PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

ALGORITMOS GENÉTICOS APLICADOS AL DISEÑO DE REDES DE DISTRIBUCIÓN CONSIDERANDO DEMANDA ESTOCÁSTICA

JESSICA MARIANELLA PIZARRO DÍAZ

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO
PROFESIONAL DE INGENIERO
CIVIL INFORMÁTICO

ABRIL 2009

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

ACTA DE APROBACIÓN

JESSICA MARIANELLA PIZARRO DIAZ

_____

GUILLERMO CABRERA

PROFESOR GUÍA

ABRIL 2009

Dedico este libro a mis padres, por todo lo que me han entregado, ya que sin su apoyo constante nada de lo que he hecho hubiese sido posible de lograr.

A mi padre, que ha sido mi amigo y ejemplo a seguir, ya que me ha entregado todos sus valores y enseñanzas para seguir por la vida en un camino correcto.

A mi madre, que me ha entregado su amor y preocupación cada día que pasa, para que pueda siempre tener fe en lo que me propongo.

A mi hermana que ha sido mi gran amiga, soporte y mi gran compañera durante todo mi camino.

A mi hermano que han sido un apoyo y pilar en todo lo que he emprendido, motivándome a siempre seguir con mis proyectos.

A mi sobrina que me entrega toda la alegría y felicidad para vivir esta vida llena de nuevos desafíos.

Ha significado un gran esfuerzo toda la investigación y desarrollo de esta tesis, pero gracias al apoyo de algunas personas que estuvieron durante todo este proceso, no fue tan difícil haber logrado finalizarla.
Primero que todo, quiero agradecer a mis padres, hermanos y sobrina por el amor incondicional que me han brindado.
De manera especial, debo agradecer a mi profesor guía, quien ha sido un gran apoyo en el desarrollo de este trabajo y por toda la ayuda que me ha brindado.
Este proyecto de tesis fue financiado y desarrollado íntegramente en el marco del proyecto FONDECYT N°1060945

ÍNDICE

LISTA DE ABREVIATURAS O SIGLAS

AG: Algoritmos Genéticos.

DND: Distribution Network Design.

DNDRP: Distribution Network Design with Risk Pooling.

TS: Tabú Search.

LS: Local Search.

SA: Simulated Annealing.

CD: Centro de Distribución.

CFLP: Capacitated Facilities Location Problem.

NP: Non Deterministic Polinomial Time.

GRASP: Greedy Randomized Adaptative Search Procedures.

MSPP: Median Shortest Path Problem.

PMMC: P-Median Problem with Mutual Communication.

DSDP: Distribution System Design Problem.

EOQ: Economic Order Quantity.

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

RESUMEN

Este trabajo de título se basa en diseñar una red de distribución, que abarque la toma de decisiones estratégicas, lo cual consiste en determinar la cantidad necesaria de instalaciones y donde localizarlas para proveer a los distintos clientes, de la forma más eficiente posible. Se tomó como base el modelo de localización Distribution Network Design with Risk Pooling (DNDRP), donde el problema apoya las decisiones estratégicas con las tácticas, es decir, considera decisiones de control de inventario para brindar el nivel de servicio exigido por los clientes, manteniendo un stock de seguridad apropiado para satisfacer la demanda de los clientes. El problema fue resuelto mediante la metaheurística, Algoritmos Genéticos, la cual parte con una red generada aleatoriamente, utilizando la codificación entera positiva. Esta población está formada por individuos, los cuales se irán cruzando y mutando iterativamente hasta arrojar una solución factible para el problema. Los resultados obtenidos, si bien no fueron óptimos, con respecto a otros métodos de resolución, arrojaron soluciones donde en algunos casos hubo una diferencia de tan solo un 0,1% de la óptima encontrada.

