las plantas.

Primer Caso de Estudio

Por parte de Orellana (2020) se han desarrollado dos funciones objetivo presentadas en la Ecuación (3) y Ecuación (4). De esta manera, la función objetivo se compone del valor de la suma de los productos distribuidos por su precio menos la suma de todos los costos de materia prima, costos de ensamblaje de los productos, costos de mantenimiento del inventario de los productos y costos de transporte de los productos desde las plantas hasta los clientes; la cual debe ser maximizada. También se debe maximizar el nivel de servicio ofrecido a los clientes.

Objetivo 1:

Maximizar Beneficio

$$= \sum_z \sum_j \sum_k \sum_b (PD_{zjkb} * PV_{kb}) - \sum_z \sum_j \sum_b P_{zjb} * (CM_{zjib} + CE_{jzb})$$

$$- \sum_z \sum_j \sum_k \sum_b (PD_{zjkb} * CT_{zjkb}) - \sum_z \sum_j \sum_b (I_{zjb} * CI_{zjb}) \qquad (3)$$

Objetivo 2:

$$\textbf{\textit{Maximizar Nivel de Servicio}} == \frac{\sum_z \sum_j \sum_k \sum_b PD_{zjkb}}{\sum_z \sum_k \sum_b D_{zkb}} \qquad (4)$$

Donde,

z *es el periodo*,

b *es el índice de productos*,

j *es el índice de plantas de ensamblaje*,

k *es el índice de mercados*,

CM_{zjb} es el costo unitario de materia prima abastecida para ensamblar b

desde el proveedor i hasta la planta j,

P_{zjb} es la cantidad total de producto b ensamblado en la planta j, en el periodo z,

CE_{zjb} es el costo de ensamblaje (conversión)del producto b en la planta j, en el periodo z,

PD_{zjkb} es la cantidad de producto b despachado desde la planta j hasta el mercado k, en el periodo z,

I_{zjb} es el inventario final de producto b en la planta j, en el periodo z,

CI_{zjb} es el costo de unitario de mantenimiento inventario de producto b en la planta j, en el periodo z,

PV_{kb} es el precio de venta del producto b en el mercado k,

CT_{zjkb} es el costo de unitario de transporte de producto b dsde la planta j hasta el mercado k,

en el periodo z,

D_{zkb} es la demanda del producto b del mercado k, en el periodo z,

I_{b0} es el nivel de inventario inicial del producto b,

pt_{jb} es el tiempo requerido para producir el producto b en la planta j,

tt_{zj} es el tiempo disponible para producir en la planta j, en el periodo z,

NS es el nivel de servicio al cliente,

SS_{zjb} es la stock de seguridad del producto b en la planta j, en el periodo z.

Restricciones de la Función.

Es necesario también limitar el modelo para conseguir la convergencia del problema sobre valores viables. Por ese motivo se han propuesto las siguientes restricciones (5-10).

1. Condición 5: La producción va a ser igual o mayor a los productos distribuidos.

$$\sum_{j} P_{zjb} \geq PD_{zjkb} \; ; \; \forall \, z, k, b \qquad (5)$$

2. Condición 6: El tiempo de producción no puede exceder el tiempo disponible de la planta.

$$\sum_{b} pt_{jb} * P_{zjb} \leq tt_{zj} \quad ; \ \forall j, z \qquad (6)$$

3. Condición 7: Limitar la cantidad de productos enviados en el periodo, para que no sea mayor a la demanda.

$$\sum_{j} PD_{zjkb} \leq D_{zkb} \ ; \ \forall z, k, b \qquad (7)$$

4. Condición 8: Balance entre el inventario final de la planta.

$$I_{zjb} = I_{(z-1)jb} + P_{zjb} - \sum_{k} PD_{zjkb} \ ; \ \forall j, b, z \qquad (8)$$

5. Condición 9: El inventario mínimo que debe tener cada planta al final de cada periodo.

$$I_{zjb} \geq SS_{bjz} \qquad (9)$$

6. Condición 10: No negatividad de las variables.

$$P_{zjb}, PD_{zjkb}, I_{zjb} \geq 0 \qquad (10)$$

Segundo Caso de Estudio

Berrezueta (2020) cuenta con dos casos de estudio, en el primero propuso la Ecuación (11), la cual es la función objetivo. Esta función objetivo comprende la suma del beneficio bruto ganado

menos el costo total de producción, menos el costo total de inventario y menos el costo total de distribución, la misma que debe ser maximizada.

Maximizar el Beneficio

$$= \sum_{k,b} PVP_b * PV_{kb} - \sum_{b} PR_b * CP_b \; - \sum_{b} CP_b * Inv_b * T_b * I_a$$

$$- \sum_{k,b} CT_b * PD_{kb} \qquad (11)$$

Donde,

i *es el índice de materias primas*,

b *es el índice de productos*,

k *es el índice de clientes*,

$\boldsymbol{PVP_b}$ es el precio de venta del producto b en el mercado k,

$\boldsymbol{PV_{kb}}$ es la cantidad de producto b vendido en el mercado k,

$\boldsymbol{PR_b}$ es la cantidad de producto b,

$\boldsymbol{CP_b}$ es el costo unitario de producción del producto b,

$\boldsymbol{Inv_b}$ es el inventario del producto b,

$\boldsymbol{T_b}$ es el tiempo promedio en inventario del producto b,

$\boldsymbol{I_a}$ es el radio definido de interes,

$\boldsymbol{PD_{kb}}$ es la cantidad de producto b distribuido a k,

$\boldsymbol{CT_b}$ es el costo de transporte del producto b,

$\boldsymbol{D_{kb}}$ es la demanda del producto b en el mercado k.

Restricciones

1. Condición 12: Los productos distribuidos deben ser mayor o igual a la demanda.

$$\sum_{k,b} PD_{kb} \geq \sum_{k,b} D_{kb} \qquad (12)$$

2. Condición 13: Restricción para controlar el modelo y evitar que se distribuya más producto del que existe disponible.

$$PR_b + Inv_{\ b} \geq PD_b \quad (13)$$

Tercer Caso de Estudio

Como tercer caso de estudio, Berrezueta (2020) propone como función objetivo la Ecuación (14). Esta función busca minimizar el desperdicio generado por una planta después de haber cortado materia prima.

$$\boldsymbol{Minimizar\ el\ desperdicio\ producido} = r = min\{cx : x \in Z_+^{\eta}\} \qquad (14)$$

Donde,

$\boldsymbol{r}$ es el desperdicio generado,

$\boldsymbol{c}$ es el patrón cortado,

$\boldsymbol{x}$ es la frecuencai de corte,

$\boldsymbol{i}$ es la materia prima,

$\boldsymbol{j}$ es el patrón,

$\boldsymbol{a_{ij}}$ son los cortes producidos en la materia prima i usando el patrón j,

$\boldsymbol{x_j}$ es la frecuencia de cortes aplicados por el patrón j,

$\boldsymbol{e_j}$ es el inventario disponible,

$\boldsymbol{d_i}$ es la demanda de cortes de la materia prima i.

Restricciones.

1. Condición 15: Los patrones de corte deben satisfacer la demanda.

$$\sum_{j=1}^{\eta} a_{ij} x_j \geq d_i \quad (15)$$

2. Condición 16: No utilizar más materia prima del que está en el stock.

$$\sum_{j=1}^{\eta} a_{ij} x_j \leq e_j \qquad (16)$$

3.7.3 Tercera etapa.

La siguiente etapa consistió en proponer un nuevo algoritmo. Este algoritmo debe tomar las bondades de otros algoritmos con el fin de realizar el proceso de optimización reduciendo el tiempo de ejecución que toma en llegar a un resultado factible. Así también, mejorar la calidad de la solución. Para comprobar que este algoritmo propuesto realiza de manera más eficiente el proceso de optimización se desarrolló con algoritmos base.

3.7.3.1 Algoritmo Propuesto.

Se ha propuesto un nuevo algoritmo híbrido que cuente con las mejores características de cada uno de los algoritmos base. Este algoritmo tiene como finalidad optimizar de una manera más eficiente el problema de CS en términos de tiempo que le toma ejecutar al algoritmo y la calidad de solución.

Este algoritmo usa como base el algoritmo NSGA-II (debido a que este algoritmo tiene una probabilidad mayor de converger al óptimo global del problema), pero tiene cambios al momento de la mutación que están relacionados con el algoritmo de enjambre de partículas. Así como los algoritmos genéticos, el cruce es basado en probabilidad generada aleatoriamente en cada iteración. Si la probabilidad es menor a 0.5, entonces se toma el gen del primer padre; si la probabilidad está entre 0.5 y (1/Número de Individuos), se toma el gen del segundo padre; o, caso contrario, el gen realiza el proceso de mutación. Este proceso se basa en la creación de sub-espacios usando una variable *xz* que cambia durante la ejecución del algoritmo. A continuación, se presenta esta estrategia.

Creación de sub-espacios.

Los algoritmos genéticos, a la hora de mutar los genes, usualmente toman un valor aleatorio entre todo el espacio del problema; sin embargo, para el algoritmo híbrido se ha propuesto la idea de crear solo un sub-espacio en donde el gen pueda tomar menos valores que estén cercanos a su valor actual.

Aun así, también se deben analizar los puntos débiles de esta teoría. Si se crea un sub-espacio, entonces los genes quizás se queden limitados a tomar valores de dicha zona específica, los cuales no pueden ser los mejores, y por ende no se pueda encontrar las soluciones óptimas o el óptimo

global. Esta situación ha sido resuelta, ya que, al inicio de la ejecución, el sub-espacio del problema va a ser igual al espacio; y, conforme se avanza en cada iteración, este sub-espacio cada vez se va reduciendo. Entonces, al inicio el algoritmo tratará de encontrar las mejores zonas donde posicionarse, y al final sus valores cambiarán ligeramente debido a la limitación del sub-espacio. El sub-espacio es generado usando el valor *xz*, el cual será sumado al valor del cromosoma del individuo, con el fin de mutar en un nuevo valor. En la Figura 5 se puede observar el comportamiento de este proceso.

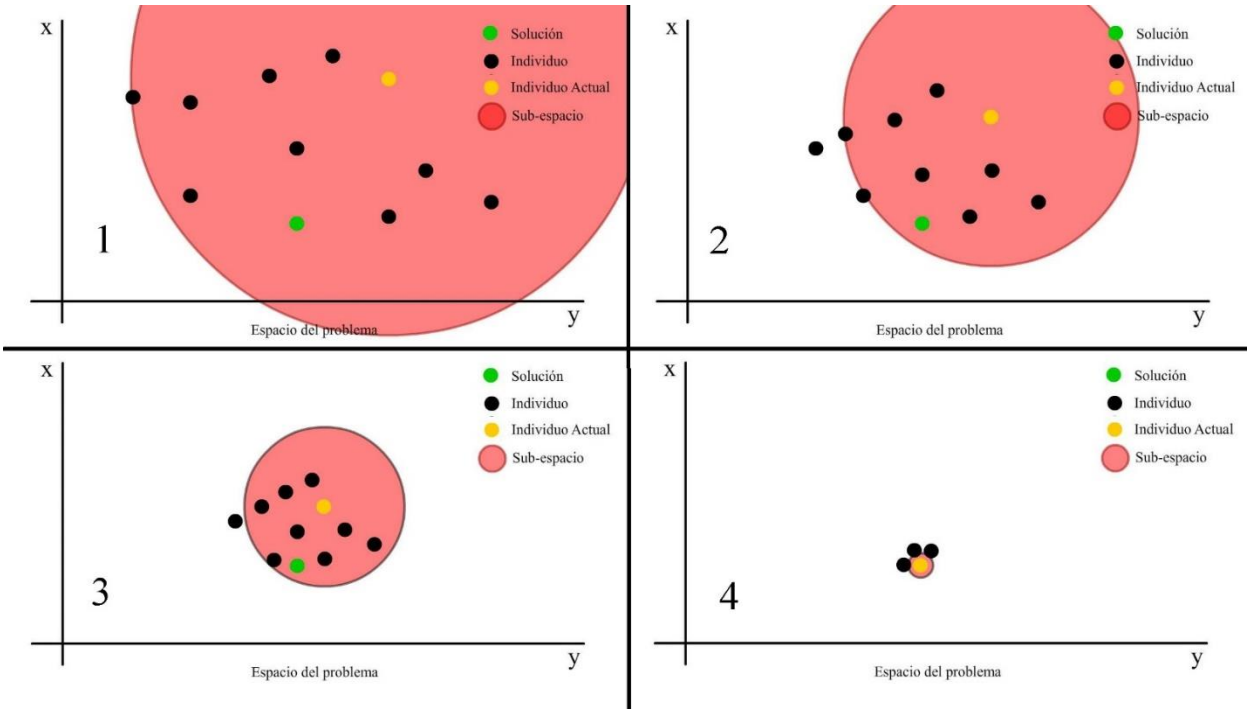

Figura 5. Creación de los sub-espacios del problema.

Regresando a las características del algoritmo se tiene una parte relacionada con los micro-algoritmos. Al inicio, el algoritmo contará con una gran población para tener más opciones de

donde escoger los mejores Individuos. Sin embargo, en las siguientes iteraciones esta población se reducirá, con el fin de disminuir el tiempo de ejecución y el uso de memoria.

A continuación, se presenta el pseudocódigo del algoritmo híbrido que presenta las bondades de los algoritmos previamente revisados. También, en la Figura 6 se presenta el diagrama de flujo del mismo que indica las partes del algoritmo que han sido basadas en otros algoritmos de optimización.

1. Inicializar Población (Pocos Individuos)
2. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
3. Realizar el Ordenamiento no dominado
    a. Buscar las distancias entre la población
    b. Seleccionar los mejores individuos que serán los nuevos padres
4. Mientras no se cumpla el criterio de parada, hacer:
    a. Generar los hijos de la población
        i. Cruzar a los individuos
        ii. Mutar: Seleccionar un valor del nuevo sub-espacio generado por el valor *xz*
    b. Evaluar las Funciones Objetivo de los nuevos hijos
        i. Dar un peso a cada una de las funciones
    c. Realizar el Ordenamiento no dominado
        i. Buscar las distancias entre la población
        ii. Seleccionar los mejores individuos que serán los nuevos padres
    d. Si se llega a un número indicado de iteraciones

i. Reducir el valor de *xz*

ii. Aumentar la población

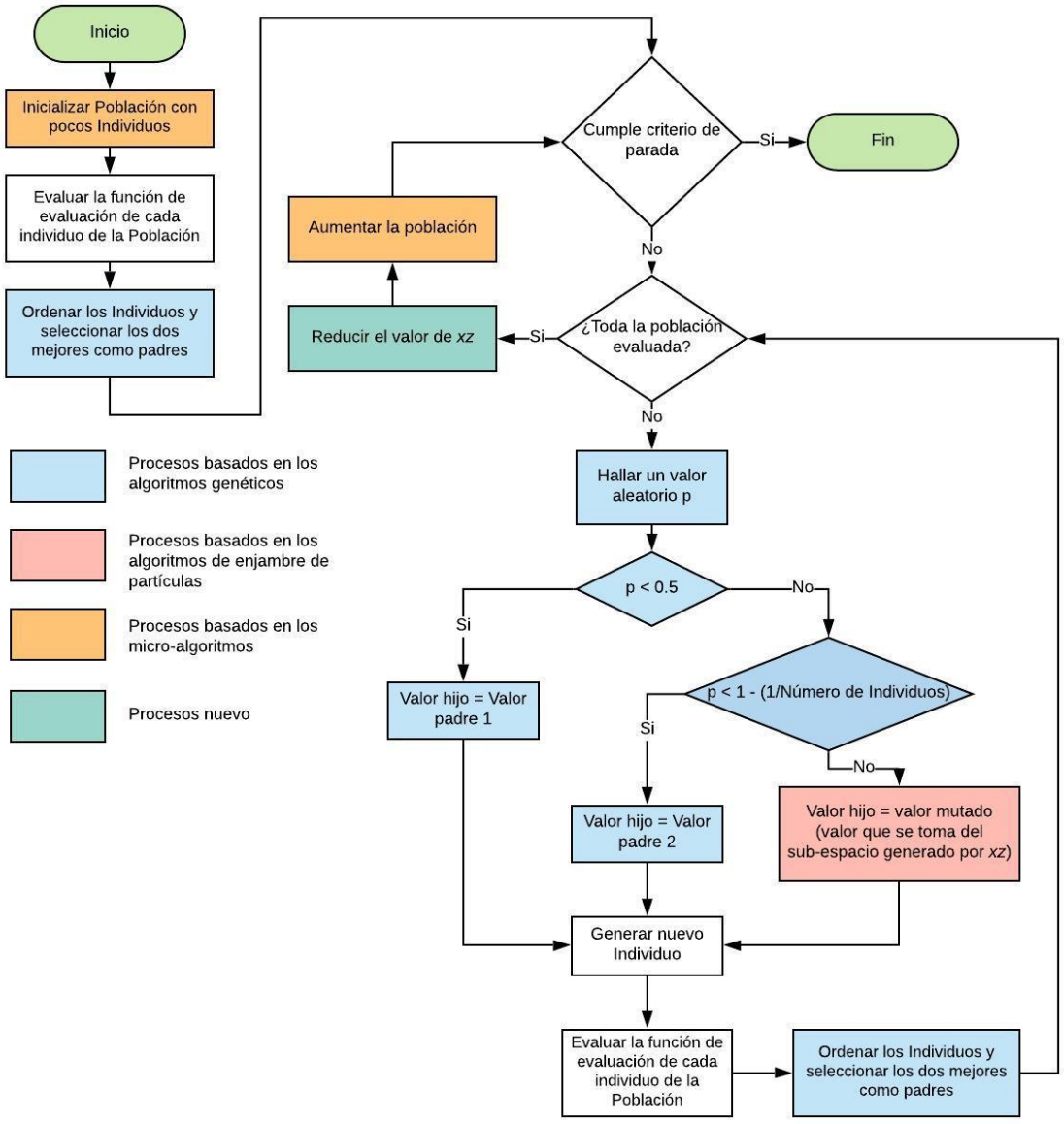

Figura 6. Diagrama de flujo del algoritmo de optimización híbrido.

Una vez entendidos los algoritmos seleccionados se procedió a realizar la implementación de los mismos. Con el fin de probar los algoritmos en las mismas condiciones, estos fueron

implementados en el mismo lenguaje de programación. El lenguaje elegido para este proceso fue el lenguaje C debido a que es de muy bajo nivel y tiene la característica de optimizar los procesos en términos de paralelismo.

Al finalizar la ejecución de estos algoritmos, se procedió a observar las bondades de los mismos y también de otros, así como las desventajas que presentan, esto tanto en complejidad temporal, espacial o computacional.

3.7.3.2 Comparación de los algoritmos.

Para realizar la verificación de la mejora del nuevo algoritmo híbrido, se hizo una comparativa con los otros algoritmos mencionados en la Etapa 1. Se tomó en cuenta dos indicadores que ayudaron a la comparación de los algoritmos: convergencia y tiempo de ejecución.

Convergencia.

La convergencia del algoritmo es la tasa de cambio del valor inicial al valor final. En la mayoría de ocasiones, un algoritmo puede converger después de transcurridas un gran número de iteraciones, y en otras divergir y nunca llegar a la solución final, que es el óptimo.

Para este trabajo, se ha realizado una serie de ejecuciones del algoritmo con un gran número de iteraciones. Y, se ha analizado cómo el algoritmo se comporta en términos de convergencia en cada número *n* de iteraciones. De esta manera, algunos algoritmos pueden encontrar una solución cercana a la óptima en pocas iteraciones, mientras que otros pueden tomar mucho más, por ende, este indicador es de mucha relevancia para la investigación.

Tiempo de Ejecución.

Como se mencionó en el punto anterior, los algoritmos pueden llegar a converger a un cierto número de iteraciones o de tiempo; sin embargo, también pueden llegar a un estado cíclico donde no encuentren una solución. Por eso, es de suma importancia contar con criterios de parada como el número de iteraciones. Basados en esto, para comprobar el tiempo de ejecución de los algoritmos se ha hecho el proceso de optimización con un mismo número de iteraciones para los cinco algoritmos, y se ha usado una marca de tiempo para determinar el tiempo de ejecución del proceso.

Se ha optado por realizar una serie de pruebas con los algoritmos en la misma configuración, de las cuales se ha determinado el promedio que será la medida final tomada para el análisis. Esto se realizó debido a que una computadora no siempre tarda el mismo tiempo en ejecutar una misma sentencia, ya que además realiza otros procesos propios del funcionamiento de la misma.

Se debe tener en cuenta también que, los algoritmos de optimización deberán pasar por un proceso de calibración y validación del modelo. El proceso de calibración o también conocido como proceso de entrenamiento se encarga de encontrar los mejores valores que se adapten al modelo para obtener los mejores resultados. La calibración se realiza con un conjunto de datos llamados datos de entrenamiento. Con este modelo, es mucho más fácil predecir cuando los datos de entrada cambien. Debido a la restricción de entrega de datos por parte de las empresas, solo se pudo generar el modelo, pero no realizar una fase de prueba para validar los mismos.

3.7.4 Cuarta etapa

Finalmente, como cuarta etapa se tuvo el desarrollo del software para la ejecución del algoritmo. Este software es una aplicación web parametrizable. Este aplicativo fue realizado en Java y JSP. En este caso se tuvo que re-implementar los algoritmos en lenguaje JAVA, para poder adaptarse al servidor web en donde se alojaría la aplicación. El servidor web utilizado fue Apache GlassFish.

Este aplicativo se encuentra enfocado para la ejecución de los algoritmos de optimización para el caso de maximización de beneficio y nivel de servicio de una CS. En los Apéndices A y B, se encuentran tanto el Manual de Usuario como el Manual Técnico de la aplicación.

Capítulo IV - Resultados

La finalidad del proyecto de titulación es encontrar un algoritmo de optimización que, frente a otros, presente una mejora en términos de tiempo de ejecución y convergencia del resultado. Estos algoritmos fueron ejecutados con datos de CS de los casos de estudio mencionados en el capítulo anterior. La función objetivo, las restricciones de los casos de estudio y el lenguaje de programación usados para la ejecución de los algoritmos son los mismos.

4.1 Ejecución de la Función Objetivo del Caso de Estudio 1

Para la representación de los resultados se realizaron varios diagramas de dispersión que tienen como ejes: el número de iteraciones y el valor de función objetivo. Para estos resultados, los algoritmos fueron ejecutados varias veces y luego se obtuvo el promedio de los mismos.

4.1.1 Algoritmo M-PAES.

El tiempo de ejecución que le tomó al algoritmo llegar a un nivel de servicio de 99% y un beneficio de $4.314.239,07 fue de 17 minutos. Este algoritmo es el que más tiempo toma en ejecutarse, aparte que trabaja con bastante memoria debido a que debe almacenar varios individuos. En la Figura 7 se presenta la convergencia del algoritmo M-PAES al beneficio y un nivel de servicio mencionados; así también, el número de iteraciones que le toma en llegar a estos valores.

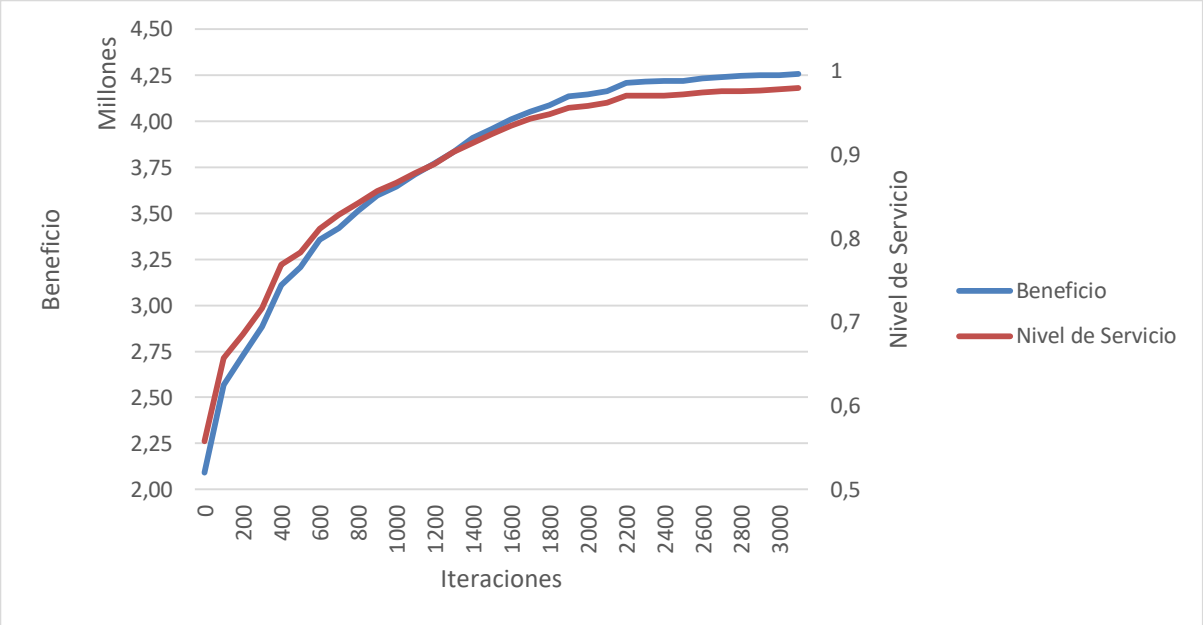

Figura 7. Convergencia y tendencia del algoritmo M-PAES para obtener el mayor beneficio y ofrecer el mejor servicio al cliente.

4.1.2 Algoritmo MOPSO.

El tiempo de ejecución de MOPSO fue de 51,32 segundos. El algoritmo llego a converger a un nivel de servicio del 98,6%; y su beneficio fue de $4.263.268,32. Este algoritmo realiza muy rápidamente su trabajo, sin embargo, no llega a converger a las mejores soluciones. En la Figura 8 se presentan los valores a los que converge el beneficio y nivel de servicio de la CS.

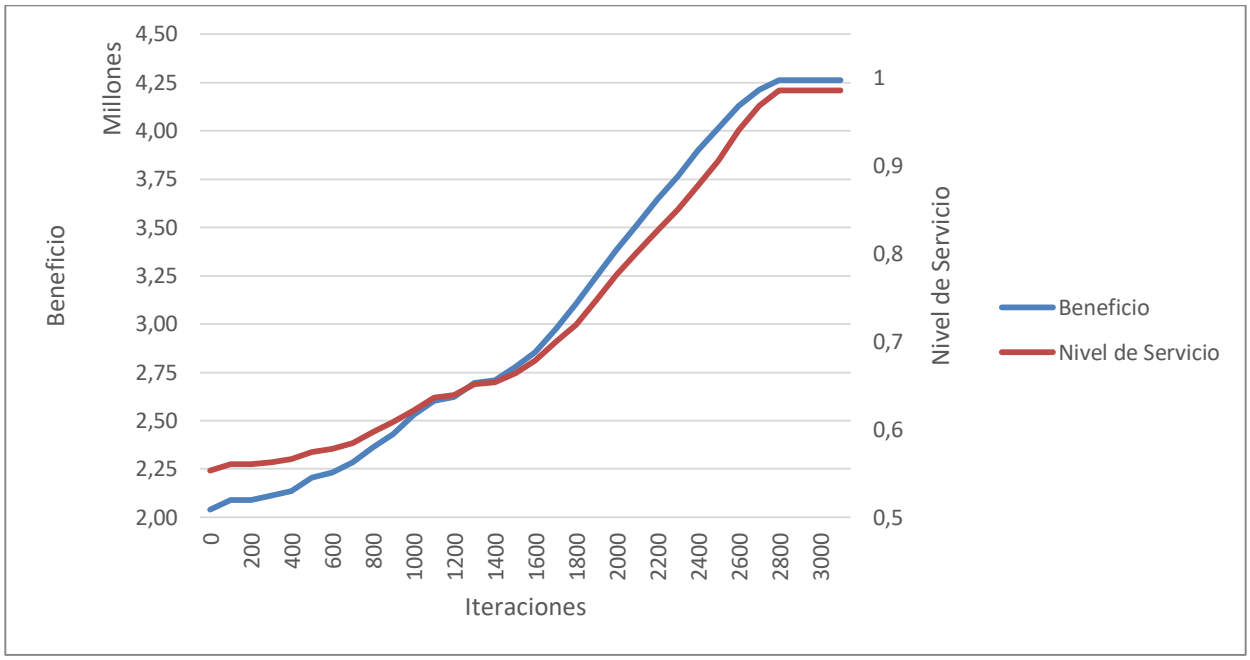

Figura 8. Convergencia y tendencia del algoritmo MOPSO para obtener el mayor beneficio y ofrecer el mejor servicio al cliente.

4.1.3 Algoritmo Híbrido.

Este algoritmo llegó a converger a un nivel de servicio del 99.9% y con un beneficio de $4.360.030,69. El tiempo de ejecución del algoritmo fue de 8,08 segundos. En la Figura 9 se presenta la convergencia del nivel de servicio y el beneficio obtenido con el algoritmo híbrido.

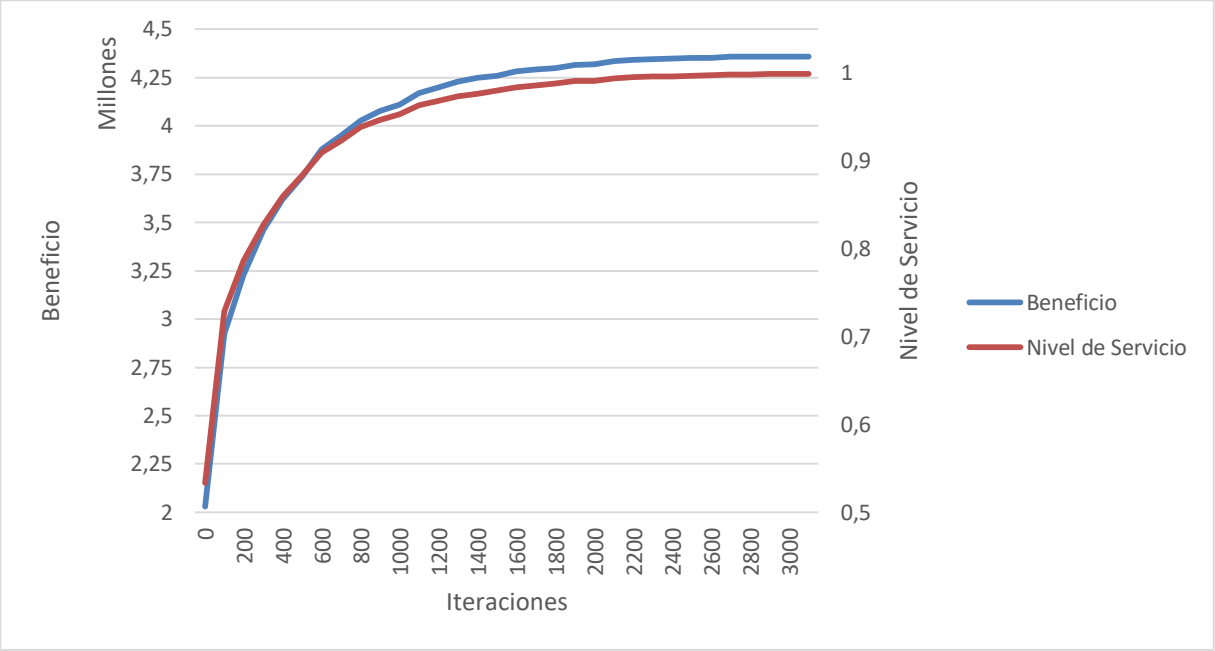

Figura 9. Convergencia y tendencia del algoritmo híbrido para obtener el mayor beneficio y ofrecer el mejor servicio al cliente.

4.2 Ejecución de la Función Objetivo del Caso de Estudio 2

Este caso de estudio es muy similar al anterior, solo que la forma de optimizar es diferente. En el anterior se realizaba un proceso completo de toda la CS. En este caso se realiza individualmente cada producto por mes. En las Tablas 2-13 se presentan los valores mensuales obtenidos por los algoritmos y cuáles eran los valores esperados que se encontraron en Berrezueta (2020).

Tabla 2. Ejecución de los algoritmos en el mes de enero para la función de distribución.

		Híbrido		**NSGA-II k-means**		**u NSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
2	1.258,25	1.258,250625	2	1.258,25063	1102	1.084,2145	10.000
7	1027,91	1027,910125	223	1027,91013	1209	801,910687	10.000
9	156,73	156,726962	1	148,648825	10.000	148,648825	10.000
12	801,87	801,868767	94	801,024525	10.000	676,726042	10.000

18	775,21	757,457037	10.000	775,207037	8295	533,48735	10.000
21	593,47	593,474608	209	593,474608	629	509,769667	10.000

Tabla 3. Ejecución de los algoritmos en el mes de febrero para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	70,4	70,396058	1	70,396058	0	70,396058	0
2	428,58	428,576333	2	418,593271	10.000	412,013271	10.000
7	12,58	12,577062	1	0	10.000	-4.214.979.468	10.000
8	1.681,5	1.681,496437	220	1.681,49644	978	1.320,278	10.000
9	519,19	519,192062	1	519,192062	310	480,795788	10.000
10	2,66	2,655796	1	0	10.000	2,655796	0
12	269,56	266,813933	10.000	266,813933	10.000	250,999692	10.000
18	913,43	913,430237	425	913,430237	1754	582,169787	10.000
21	609,7	609,697021	249	609,697021	5940	463,119633	10.000

Tabla 4. Ejecución de los algoritmos en el mes de marzo para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	525,63	525,628613	38	525,628613	829	332,88635	10.000
2	1.189,92	1.189,921438	53	1.189,92144	55	1.069,857354	10.000
7	1.184,32	1.184,316	249	1.184,316	382	995,1995	10.000
8	1.217,46	1.217,458562	185	1.217,45856	27	939,292937	10.000
9	840,5	840,501575	2	840,501575	153	725,210888	10.000
10	0	0	0	0	0	0	0
11	546,56	546,559792	0	546,559792	0	491,903812	10.000
12	1.192,4	1.192,399667	24	1.192,39967	717	1.010,499975	10.000
18	1.018,34	1.018,335163	425	1.012,44933	10.000	791,114862	10.000

21	2.972,21	2.972,211063	434	2.972,21106	3099	2.565,101312	10.000
29	160,4	160,399492	185	160,399492	4883	121,731917	10.000

Tabla 5. Ejecución de los algoritmos en el mes de abril para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	544,7	544,696496	205	544,696496	0	458,522408	10.000
2	1.245,6	1.245,599604	217	1.245,5996	0	969,896333	10.000
7	2.171,03	2.171,034312	217	2.171,03431	732	1.884,963687	10.000
8	3.905,37	3.905,365687	249	3.871,97513	10.000	3.231,784437	10.000
9	655,16	655,157163	249	655,157163	75	519,164613	10.000
10	0	0	0	0	0	0	0
11	553,16	553,159688	90	522,653708	10.000	500,267729	10.000
12	1.226,76	1.226,763008	249	1.226,76301	57	787,672108	10.000
18	2.519,69	2.519,693813	1.266	2.507,92214	10.000	1.889,486412	10.000
21	413,25	413,24975	221	413,24975	27	359,462279	10.000
29	729,69	729,689467	438	729,689467	2325	544,915887	10.000

Tabla 6. Ejecución de los algoritmos en el mes de mayo para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	1.510,96	1.510,964817	13	1.510,96482	1.894	1.412,274671	10.000
2	1.034,84	1.034,842458	127	1.034,84246	3.954	926,898375	10.000
7	1.521,68	1.518,075438	10.000	1.495,68544	10.000	1.115,296562	10.000
8	2.356,92	2.356,922625	40	2.356,92263	90	2.089,758125	10.000
9	474,09	474,093925	1	409,83765	10.000	409,85765	10.000

10	0	0	0	0	0	0	0
11	1.888,46	1.888,457104	258	1.888,4571	441	1.204,911437	10.000
12	1.735,34	1.735,342083	259	1.735,34208	460	1.220,345733	10.000
18	4.377,1	4.377,102675	1.687	4.366,50516	10.000	2.913,316862	10.000
21	1.023,58	1.023,584375	234	1.023,58438	1.947	800,119533	10.000

Tabla 7. Ejecución de los algoritmos en el mes de junio para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	393,13	393,126204	0	393,126204	0	357,928175	10.000
2	2.641,51	2.641,513896	174	2.641,5139	285	2.166,565729	10.000
7	1.799,48	1.799,483125	64	1.799,48313	2	1.429,816625	10.000
8	2.398,16	2.398,160375	64	2.398,16038	405	2.068,204187	10.000
9	1.071,12	1.071,1204	64	1.071,1204	541	751,199025	10.000
10	0	0	0	0	0	0	0
11	551,58	551,577729	1	521,07175	10.000	432,399792	10.000
12	875,26	875,263317	107	875,263317	134	609,473625	10.000
18	1.414,33	1.406,412925	10.000	1.405,21293	10.000	1.037,3276	10.000
21	882,92	882,919333	400	882,919333	8.247	781,474425	10.000

Tabla 8. Ejecución de los algoritmos en el mes de julio para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	863,45	863,446642	108	863,446642	304	658,788467	10.000
2	2.286,22	2.286,21675	152	2.286,21675	190	1.910,448583	10.000

7	657,33	657,325375	3	657,325375	3.429	554,707125	10.000
8	1.536,02	1.536,023625	128	1.536,02363	2.364	1.178,057437	10.000
9	923,78	923,7804	240	923,7804	595	594,749025	10.000
10	0	0	0	0	0	0	0
11	1.012,34	1.012,341542	122	1.012,34154	145	1.012,341542	1
12	2.176,75	2.176,746017	122	2.176,74602	423	1.827,0136	10.000
18	3.834,62	3.834,62005	599	3.834,62005	7.681	2.997,609	10.000
21	815,25	815,246921	83	815,246921	969	587,384608	10.000

Tabla 9. Ejecución de los algoritmos en el mes de agosto para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	1.108,69	1.108,688758	302	997,424671	10.000	934,686642	10.000
2	2.601,99	2.601,985938	83	2.601,98594	82	1.947,461646	10.000
7	2.389,69	2.389,692563	150	2.300,57019	10000	2.030,371938	10.000
8	4.331,35	4.331,351312	298	4.331,35131	1.395	3.495,148375	10.000
9	284,98	284,97765	116	284,97765	206	266,041375	10.000
10	0	0	0	0	0	0	0
11	531,07	531,069792	2	531,069792	0	422,247833	10.000
12	1.352,37	1.352,3727	110	1.352,3727	1.115	902,387558	10.000
18	3.089,69	3.089,68625	659	3.083,80041	10.000	2.416,223625	10.000
21	3.773,19	3.773,18785	425	3.773,18785	7.771	2.837,640979	10.000
29	197,56	197,559792	371	197,559792	6.075	140,717067	10.000
32	6,94	6,944262	0	6,944262	0	6,944262	1

Tabla 10. Ejecución de los algoritmos en el mes de septiembre para la función de distribución.

		Híbrido	**NSGA-II k-means**	**uNSGA-II**

Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	73,21	73,212117	1	73,212117	0	66,064087	10.000
2	682,38	682,378375	116	602,364292	10.000	570,314292	10.000
7	557,82	557,818875	1	557,818875	1.380	525,13475	10.000
8	1.794,08	1.794,08475	29	1.794,08475	947	1.402,929125	10.000
9	338,36	338,361375	1	274,1251	10.000	264,6351	10.000
10	0	0	0	0	0	0	0
11	838,69	838,685667	122	838,685667	154	745,953708	10.000
12	2.459,03	2.459,026325	2.215	2.459,02633	13	1.971,963908	10.000
18	4.177,37	4.203,7698	1	4.287,27483	0	4.018,110512	10.000
21	986,68	986,676804	364	986,676804	3255	769,059433	10.000
32	23,08	24,095083	0	22,9141	0	24,618854	1

Tabla 11. Ejecución de los algoritmos en el mes de octubre para la función de distribución.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	1.679,75	1.679,747225	38	1.679,74723	181	1.210,108904	10.000
2	1.512,57	1.512,567563	38	1.512,56756	317	1.130,540417	10.000
7	1.474,76	1.464,7825	10.000	1.464,7825	10.000	1.338,760125	10.000
8	1.769,96	1.769,95925	98	1.555,72588	10.000	1.549,844187	10.000
9	150,88	150,88255	1	150,88255	0	118,754413	10.000
10	0	0	0	0	0	0	0
11	1.597,92	1.597,915354	61	1.597,91535	3	1.322,4435	10.000
12	515,64	500,322417	10.000	503,932417	10.000	434,812417	10.000
18	2.401,95	2.401,945	2.822	2.397,045	10.000	2.351,505	10.000
21	727,2	727,199667	78	727,199667	261	654,4797	10.000
32	15,34	15,342787	1	15,342787	0	15,342787	1

Tabla 12. Ejecución de los algoritmos en el mes de noviembre para la función de distribución.

		Híbrido		NSGA-II k-means		uNSGA-II	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	331	331,002554	239	331,002554	207	237,35635	10.000
2	1.438,93	1.438,930833	136	1.438,93083	2.347	930,399604	10.000
7	1.474,57	1.460,876625	10.000	1.460,87663	10.000	1.144,716	10.000
8	1.810,65	1.810,647563	135	1.810,64756	245	1.398,44025	10.000
9	1.004,71	1.004,71295	135	1.004,71295	0	980,478538	10.000
10	0	0	0	0	0	0	0
11	1.308,95	1.285,599375	10.000	1.281,53938	10.000	929,189583	10.000
21	256,27	256,274875	1	256,274875	6.285	256,274875	0
29	379,03	379,032562	107	379,032562	2.684	261,275496	10.000
32	146,42	1,704754	10.000	0	10.000	1,704754	10.000

Tabla 13. Ejecución de los algoritmos en el mes de diciembre para la función de distribución.

		Híbrido		NSGA-II k-means		uNSGA-II	
Producto	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	506,89	506,890875	236	506,890875	190	457,194671	10.000
2	1.766,04	1.766,044917	180	1.546,43369	10.000	1.683,048792	10.000
7	870,33	869,616562	10.000	869,616562	10.000	808,398312	10.000
8	1.484,23	1.487,26475	325	1.487,26475	598	1.078,24575	10.000
9	185,2	185,199512	2	185,199512	8.130	140,818825	10.000
10	0	0	0	0	0	0	0
18	441,44	441,437813	1	435,551975	10.000	423,7803	10.000
21	256,27	256,274875	1	256,274875	0	256,274875	0
29	190,91	190,906813	219	190,906813	360	162,090046	10.000
32	162,81	0	10.000	0	10.000	0	10.000

4.3 Ejecución de la Función Objetivo del Caso de Estudio 3

Así mismo, como en la ejecución anterior, en este caso también se hizo una optimización mensual de cada materia prima. Estos datos (Tablas 14-25), también han sido comparados con los valores encontrados por Berrezueta (2020).

Tabla 14. Ejecución de los algoritmos en el mes de enero para la función de corte.

		Híbrido		**NSGA-II k-means**		**Unsga-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
1	0	0	151	0	94	1.462,5	1.000
2	0	0	5	0	51	1.404	1.000
3	747.315	747.315	73	733.613	62	1.060.879	1.000
4	202.936,5	202.936,5	31	202.936,5	87	202.936,5	946
5	18.042,57	18.042,57	8	17.362,8	51	15.118,74	27
6	1.295.014,5	1.295.014,5	109	1.295.014,5	318	1.304.608,5	1.000
7	2.577,6	2.577,6	256	129,6	141	10.749,6	1.000

Tabla 25. Ejecución de los algoritmos en el mes de febrero para la función de corte.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	181.800	181.800	30	181.800	51	181.800	94
9	3.188,25	3.188,25	242	2.913,3	279	5.136,3	1.000
4	1.2762,95	1.2762,95	526	551.690,86	1.000	65.140,5	1.000
1	183.397,5	183.397,5	2	164.385	51	165.262,5	210
10	123.777,23	137.424,9	1.000	267.605,62	1.000	241.821,97	1.000
6	716.274	716.274	124	716.274	87	724.932	1.000
11	45.000	45.000	1	45.000	51	45.000	839
12	3.575,4	58.878,6	1.000	22.686,6	1.000	73.685,4	1.000

3	298.139,95	298.139,95	83	295.326,65	151	333.314,15	1.000
7	79.933,2	79.933,2	31	79.455	73	80.663,4	1.000

Tabla 36. Ejecución de los algoritmos en el mes de marzo para la función de corte.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	77.400	77.400	161	77.400	51	77.400	351
13	9.945	9.945	35	9.945	51	9.945	104
4	69.751,89	69.751,89	390	76.268,58	1.000	90.443,78	1.000
1	78.975	78.975	30	78.682,5	173	78.097,5	321
14	84.731,4	84.731,4	1.000	84.731,4	1.000	84.731,4	1.000
2	0	0	0	0	51	0	0
5	5.850	5.850	29	5.850	51	5.850	12
3	535.532,83	852.883,8	1.000	15.403.359,69	1.000	4.193.368,41	1.000
15	1.354,28	1.354,28	171	793,42	257	1.377,7	1.000
9	5.539,95	5.539,95	54	5.534,1	51	7.131,15	1.000
6	810.576	1.744.411,5	1.000	945.594	1.000	2.050.249,5	1.000
11	177.000	177.000	46	177.000	51	177.000	636
7	11.475,6	11.475,6	207	42.211,2	1.000	48.963,6	1.000

Tabla 47. Ejecución de los algoritmos en el mes de abril para la función de corte.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	12.600	12.600	1	12.600	51	12.600	2
13	11.700	11.700	1	11.700	51	11.700	637
4	90.148,5	88.569	1	20.650,5	51	74.997	6

1	30.712,5	30.712,5	2	30.712,5	51	30.712,5	2
3	1.932.963,88	1.932.963,88	910	47.894.705,01	1.000	36.291.122,7	1.000
5	11.700	11.700	23	11.700	51	11.700	593
9	737,1	602,55	2	181,35	51	544,05	13
7	1.752	1.062	53	1.272	51	1.512	525
11	72.000	72.000	67	72.000	51	72.000	187
15	61.583	61.583	49	59.245	51	73.936	1.000
6	1.042.294,5	1.042.294,5	757	1.345.383	1.000	1.175.850	1.000

Tabla 58. Ejecución de los algoritmos en el mes de mayo para la función de corte.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	81.000	81.000	91	81.000	51	81.000	613
4	88.602,35	88.602,35	74	87.171	51	77.410,25	807
1	0	0	293	0	228	9.664,5	1.000
14	141.219	141.219	45	141.219	51	141.219	15
2	0	0	0	0	51	0	0
3	1.055.146,95	1.055.146,95	127	1.025.300,32	111	874.856,35	809
15	226.395	226.395	1	226.395	51	226.395	650
13	17.550	17.550	20	17.550	51	17.550	2
5	10.530	10.530	95	10.530	51	10.530	9
9	10.354,5	10.354,5	129	9.734,4	77	10.032,75	34
6	2.844.328,5	2.844.328,5	233	0	1.000	2.856.379,5	1.000
11	84.000	84.000	16	84.000	51	84.000	703
7	333.552	333.552	23	175.424,4	59	326.107,8	301

Tabla 69. Ejecución de los algoritmos en el mes de junio para la función de corte.

		Híbrido		NSGA-II k-means		uNSGA-II	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	162.000	162.000	66	162.000	51	162.000	175
4	473.539,37	436.164,16	14	309.442,28	51	359.047,03	38
1	287.820	287.820	55	269.977,5	51	287.235	447
14	141.219	141.219	1	141.219	51	141.219	300
2	0	0	0	0	51	0	0
3	242.435,7	242.435,7	1.000	574.976,34	1.000	329.239,01	1.000
15	266.643	266.643	108	266.643	55	266.643	117
13	21.060	21.060	108	21.060	51	21.060	3
5	12.870	12.870	67	12.870	51	12.870	25
9	26.851,5	26.851,5	116	101.725,65	1.000	29.589,3	1.000
6	2.408.035,5	2.408.035,5	83	2.418.916,5	1.000	2.430.499,5	1.000
11	216.000	216.000	25	216.000	51	216.000	445
7	63.978	63.978	103	57.203,4	168	133.191	1.000

Tabla 20. Ejecución de los algoritmos en el mes de julio para la función de corte.