ABSTRACT

This work is based on designing a distribution network, including strategic decisions, which is to determine the amount needed and where to locate facilities to provide individual customers, as efficiently as possible. Will be based on the model of localization Distribution Network Design with Risk Pooling (DNDRP), where the problem supports strategic decisions with the tactics, ie, considered decisions of inventory control to provide the level of service demanded by customers, keeping an appropriate safety stock to meet customer demand. The problem was solved by the metaheuristic, genetic algorithms, this start with a randomly generated network, using encoding entire positive. This population was comprised of individuals or chromosomes, which are iteratively crossing and mutating to produce a workable solution to the problem. The results obtained, were not optimal, with respect to exact methods of resolution. In some case the solutions was a difference of only 0.1% of the optimum found.

CAPÍTULO 1

Descripción del Proyecto

1.1 Introducción

En la mayoría de las compañías, se debe prestar real atención sobre la cadena de suministro de forma completa, ya que éstas deben analizar particularmente cómo abastecer y mejorar el nivel de servicio a los clientes sin descontrolar el crecimiento de los costos. En resumen, las compañías deben incrementar la eficiencia de las operaciones logísticas, siendo esto de fundamental importancia para optimizar tanto los flujos de bienes como los de información entre los diferentes actores de la cadena de suministro, entre los cuales se encuentran los proveedores, distribuidores y clientes.

Este trabajo de título se enfocó en analizar el problema de diseño de redes de distribución que es típica en compañías productoras y distribuidoras, por lo tanto el objetivo común entre ellas, es optimizar el flujo de bienes de la red productora, también llamada red de distribución, que va desde una instalación (Centro de Distribución) donde los bienes son almacenados, hasta los puntos de demanda, que son principalmente los clientes.

Se analizó el diseño de una red de distribución teniendo siempre en mente abastecer a los clientes con el nivel de servicio dado, para poder satisfacer la demanda, en cuyo caso se consideró una red de distribución existente, la cual se mejorará con algoritmos genéticos

para determinar la mejor configuración de la red, es decir, dónde localizar las instalaciones para satisfacer a los respectivos clientes minimizando los costos.

El problema del diseño de redes de distribución involucra dos clases de análisis, uno consiste en el problema de determinar la mejor forma de transferir o distribuir los bienes desde la planta proveedora hasta los puntos de demanda teniendo que elegir la estructura de la red (número de instalaciones, que centro abastece a un determinado cliente), minimizando todos los costos.

El otro problema consiste en la toma de decisiones estratégicas, influenciadas por las decisiones tácticas. En particular, estas decisiones involucran las decisiones de transporte y de inventario, las cuales afectan a los costos de almacenamiento, ordenamiento, transporte y el costo de mantener un stock de seguridad, además de influir en la calidad del nivel de servicio a los clientes. Ambos problemas son básicos para cada compañía.

Como se ha podido ver, el problema de diseño de redes de distribución involucra varias decisiones, las cuales son difíciles de considerar todas juntas. Generalmente, se han adoptado suposiciones en la literatura existente, donde se toman en cuenta ciertos aspectos a la hora de modelar el problema, en algunos casos, como CFLP (Capacitated Facility Location Problem) o DSDP (Distribution System Design with *k*-Products), los autores discuten acerca del problema de diseño de redes de distribución como problemas de localización puro sin la integración de los diferentes tipos de decisiones tácticas.

En este trabajo se tomaron en cuenta las decisiones tácticas para la toma de decisiones estratégicas, es decir, involucrar las decisiones de inventario en la localización de las instalaciones, además de considerar todos los tipos de costos para la red (Planta o bodega central, bodegas regionales o centros de distribución, clientes). Donde la planta central envía los bienes a los centros, logrando satisfacer la demanda de cada cliente.

Para encontrar una buena solución, se utilizó la metaheurística Algoritmos Genéticos, con lo cual se redujo la amplia gama de metaheurísticas que se encuentran en la literatura.

1.2 Definición de Objetivos

❖ Objetivo General

- ✓ Resolver el problema de Diseño de una Red de Distribución, que toma en cuenta el control de inventario considerando una demanda estocástica, a través de algoritmos genéticos.

❖ Objetivos Específicos

- ✓ Realizar un estudio de los modelos para el diseño de redes de distribución con información bibliográfica existente.
- ✓ Encontrar un modelo que represente el problema de diseño de Redes de Distribución, considerando la demanda de tipo estocástica y decisiones para el control de inventario.
- ✓ Realizar un estudio del funcionamiento de los algoritmos genéticos para la solución del problema modelado, basándose en la literatura existente.
- ✓ Diseñar una solución para un prototipo basándose en una simplificación del problema modelado, basada en algoritmos genéticos. (CFLP)
- ✓ Diseñar e implementar una solución para el problema modelado, basada en algoritmos genéticos.

1.3 Metodología

Según (Hernández R., 1998), los tipos de investigación son cuatro:

- Exploratorio
- Descriptivo
- Correlacional
- Explicativo

La metodología exploratoria es la investigación que pretende darnos una visión general y sólo aproximada de los objetos de estudio. Este tipo de investigación se realiza especialmente cuando el tema elegido ha sido poco explorado, cuando no hay suficientes estudios previos y cuando aún, sobre él, es difícil formular hipótesis precisas o de cierta generalidad.

Para la metodología Descriptiva, su preocupación primordial radica en describir situaciones y eventos, utilizando criterios sistemáticos que permiten poner de manifiesto la estructura o el comportamiento de los fenómenos en estudio, proporcionando de ese modo información comparable con la de otras fuentes.

La metodología Correlacional mide dos o más variables, en las cuales se pretende ver si están o no relacionadas en los mismos sujetos y después se analiza la correlación. La utilidad y propósito son saber cómo se puede comportar un concepto o variable conociendo el comportamiento de otras variables relacionadas.

Finalmente, la metodología Explicativa, es aquella en donde la preocupación se centra en determinar los orígenes o las causas de un determinado conjunto de fenómenos. Su objetivo, por lo tanto, es conocer por qué suceden ciertos hechos, analizando las relaciones causales existentes o, al menos, las condiciones en que ellos se producen.

Para una primera parte del proyecto se utilizó una metodología exploratoria, la cual ayuda a tener una visión general de tipo aproximativo respecto del problema a investigar. Este tipo de investigación se realizó tomando en cuenta la literatura existente sobre diseño de redes de distribución, modelos de localización de instalaciones y las diferentes heurísticas existentes. Luego de finalizar con la etapa exploratoria, se proseguió utilizando una metodología descriptiva, donde su objetivo es lograr una descripción de las principales características de los temas estudiados, lo cual es de vital importancia para la integración de éstos para generar el modelamiento del problema.

Finalmente se utilizó la metodología explicativa donde su objetivo es determinar justificación de los resultados obtenidos, los cuales se deben acercar a la realidad y donde se debe tener un real cuidado para no cometer errores que aumenten considerablemente una mala conclusión final.

1.4 Organización del Texto

El presente trabajo de título se divide en seis capítulos, el cual se iniciará con la descripción del proyecto, los objetivos y la metodología utilizada para este trabajo de investigación en el capítulo 1.

En el capítulo 2 se hará un estudio de los diferentes modelos de localización existentes en la literatura para la mejor comprensión del problema, además de revisar en qué consiste la gestión de inventario. En el capítulo 3 se hace un estudio del estado del arte, el cual consiste en investigar acerca de los Algoritmos Genéticos.