		Híbrido		NSGA-II k-means		uNSGA-II	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	180.000	180.000	38	180.000	51	180.000	893
4	89.454,89	89.454,89	1.000	401.238,84	1.000	1.336.803,34	1.000
1	303.322,5	303.322,5	14	132.502,5	51	288.697,5	119
2	0	0	0	0	51	0	0
3	1.893.782,48	1.893.782,48	138	1.748.090,34	51	1.885.682,08	881
15	201.240	201.240	53	201.240	51	201.240	293
13	15.210	15.210	1	15.210	51	15.210	305
5	9.360	9.360	5	9.360	51	9.360	273

9	1.994,85	1.994,85	19	1.994,85	51	1.994,85	315
6	3.809.052	3.809.052	1.000	5.502.744	1.000	3.877.906,5	1.000
11	184.800	184.800	1.000	0	1.000	348.900	1.000
7	77.357,4	77.357,4	140	346.230	1.000	150.012	1.000

Tabla 21. Ejecución de los algoritmos en el mes de agosto para la función de corte.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	18.000	18.000	1	18.000	51	18.000	221
4	56.355	56.355	213	53.644,5	83	66.261	1.000
1	31.590	31.590	22	29.250	51	30.712,5	75
2	0	0	0	0	51	0	0
3	292.426,88	292.426,88	663	568.646,85	1.000	447.952,7	1.000
15	37.323	37.323	435	37.323	52	65.754	1.000
13	11.115	11.115	1	11.115	51	11.115	347
5	6.435	6.435	46	6.435	51	6.435	201
9	181,35	181,35	1	181,35	51	181,35	33
6	2.427.282	2.427.282	104	2.428.861,5	1.000	2.451.793,5	1.000
11	229.200	229.200	244	229.200	51	229.200	472
7	10.201,2	10.201,2	339	21.174,6	1.000	482.267,4	1.000

Tabla 22. Ejecución de los algoritmos en el mes de septiembre para la función de corte.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	72.000	72.000	9	72.000	51	72.000	215
4	35.895,6	35.896	1.000	42.626,5	1.000	35.896	1.000

1	42.705	42.705	1	42.705	51	42.705	119
3	266.385,6	266.385,6	10	229.530,82	58	352.691,4	1.000
15	80.496	80.496	25	80.496	51	80.496	42
13	6.435	6.435	42	6.435	51	6.435	124
5	4.095	4.095	0	4.095	51	4.095	50
9	725,4	725,4	21	725,4	51	906,75	1.000
6	1.232.536,5	1.232.536,5	109	1.232.536,5	194	1.240.609,5	1.000
11	0	0	0	0	51	0	0
7	25.920	25.920	2	25.920	51	25.920	72

Tabla 23. Ejecución de los algoritmos en el mes de octubre para la función de corte.

		Híbrido		**NSGA-II k-means**		**uNSGA-II**	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	86.400	86.400	1	86.400	51	86.400	336
4	18.321,03	47.121,99	1.000	75.728,47	1.000	76.572,09	1.000
1	121.398,75	121.398,75	157	121.398,75	53	121.983,75	1.000
14	56.487,6	56.487,6	1.000	0	1.000	56.487,6	1.000
2	0	0	0	0	51	0	0
3	345.264,08	345.264,08	45	268.633,24	51	325.563,03	211
15	25.155	25.155	1	25.155	51	25.155	1
13	1.755	1.755	2	1.755	51	1.755	45
5	3.510	3.510	1	3.510	51	3.510	0
9	906,75	1.433,25	1.000	982.911,15	1.000	1.433,25	1.000
6	774.540	774.540	163	774.540	110	782.730	1.000
11	106.500	106.500	34	106.500	51	106.500	923
7	3.609	3.609	140	6.955,2	1.000	30.329,4	1.000

Tabla 24. Ejecución de los algoritmos en el mes de noviembre para la función de corte.

		Híbrido		NSGA-II k-means		uNSGA-II	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	18.000	18.000	1	18.000	51	18.000	139
4	37.709,1	37.709,1	63	37.670,52	54	39.562,42	1.000
1	70.785	70.785	65	70.785	51	70.785	218
14	112.975,2	112.975,2	1.000	112.975,2	1.000	112.975,2	1.000
2	0	0	0	0	51	0	0
3	0	0	0	0	51	0	0
15	90.733,5	90.733,5	24	90.733,5	51	90.733,5	44
9	4.633,2	4.147,65	2	1.053	51	1.649,7	511
11	0	0	0	0	51	0	0
7	4.377	4.377	1	1.021,2	51	3.595,8	2

Tabla 25. Ejecución de los algoritmos en el mes de julio para la función de corte.

		Híbrido		NSGA-II k-means		uNSGA-II	
Materia Prima	Valor Esperado	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones	Valor Obtenido	Iteraciones
8	54.000	54.000	36	54.000	180	54.000	180
4	38.848,1	38.848,1	187	37.518,5	1.000	37.518,5	702
1	92.137,5	92.137,5	67	92.137,5	1.000	92.137,5	451
14	56.487,6	56.487,6	1.000	56.487,6	1.000	56.487,6	1.000
2	0	0	0	0	51	0	0
3	47.502	47.502	641	109.986,08	1.000	109.986,08	1.000
15	45.279	45.279	1	45.279	51	45.279	15
13	3.510	3.510	1	3.510	132	3.510	132
5	2.340	2.340	1	2.340	143	2.340	143
9	6.230,25	6.230,25	330	6.230,25	1.000	6.230,25	471
6	607.815	607.815	217	607.815	1.000	607.815	390

11	42.000	42.000	13	42.000	1.000	42.000	794
7	10.236	10.236	126	5.647,8	1.000	5.647,8	961

Capítulo V - Discusión

En el capítulo anterior se presentaron todos los resultados obtenidos después de haber ejecutado los algoritmos en cada uno de los casos de estudio del Capítulo 3. Ahora, continuando con el proceso, y después de haber obtenido estos resultados (datos e información), se puede realizar la comparación entre los algoritmos y así poder establecer las bondades y desventajas de los mismos.

5.1 Comparación de los Algoritmos del Caso de Estudio 1

En las Figuras 10 y 11 se puede observar la convergencia de los tres algoritmos usados en el caso de estudio 1. Con estas gráficas se pueden determinar varios puntos:

- El algoritmo M-PAES llega a muy buenos resultados, pero también toma mucho tiempo de ejecución para alcanzar a los mejores valores.
- El algoritmo MOPSO, por otra parte, realiza su trabajo en poco tiempo, pero no siempre se obtienen las mejores soluciones.
- El algoritmo Híbrido es el más rápido de los demás, y converge a valores casi óptimos. Además, converge muy rápido en las primeras iteraciones, e incluso, con menos iteraciones es capaz de devolver valores cercanos al óptimo.
- La ejecución de una iteración del algoritmo híbrido es más rápida que una ejecución del MOPSO o M-PAES.

Por otra parte, en la Tabla 26, se presenta un resumen de los resultados obtenidos en la ejecución de cada uno de los algoritmos de optimización, en términos de convergencia y tiempo de ejecución.

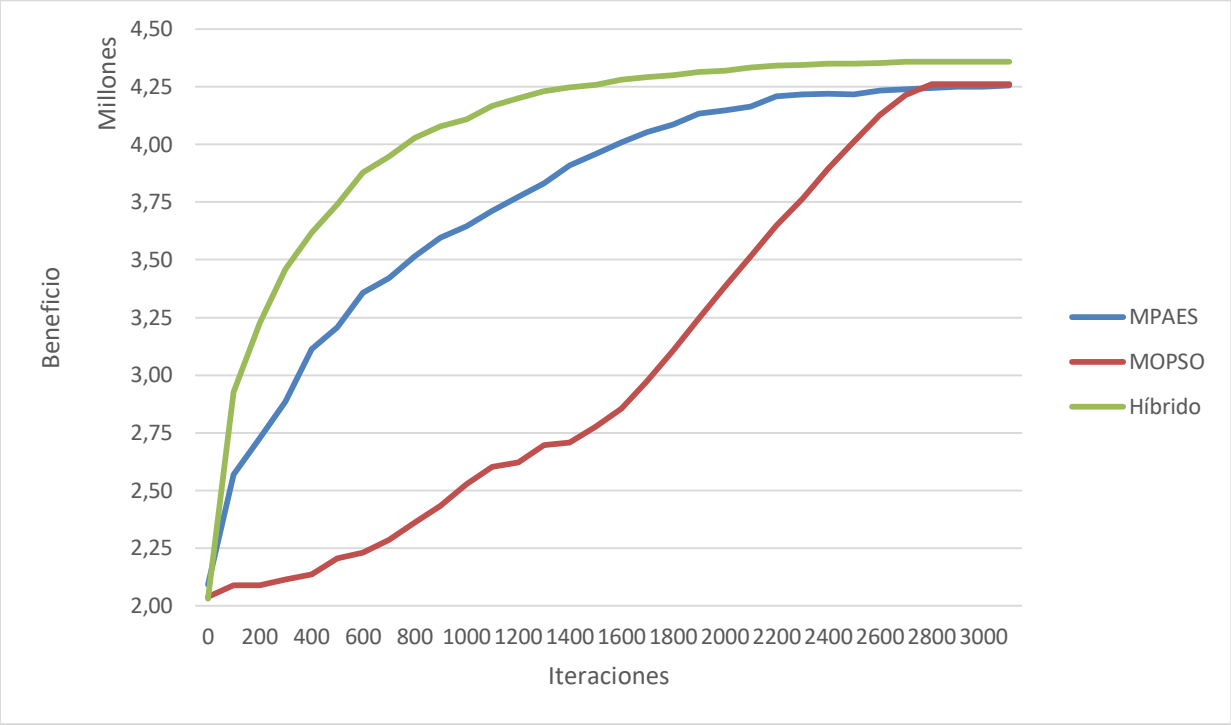

Figura 10. Convergencia de los algoritmos de optimización usados para maximizar el beneficio de la CS.

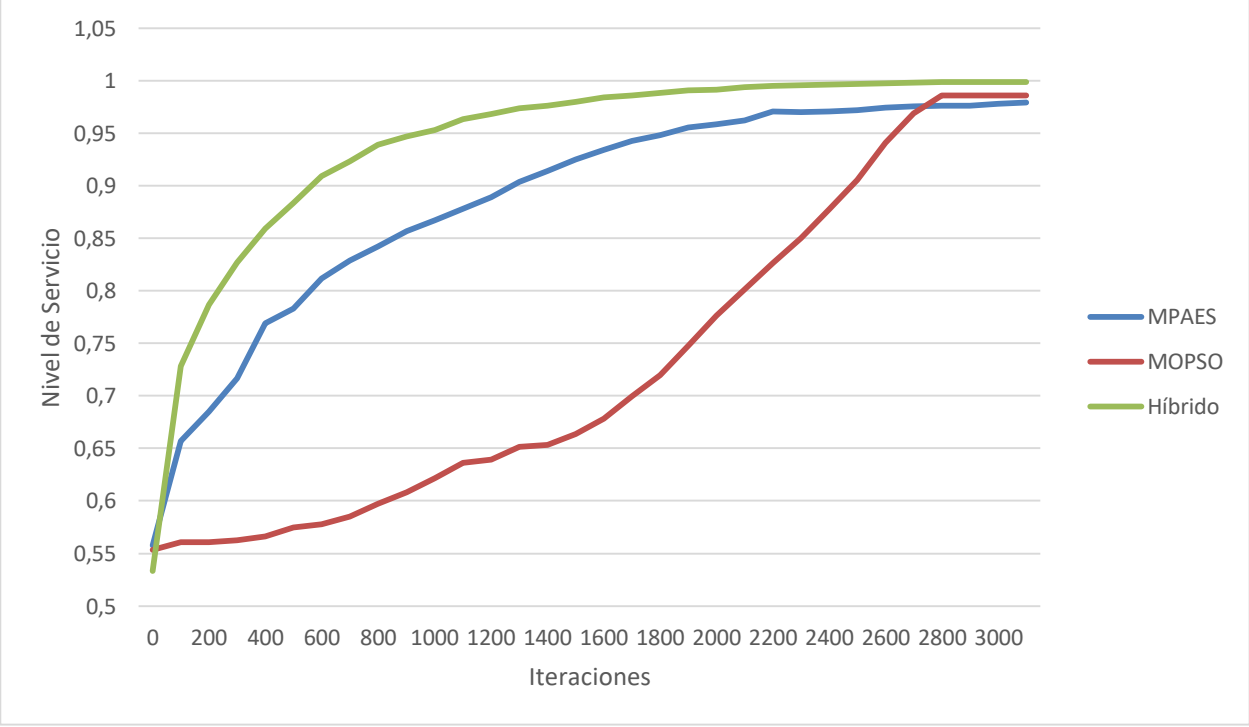

Figura 11. Convergencia de los algoritmos de optimización usados para maximizar el nivel de servicio ofrecido a los clientes finales.

Tabla 26. Comparación de los algoritmos de optimización para el caso de estudio 1.

Algoritmo	M-PAES	MOPSO	Híbrido
Tiempo de ejecución del algoritmo al llegar a converger a un valor constante para la función de nivel de servicio/Relación con el valor del algoritmo M-PAES	17 min 100%	51,32 s 5,03%	8,08 s 0,79%
Número de iteraciones donde el algoritmo converge a un valor constante para la función de nivel de servicio /Relación con el valor del algoritmo M-PAES	8141 100%	2848 34,98%	3050 37,46%
Valor al que converge la función de Beneficio/Relación con el valor del algoritmo MOPSO	$4.314.239,065 101,19%	$4.263.268,324 100%	$4.360.030,69 102,6%
Valor al que converge la función de Nivel de Servicio	99%	98,6%	99,9%

Se puede observar que la convergencia del algoritmo híbrido es de mejor que la del MOPSO. En la Tabla 26 se observa que el algoritmo M-PAES y el híbrido tienden a converger a valores casi cercanos a lo óptimo. Por otra parte, el algoritmo MOPSO llega también a buenos resultados, pero no tan buenos como los otros dos algoritmos. Sin embargo, el algoritmo híbrido es el que más cerca se encuentra de la solución óptima.

Para la comparativa también se debe tener en cuenta el tiempo de ejecución de cada algoritmo. En el caso del M-PAES, el tiempo de ejecución es demasiado grande. Esto se puede volver un problema a largo plazo, debido a que mientras más grande el problema, más tiempo en encontrar una solución le tomará al algoritmo. En el caso del algoritmo MOPSO y el híbrido, los tiempos son bastante más bajos con respecto al M-PAES, siendo el algoritmo híbrido el más rápido.

El número de iteraciones está relacionado directamente al tiempo de ejecución, el algoritmo que más iteraciones toma para converger es el algoritmo M-PAES; mientras que, el que menos iteraciones tiene es el algoritmo híbrido.

5.2 Comparación de los Algoritmos del Caso de Estudio 2

Con los resultados obtenidos en el capítulo anterior, se puede analizar el número de iteraciones que le tomó a cada algoritmo ejecutar todo el proceso, y también el tiempo de ejecución en el caso de estudio 2.

En la Tabla 27 se muestran los valores de tiempo de ejecución y número de iteraciones que le tomó al algoritmo ejecutar todos los productos en todos los meses para el segundo caso de estudio.

Tabla 27. Comparación de los algoritmos de optimización para el caso de estudio 2.

Algoritmo	uNSGA-II	NSGA-II k-means	Híbrido
Tiempo de ejecución/Relación con el valor del algoritmo NSGA-II k-means	0,33 s 0,55%	59,54 s 100%	7,32 s 12,39%
Número de iteraciones/Relación con el valor del algoritmo uNSGA-II	1013088 100%	390410 38,53%	131695 12,99%

En la Figura 12 se presenta el número de iteraciones que toma el algoritmo para llegar al beneficio total de la CS.

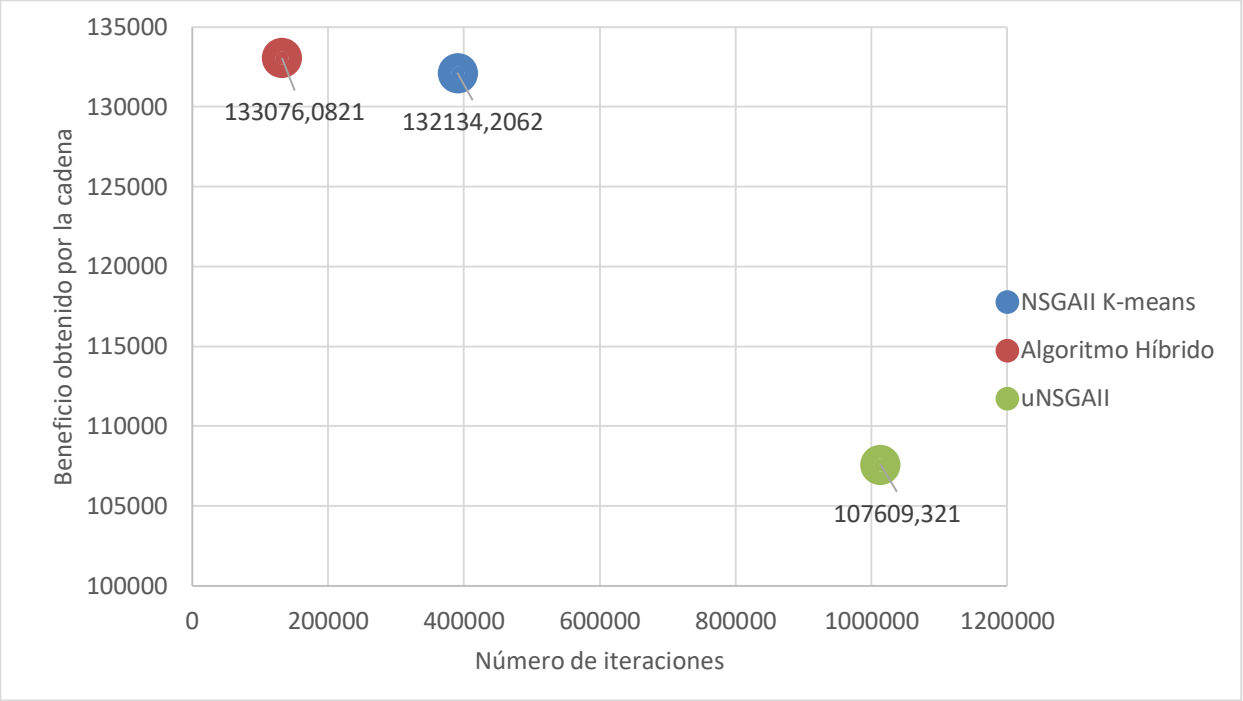

Figura 12. Beneficio total obtenido por la CS.

Comparación de los Algoritmos del Caso de Estudio 3

En este apartado se discuten los resultados obtenidos en el tercer caso de estudio. La Tabla 27 presenta el tiempo de ejecución y numero de iteraciones totales al ejecutar el algoritmo para todos los productos en todos los meses.

Tabla 28. Comparación de los algoritmos de optimización para el caso de estudio 3.

Algoritmo	uNSGA-II	NSGA-II k-means	Híbrido
Tiempo de ejecución/Relación con el valor del algoritmo NSGA-II k-means	0,66 s 0,63%	104,4 s 100%	18,7 s 17,91%
Número de iteraciones/Relación con el valor del algoritmo uNSGA-II	71180 100%	40178 56,44%	26570 37,32%

En la Figura 13 se presenta el número de iteraciones en el que el algoritmo llega a obtener el menor desperdicio generado. Este desperdicio se encuentra expresado en mm.

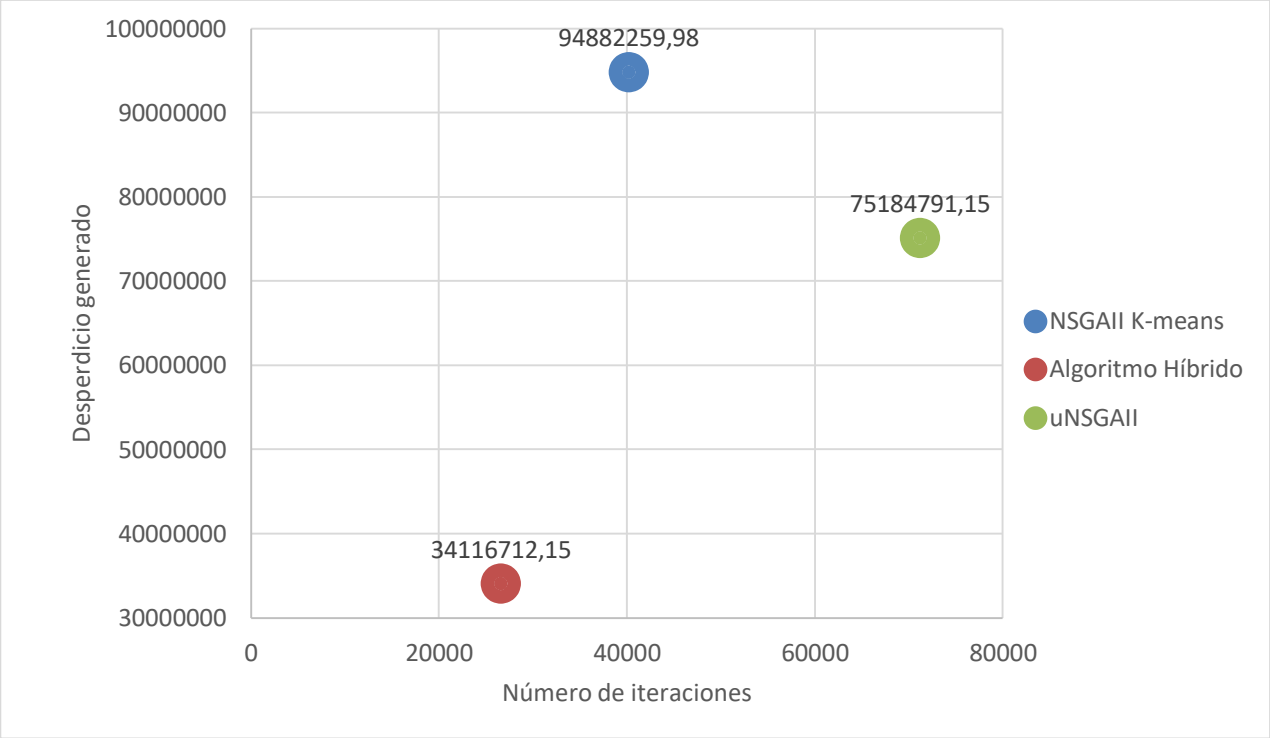

Figura 13. Desperdicio generado por el corte de la materia prima.

En las Tablas 2-13, se puede observar que el micro-algoritmo no llega a converger en todos los productos y materias primas. Por otra parte, el algoritmo NSGA-II k-means llega a converger en casi todos los productos y materias primas. Finalmente, el algoritmo híbrido también converge en la mayoría de sus productos y materias primas.

El micro-algoritmo realiza su ejecución en poco tiempo, en centésimas de segundos, por lo que éste es el algoritmo más rápido de todos. El algoritmo NSGA-II k-means es el que más tiempo toma en ejecutarse. Y, por último, el algoritmo híbrido tiene un tiempo relativamente bajo, mucho mejor que el algoritmo NSGA-II k-means.

El tiempo de ejecución del micro-algoritmo fue el más bajo, pero el número de iteraciones para converger es muy alto. El algoritmo NSGA-II k-means converge en menos iteraciones. Finalmente, el algoritmo híbrido es el que menos iteraciones toma para converger a los resultados.

Al haber discutido todos estos puntos, se ha llegado a determinar que el algoritmo híbrido logra cumplir el objetivo de mejorar en velocidad y en calidad de los valores de convergencia. En comparación con cada uno de los algoritmos se tiene que:

- M-PAES tiene una buena convergencia de valores, pero a costa de un mayor del tiempo de ejecución. El algoritmo híbrido presenta un tiempo de ejecución muy corto, reemplazando completamente este algoritmo.

- MOPSO tiene un buen tiempo de ejecución, pero se queda en el problema de que no llega a converger al óptimo debido al límite de velocidad de cada partícula. Mientras que, el algoritmo híbrido inicialmente maneja todo el espacio del problema, y en las siguientes iteraciones lo reduce para mejorar el tiempo de ejecución y la obtención de la solución óptima.