En el capítulo 4 se realizará la descripción para DNDRP, planteando la resolución matemática para su futura implementación. Se aplicarán los algoritmos genéticos al CFLP, que es una simplificación del problema propuesto, detallando cada uno de sus elementos y realizando una breve explicación de la estructura de datos utilizada en el diseño del algoritmo, para finalizar este capítulo 5, se mostrará el algoritmo asociado a la implementación de AG aplicado al CFLP en un diagrama de flujo, además se enseñarán los resultados obtenidos. La aplicación de AG a CFLP es necesaria para verificar cómo se comporta AG en este tipo de problemas de localización, utilizando la representación o codificación escogida, y así utilizar este prototipo como base para el problema propuesto.

Luego de haber aplicado AG al CFLP se continuará con la aplicación para DNDRP, describiendo cada uno de sus elementos, estructuras de datos necesarias para la implementación y algoritmo realizado a través de un diagrama de flujo. Se propondrá una extensión que consiste en agregar una variable de decisión para hacer más completo la resolución del problema. Los resultados obtenidos para DNDRP aplicando AG serán mostrados en un cuadro comparativo, el cual entregará resultados obtenidos con otros tipos de métodos para su comparación, lo cual se explica en el capítulo 6.

CAPÍTULO 2

Modelos de Localización e Inventario

2.1 Modelos de Localización

Los problemas de localización de instalaciones es un problema que involucra a una serie de clientes repartidos físicamente en distintas áreas geográficas originando demanda por algún bien o servicio.

La diversidad de modelos de localización de instalaciones nace de las aplicaciones del mundo real, donde la demanda de los clientes debe ser abastecida por una o más instalaciones.

El proceso de decisión debe establecer dónde localizar las instalaciones en un espacio según los requerimientos de los usuarios y posibles restricciones de territorio. Cada elección particular para localizar un centro de distribución implica una serie de costos, algunos operacionales y otros fijos, como los de instalación. Características como, reducción de costos, satisfacción de la demanda, abastecimiento de bienes, tiempo rápido de respuesta y otros, deben manejar o guiar hacia una mejor selección del lugar dónde localizar una instalación.

El problema de localización de instalaciones abarca tres términos muy importantes: instalaciones, clientes y localización.

El término instalación, hace referencia a objetos, para los cuales se debe determinar dónde ubicarlos para que interactúen entre ellos, además de interactuar con los objetos ya existentes. Los ejemplos más clásicos de instalaciones, que se pueden mencionar, son las plantas de distribución, escuelas, almacenes, hospitales, y otras estructuras industriales, comerciales o públicas.

El segundo término, localización, se refiere al espacio físico donde ubicar una instalación, existen tres tipos de representaciones para las diferentes formas de localización: discreta, continua y redes. En la localización de tipo discreta, la decisión se toma basándose en una lista de sitios específicos donde realizar la instalación. Este tipo de solución es muy flexible a la hora de agregar más características al modelo. Para la localización de tipo continua asume que el espacio donde localizar las instalaciones es continuo, donde todos los puntos pueden ser determinados por una o más coordenadas que varían continuamente. Y para la localización de tipo red, que son una excelente representación al problema, generalmente es más eficiente, ya que puede resolver problemas de tamaño más grande.

Por último, el término cliente, este hace referencia a los clientes que necesitan de los servicios o bienes de las instalaciones, por lo que son los clientes los que hacen nacer los problemas de localización de instalaciones (Scaparra, 2001).

Se describirá brevemente algunos de los modelos más mencionados en la literatura, uno de ellos es el llamado *p*-Median Problem, formulado cerca de los 1960s. (Bramel, 2000), explica que este problema es el más general y simple, donde dado un conjunto de *p* instalaciones, se debe encontrar la mejor configuración para minimizar la distancia entre los clientes y la instalación a la cual fueron asignados, y así lograr la mínima suma de la distancia total o promedio.

Algunas modificaciones que se han realizado a este problema a lo largo de los años son el MSPP (median shortest path problems) y el PMMC (p-median problem with mutual communication). Otro problema es el conocido CFLP (capacitated facility location problem), decrito por (Zhang, 2005), el cual consiste en asignar a una serie de n clientes, donde sus demandas son conocidas, a una serie m de posibles instalaciones. Cada cliente es servido por sólo una instalación, sin sobrepasar los límites de capacidad que tiene cada instalación.

El tercer modelo explicado por (Bramel, 2000), es el DSDP, Problema de diseño de sistemas de distribución, el cual considera múltiples plantas con capacidades fijas, pero considerando *K* tipos de productos diferentes.

(Cachon, 2000), (Chen, 2000) y (Lee, 2000) investigaron diferentes aspectos sobre el control de inventario. El estudio de cómo diseñar una red de distribución, con decisiones de inventario involucra varias decisiones con respecto a la localización de las instalaciones.

(Miranda, 2004), (Sourirajan, 2006) han estudiado el modelo DNDRP considerando diferentes variables que influyen en las decisiones de control de inventario.

(Sourirajan, 2008) aplica los algoritmos genéticos al modelo DNDRP con diferentes estrategias, una considerando un vector binario para su representación, otra es utilizando un clave aleatoria y por último un algoritmo híbrido que mezcla lo mejor de ambos.

Para tener una primera visión, sobre si los AG se pueden aplicar a los problemas de diseño de redes de distribución obteniendo resultados aceptables, es que como primera instancia en este trabajo se tomó como base para un prototipo el modelado que presenta (Zhang, 2005) para CFLP, que es un problema dónde sólo se toma en cuenta los costos de instalación y transporte, además de la restricción de capacidad que mantiene cada centro.

Luego para desarrollar el problema de localización propuesto, es decir, considerando una demanda estocástica es que se consideró la investigación realizada en (Miranda, 2004) con respecto al modelado del problema DNDRP , ya que cuenta con el problema básico de decidir la localización de instalaciones pero apoyado por decisiones de tipo tácticas, es decir considerando el control de inventario para poder satisfacer el nivel de servicio exigido por los clientes, está inclusión requiere especial relevancia en la presencia de los costos de almacenamiento, ordenamiento y la alta variabilidad de la demanda.

1.1 Definición

La ilustración *2.1.1* muestra una red de distribución donde una planta central abastece a los demás centros de distribución (*CD*) y estos *CD* distribuyen los bienes a los clientes (*C*), entonces se debe tomar en cuenta a la hora de realizar decisiones, los costos de almacenamiento de cada instalación, además de los costos de transporte.