- uNSGA-II es el más veloz en tiempo de ejecución, pero al tener pocos individuos, no existe un cambio a la hora de seleccionar los nuevos padres, por lo que, toma más iteraciones en llegar a buenos valores, y en muchos casos no llega a una buena solución. Por otra parte, el algoritmo híbrido inicia con una población pequeña, en donde los diferentes individuos tienen valores diferentes. De esta manera, se puede seleccionar más rápido los mejores valores, pero a medida que el algoritmo se ejecuta estos individuos aumentan, para poder tener un mayor conjunto de individuos para seleccionar como nuevos padres.

- NSGA-II K-means trata de crear grupos y luego ejecutar sobre cada grupo el algoritmo NSGA-II. Este proceso ha llegado a tomar bastante tiempo, y aunque se realice el proceso de micro-algoritmos también se debe tener en cuenta el número de grupos, por lo que hace que su tiempo de ejecución sea similar al NSGA-II. En contra parte, el algoritmo híbrido presenta solo un grupo que al inicio es muy pequeño, pero va creciendo.

Teniendo en cuenta toda esta información, se puede responder a la pregunta de investigación "¿Un algoritmo metaheurístico híbrido puede realizar la optimización de una CS de una manera más eficiente (en términos de tiempo y calidad de la solución)?". Se concluyó que, si se pudo desarrollar un algoritmo que comparta bondades de otros algoritmos de optimización para maximizar el beneficio de una CS en términos de calidad de convergencia y tiempo de ejecución. Se puede mencionar también que este algoritmo puede ser usado para problemas con un gran número de variables y que cuente con un gran espacio de búsqueda. Es importante aclarar que este algoritmo solo ha sido probado en problemas de CS, por lo que, en otros ámbitos, los resultados pueden ser diferentes. Además, se debe tener en cuenta que el algoritmo creado no siempre puede encontrar la solución óptima a un problema de optimización.

Comparación con otros trabajos

En el artículo de Kuo y Han (2011), se presentan tres algoritmos híbridos. El primero tiene como base el algoritmo de enjambre de partículas PSO, con la diferencia que al final del cambio de velocidades se procede a dividir las partículas en dos partes. La primera queda intacta, pero la segunda pasa por un proceso de mutación y cruce perteneciente a los algoritmos genéticos (GA por las palabras en inglés Genetic Algorithm). El segundo es muy parecido, solo que, en vez de dividir las partículas, realiza un cruce y mutación entre todas ellas. Finalmente, el tercero, igual

que los otros dos, tiene como base los PSO, pero este algoritmo se basa en verificar si la partícula global y local cambian. Si no cambian, se realizar el proceso de cruce y mutación de todo el enjambre. Todos estos algoritmos tienen la lógica de usar la velocidad de convergencia de los PSO; pero, para no estancarse en un óptimo que no sea el global, deciden usar los GA. Sin embargo, como se había mencionado, el proceso PSO tiende a tomar mucho más tiempo de ejecución, por lo que, en problemas con un mayor número de variables, puede demorar demasiado en encontrar una solución. Comparado con el algoritmo de este trabajo, se puede observar una reducción de tiempo de ejecución debido a que los procesos en este algoritmo pertenecen a los NSGA-II, los cuales son mucho más veloces, haciendo que este algoritmo sea mucho más eficiente.

Jamshidi, Fatemi Ghomi, y Karimi (2012) presentan un algoritmo memético o híbrido. Este algoritmo se basa en el método Taguchi, que es usado para la selección de padres dentro de un conjunto de individuos. Esta selección se realiza mediante unas reglas especiales del método, y luego de eso se procede a realizar la combinación. En ese artículo, se presenta una nueva función que ayuda a obtener mejores resultados en la optimización, por ende, toma menor tiempo llegar a la solución. Comparado al algoritmo híbrido de este trabajo, se puede decir que tiene una gran semejanza, ya que para la selección de padres se usan los PSO, y con este mismo proceso se reduce el tiempo de ejecución del algoritmo original (GA). Por otra parte, una de las grandes diferencias de este algoritmo, desarrollado por Jamshidi et al (2012). es que funciona muy bien pero específicamente para el problema planteado en ese documento, mientras que de manera general podría no llegar a encontrar las mejores soluciones. Mientras tanto, el algoritmo diseñado en este trabajo si puede llegar a generalizar de mejor manera los problemas de CS por lo que le otorga más valor a este último algoritmo.

Otro de los algoritmos híbridos encontrados en la literatura es el presentado por Soleimani y Kannan (2015). Este algoritmo trata de usar las técnicas de los GA y los PSO. Primero, se genera una población inicial que pasa por un proceso de cruce y mutación de los individuos. Así mismo, se usan dos nuevos operadores para encontrar los nuevos hijos de la iteración y, de esta manera descubrir el mejor global y los mejores locales. Luego, se calcula la función objetivo y se escogen los mejores individuos. Este proceso se realiza hasta que se cumpla el criterio de parada. A diferencia del algoritmo de este trabajo que utiliza al inicio pocos individuos, pero luego se va aumentando esta población, el algoritmo de Soleimani y Kannan utiliza mucha memoria, debido a que tiene que almacenar los padres, los hijos, los mejores valores locales y el valor global. Otra diferencia es el proceso de ordenamiento de las soluciones, debido a que este proceso consume mucho más tiempo de ejecución que puede afectar directamente a encontrar la mejor solución del problema deseado.

En el trabajo realizado por Xiong, Gong, Jin, y Fan (2018) se desarrolla un nuevo algoritmo híbrido. Este algoritmo consiste en generar dos poblaciones; la primera pasa por los procesos de GA (mutación y cruce de los individuos), mientras que, el segundo grupo pasa por los procesos básicos de PSO (cambio de velocidad de las partículas). Cabe destacar que ambos grupos comparten el mejor individuo o el mejor global. Al tener que realizar el proceso de PSO, el tiempo de ejecución será mucho mayor al algoritmo propuesto en este trabajo. Por lo que, se puede decir que, el algoritmo de este trabajo utiliza menos recursos para llegar a la solución del problema, considerando tanto tiempo como memoria.

Capítulo VI – Conclusión

El objetivo de la investigación fue desarrollar un algoritmo metaheurístico híbrido usando las bondades de otros algoritmos que pueda optimizar una CS de una manera mucho más eficiente, en términos de tiempo y calidad de convergencia. Así mismo, han sido cumplidos los objetivos específicos: se ha realizado una revisión sistemática de los algoritmos metaheurísticos más utilizados; se analizaron las variables de entrada, las funciones objetivo y las restricciones de los tres casos de estudio; se realizó un análisis de los algoritmos de optimización para encontrar los mejores candidatos para el desarrollo del algoritmo híbrido; y se desarrolló el algoritmo híbrido y la aplicación que ejecuta dicho algoritmo de manera satisfactoria. Los resultados de esta investigación muestran que el nuevo algoritmo ha logrado maximizar los beneficios de las CS y también ha reducido el tiempo de ejecución y el uso de memoria en las otras funciones objetivo propuestas usando las variables de la cadena. Los hallazgos de esta investigación proporcionan información para decir que un algoritmo híbrido que toma las bondades de los micro-algoritmos, NSGA-II y MOPSO puede mejorar los procesos en general, no solo en el caso de la optimización, sino se puede llegar más allá como algoritmos de inteligencia artificial o algoritmos de ordenamiento. Este algoritmo híbrido llega a obtener mejores resultados debido a la combinación de las técnicas de los algoritmos de optimización de CS. El algoritmo base usado fue el GA debido a que es uno de los más conocidos y usualmente tiende a llegar a converger a la solución óptima; luego, se ha usado el algoritmo PSO para la selección de padres para el proceso de cruce y mutación; y, finalmente, los micro-algoritmos fueron usados para reducir el uso de memoria de la máquina. Los dos primeros casos de estudio presentaron un modelo para aumentar el beneficio de una CS. El último caso de estudio presentó un modelo para reducir desperdicio generado por las plantas al cortar materia prima. Estos modelos son bastantes precisos debido a los datos reales

proporcionados por las empresas, por lo que, pueden ser usados en el futuro para predecir costes o cantidades de las distintas etapas de fabricación. Lastimosamente, dentro de este trabajo no fue posible realizar esta predicción ya que las empresas no pudieron entregar todos los datos necesarios. Por otro lado, la comparación realizada entre los algoritmos fue exhaustiva, debido a que se analizaron los puntos más fuertes de un algoritmo como: complejidad temporal, espacial, computacional y la convergencia al resultado.

La generalización de estos resultados está sujeta a ciertas limitaciones. Por ejemplo, este algoritmo específicamente ha sido diseñado para la resolución de problemas basados en CS, por lo que no se puede decir que éste sea mucho mejor en otros ámbitos. Se tiene como sugerencia, implementar el algoritmo y realizar otras pruebas en diferentes casos con el fin de demostrar que es mucho más eficiente a la hora de optimizar en otras áreas. Se debe tener en cuenta, que el nuevo algoritmo tiene una mejora mucho más significativa cuando el tamaño del problema es grande. Esto quiere decir que, mientras más datos o más grande sea el problema, la diferencia entre la comparativa de resultados será más notable que cuando se tengan pocos datos. Otro punto a analizar es que este algoritmo combina diferentes técnicas según las bondades presentadas por algunos algoritmos de optimización, pero esto no significa que la combinación de ventajas sea la mejor debido a que existe una gran posibilidad de combinaciones de técnicas para mejorar el proceso de optimización. Sin embargo, cabe destacar que en la literatura se han visto varios algoritmos híbridos de optimización, que usualmente unen técnicas de GA y algoritmos PSO. Como una mejora a este trabajo se puede agregar técnicas de micro-algoritmos para reducir el tiempo de ejecución y la complejidad espacial (es decir, la memoria que tendrá que usar la máquina para almacenar los datos de los individuos) de cada iteración del algoritmo híbrido desarrollado.

En este documento se ha presentado una de estas combinaciones, pero es posible que en trabajos futuros se puedan encontrar nuevas combinaciones de técnicas para reducir tiempos de ejecución y números de iteraciones para la convergencia de la solución. Por lo que, se motiva al lector a analizar más de las bondades de otros algoritmos para encontrar un nuevo algoritmo híbrido. En el Apéndice C se encuentran las publicaciones derivadas de este Trabajo de Titulación. Finalmente, se puede concluir que, con el paso de los años, existirán nuevos algoritmos y nuevas técnicas no solo de optimización sino de cualquier área relacionada con la tecnología.

Bibliografía

Altiparmak, F., Gen, M., Lin, L., & Paksoy, T. (2006). A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers & Industrial Engineering*, *51*(1), 196-215. https://doi.org/10.1016/j.cie.2006.07.011

Arifovic, J. (1994). Genetic algorithm learning and the cobweb model. *Journal of Economic Dynamics and Control*, *18*(1), 3-28. https://doi.org/10.1016/0165-1889(94)90067-1

Ballou, R. H. (1978). *Basic business logistics: Transportation, materials management, & physical distribution*. Prentice-Hall.

Ballou, R. H. (2004). *Logística: Administración de la cadena de suministro*. Pearson Educación.

Beheshti, Z., & Shamsuddin, S. M. (2013). A review of population-based meta-heuristic algorithm. *International Journal of Advances in Soft Computing and Its Applications*, *5*, 1-35.

Berrezueta, N. (2020). *Optimización de la cadena de suministro mediante el uso de un algoritmo genético basado en clusterización*. http://dspace.ucuenca.edu.ec/handle/123456789/34050

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, *2*(4), 353-373. https://doi.org/10.1016/j.plrev.2005.10.001

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.*, *35*(3), 268–308. https://doi.org/10.1145/937503.937505

Chan, F. T. S., Chung, S. H., & Wadhwa, S. (2005). A hybrid genetic algorithm for production and distribution. *Omega*, *33*(4), 345-355. https://doi.org/10.1016/j.omega.2004.05.004

Chopra, S., & Meindl, P. (2007). Supply Chain Management. Strategy, Planning & Operation. En C. Boersch & R. Elschen (Eds.), *Das Summa Summarum des Management: Die 25 wichtigsten Werke für Strategie, Führung und Veränderung* (pp. 265-275). Gabler. https://doi.org/10.1007/978-3-8349-9320-5_22

Coello Coello, C.A., & Lechuga, M. S. (2002). MOPSO: A proposal for multiple objective particle swarm optimization. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, *2*, 1051-1056 vol.2. https://doi.org/10.1109/CEC.2002.1004388

Coello Coello, Carlos A., & Toscano Pulido, G. (2001). A Micro-Genetic Algorithm for Multiobjective Optimization. En E. Zitzler, L. Thiele, K. Deb, C. A. Coello Coello, & D. Corne (Eds.), *Evolutionary Multi-Criterion Optimization* (pp. 126-140). Springer. https://doi.org/10.1007/3-540-44719-9_9

Cooper, M., & Ellram, L. (1993). Characteristics of Supply Chain Management & the Implications for Purchasing & Logistics Strategy. *International Journal of Logistics Management, The*, *4*, 13-24. https://doi.org/10.1108/09574099310804957

Dantzig, G. B. (1998). *Linear Programming and Extensions*. Princeton University Press.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182-197. https://doi.org/10.1109/4235.996017

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *26*(1), 29-41. https://doi.org/10.1109/3477.484436

Dorigo, Marco, & Di Caro, G. (1999). Ant colony optimization: A new meta-heuristic. *IEEE.*, *2*, 1477 Vol. 2. https://doi.org/10.1109/CEC.1999.782657

Drucker, P. F. (2007). *The practice of management* (Rev. ed). Amsterdam ; London : Butterworth-Heinemann. https://trove.nla.gov.au/work/10359984

Elkhechafi, M., Benmamoun, Z., Hachimi, H., Amine, A., & Elkettani, Y. (2018). Firefly Algorithm for Supply Chain Optimization. *Lobachevskii Journal of Mathematics*, *39*(3), 355-367. https://doi.org/10.1134/S1995080218030125

Fink, A. (2010). *Conducting Research Literature Reviews: From the Internet to Paper*. SAGE.

Gao, Y., Zhang, G., Lu, J., & Wee, H.-M. (2011). Particle swarm optimization for bi-level pricing problems in supply chains. *Journal of Global Optimization*, *51*(2), 245-254. https://doi.org/10.1007/s10898-010-9595-8

Garcia, D. J., & You, F. (2015). Supply chain design and optimization: Challenges and opportunities. *Computers & Chemical Engineering*, *81*, 153-170. https://doi.org/10.1016/j.compchemeng.2015.03.015

Glover, F., & Laguna, M. (1999). Tabu search I. En *ORSA Journal on Computing* (Vol. 1). https://doi.org/10.1287/ijoc.1.3.190

Glover, F. W., & Kochenberger, G. A. (Eds.). (2003). *Handbook of Metaheuristics*. Springer US. https://doi.org/10.1007/b101874

Grossmann, I. (2005). Enterprise-wide Optimization: A New Frontier in Process Systems Engineering. *AIChE Journal*, *51*, 1846-1857. https://doi.org/10.1002/aic.10617

Guamán Ochoa, M. M., Cárdenas Arias, B. E., Siguenza-Guzman, L., & Segarra, L. (2020). Integración de información de costos para la toma de decisiones en industrias de ensamblaje. *Revista Economía y Política*, 100-117.

Guerrero, P. (2018). *Tiempos estándar y modelización de procesos de ensamblaje: Televisores y tarjetas electrónicas usando programación no lineal y BPMN* [Tesis de Grado, Universidad de Cuenca]. http://dspace.ucuenca.edu.ec/handle/123456789/31279

Habib, M. A., Rahman, N., Alam, Z., Zoarder, A., & Haque, M. (2016). Route Optimization in Supply chain Network. *Imperial Journal of Inter disciplinary Research*.

Hasançebi, O., Çarbaş, S., Doğan, E., Erdal, F., & Saka, M. P. (2009). Performance evaluation of metaheuristic search techniques in the optimum design of real size pin jointed structures. *Computers & Structures*, *87*(5), 284-302. https://doi.org/10.1016/j.compstruc.2009.01.002

Heskett, J. L. (1964). *Business logistics: Management of physical supply and distribution*. Ronald Press Co.

Holland, J. H. (1992a). Genetic Algorithms. *Scientific American*, *267*(1), 66-73. JSTOR.

Holland, J. H. (1992b). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence* (Edición: Reprint). A Bradford Book.

Hussain, K., Mohd Salleh, M. N., Cheng, S., & Shi, Y. (2019). Metaheuristic research: A comprehensive survey. *Artificial Intelligence Review*, *52*(4), 2191-2233. https://doi.org/10.1007/s10462-017-9605-z

Jamshidi, R., Fatemi Ghomi, S. M. T., & Karimi, B. (2012). Multi-objective green supply chain optimization with a new hybrid memetic algorithm using the Taguchi method. *Scientia Iranica*, *19*(6), 1876-1886. https://doi.org/10.1016/j.scient.2012.07.002

Kaur, G., & Chhabra, A. (2020). *Evolution and Growth of Metaheuristics for Solving Variety of Problems*.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, *4*, 1942-1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671-680. https://doi.org/10.1126/science.220.4598.671

Knowles, J., & Corne, D. (2000). M-PAES: A Memetic Algorithm for Multiobjective Optimization. *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, *1*. https://doi.org/10.1109/CEC.2000.870313

Koziel, S., & Yang, X.-S. (Eds.). (2011). *Computational Optimization, Methods and Algorithms*. Springer-Verlag. https://doi.org/10.1007/978-3-642-20859-1

Kuo, R. J., & Han, Y. S. (2011). A hybrid of genetic algorithm and particle swarm optimization for solving bi-level linear programming problem—A case study on supply chain model. *Applied Mathematical Modelling*, *35*(8), 3905-3917. https://doi.org/10.1016/j.apm.2011.02.008

Lazar, A. (2000). *Heuristic Knowledge Discovery for Archaeological Data Using Genetic Algorithms and Rough Sets*. https://doi.org/10.4018/9781930708266.ch014

Lee, K. S., & Geem, Z. W. (2004). A new structural optimization method based on the harmony search algorithm. *Computers & Structures*, *82*(9), 781-798. https://doi.org/10.1016/j.compstruc.2004.01.002

Lummus, R. R., & Alber, K. L. (1997). *Supply Chain Management: Balancing the Supply Chain with Customer Demand*. APICS Educational & Research Foundation, Incorporated.

Mastrocinque, E., Yuce, B., Lambiase, A., & Packianather, M. S. (2013). A Multi-Objective Optimization for Supply Chain Network Using the Bees Algorithm. *International*

Journal of Engineering Business Management, *5*(Godište 2013), 5-38. https://doi.org/10.5772/56754

Michalewicz, Z., & Fogel, D. B. (2004). *How to Solve It: Modern Heuristics* (Edición: 2nd ed. 2004). Springer.

Mousavi, S. M., Alikar, N., Niaki, S. T. A., & Bahreininejad, A. (2015). Optimizing a location allocation-inventory problem in a two-echelon supply chain network: A modified fruit fly optimization algorithm. *Computers & Industrial Engineering*, *87*, 543-560. https://doi.org/10.1016/j.cie.2015.05.022

Mousavi, S. M., Bahreininejad, A., Musa, S. N., & Yusof, F. (2017). A modified particle swarm optimization for solving the integrated location and inventory control problems in a two-echelon supply chain network. *Journal of Intelligent Manufacturing*, *28*(1), 191-206. https://doi.org/10.1007/s10845-014-0970-z

Oliver, R. K., & Webber, M. D. (1982). *Supply-chain management: Logistics catches up with strategy*. https://doi.org/null

Orellana, J. (2020). *Optimización de costos en la cadena de suministro en empresas de ensamblaje: Comparación de algoritmos evolutivos en el caso de estudio de ensamblaje de televisores*. http://dspace.ucuenca.edu.ec/handle/123456789/34007

Pauling, L. (1960). *The Nature of the Chemical Bond and the Structure of Molecules and Crystals: An Introduction to Mode* (Third edition). Cornell University Press.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley Longman Publishing Co.

Pham, D., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., & Zaidi, M. (2005). The Bees Algorithm Technical Note. *Manufacturing Engineering Centre, Cardiff University, UK*, 1–57.

Rayward-Smith, V., Osman, I., Reeves, C., & Simth, G. (1996). *Modern Heuristic Search Methods*.

Robeson, J. F. (1994). *Logistics Handbook*. Simon and Schuster.

Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3 edition). Pearson.

Singh, J. (1996). The importance of information flow within the supply chain. *Logistics Information Management*, *9*(4), 28-30. https://doi.org/10.1108/09576059610123132

Soleimani, H., & Kannan, G. (2015). A hybrid particle swarm optimization and genetic algorithm for closed-loop supply chain network design in large-scale networks. *Applied Mathematical Modelling*, *39*(14), 3990-4012. https://doi.org/10.1016/j.apm.2014.12.016

Spencer, H. (1864). *The Principles of Biology*. William and Norgate.

Tan, K.-C., Kannan, V., & Handfield, R. (1998). Supply Chain Management: Supplier Performance and Firm Performance. *International Journal of Purchasing & Materials Management*, *34*(3). https://digitalcommons.usu.edu/manage_facpub/278

Varma, V. A., Reklaitis, G. V., Blau, G. E., & Pekny, J. F. (2007). Enterprise-wide modeling & optimization—An overview of emerging research challenges and opportunities. *Computers & Chemical Engineering*, *31*(5), 692-711. https://doi.org/10.1016/j.compchemeng.2006.11.007

Weise, T. (2009). *Global Optimization Algorithm: Theory and Application*.

Xiong, F., Gong, P., Jin, P., & Fan, J. F. (2018). Supply chain scheduling optimization based on genetic particle swarm optimization algorithm. *Cluster Computing*. https://doi.org/10.1007/s10586-018-2400-z

Yang, X.-S. (2011). *Computational Optimization, Methods and Algorithms* (Vol. 356). https://doi.org/10.1007/978-3-642-20859-1

Yang, X.-S. (2014). Cuckoo Search and Firefly Algorithm: Overview and Analysis. En X.-S. Yang (Ed.), *Cuckoo Search and Firefly Algorithm: Theory and Applications* (pp. 1-26). Springer International Publishing. https://doi.org/10.1007/978-3-319-02141-6_1

Yeh, W.-C., & Chuang, M.-C. (2011). Using multi-objective genetic algorithm for partner selection in green supply chain problems. *Expert Systems with Applications*, *38*(4), 4244-4253. https://doi.org/10.1016/j.eswa.2010.09.091

Yuce, B., & Mastrocinque, E. (2016). Supply Chain Network Design Using an Enhanced Hybrid Swarm-Based Optimization Algorithm. *Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics*, 95-112. https://doi.org/10.4018/978-1-4666-9644-0.ch003

Plataforma de Optimización en la cadena de suministro y reducción de desperdicio de material

Manual de Usuario

Versión: 1.2

Fecha: 22 de marzo del 2020

HOJA DE CONTROL

Organismo	Universidad de Cuenca		
Proyecto	Plataforma de Optimización en la cadena de suministro, CS y reducción de desperdicio de material		
Entregable	Manual de Usuario		
Autor	Carlos Cevallos		
Versión/Edición	1.2	**Fecha Versión**	22/03/2020
		Nº Total de Páginas	25

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.0	Versión inicial	Carlos Cevallos Tapia	24/02/2020
1.1	Versión corregida por Erick Sigcha	Carlos Cevallos Tapia	04/03/2020
1.2	Versión corregida por Lorena Sigüenza	Carlos Cevallos Tapia	22/03/2020

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos
Carlos Cevallos Tapia

Contenido

Glosario

Cadena de suministro: Es una red compuesta por un conjunto de entidades que pueden ser: proveedores, distribuidores, minoristas, transportistas, producto, entre otros. Esta red cuenta con indicadores que pueden ofrecer información para la toma de decisiones dentro de la empresa; por ejemplo, cómo aumentar el beneficio de la empresa o mejorar la calidad del producto y el servicio al cliente.

Optimización: Es el proceso para encontrar la mejor o muy buena solución a un problema específico. En este proceso se necesita un modelo que puede ser una ecuación y un optimizador que puede ser un algoritmo.

Algoritmo: Secuencia de pasos bien definidos que ayudan a resolver un problema determinado.

Algoritmo de Optimización: Es un algoritmo encargado de encontrar las mejores soluciones de un conjunto con el fin de resolver un problema dado. Existen una gran cantidad de algoritmos, y estos a su vez pueden ser clasificados en diferentes grupos. Algunos de estos algoritmos son: algoritmo genético, GA, algoritmo de optimización de enjambre de partículas, PSO y algoritmo de colonia de hormigas.

Convergencia: Es un valor al que tiende alguna función luego de varias iteraciones.

Tiempo de ejecución: Es el tiempo total que le toma a un programa ejecutarse desde el inicio hasta su fin.

Población: Conjunto de individuos.

Individuo: Es una entidad que pertenece a una población, la cual cambia sus valores durante el entrenamiento o ejecución de los algoritmos de Optimización.

Descripción del Documento

Propósito

Este documento tiene el propósito de brindar información a los usuarios que vayan a usar este programa. Así mismo, este documento sirve como un control de versiones, ya que en un futuro se pueden agregar nuevas funcionalidades. A su vez, este documento ayuda al mantenimiento del software.

Alcance

En el documento presentado a continuación se describirá la funcionalidad del programa, las plataformas en las que trabaja, sus características, y las preguntas más frecuentes que se pueden generar por los usuarios finales del software.

Funcionalidad

Este sistema tiene como propósito el uso de algoritmos de optimización para entrenar modelos y obtener los mejores valores de estos. Con los valores entrenados, en un futuro se puede probar con nuevos datos con el fin de encontrar soluciones a los problemas más eficientemente. Cada uno de estos algoritmos cuenta con parámetros que pueden ser modificados con el fin de obtener mejores valores.

Mapa del Sistema

Navegación

En esta sección se presenta la forma de navegación a través de las diferentes páginas del aplicativo. En la Figura A.1 se puede ver la navegación de cada uno de los problemas que se pueden resolver en la aplicación de algoritmos de optimización. En la sección de Descripción del Sistema se presentan las pantallas de estas páginas, así como la explicación de cada uno de sus elementos y secciones.

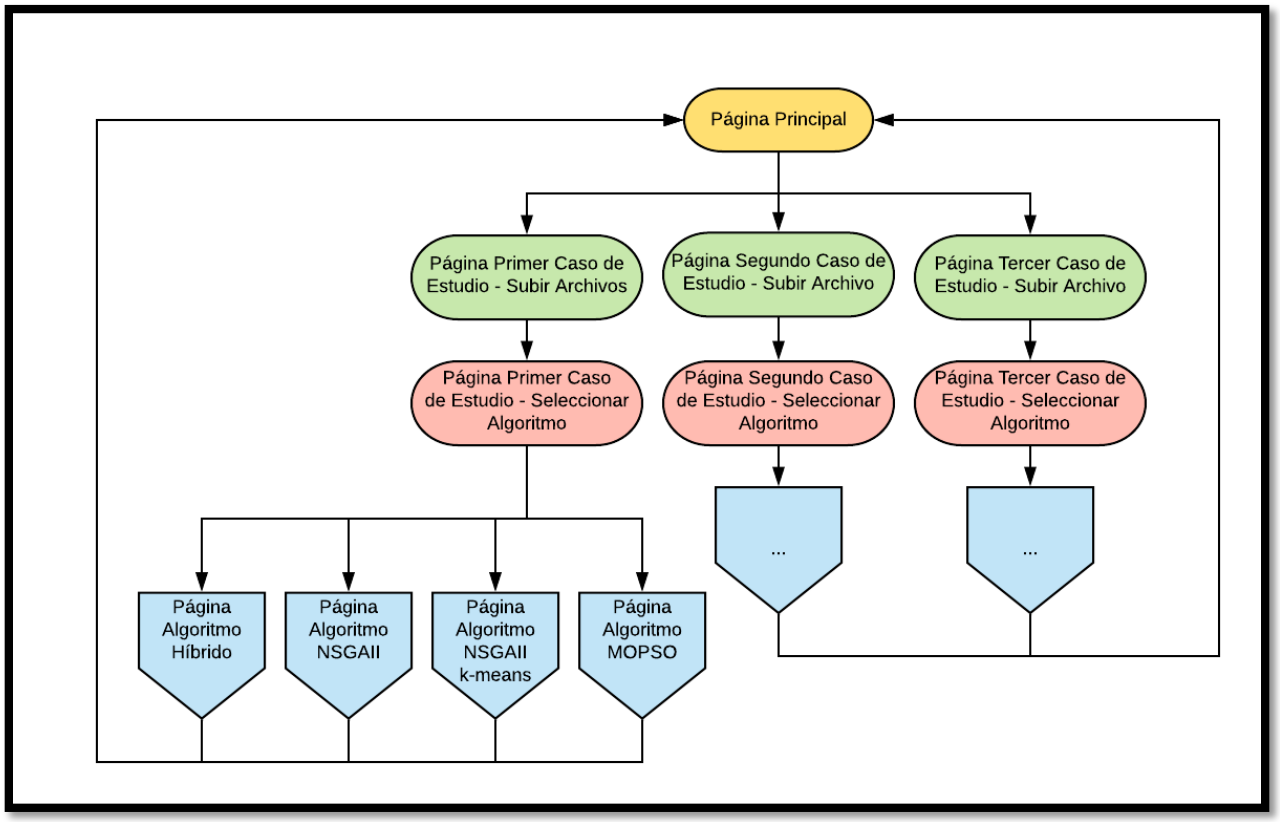

Figura A.2. Navegación del programa.

Página Principal

Esta es un página netamente informativa en la cual se describe un poco sobre la aplicación, esta página es redirigida desde la página principal del proyecto, https://imagineresearch.org.

Página Caso de Estudio – Subir Archivo

Este sistema cuenta con una página por cada uno de los tres casos de estudios que serán redirigidas desde la página principal.

En cada una de estas páginas que corresponden a cada problema a resolver se encuentra el formato de archivos que se puede descargar, y a su vez en esta página se lo puede subir al servidor para poder entrenar los algoritmos. , así como la opción de subir el archivo al servidor para poder entrenar a los diferentes algoritmos diferentes parámetros que el usuario puede ingresar, así como los algoritmos que puede elegir. Ésta a su vez redirige a la ejecución de cada uno de los algoritmos de ejecución programados.

Página Caso de Estudio – Seleccionar Algoritmo

Así mismo cuenta con una página por cada uno de los tres casos de estudio, esta es redirigida por la Página Caso de Estudio – Subir Archivo al haber presionado el botón “Subir Archivo”. En esta página se puede seleccionar el algoritmo y modificar los diferentes parámetros con el que cuente el algoritmo a entrenar. Estas tres páginas se encargan de dirigir hacia las secciones de los algoritmos donde se entrenan y se muestran los resultados.

Página Algoritmo Híbrido

Se encarga de obtener los datos ingresados en la página principal, ejecutar el algoritmo híbrido y mostrar los resultados. Ésta a su vez puede regresar a la Página Caso de Estudio – Subir Archivo.

Página Algoritmo NSGA-II

Se encarga de obtener los datos ingresados en la página principal, ejecutar el algoritmo NSGA-II y mostrar los resultados. Ésta a su vez puede regresar a la Página Caso de Estudio – Subir Archivo.

Página Algoritmo NSGA-II K-means

Se encarga de obtener los datos ingresados en la página principal, ejecutar el algoritmo NSGA-II K-means y mostrar los resultados. Ésta a su vez puede regresar a la Página Caso de Estudio – Subir Archivo.

Página Algoritmo MOPSO

Se encarga de obtener los datos ingresados en la página principal, ejecutar el algoritmo MOPSO y mostrar los resultados. Ésta a su vez puede regresar a la Página Caso de Estudio – Subir Archivo.

Algoritmos de Optimización usados

Algoritmo de enjambre de partículas multi-objetivo

El algoritmo trata de encontrar óptimos locales y uno global de un conjunto de partículas o individuos Estos individuos cambian según su velocidad que es calculada por la ecuación (A.1).

$$v_i = \omega v_i + c1 * r1(p_i - x_i) + c2 * r2(g_i - x_i), donde \qquad (A.1)$$

$w, c1\ y\ c2\ son\ constantes,$

$r1\ y\ r2\ son\ enteros\ aleatorios\ que\ pueden\ tomar\ el\ valor\ de\ 0\ o\ 1,$

$v_i\ es\ la\ velocidad\ de\ la\ particula\ i,$

$x_i\ es\ la\ particula\ i,$

$p_i\ es\ la\ mejor\ particula\ local\ i,$

$g_i\ es\ la\ mejor\ particula\ global\ i$

Pseudocódigo

4. Por cada partícula hacer:
    a. Inicializar la posición de la partícula mediante un vector.
    b. Inicializar la mejor posición de la partícula conocida.
    c. Actualizar el valor de la mejor posición conocida.
    d. Inicializar la velocidad de la partícula.

5. Mientras no se cumpla el criterio de parada (número de iteraciones o variación entre la solución actual y la anterior) repetir:
    a. Por cada partícula hacer:
        i. Por cada dimensión hacer:
            1. Elegir números aleatorios.
            2. Modificar la velocidad de la partícula.
        ii. Modificar la posición de la partícula.
        iii. Si la función objetivo en la posición es menor a la función objetivo en la mejor posición local entonces:
            1. Modificar la mejor posición de la partícula.
                a. Si la función objetivo en la mejor posición local es menor a la función objetivo en la posición global entonces:
                b. Modificar la mejor posición global.
6. Retornar la mejor posición global.

NSGA-II

El algoritmo NSGA-II cuenta con el ordenamiento no dominado y la distancia entre la población. Las operaciones importantes son: selección, cruce y mutación. El cruce es basado en probabilidad; si la probabilidad es menor a 0.5 entonces se toma el gen del primer padre; si la probabilidad esta entre 0.5 y (1/Número de Individuos) se toma el gen del segundo padre; o, caso contrario, el gen realiza el proceso de mutación y toma un valor aleatorio del espacio del problema.

Pseudocódigo

5. Inicializar Población
6. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
7. Realizar el Ordenamiento no dominado
    a. Buscar las distancias entre la población
    b. Seleccionar los mejores individuos que serán los nuevos padres
8. Mientras no se cumpla el criterio de parada, hacer:
    a. Generar los hijos de la población
        i. Recombinación y Mutación
    b. Evaluar las Funciones Objetivo
        i. Dar un peso a cada una de las funciones
    c. Realizar el Ordenamiento no dominado
        i. Buscar las distancias entre la población
        ii. Seleccionar los mejores individuos que serán los nuevos padres

Algoritmo NSGA-II k-means

Es una combinación entre el algoritmo NSGA-II y la teoría de la agrupación por k-means (k grupos o clusters). Lo que el algoritmo trata de hacer es probar los individuos con NSGA-II en un

número fijo de iteraciones y luego de eso, empezar a agrupar los individuos. Por cada uno de estos grupos volver a ejecutar NSGA-II.

Pseudocódigo

7. Inicializar Población
8. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
9. Realizar el Ordenamiento no dominado
    a. Buscar las distancias entre la población
    b. Seleccionar los mejores individuos que serán los nuevos padres
10. Realizar un número específico de operaciones:
    a. Generar los hijos de la población
        i. Recombinación y Mutación
    b. Evaluar las Funciones Objetivo
        i. Dar un peso a cada una de las funciones
    c. Realizar el Ordenamiento no dominado
        i. Buscar las distancias entre la población
        ii. Seleccionar los mejores individuos que serán los nuevos padres
11. Crear 10 grupos y colocar individuos

12. Mientras no se cumpla el criterio de parada:
    a. Por cada grupo, hacer:
        i. Generar los hijos de la población
            1. Recombinación y Mutación
        ii. Evaluar las Funciones Objetivo
            1. Dar un peso a cada una de las funciones
        iii. Realizar el Ordenamiento no dominado
            1. Buscar las distancias entre la población
            2. Seleccionar los mejores individuos que serán los nuevos padres

Algoritmo Híbrido

Este algoritmo es una modificación del algoritmo NSGA-II, que combina características del algoritmo MOPSO y micro-algoritmo con el fin de reducir tiempo de ejecución y obtener una mejor solución.

Pseudocódigo

5. Inicializar Población
6. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
7. Realizar el Ordenamiento no dominado

a. Buscar las distancias entre la población

b. Seleccionar los mejores individuos que serán los nuevos padres

8. Mientras no se cumpla el criterio de parada, hacer:

a. Generar los hijos de la población

i. Generar un sub-espacio

ii. Recombinar

iii. Mutar según ese nuevo sub-espacio

b. Evaluar las Funciones Objetivo de los nuevos hijos

i. Dar un peso a cada una de las funciones

c. Realizar el Ordenamiento no dominado

i. Buscar las distancias entre la población

ii. Seleccionar los mejores individuos que serán los nuevos padres

d. Reducir la población

Descripción del Sistema

Este sistema tiene tres optimizaciones. La primera es la maximización del beneficio y nivel de servicio ofrecido por una CS. La segunda se basa en la maximización de beneficio de un producto en un periodo específico dentro de una CS. Y, la tercera corresponde a minimización de desperdicio producido por el corte de materia prima.

El sistema cuenta con una página principal; tres páginas de caso de estudio para subir los archivos; tres páginas de caso de estudio para seleccionar el algoritmo; diez páginas en donde se entrenan y se muestran los resultados de los algoritmos; cuatro páginas de errores y una página de "Acerca de", en donde se muestra información del sistema. A continuación, se describe cada una de estas páginas y que funcionalidades ofrecen cada uno.

Cabecera

Todas las páginas en este sistema cuentan con la misma cabecera, la cual es presentada en la Figura A.2. Esta cabecera muestra el logo del equipo de investigación, así como varios botones para redirigir a los diferentes tipos de optimizaciones.

1. Botón para redirigir a la primera optimización.
2. Botón para redirigir a la segunda optimización.
3. Botón para redirigir a la tercera optimización.
4. Botón para descargar el Manual de Usuario.
5. Botón para redirigir a la página de "Acerca De".
6. Imagen para redirigir a la página de inicio de IMAGINE.

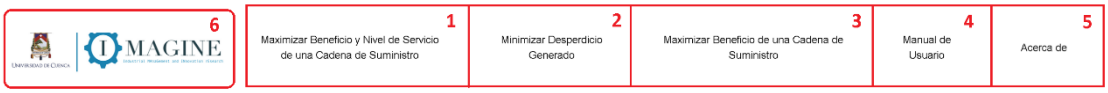

Figura A.2. Cabecera de las páginas.

Página Principal

Es de tipo informativa, y desde aquí se puede usar la cabecera para redirigir a los diferentes problemas.

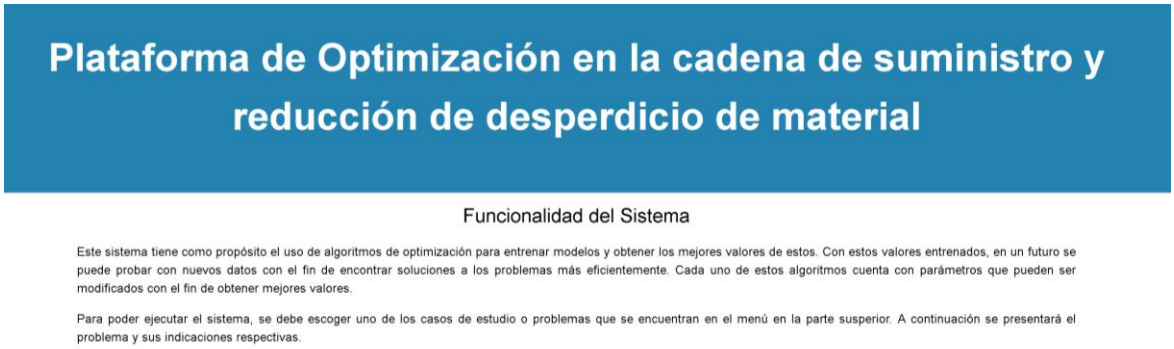

Figura A.3. Página Principal.

Página Caso de Estudio – Subir Archivo

Debido a que existen tres páginas similares, solo se indicará la estructura de ellas. En las Figuras A.4-A.6 se presentan las tres paginas principales.

1. Titulo “Algoritmos de Optimización”.
2. Información preliminar sobre el problema.
3. Gráfica para ayudar a entender las entradas y salidas del algoritmo.

En la Figura 4 se visualiza la estructura de la primera sección de la página.

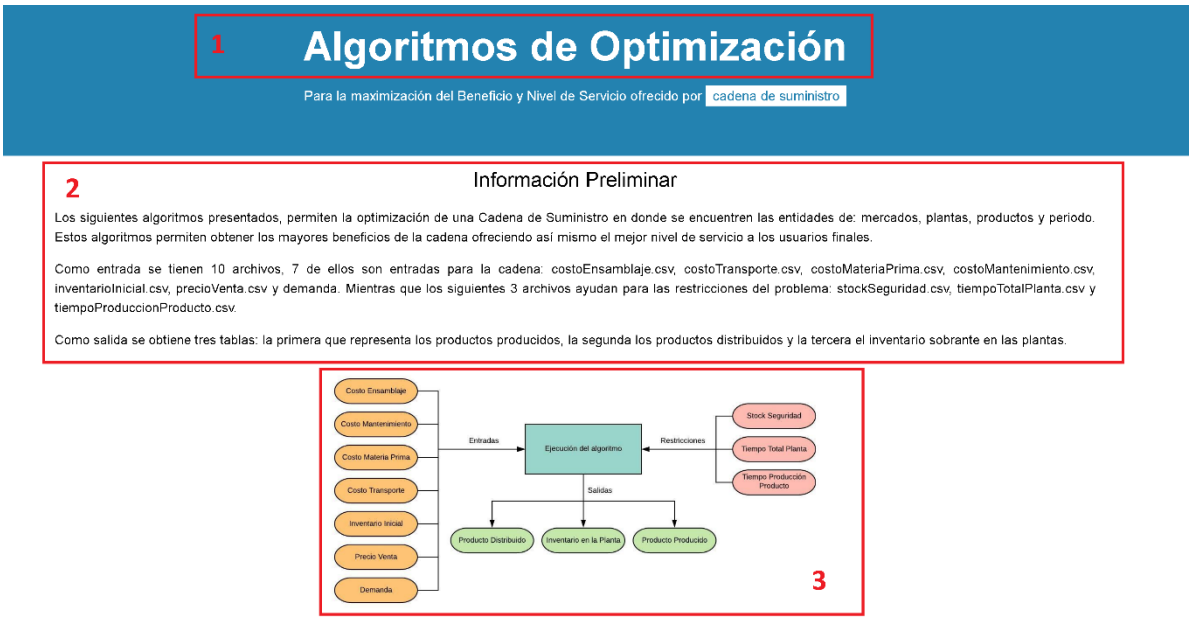

Figura A.4. Primera Sección de la Página Primer Caso de Estudio – Subir Archivo.

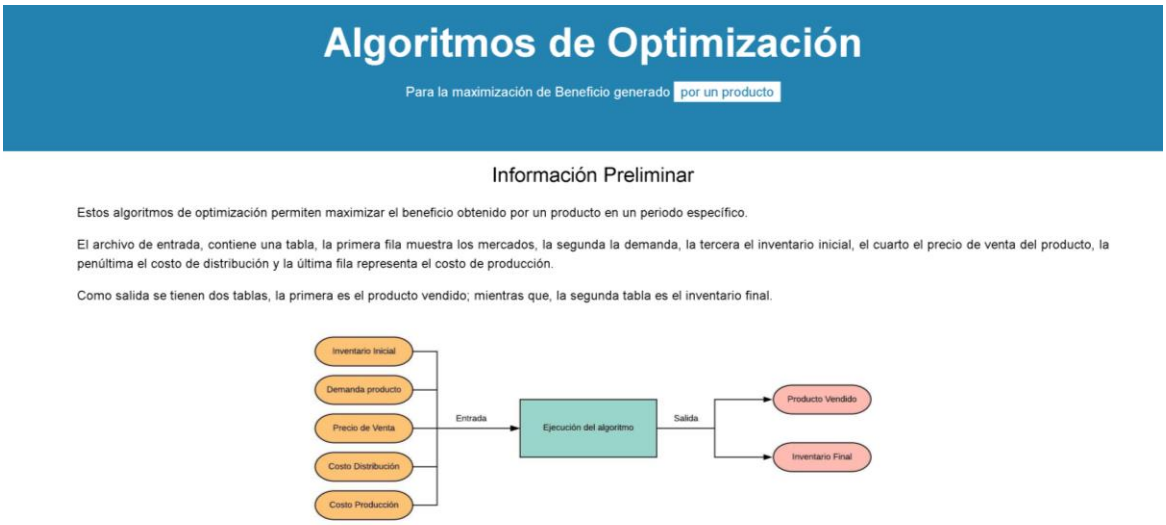

Figura A.5. Primera Sección de la Página Segundo Caso de Estudio – Subir Archivo.

Figura A.6. Primera Sección de la Página Tercer Caso de Estudio – Subir Archivo.

En la segunda sección, mostrada en la Figura A.7, se pueden apreciar:

1. Los pasos para entrenar el algoritmo.
2. El enlace para descargar el formato de los archivos.
3. El botón para seleccionar el archivo de la máquina del cliente.
4. Botón para subir el o los archivos al servidor.

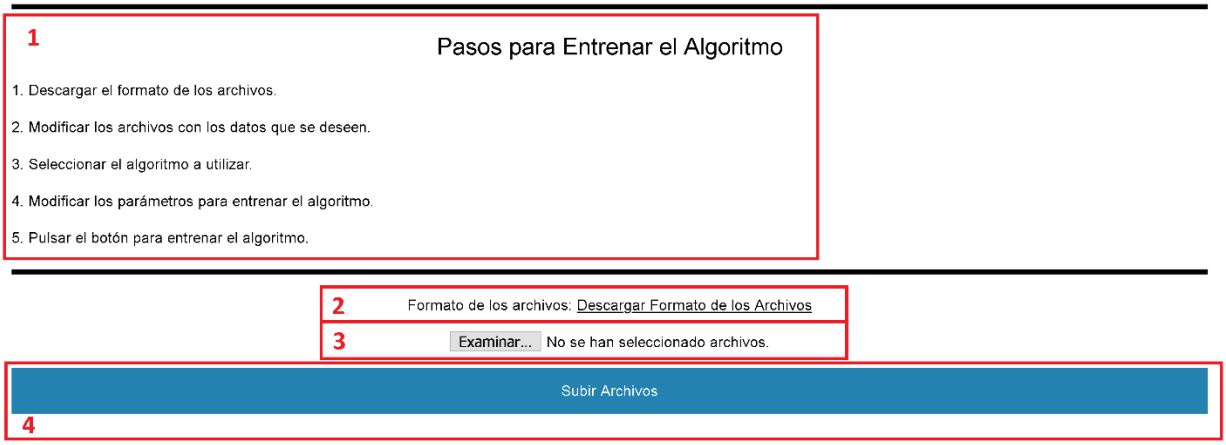

Figura A.7. Segunda Sección de la Página Caso de Estudio – Subir Archivo.

Página Caso de Estudio – Seleccionar Algoritmo

Una vez subido el archivo, se procede a esta pagina, en la primera sección se elije el algoritmo. Luego se tiene diferentes entradas. A continuación se describen estas entradas.

1. Ingresar el número de individuos, se refiere al número individuos que va a tener el algoritmo. Cada individuo tendrá un valor de la función objetivo, y éste cambiará durante la ejecución del algoritmo; mientras más existan, muchas mas opciones tendrá el algoritmo para escoger.
2. Ingresar el número de iteraciones máximas del algoritmo; esto quiere decir que, si no se llega al nivel de servicio deseado, el algoritmo finalizará cuando alcance el número máximo de iteraciones.
3. Ingresar el nivel de servicio mínimo (Solo para la primera optimización), es usado como criterio de parada para obtener un nivel de servicio oportuno para los interesados.
4. Ingresar el valor de la constante 1 (Solo algoritmo MOPSO), es parte de la ecuación para calcular la velocidad.
5. Ingresar el valor de la constante 2 (Solo algoritmo MOPSO), es parte de la ecuación para calcular la velocidad.
6. Ingresar la velocidad inicial (Solo algoritmo MOPSO), es parte de la ecuación para calcular la velocidad, este valor cambia en cada iteración.
7. Ingresar la máxima velocidad (Solo algoritmo MOPSO), este valor permite limitar los valores a los que cambia cada partícula.

Finalmente, en la página se tiene un botón para entrenar el algoritmo, el cual reedigirá a las páginas de los algoritmos a entrenarse. La estructura de la última sección se puede ver en la Figura A.8. Y las entradas se muestran en la Figura A.9.

Figura A.8. Página Caso de Estudio – Seleccionar Algoritmo.

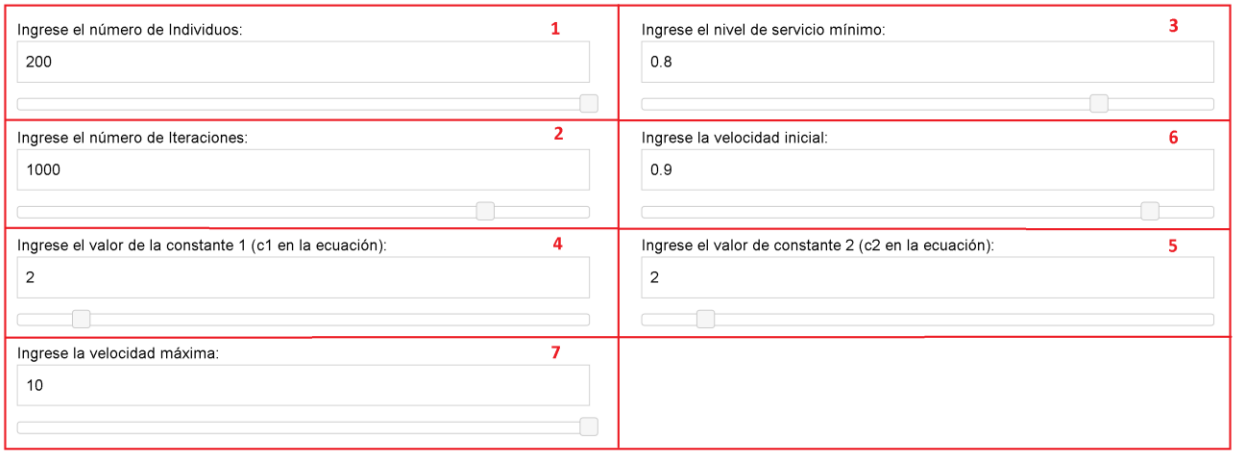

Figura A.9. Entradas de los algoritmos.

Sección de los Algoritmos

En estas páginas se realiza la ejecución de los algoritmos, se hace la llamada de los respectivos métodos, y se devuelven los resultados que son mostrados en texto o por gráficos.

En la primera sección se puede corroborar que el algoritmo se ha entrenado correctamente, y se muestra información importante del algoritmo:

1. Número de iteraciones: Es un valor entero que representa el número de ciclos que le tomó al algoritmo para terminar con su ejecución.
2. Beneficio Obtenido: Es un valor en dólares que representa el beneficio total obtenido por la CS.
3. Nivel de Servicio: Es un porcentaje que representa la satisfacción del cliente.
4. Tiempo de Ejecución: Es un valor en segundos que representa cuánto tiempo tomó la ejecución del algoritmo.

Así mismo, se tiene un botón para regresar a la página principal. Esta sección puede ser visualizada en la Figura A.10.

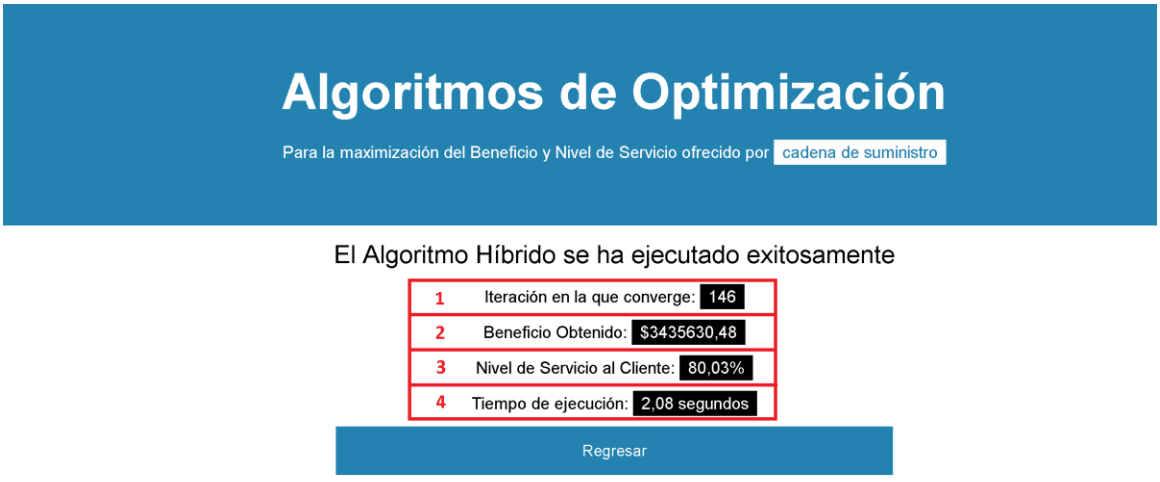

Figura A.10. Primera Sección de la Página del Algoritmo.

En la siguiente sección se muestran gráficas de la convergencia del algoritmo según cada función objetivo; en algunos casos, se tendrá dos gráficas y en otras solo una. Estas gráficas son de puntos. La Figura A.11 presenta un gráfico de convergencia obtenido al ejecutar un ejemplo.

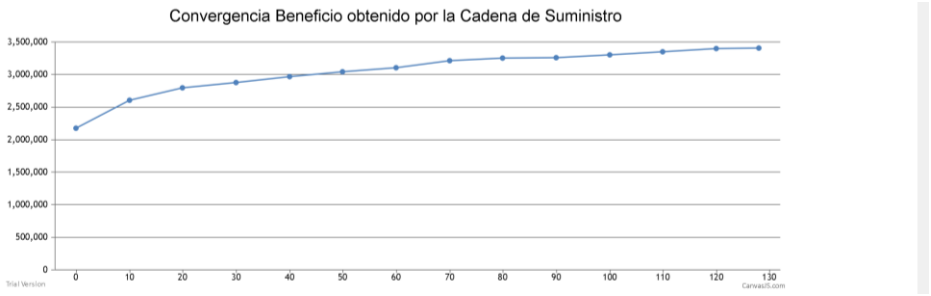

Figura A.11. Convergencia de la función objetivo.

En la Figura A.12 se puede observar la última sección, en donde se muestran:

1. Seleccionar el mejor individuo (solo para la primera optimización). Esta información de los individuos cambia dinámicamente dependiendo de la selección escogida en la lista desplegable.
2. Tablas con las correspondientes salidas del individuo seleccionado.
3. Botón para regresar a la página principal.

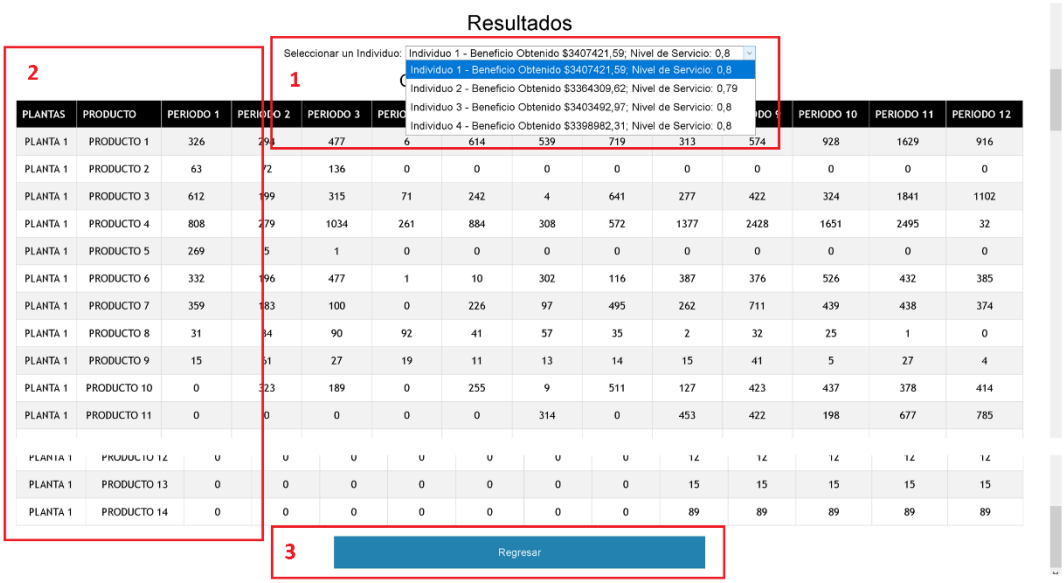

Figura A.12. Resultados del mejor Individuo y botón para regresar a la página principal.

Páginas de Error

Son tres páginas informativas, las cuales son errorLeerDirectorio.jsp, errorLeerArchivos.jsp, errorLeerArchivo.jsp y errorArchivoVacio.jsp. Éstas contienen el nombre del error y una pequeña descripción del mismo (Figuras A.13-A.16).

Figura A.13. Página de "Error al abrir el Directorio".

Figura A.14. Página de "Error al leer los Archivos".

Figura A.15. Página de “Error al leer el Archivo”.

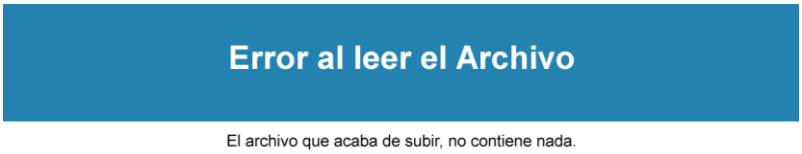

Figura A.16. Página de “Error al leer el Archivo”.

Página Acerca De

Página enfocada a brinda información. Aquí se puede ver la descripción del Sistema, así como un contacto para cualquier problema que ocurra con el mismo. Esto se aprecia de mejor manera en la Figura A.17.

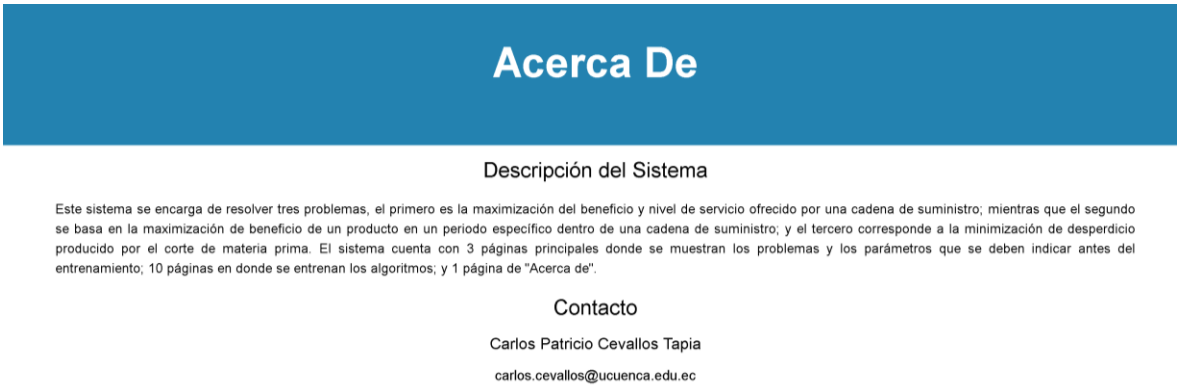

Figura A.17. Página de Acerca De.

Formato de Archivos de los Problemas

Formato Archivos (Primera Optimización – Maximización de Beneficio y Nivel de Servicio de la Cadena de Suministro)

Es una carpeta en donde se encuentran diez archivos de valores separados por coma. Los archivos se presentan en la Figura A.16.

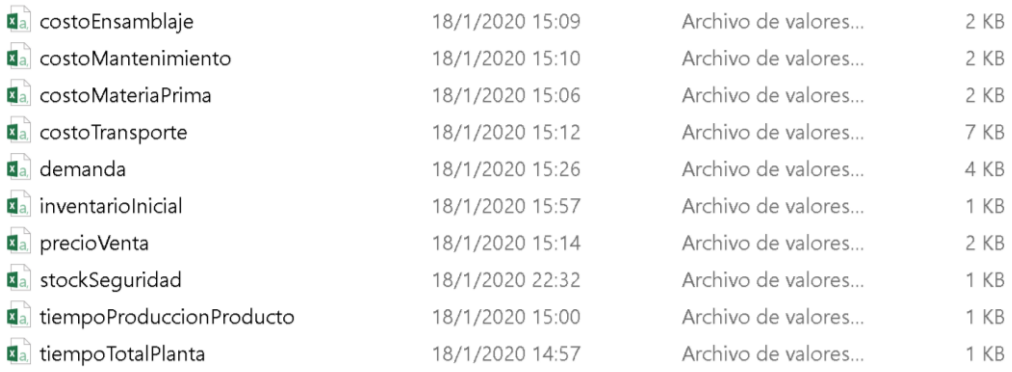

Figura A.18. Archivos contenidos en la carpeta del formato.

Los archivos cuentan con el mismo formato que se explica, a continuación:

- Como Filas se tiene:
    1. Las plantas en donde van a ser producidos los productos.
    2. Los mercados en donde van a ser vendidos los productos.
    3. Los diferentes productos que se van a producir.
- Como Columnas se tiene:
    4. El periodo de producción y distribución de los productos.

El formato de los archivos se puede ver en la Figura A.17.

1	2	3											4
PLANTA	MERCADO	PRODUCTO	ENERO	FEBRERO	MARZO	ABRIL	MAYO	JUNIO	JULIO	AGOSTO	SEPTIEMBRE	OCTUBRE	NOVIEMBRE	DICIEMBRE
PLANTA 1	PICHINCHA	7T1080H	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
PLANTA 1	PICHINCHA	7T1080I	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
PLANTA 1	PICHINCHA	7T1080J	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
PLANTA 1	PICHINCHA	7T10900	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
PLANTA 1	PICHINCHA	7T10901	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
PLANTA 1	PICHINCHA	7T10902	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
PLANTA 1	PICHINCHA	7T10903	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
PLANTA 1	PICHINCHA	7T10904	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00
PLANTA 1	PICHINCHA	7T10905	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00
PLANTA 1	PICHINCHA	7T10906	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
PLANTA 1	PICHINCHA	7T10907	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
PLANTA 1	PICHINCHA	7T10909	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
PLANTA 1	PICHINCHA	7T1090A	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
PLANTA 1	PICHINCHA	7T1090B	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
PLANTA 1	GUAYAS	7T1080H	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
PLANTA 1	GUAYAS	7T1080I	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
PLANTA 1	GUAYAS	7T1080J	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
PLANTA 1	GUAYAS	7T10900	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
PLANTA 1	GUAYAS	7T10901	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00
PLANTA 1	GUAYAS	7T10902	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00

Figura A.19. Estructura del formato del archivo.

Formato Archivos (Segunda Optimización – Maximizar Beneficio de un Producto)

En este caso solo se tiene un archivo el cual cuenta con la siguiente estructura:

- En las filas se tiene
    1. Los mercados a donde se distribuirá el producto.
    2. La demanda de producto en cada mercado.
    3. El inventario inicial del producto en cada mercado.
    4. El precio de venta del producto en cada mercado.
    5. El costo de distribución del producto para cada mercado.
    6. Costo de producción del producto en la planta.

Esta estructura se puede ver en la Figura A.18.

1	MERCADOS	1	2	3	4	5	6	7	8	9	10	11	12	13
2	DEMANDA	0	2	0	4	7	2	1	0	0	2	10	0	2
3	INVENTARIO INICIAL	0	0	0	0	0	0	0	0	0	0	0	0	65
4	PRECIO DE VENTA	0 65.43	64.41	62.65	118.23	63.02	116.27	141.43	65.43	135.3	112.07	60.69	131.32	
5	COSTO DE DISTRIBUCION	0	0	0	0	3	0	3	3	0	3	3	0	0
6	COSTO DE PRODUCCION 59.15													

Figura A.20. Estructura del formato del archivo

Formato Archivos (Tercera Optimización – Minimización de Desperdicio)

En este caso sólo se tiene un archivo el cual cuenta con la siguiente estructura:

- En las columnas se tiene:
    1. El número de patrones de corte para dicha materia prima.
    2. La longitud de la materia prima a ser cortada.
    3. Los valores de los patrones para cortar X mm de materia prima.
    4. El desperdicio que se genera cuando se utiliza el patrón.
- En las filas se tiene
    5. La longitud que se va a cortar en mm.
    6. La demanda de piezas de longitud X que se deben cortar como mínimo.

Esta estructura se puede ver en la Figura A.19.

	1 ⇩	2 ⇩	3 ⇩						4 ⇩
5 ⇨	K	Longitud	1206	1612	2165	2925	3481	3820 Total	Desperdicio
6 ⇨		9MMB0HP	5	366	289	50	65	36 811	
	1	5850	4	0	0	0	0	0 0	1026
	2	5850	1	1	0	0	0	0 0	3032
	3	5850	1	0	1	0	0	0 0	2479
	4	5850	2	0	0	1	0	0 0	513
	5	5850	1	1	1	0	0	0 36	867
	6	5850	1	1	0	1	0	0 0	107
	7	5850	1	0	0	0	1	0 0	1163
	8	5850	1	0	0	0	0	1 7	824
	9	5850	0	2	0	0	0	0 131	2626
	10	5850	0	1	1	0	0	0 12	2073
	11	5850	0	1	0	1	0	0 0	1313
	12	5850	0	1	0	0	1	0 26	757
	13	5850	0	1	0	0	0	1 31	418
	14	5850	0	0	1	0	0	0 62	3685
	15	5850	0	0	1	1	0	0 42	760
	16	5850	0	0	1	0	1	0 137	204
	17	5850	0	0	0	1	0	0 7	2925
	18	5850	0	0	0	0	1	0 0	2369
	19	5850	0	0	0	0	0	1 0	2030

Figura A.21. Estructura del formato del archivo.

Pasos para el entrenamiento del algoritmo

1. Seleccionar desde la página principal el caso de estudio que se desea realizar (Existen tres tipos de optimizaciones).
    a. Maximización de Beneficio y Nivel de Servicio de una CS.
    b. Minimización de Desperdicio de materia prima.
    c. Maximización de Beneficio de un producto en una CS.
2. Leer las instrucciones de la optimización a realizar.
3. Descargar el formato de los archivos.
4. Modificar el formato de los archivos.
5. Seleccionar y Subir el archivo.
6. Seleccionar el algoritmo para el entrenamiento.
    a. Híbrido.
    b. NSGA-II.
    c. MOPSO.
    d. NSGA-II K-means.
7. Cambiar los parámetros para el algoritmo.
    a. En la primera optimización se puede elegir un mínimo de nivel de servicio.
8. Dar click en el botón “Entrenar Algoritmo”.
9. Se abrirá una nueva página en donde aparecerán los resultados. En la parte que dice “Resultados” se podrá cambiar el individuo para ver diferentes resultados (solo para la primera optimización).

Preguntas Frecuentes

1. **¿Se puede ingresar otro formato de archivos para los diferentes problemas?**

   No, solo los archivos de valores separados por coma y con el mismo formato de entrada.

2. **¿Qué pasa si los datos que ingreso no son correctos?**

   El sistema, enviará un mensaje de error, diciendo que no se puede entrenar el algoritmo, porque los datos no son correctos.

3. **¿Los problemas funcionan para todos los casos de CS o de reducción de desperdicio?**

   Si, si se tiene todos los datos que se necesita poner en el formato.

4. **¿Estos algoritmos pueden ser utilizados para resolver otros problemas?**

   Este programa está diseñado para resolver tres problemas, pero se podrían resolver otros problemas, cambiando la lógica del algoritmo.

Plataforma de Optimización en la cadena de suministro y reducción de desperdicio de material

Manual Técnico

Versión: 1.2

Fecha: 22 de marzo del 2020

HOJA DE CONTROL

Organismo	Universidad de Cuenca		
Proyecto	Plataforma de Optimización en la cadena de suministros, CS y reducción de desperdicio de material		
Entregable	Manual de Usuario		
Autor	Carlos Cevallos		
Versión/Edición	1.2	**Fecha Versión**	22/03/2020
		Nº Total de Páginas	35

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.0	Versión inicial	Carlos Cevallos Tapia	24/02/2020
1.1	Versión corregida por Erick Sigcha	Carlos Cevallos Tapia	04/03/2020
1.2	Versión corregida por Lorena Sigüenza	Carlos Cevallos Tapia	22/03/2020

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos
Carlos Cevallos Tapia

Contenido

Objetivos

Objetivo General

Ofrecer a los interesados de este documento información acerca de la aplicación creada para la optimización de tres diferentes problemas.

Objetivos Específicos

- Describir las herramientas usadas para la construcción de la aplicación.
- Indicar la arquitectura del sistema.
- Mostrar la funcionalidad del aplicativo.

Introducción

En este documento se abordan los aspectos técnicos del aplicativo. Aquí se presenta la metodología de desarrollo usada, la arquitectura del sistema, los diagramas de clase, casos de uso, pseudocódigos de los algoritmos usados y el código fuente.

Metodología de Desarrollo usada

Para el desarrollo de este trabajo se ha utilizado la metodología ágil. La metodología ágil permite analizar y mejorar el aplicativo o software mediante su desarrollo. Este proceso se enfoca más en las personas que en los procesos. La documentación no es tan primordial, mas importantes es que el aplicativo se encuentre en funcionamiento. Cuando se realice un cambio, la metodología debe responder inmediatamente. Más información: https://luis-goncalves.com/es/que-es-la-metodologia-agil/.

Herramientas usadas para el desarrollo

HTML

Es un lenguaje que trabaja con marcas o etiquetas, el cual es utilizado para el desarrollo de páginas web. Con este lenguaje se puede mostrar a los usuarios mucha información de una manera muy fácil, y si se desea que se vea más llamativa HTML usa CSS (Hojas de Estilo en español) para cambiar el estilo de las páginas. Para este aplicativo ha sido usado HTML 5. Más información: https://html.spec.whatwg.org/multipage/.

Java

Java es un lenguaje de programación comercializada en 1995 por Sun Microsystems. Java es muy utilizado para el desarrollo de todo tipo de aplicaciones, tanto de escritorio como web. Java es uno de los lenguajes más rápidos, seguros y confiables. Java 7 ha sido usado durante el desarrollo de este aplicativo. Más información: https://www.java.com/es/.

JSP

JSP (JavaServer Pages) es un lenguaje de programación del lado servidor que permite la creación de páginas web dinámicas basadas en HTML y XML (Lenguaje de Marcado Extensible). Este lenguaje usa JAVA, así que básicamente su notación, palabras claves y semántica es muy similar. Para poder utilizar este lenguaje es necesario tener un servidor Web. Más información: https://es.wikipedia.org/wiki/JavaServer_Pages.

Javascript

Es un lenguaje de programación orientado a objetos. Se le denomina JS. Es un lenguaje imperativo, débilmente tipado, basado en prototipos y dinámico. Es usado en la creación de

aplicativos webs, del lado del cliente. Tiene una sintaxis muy similar a C. Todos los navegadores pueden interpretar el código de JavaScript. Más información: https://www.javascript.com/.

Librerías usadas

- jquery.canvasjs.min.js: Ayuda a crear la gráfica de puntos para mostrar la convergencia de los algoritmos.

Apache GlassFish

Es un servidor de aplicaciones web desarrollado también por Sun Microsystems. Permite ejecutar aplicaciones desarrolladas con Java EE. Al igual que Java es gratuita, de código libre. Aunque, también cuenta con una versión comercial conocida como "Oracle GlassFish Enterprise Server". La versión usada para el desarrollo del aplicativo es la 4.1. Más información: https://javaee.github.io/glassfish/download.

Casos de Estudio

Este sistema cuenta con tres casos de estudio:

1. Maximización del beneficio y nivel de servicio ofrecido por una CS.
2. Maximización de beneficio de un producto en un periodo específico dentro de una cadena de suministro
3. Minimización de desperdicio producido por el corte de materia prima.

Algoritmos de Optimización usados

Algoritmo de enjambre de partículas multi-objetivo

El algoritmo trata de encontrar óptimos locales y uno global de un conjunto de partículas o individuos Estos individuos cambian según su velocidad que es calculada por la ecuación (B.1).

$$v_i = \omega v_i + c1 * r1(p_i - x_i) + c2 * r2(g_i - x_i), donde \qquad (B.1)$$

$w, c1\ y\ c2\ son\ constantes,$

$r1\ y\ r2\ son\ enteros\ aleatorios\ que\ pueden\ tomar\ el\ valor\ de\ 0\ o\ 1,$

$v_i\ es\ la\ velocidad\ de\ la\ particula\ i,$

$x_i\ es\ la\ particula\ i,$

$p_i\ es\ la\ mejor\ particula\ local\ i,$

$g_i\ es\ la\ mejor\ particula\ global\ i$

Pseudocódigo

7. Por cada partícula hacer:
    a. Inicializar la posición de la partícula mediante un vector.
    b. Inicializar la mejor posición de la partícula conocida.
    c. Actualizar el valor de la mejor posición conocida.
    d. Inicializar la velocidad de la partícula.

8. Mientras no se cumpla el criterio de parada (número de iteraciones o variación entre la solución actual y la anterior) repetir:
    a. Por cada partícula hacer:
        i. Por cada dimensión hacer:
            1. Elegir números aleatorios.
            2. Modificar la velocidad de la partícula.
        ii. Modificar la posición de la partícula.
        iii. Si la función objetivo en la posición es menor a la función objetivo en la mejor posición local entonces:
            1. Modificar la mejor posición de la partícula.
                a. Si la función objetivo en la mejor posición local es menor a la función objetivo en la posición global entonces:
                b. Modificar la mejor posición global.
9. Retornar la mejor posición global.

NSGA-II

El algoritmo NSGA-II cuenta con el ordenamiento no dominado y la distancia entre la población. Las operaciones importantes son: selección, cruce y mutación. El cruce es basado en probabilidad; si la probabilidad es menor a 0.5 entonces se toma el gen del primer padre; si la probabilidad esta entre 0.5 y (1/Número de Individuos) se toma el gen del segundo padre; o, caso contrario, el gen realiza el proceso de mutación y toma un valor aleatorio del espacio del problema.

Pseudocódigo

9. Inicializar Población
10. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
11. Realizar el Ordenamiento no dominado
    a. Buscar las distancias entre la población
    b. Seleccionar los mejores individuos que serán los nuevos padres
12. Mientras no se cumpla el criterio de parada, hacer:
    a. Generar los hijos de la población
        i. Recombinación y Mutación
    b. Evaluar las Funciones Objetivo
        i. Dar un peso a cada una de las funciones
    c. Realizar el Ordenamiento no dominado
        i. Buscar las distancias entre la población
        ii. Seleccionar los mejores individuos que serán los nuevos padres

Algoritmo NSGA-II k-means

Es una combinación entre el algoritmo NSGA-II y la teoría de la agrupación por k-means (k grupos o clusters). Lo que el algoritmo trata de hacer es probar los individuos con NSGA-II en un

número fijo de iteraciones y luego de eso, empezar a agrupar los individuos. Por cada uno de estos grupos volver a ejecutar NSGA-II.

Pseudocódigo

13. Inicializar Población
14. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
15. Realizar el Ordenamiento no dominado
    a. Buscar las distancias entre la población
    b. Seleccionar los mejores individuos que serán los nuevos padres
16. Realizar un número específico de operaciones:
    a. Generar los hijos de la población
        i. Recombinación y Mutación
    b. Evaluar las Funciones Objetivo
        i. Dar un peso a cada una de las funciones
    c. Realizar el Ordenamiento no dominado
        i. Buscar las distancias entre la población
        ii. Seleccionar los mejores individuos que serán los nuevos padres
17. Crear 10 grupos y colocar individuos

18. Mientras no se cumpla el criterio de parada:
    a. Por cada grupo, hacer:
        i. Generar los hijos de la población
            1. Recombinación y Mutación
        ii. Evaluar las Funciones Objetivo
            1. Dar un peso a cada una de las funciones
        iii. Realizar el Ordenamiento no dominado
            1. Buscar las distancias entre la población
            2. Seleccionar los mejores individuos que serán los nuevos padres

Algoritmo Híbrido

Este algoritmo es una modificación del algoritmo NSGA-II, que combina características del algoritmo MOPSO y micro-algoritmo con el fin de reducir tiempo de ejecución y obtener una mejor solución.

Pseudocódigo

9. Inicializar Población
10. Evaluar las Funciones Objetivo
    a. Dar un peso a cada una de las funciones
11. Realizar el Ordenamiento no dominado

a. Buscar las distancias entre la población

b. Seleccionar los mejores individuos que serán los nuevos padres

12. Mientras no se cumpla el criterio de parada, hacer:

a. Generar los hijos de la población

i. Generar un sub-espacio

ii. Recombinar

iii. Mutar según ese nuevo sub-espacio

b. Evaluar las Funciones Objetivo de los nuevos hijos

i. Dar un peso a cada una de las funciones

c. Realizar el Ordenamiento no dominado

i. Buscar las distancias entre la población

ii. Seleccionar los mejores individuos que serán los nuevos padres

d. Reducir la población

Modelo Lógico

El aplicativo muestra una página web desarrollada con HTML y con JSP. Estas páginas cuentan con varias entradas de datos. Luego se tiene la lectura de los datos. Esta lectura se hace con JSP, algunos datos se leen desde el HTML, y otros son leídos desde una carpeta donde se encuentran los archivos de valores separados por comas. Luego de eso, los datos pasan a Java donde se llama a los métodos para ejecutar los algoritmos de optimización. Finalmente, el resultado del algoritmo es devuelto a JSP, donde se muestran al usuario en una página HTML. El aplicativo ha sido colocado en un servidor GlassFish, que cuenta con el siguiente dominio http://cidi.ingenieria.ucuenca.edu.ec/optimizacionsc/. En la Figura B.1 se presenta este modelo.

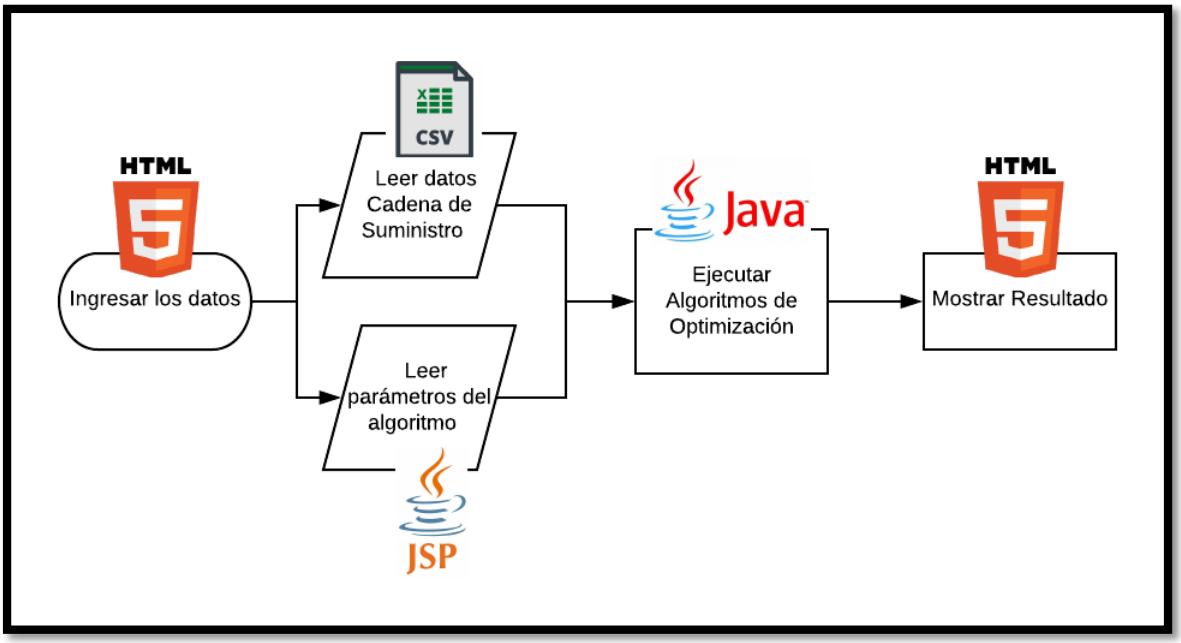

Figura B.3. Modelo de la Aplicación.

Arquitectura del Sistema

En la Figura B.2 se presenta la arquitectura del sistema. El usuario se conecta a una computadora o dispositivo móvil para ingresar al aplicativo. Después de eso, el usuario ingresa los parámetros necesarios para entrenar el algoritmo. Estos parámetros son enviados a través de la nube hasta llegar al servidor. El servidor se encarga de entrenar el algoritmo y devuelve resultados que pasan nuevamente a través de la nube. Finalmente, estos resultados son presentados en el lado del cliente.

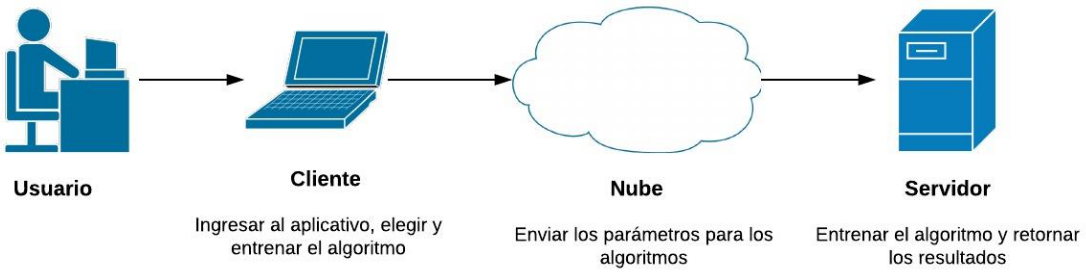

Figura B.2. Arquitectura del Sistema.

Casos de Uso

En esta sección se describen los casos de uso que fueron utilizados durante el desarrollo de las funcionalidades de la aplicación. Han sido desarrollados cuatro casos de estudio por cada entrenamiento de los algoritmos. Los casos de uso son presentados en la Figura B.3 y son descritos en las Tablas B.1-B.5.

Participantes

Usuario Final: Es el participante que se encarga de usar la aplicación, es el único actor en este proceso. A continuación, se presentan sus casos de estudio.

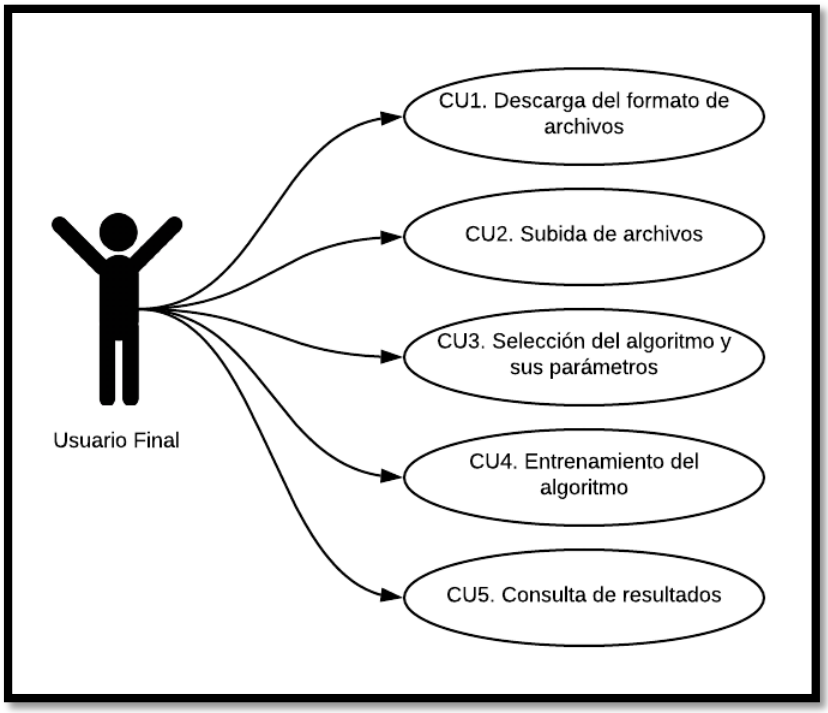

Figura B.3. Casos de Uso del Usuario Final.

Tabla B.1. Caso de Estudio – Descarga del formato de archivos.

Nombre	CU1. Descarga del formato de archivos
Actor:	Usuario Final
Descripción:	El usuario descarga el formato de archivos para poder luego subir y ejecutar los algoritmos.
Precondiciones:	Ninguna.
Flujo Normal:	1. El usuario ingresa a la página principal. 2. El usuario presiona el enlace para descargar. 3. El usuario debe tener el archivo descargado.
Post condición:	1. Archivo descargado.

Tabla B.2. Caso de Estudio – Subida de archivos.

Nombre	CU2. Subida de archivos
Actor:	Usuario Final
Descripción:	El usuario sube los archivos necesarios para el entrenamiento del algoritmo.
Precondiciones:	1. Archivo descargado
Flujo Normal:	1. El selecciona el o los archivos necesarios. 2. El usuario presiona el botón “Subir Archivo”
Post condición:	1. Archivo/s subidos al servidor.

Tabla B.3. Caso de Estudio – Selección del algoritmo y sus parámetros.

Nombre	CU3. Selección del algoritmo y sus parámetros
Actor:	Usuario Final
Descripción:	El usuario selecciona el algoritmo y modifica los parámetros a su conveniencia.
Precondiciones:	1. Archivo/s subidos al servidor.
Flujo Normal:	1. El usuario selecciona el algoritmo. 2. El usuario modifica los parámetros del algoritmo.
Post condición:	1. Algoritmo seleccionado.

Tabla B.4. Caso de Estudio – Entrenamiento del algoritmo.

Nombre	CU4. Entrenamiento del algoritmo
Actor:	Usuario Final
Descripción:	El usuario presiona el botón para entrenar el algoritmo deseado.

Precondiciones:	1. Algoritmo seleccionado.
Flujo Normal:	1. El usuario presiona el botón para entrenar el algoritmo.
Post condición:	1. Algoritmo entrenado.

Tabla B.5. Caso de Estudio – Consulta de resultados.

Nombre	CU5. Consulta de resultados
Actor:	Usuario Final
Descripción:	El usuario revisa los resultados devueltos por el algoritmo.
Precondiciones:	1. Algoritmo entrenado.
Flujo Normal:	1. El usuario revisa los resultados devueltos por el algoritmo seleccionado.
Post condición:	

Instalación y configuración

Para poder abrir el proyecto se necesitan seguir los siguientes pasos.

1. Descargar JDK. Página de Descarga: https://www.oracle.com/technetwork/es/java/javase/downloads/index.html.
2. Descargar Netbeans 8.0.2 (Versión All). Página de Descarga: https://netbeans.org/downloads/8.0.2/.
3. Instalar Netbeans.
4. Abrir el Proyecto.

Para ejecutar el proyecto:

1. Click derecho sobre el proyecto y dar click en “Clean and Build”.
2. Click derecho sobre el proyecto y dar click en “Run”.

Árbol del Proyecto

Web Pages

Aquí se presentan todas las páginas web usadas para el sistema. Así como las imágenes que aparecerán en las distintas páginas. También se presentan los archivos de formato que serán descargados por los usuarios y el manual de usuario. Este árbol se muestra en la Figura B.4.

Figura B.4. Árbol del Proyecto.

Source Packages

Aquí se presentan todas las Clases y métodos usados para la ejecución del sistema. Se tienen tres paquetes. Los tres primeros representan los algoritmos de cada caso de estudio. El cuarto paquete presenta los archivos Handlers que se comunican con las páginas web. Y finalmente, el

último paquete almacena los resultados obtenidos después del entrenamiento de los algoritmos. Estos archivos se muestran en la Figura B.5.

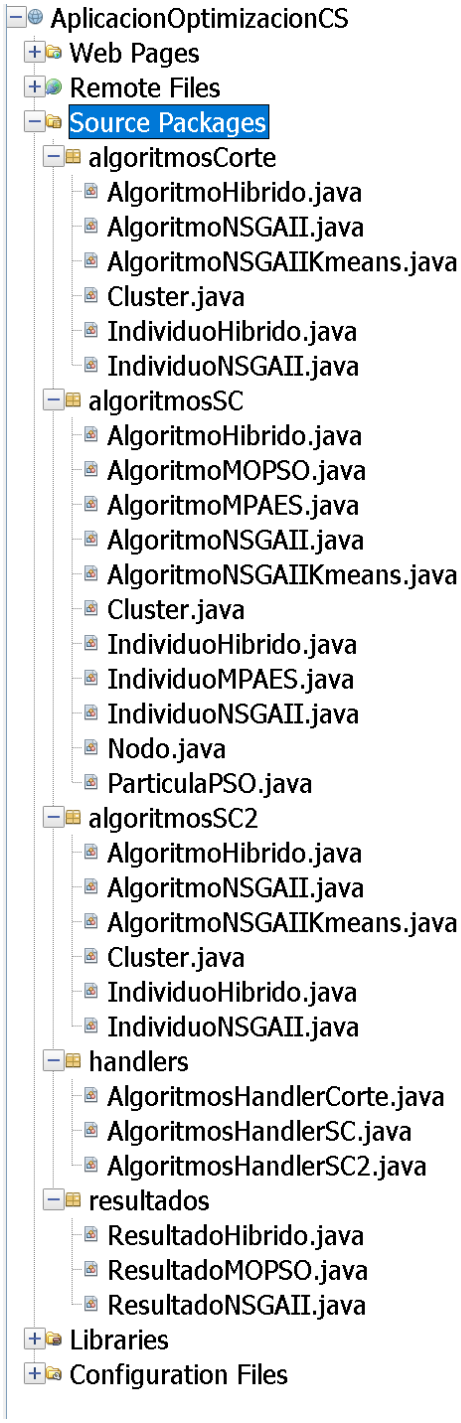

Figura B.5. Source Packages del Proyecto.

Páginas Web

En la siguiente sección se analizará el código de las diferentes páginas web que se encuentran en el aplicativo. Se revisará el código Javascript así como jsp.

Página Principal

Solo es una página, index.jsp, es solo informativa.

Página Caso de Estudio – Subir Archivos

Estas son tres páginas: problemasSC.jsp, problemaSC2.jsp y problemaCorte.jsp. En esta página solo se tiene el formato del archivo que puede ser descargado, asi como un botón para subir los archivos desde la máquina del cliente.

Página Caso de Estudio – Seleccionar Algoritmo

Estas son tres páginas: seleccionarAlgoritmoSC.jsp, seleccionarAlgoritmoSC2.jsp y seleccionarAlgoritmoCorte.jsp.

Métodos Javascript

window.onload: Se encarga de que cuando la ventana se cargue solo se muestre un algoritmo para entrenar, mientras los otros se esconden para que exista más espacio en la página.

```
window.onload = function() {


    document.getElementById("formNSGAII").style.visibility = "hidden";
    document.getElementById("formNSGAIIK").style.visibility = "hidden";
    document.getElementById("formMOPSO").style.visibility = "hidden";

}
```

mostrarAlgoritmo(valor): Se encarga de mostrar un algoritmo especifico, y de ocultar los otros que no vayan a ser utilizados.

```
function mostrarAlgoritmo(valor) {

    if(valor === "Algoritmo Híbrido"){

        if(document.getElementById("formHib").style.visibility === "hidden"){

            document.getElementById("formNSGAII").style.visibility = "hidden";
            document.getElementById("formNSGAIIK").style.visibility = "hidden";
            document.getElementById("formMOPSO").style.visibility = "hidden";
            document.getElementById("formHib").style.visibility = "visible";

        }
```

slider: Se encarga de crear un deslizador para elegir valores y limitar al usuario.

```
$( function() {
$( "#slider1-1" ).slider({
min : 30,
max: 500,
step : 5,
value : 200,
change: function( event, ui ) {
$('#slide1-1').val(ui.value);
}
});
});
```

Métodos JSP

En esta sección se sube el archivo al servidor.

```
            File file ;
   int maxFileSize = 5000 * 1024;
   int maxMemSize = 5000 * 1024;
   ServletContext context = pageContext.getServletContext();

   String filePath = System.getProperty("user.dir") + "/";


   // Verify the content type
   String contentType = request.getContentType();

   if ((contentType.indexOf("multipart/form-data") >= 0)) {
      DiskFileItemFactory factory = new DiskFileItemFactory();
      // maximum size that will be stored in memory
      factory.setSizeThreshold(maxMemSize);

      // Location to save data that is larger than maxMemSize.
      factory.setRepository(new File("C:/Users/carlo/Downloads"));

      // Create a new file upload handler
      ServletFileUpload upload = new ServletFileUpload(factory);

      // maximum file size to be uploaded.
      upload.setSizeMax( maxFileSize );

      try {
         // Parse the request to get file items.
         List<FileItem> fileItems =  upload.parseRequest(request);


         fileItems.sort(new Comparator<FileItem>() {

    @Override

    public int compare(FileItem f1, FileItem f2) {

        return f1.getName().compareTo(f2.getName());

     }

});
```

```
        List<String> nombresArchivos = new ArrayList<String>();
        int cont = 0;
        nombresArchivos.add("costoEnsamblaje.csv");
        nombresArchivos.add("costoMantenimiento.csv");
        nombresArchivos.add("costoMateriaPrima.csv");
        nombresArchivos.add("costoTransporte.csv");
        nombresArchivos.add("demanda.csv");
        nombresArchivos.add("inventarioInicial.csv");
        nombresArchivos.add("precioVenta.csv");
        nombresArchivos.add("stockSeguridad.csv");
        nombresArchivos.add("tiempoProduccionProducto.csv");
        nombresArchivos.add("tiempoTotalPlanta.csv");
        for(int i = 0; i < nombresArchivos.size(); i++){
            File f = new File(filePath + nombresArchivos.get(i));
            if(f.exists()){
                f.delete();
            }
        }
        Iterator<FileItem> i = fileItems.iterator();
        while ( i.hasNext () ) {
           FileItem fi = i.next();
           if ( !fi.isFormField () ) {
              // Get the uploaded file parameters
              String fieldName = fi.getFieldName();
              String fileName = nombresArchivos.get(cont);
              boolean isInMemory = fi.isInMemory();
              long sizeInBytes = fi.getSize();
              // Write the file
              if( fileName.lastIndexOf("/") >= 0 ) {
                 file = new File( filePath +
                 fileName.substring( fileName.lastIndexOf("/"))) ;
              } else {
                 file = new File( filePath +
                 fileName.substring(fileName.lastIndexOf("/")+1)) ;
              }
              fi.write( file ) ;
               cont +=1;
           }
        }
    } catch(Exception ex) {
    }
} else {
}
```

Página de Entrenamiento del Algoritmo

En estas páginas se realiza el entrenamiento de varios algoritmos, existen diez archivos jsp.

Métodos JavaScript

window.onload: Ayuda a mostrar las gráficas de puntos de la convergencia de los algoritmos de optimización.

```
window.onload = function() {

    var chart = new CanvasJS.Chart("chartContainer", {
            theme: "light1", // "light2", "dark1", "dark2"
            animationEnabled: true,
            zoomEnabled: true,
            title: {
                    text: ""
            },
            data: [{
                    type: "line",
                    dataPoints: <%=algoritmoBean.getRh().getDatos1()%>
            }]
    });
    chart.render();

    var chart2 = new CanvasJS.Chart("chartContainer2", {
            theme: "light1", // "light2", "dark1", "dark2"
            animationEnabled: true,
            zoomEnabled: true,
            title: {
                    text: ""
            },
            data: [{
                    type: "line",
                    dataPoints: <%=algoritmoBean.getRh().getDatos2()%>
            }]
    });
    chart2.render();
```

cambiarDatos(valor): Ayuda a cambiar los valores de las tablas dinámicamente dependiendo del mejor individuo que haya sido seleccionado.

```
function cambiarDatos(valor) {


    var table = document.getElementById("tabla");
    var table2 = document.getElementById("tabla2");
    var table3 = document.getElementById("tabla3");

    if(valor == 0){

        <%
        int b = 0; int j = 0; int k =0;
        for(int i = 0; i < algoritmoBean.getNumProveedores() * algoritmoBean.getNumProductos(); i ++){

            if(b==algoritmoBean.getNumProductos()){

                b = 0;
                j++;

            }
            for(int z = 0; z < algoritmoBean.getPeriodo(); z++){

               %>
                        table.rows[<%=i+1%>].cells[<%=z+2%>].innerHTML = "<%=algoritmoBean.getRh().getIndivid
                                                                        <%

            }
            b++;
```

JSP

Se encarga de crear un Bean de petición para comunicarse con la clase Handler, y mapear los parámetros con las variables de la clase Handler.

```
<jsp:useBean id="algoritmoBean" scope="request" class="handlers.AlgoritmosHandlerSC" />
<jsp:setProperty name="algoritmoBean" property="directorio"/>
<jsp:setProperty name="algoritmoBean" property="nivelServicio" />
<jsp:setProperty name="algoritmoBean" property="numIndividuos" />
<jsp:setProperty name="algoritmoBean" property="numIteraciones" />
```

Métodos JSP

En este apartado se verifica si existe el directorio y si los archivos se encuentran dentro de él, caso contrario, se redirigirá a una página donde se muestra el error. Los métodos “obtenerDatos()”, “leerArchivos()” y “ejecutarAlgoritmo()” se encuentran explicados más adelante en la sección de Clases y métodos en Handler.java.

```
String resultado = algoritmoBean.obtenerDatos();
if(resultado.equalsIgnoreCase("Error")){

    response.sendRedirect("errorLeerDirectorio.jsp");
    return;

}
resultado = algoritmoBean.leerArchivos();
if(resultado.equalsIgnoreCase("Error al leer los archivos")){

    response.sendRedirect("errorLeerArchivos.jsp");
    return;

}

algoritmoBean.ejecutarAlgoritmoHibrido();
```

JSP es usado también para mostrar los datos en tablas.

```
    <th>MERCADO</th>
    <th>PRODUCTO</th>
    <%b = 0; j = 0; k = 0;
    for(int i = 0; i < algoritmoBean.getPeriodo(); i ++){

        %><th>PERIODO <%=i+1%></th><%}%></tr>
        <%for(int i = 0; i < algoritmoBean.getNumProveedores() * algoritmoBean.getNumProductos() * algoritmoBean.getNumMercados(); i ++){
            %><tr>
            <%if(b==algoritmoBean.getNumProductos()){

                b = 0;
                k++;

            }
            if(k==algoritmoBean.getNumMercados()){

                k = 0;
                j++;

            }%>
            <td>PLANTA <%=j+1%></td>
            <td>MERCADO <%=k+1%></td>
            <td>PRODUCTO <%=b+1%></td>
            <%for(int z = 0; z < algoritmoBean.getPeriodo(); z++){


            %><td><%=algoritmoBean.getRh().getIndividuos().get(0).pdzjkb.get(z).get(j).get(b).get(k)%></td><%}%>
        <% b++;}%></tr>

</table>
<h3>Cantidad de Inventario Sobrante en las Plantas</h3>
<table id="tabla3">
```

Página Acerca De y Errores

Son páginas informativas por lo que no es necesario indicar ningún método o funcionalidad extra.

Clases y métodos

Para este apartado se explicará las diferentes clases de manera general ya que existen tres problemas en donde se usan los algoritmos. Cada uno de estos problemas cuenta con las mismas clases. En la Figura B.6 se presenta el diagrama de clases del aplicativo.

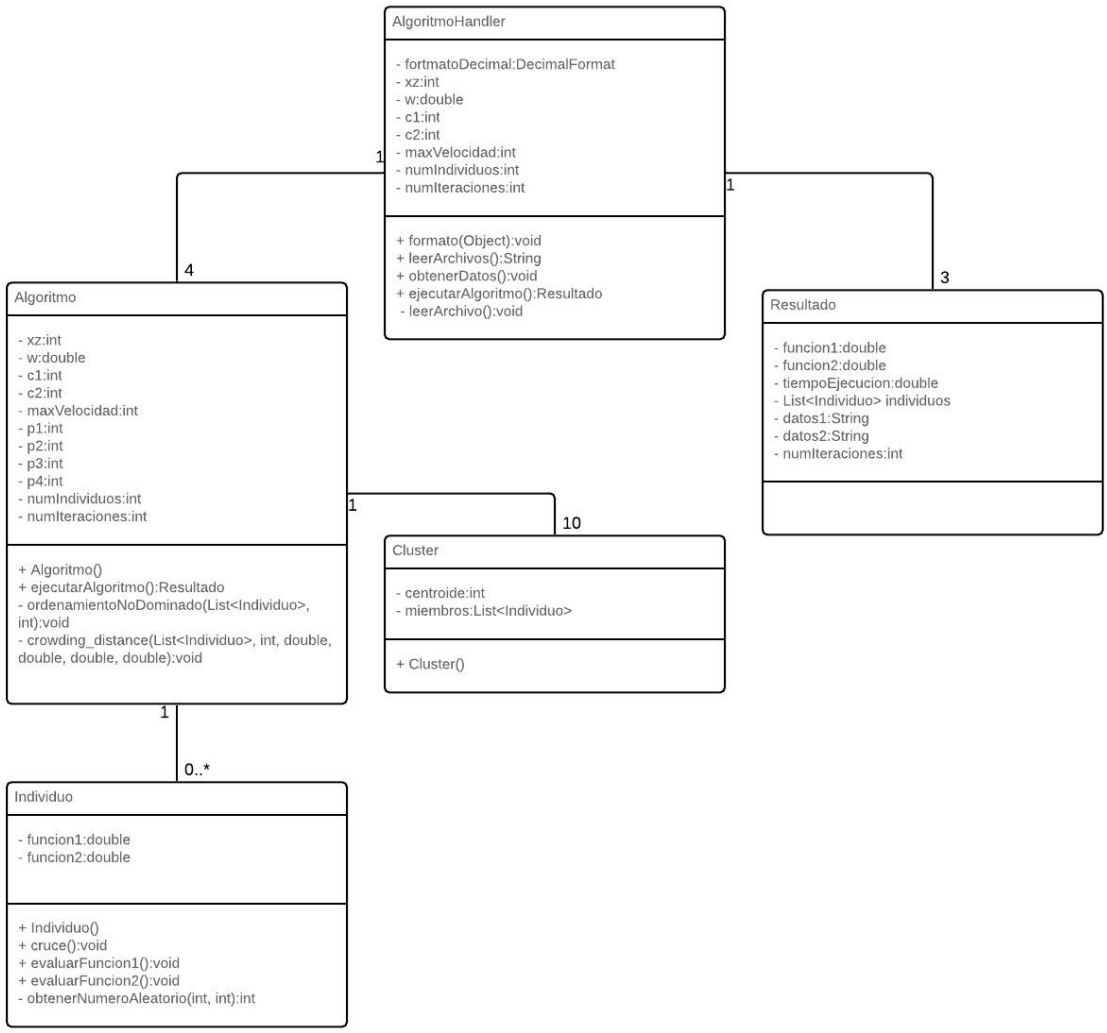

Figura B.6. Diagrama de Clases del sistema.

AlgoritmoHandler.java

Es la clase principal, la cual se comunica directamente con JSP. En ésta se encuentra la llamada a los algoritmos de optimización y la lectura de los archivos y parámetros ingresados por el usuario.

Atributos

- Las variables que se obtienen después de la lectura del formato de los archivos.
- Las variables que contienen los parámetros ingresados por el usuario desde la página web: numIndividuos, numIteraciones, nivelServicio, directorio, w, c1, c2, maxVelocidad.
- Las clases de los algoritmos a ejecutar: AlgoritmoHibrido, AlgoritmoNSGAII, AlgoritmoMOPSO y AlgoritmoNSGAIIKmeans.
- Los resultados devueltos por los algoritmos: ResultadoHibrido, ResultadoNSGAII, ResultadoMOPSO.
- Variable para mostrar los números con solo dos decimales.

```
private List<List<Double>> ptjb;
private List<List<List<Double>>> cmzjb;
private List<List<List<Double>>> cezjb;
private List<List<List<Double>>> czjb;
private List<List<List<Integer>>> sszjb;
private List<List<Double>> pvp;
private List<List<List<Integer>>> dzkb;
private List<List<List<List<Double>>>> ctzjkb;
private List<List<Double>> ttzj;
private List<Integer> ib0;
private int numProductos;
private int numMercados;
private int numProveedores;
private int numIndividuos;
private int numIteraciones;
private int periodo;
private double nivelServicio;
private String directorio;
private int xz;
private double w;
private int c1;
private int c2;
private int maxVelocidad;
private AlgoritmoHibrido ah;
private AlgoritmoNSGAII ns;
private AlgoritmoMOPSO mpso;
private AlgoritmoNSGAIIKmeans nsk;
private ResultadoHibrido rh;
private ResultadoNSGAII rns;
private ResultadoMOPSO rm;
private DecimalFormat formatoDecimal = new DecimalFormat("#.##");
```

Nota: Todas estas variables cuentan con sus respectivos métodos get() y set().

Métodos

void formato(Object valor): Se encarga de modificar un valor double y mostrarlo solo con dos decimales.

```
public String formato(Object valor){

    return formatoDecimal.format(valor);

}
```

String leerArchivos(): Se encarga de leer todos los archivos que se encuentren en el directorio ingresado por el usuario.

```
public String leerArchivos(){

    try {

        leerTiempoProducto(getDirectorio() + "\\tiempoProduccionProducto");
        leerDemanda(getDirectorio() + "\\demanda");
        leerCostoEnsamblaje(getDirectorio() + "\\costoEnsamblaje");
        leerCostoMateriaPrima(getDirectorio() + "\\costoMateriaPrima");
        leerCostoMantenimiento(getDirectorio() + "\\costoMantenimiento");
        leerCostoTransporte(getDirectorio() + "\\costoTransporte");
        leerPrecioVenta(getDirectorio() + "\\precioVenta");
        leerTiempoTotalPlanta(getDirectorio() + "\\tiempoTotalPlanta");
        leerInventarioInicial(getDirectorio() + "\\inventarioInicial");
        leerStockSeguridad(getDirectorio() + "\\stockSeguridad");
        return "Archivos leidos exitosamente";


    } catch (Exception e) {

        return "Error al leer los archivos";

    }

}
```

String obtenerDatos(): Se encarga de leer el archivo y almacenarlo en las variables correspondientes.

```
public String obtenerDatos(){

    try {
        BufferedReader br = null;
        br =new BufferedReader(new FileReader(getDirectorio() + "\\costoTransporte.csv"));
        List<String> nombrePlantas = new LinkedList<>();
        List<String> nombreMercados = new LinkedList<>();
        List<String> nombreProductos = new LinkedList<>();
        String line = br.readLine();
        if(null!=line){

            String [] fields = line.split(";");
            periodo = fields.length - 3;

        }
        line = br.readLine();
        while (null!=line) {
```

Resultado ejecutarAlgoritmo(): Son métodos que ayudan a ejecutar los algoritmos, recuperar el resultado y almacenarlo en su respectiva variable.

```
public void ejecutarAlgoritmoHibrido(){

    setAh(new AlgoritmoHibrido(ptjb, cmzjb, cezjb, czjb, pvp, dzkb, ctzjkb, ttzj, sszjb, ib0, numProductos,
    setRh(getAh().ejecutarAlgoritmo());

}
```

Nota: En el código se encuentran cuatro de estos métodos.

void leerArchivo(): Son métodos que ayudan a leer un archivo específico con un formato ya predefinido, dependiente de los valores ingresados por el usuario.

```
private void leerDemanda(String nombreArchivo) throws IOException{

    BufferedReader br = null;
    setDzkb(new ArrayList<>());
    int z, k, b;
    for(z = 0; z < getPeriodo(); z++){
        getDzkb().add(new ArrayList<>());
        for(k = 0; k < getNumMercados(); k++){
            getDzkb().get(z).add(new ArrayList<>());
            for(b = 0; b < getNumProductos(); b++){
                getDzkb().get(z).get(k).add(0);
            }
        }
    }
    z = 0;
    k = 0;
    b = 0;
    br =new BufferedReader(new FileReader(nombreArchivo+".csv"));
    String line = br.readLine();
    line = br.readLine();
    while (null!=line) {

       String [] fields = line.split(";");
       if(b == getNumProductos()){

          b=0;
          k++;

       }
       for(z = 0; z < getPeriodo(); z++){

            getDzkb().get(z).get(k).set(b, Integer.parseInt(fields[z+2]));
       }
       b++;
       line = br.readLine();
    }
    br.close();

}
```

Nota: En el código se encuentran nueve de estos métodos.

Algoritmo.java

Para el siguiente apartado, se hará una generalización de cuatro clases, ya que éstas se comportan igual. Estas clases son: AlgoritmoNSGAII.java, AlgoritmoHibrido.java, AlgoritmoNSGAIIKmeans.java y AlgoritmoMOPSO.java.

Atributos

Estos atributos son similares a los presentados en la clase AlgoritmoHandler, solo que algunos no se encuentran; por otra parte, también se tiene p1, p2, p3 y p4 que son la posición de los valores de los mejores individuos en cada iteración.

```
private List<List<Double>> ptjb;
private List<List<List<Double>>> cmzjb;
private List<List<List<Double>>> cezjb;
private List<List<List<Double>>> czjb;
private List<List<List<Integer>>> sszjb;
private List<List<Double>> pvp;
private List<List<List<Integer>>> dzkb;
private List<List<List<List<Double>>>> ctzjkb;
private List<List<Double>> ttzj;
private List<Integer> ib0;
private int numProductos;
private int numMercados;
private int numProveedores;
private int numIndividuos;
private int numIteraciones;
private int periodo;
private double nivelServicio;
private int xz;
private int p1;
private int p2;
private int p3;
private int p4;
```

Nota: El AlgoritmoMOPSO.java cuenta con otros atributos que son c1, c2, 2 y maxVelocidad.

Métodos

Algoritmo(): Aquí se tiene el método para la construcción de la clase, en donde se asignan valores a las variables.

```
public AlgoritmoHibrido(List<List<Double>> ptjb, List<List<List<Double>>> cmzjb,
    this.ptjb = ptjb;
    this.cmzjb = cmzjb;
    this.cezjb = cezjb;
    this.czjb = czjb;
    this.pvp = pvp;
    this.dzkb = dzkb;
    this.ctzjkb = ctzjkb;
    this.ttzj = ttzj;
    this.sszjb = sszjb;
    this.ib0 = ib0;
    this.numProductos = numProductos;
    this.numMercados = numMercados;
    this.numProveedores = numProveedores;
    this.numIndividuos = numIndividuos;
    this.numIteraciones = numIteraciones;
    this.nivelServicio = nivelServicio;
    this.periodo = periodo;
    this.xz = numIndividuos;

}
```

Resultado ejecutarAlgoritmo(): Este método sirve para ejecutar el algoritmo como tal, al final se retorna el resultado con los datos más importantes.

```
public ResultadoHibrido ejecutarAlgoritmo(){

    ResultadoHibrido rh = new ResultadoHibrido();
    long time = new Date().getTime();
    Gson gsonObj = new Gson();
    Map<Object,Object> map;
    List<Map<Object,Object>> list = new ArrayList<>();
    Map<Object,Object> map2;
    List<Map<Object,Object>> list2 = new ArrayList<>();
    int generation = 0;
    List<IndividuoHibrido> individuos = new ArrayList<>();
    int pop = numIndividuos;
    for(int i =0; i < numIndividuos; i++){

        IndividuoHibrido ind = new IndividuoHibrido(periodo, numProveedores, numProductos, numMercados, ttzj, dzkb, ptjb, sszjb, ib0);
        ind.evaluarFuncion1(periodo, numProveedores, numProductos, numMercados, cmzjb, cezjb, czjb, ctzjkb, dzkb, pvp);
        ind.evaluarFuncion2(periodo, numProveedores, numProductos, numMercados, dzkb);
        individuos.add(ind);

    }
```

void ordenamientoNoDominado(List<Individuo>, int): Este método es usado por el algoritmo híbrido, NSGA-II y NSGA-II K-means para obtener los cuatro mejores individuos.

```
private void ordenamientoNoDominado(List<IndividuoHibrido> ind, int pob){

    List<Nodo> f1 = new ArrayList<>();
    List<Nodo> f2 = new ArrayList<>();
    double maxf1 = 0;
    double maxf2 = 0;
    double minf1 = 999999999;
    double minf2 = 999999999;
    List<Integer> np = new ArrayList<>();
    for(int i =0; i< pob; i++){

        np.add(0);
        if(ind.get(i).funcion1 > maxf1){

            maxf1 = ind.get(i).funcion1;

        }
        if(ind.get(i).funcion2 > maxf2){

            maxf2 = ind.get(i).funcion2;

        }
        if(ind.get(i).funcion1 < minf1){

            minf1 = ind.get(i).funcion1;

        }
        if(ind.get(i).funcion2 < minf2){

            minf2 = ind.get(i).funcion2;

        }

    }
```

void crowding_distance(List<Individuo>, int, double, double, double, double): Este método es llamado por el ordenamientoNoDominado(), para calcular la distancia de población entre los Individuos y encontrar los mejores.

```
private void crowding_distance(List<IndividuoHibrido> ind, List<Nodo> f, int dev, double maxf1, double maxf2, double minf1, double minf2){

    Nodo aux;
    int padre2 = 0;
    int padre3 = 0;
    for(int i = 0; i < f.size() ; i++){

        f.get(0).distancia= 0.0;

    }
    for(int i = 1; i < f.size(); i++){

        for(int j = 0; j < f.size() - i; j++){

            if(ind.get(f.get(j).valor).funcion1 > ind.get(f.get(j+1).valor).funcion1 ){

                aux = f.get(j+1);
                f.set(j+1, f.get(j));
                f.set(j, aux);

            }

        }

    }
```

Individuo.java

Para el siguiente apartado, se hará una generalización de tres clases, ya que éstas se comportan igual. Estas clases son: IndividuoNSGAII.java, IndividoHibrido.java, y ParticulaPSO.java.

Atributos

- Se tienen las variables a optimizar.
- Se tiene el valor de las funciones objetivo.

```
public List<List<List<Integer>>> pzjb;
public List<List<List<Integer>>> izjb;
public List<List<List<List<Integer>>>> pdzjkb;
double funcion2;
double funcion1;
double padre1_sel=250;
double padre2_sel=500;
double padre3_sel=750;
double padre4_sel;
```

Atributos ParticulaPSO.java

- Este algoritmo cuenta con los mejores valores de las variables a optimizar.

- Se tienen los mejores valores de las funciones objetivo.
- Se tiene la velocidad de las variables que van a cambiar.

```
public List<List<List<Integer>>> pzjb;
public List<List<List<Integer>>> izjb;
public List<List<List<List<Integer>>>> pdzjkb;
public List<List<List<Integer>>> bpzjb;
public List<List<List<Integer>>> bizjb;
public List<List<List<List<Integer>>>> bpdzjkb;
List<List<List<List<Double>>>> vpdzjkb;
double funcion2;
double funcion1;
double mejorf1;
double mejorf2;
```

Métodos

Individuo(): Sirve para inicializar las variables del algoritmo.

```
public IndividuoHibrido(int periodo, int numProveedores, int numProductos, int numMercados, List<List<Double>> ttzj,

    pdzjkb = new ArrayList<>();
    pzjb = new ArrayList<>();
    izjb = new ArrayList<>();
    double tf;
    int  sumpdzjkb;
    for(int z=0; z<periodo; z++){

        pzjb.add(new ArrayList<>());
        izjb.add(new ArrayList<>());
        pdzjkb.add(new ArrayList<>());
        for(int j=0; j<numProveedores; j++){

            do{

                tf = 0.0;
                pzjb.get(z).add(new ArrayList<>());
                izjb.get(z).add(new ArrayList<>());
                pdzjkb.get(z).add(new ArrayList<>());
                for(int b=0; b<numProductos; b++){
```

void cruce(): Realizar el proceso de cruce y mutación de los Individuos. Este método es usado por el algoritmo híbrido, NSGA-II y NSG-II K-means.

```
public void cruce(int periodo, int numProveedores, int numProductos, int numMercados, IndividuoHibrido ind1,

    double tf;
    int  sumpdzjkb, max, min;
    double pobd = pob;
    padre4_sel = (1 - 1/pobd)*1000;
    for(int z=0; z<periodo; z++){

        for(int j=0; j<numProveedores; j++){

            do{

                tf = 0.0;
                for(int b=0; b<numProductos; b++){

                    sumpdzjkb = 0;
                    for(int k=0; k<numMercados; k++){
```

void evaluarFuncion1(): Este método sirve para encontrar el valor de la primera función objetivo.

```
public void evaluarFuncion1(int periodo, int numProveedores, int numProductos, int numMercados, List<List<List<Double>>> cmzjb,

    double costo_total = 0.0;
    double ingresos = 0.0;
    for(int z=0; z<periodo; z++){

        for(int j=0; j<numProveedores; j++){

            for(int b=0; b<numProductos; b++){

                costo_total += pzjb.get(z).get(j).get(b)*cmzjb.get(z).get(j).get(b);
                for(int k=0; k<numMercados; k++){

                    costo_total += pdzjkb.get(z).get(j).get(b).get(k)*ctzjkb.get(z).get(j).get(k).get(b);

                }
                costo_total += pzjb.get(z).get(j).get(b)*cezjb.get(z).get(j).get(b);
                costo_total += izjb.get(z).get(j).get(b)*czjb.get(z).get(j).get(b);

            }


        }

    }
```

```
public void evaluarFuncion1(List<Double> desperdicio, int numPatrones){

    funcion1 = 0.0;
    for(int i = 0; i < numPatrones; i++){

        funcion1 += soluciones.get(i)*desperdicio.get(i);

    }
}
```

```
public void evaluarFuncion1(List<Double> pvp, List<Integer> inventarioIni, Double cp, List<Double> cd, int numMercados){

    funcion1 = 0.0;
    double vb = 0;
    double cti = 0;
    double sum = 0;
    double ctp = 0;
    double ctd = 0;
    for(int i = 0; i < numMercados; i ++){

        vb += pvp.get(i)*vend.get(i);
        sum += vend.get(i);
        cti += (Math.abs(invF.get(i)- inventarioIni.get(i)) * 0.5);
        ctd += cd.get(i)*dist.get(i);

    }
    ctp = sum*cp;
    cti = cti * cp * 30 * (0.17/360);
    funcion1 = vb - ctp - cti - ctd;

}
```

void evaluarFuncion2(): Este método sirve para encontrar el valor de la segunda función objetivo.

```
public void evaluarFuncion2(int periodo, int numProveedores, int numProductos, int numMercados, List<List<List<Integer>>> dzkb){

    double cantidad = 0.0;
    double demanda = 0.0;
    for(int z=0; z<periodo; z++){

        for(int j=0; j<numProveedores; j++){

            for(int b=0; b<numProductos; b++){

                for(int k=0; k<numMercados; k++){

                    cantidad += pdzjkb.get(z).get(j).get(b).get(k);
                    demanda += dzkb.get(z).get(k).get(b);

                }

            }


        }

    }
    this.funcion2 = cantidad/demanda;

}
```

int obtenerNumeroAleatorio(int, int): Devuelve un número aleatorio entero entre un mínimo y máximo.

```
private int obtenerNumeroAleatorio(int min, int max){

    Random random = new Random();
    return random.ints(min,(max+1)).findFirst().getAsInt();

}
```

Resultado.java

Esta clase se encarga de recopilar los datos de los mejores individuos y los valores informativos después de la ejecución de los algoritmos como: tiempo de ejecución y número de iteraciones.

Atributos

- Se tiene el valor final de las funciones objetivo.
- Se tiene en número de Iteraciones.
- Se tiene el tiempo de ejecución.
- Se tiene una lista con los mejores Individuos.
- Se tiene los datos para ser graficados.

```
private double funcion1;
private double funcion2;
private int numIteraciones;
private List<IndividuoHibrido> individuos;
private String datos1;
private String datos2;
private double tiempoEjecucion;
```

Métodos

Resultado(): Sirve para inicializar la clase.

```
public ResultadoHibrido() {
    this.individuos = new ArrayList<>();
}
```

Cluster.java

Esta clase es utilizada por AlgoritmoNSGAIIKmeans.java.

Atributos

- Se tiene el valor del centroide de dicho Clúster.
- Se tiene una lista de los miembros del Clúster.

```
int centroide;
List<IndividuoNSGAII> miembros;
```

Métodos

Cluster(): Sirve para inicializar la clase Cluster.

```
public Cluster() {
    this.centroide = 0;
    this.miembros = new ArrayList<>();
}
```

Apéndice C: Publicaciones derivadas del trabajo de titulación

Cevallos, C., Peña, M., & Siguenza-Guzman, L. (2021). New hybrid algorithm for supply chain optimization. Aceptado para ser publicado en la International Conference Innovation in Engineering – ICIE'2020, Guimarães, Portugal.

Cevallos, C., Siguenza-Guzman, L., & Peña, M. (2019). A hybrid algorithm for supply chain optimization of assembly companies. 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI), 1-6. https://doi.org/10.1109/LA-CCI47412.2019.9037050