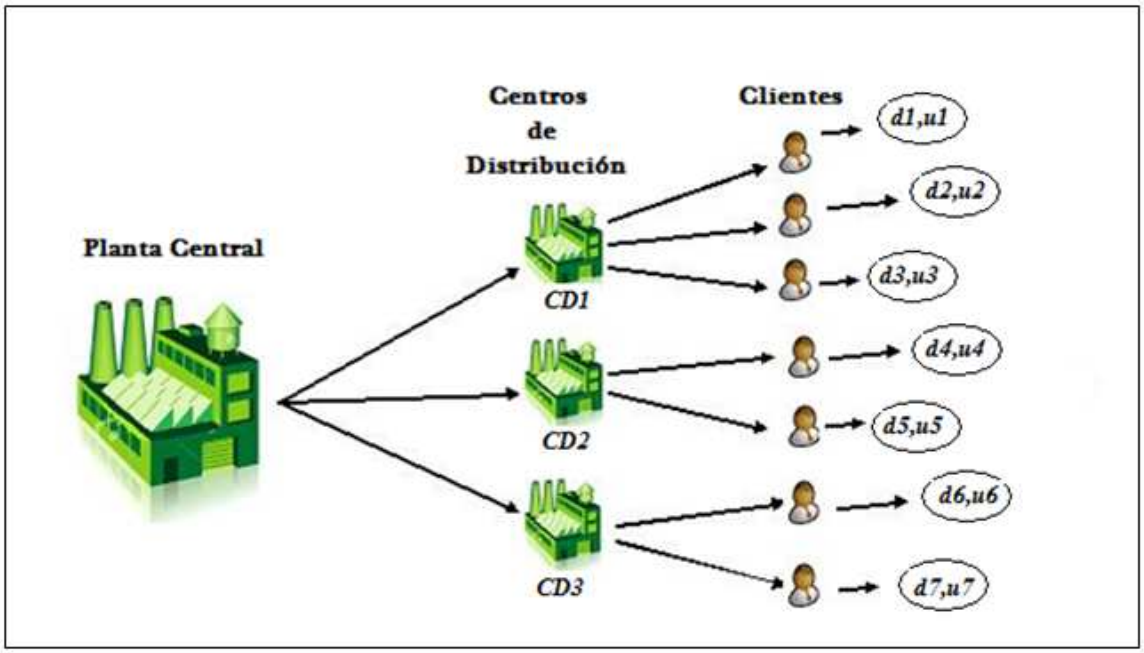


Ilustración 2.1.1 Red de Distribución.

Es por esto que las instalaciones deben considerar dos tipos de costos: uno es proporcional a la media suministrada por la demanda, es decir a los costos de almacenamiento y de administración, y el otro tipo de costo es proporcional a la desviación estándar de la demanda suministrada, es decir costos de mantener un stock de seguridad.

Bajo las constantes, nivel de servicio y tiempo de reposición, el stock de seguridad es proporcional a la desviación estándar de la demanda suministrada (Miranda, 2004).

Por lo tanto, si tenemos 3 *CD*, donde CD_1 abastece al cliente C_1, C_2 y C_3, el CD_2 abastece a C_4 y C_5, y el CD_3 abastece a C_6 y C_7, el stock de seguridad del CD_1 debe ser proporcional a demanda suministrada dada por $\sqrt{u1+u2+u3}$, el stock de seguridad del CD_2 está dada por $\sqrt{u4+u5}$, y el stock de seguridad de CD_3 dado por $\sqrt{u6+u7}$, entonces claramente los costos dependen del diseño de la red. Si se cierra el CD_1 y se asignan los clientes del CD_1 al CD_2, existen cambios que afectan considerablemente a la red.

Los costos fijos de instalación del CD_1 desaparecen, los costos de trasporte desde la planta a los *CD* y desde los *CD* a los clientes cambian, los costos que involucra mantener el stock de seguridad se ven reducidos, debido que el costo del CD_2, ahora será proporcional a $\sqrt{u1+u2+u3+u4+u5}$ que es menor que el costo de ambos *CD* abiertos $\left(\sqrt{u1+u2+u3}+\sqrt{u4+u5}\right)$.

En resumen, el efecto "risk pooling" consiste en disminuir la cantidad de centros de distribución abiertos para reducir los costos de stock de seguridad, es decir centralizar la red de distribución.

Debido al efecto "risk pooling", es que el problema de localizar instalaciones se hace más compleja, ya que por un lado se tienen los costos de transporte asociados, donde el diseño de una red desea aspirar a que cada instalación esté lo más cerca posible de los clientes, por lo que mientras más instalaciones se localicen en la red más se reducirán los

costos de transporte, por otro lado están los costos del stock de seguridad, los cuales tratan de minimizar la cantidad de instalaciones para reducir el costo de stock, por lo tanto, el problema debe encontrar la cantidad óptima de instalaciones a localizar para que haya un equilibrio entre los costos.

2.2 Modelos de Control de Inventario

El control de inventario es de gran importancia en la cadena de abastecimiento de las compañías distribuidoras, por lo que se debe considerar este tipo de gestión en la toma de decisiones tácticas.

Según (Gutiérrez, 2008) una política de inventarios debe dar respuesta a las preguntas de cada cuánto debe revisarse el inventario, cuándo ordenar y cuánto ordenar, bien sea ítems de demanda independiente o dependiente.

Las principales variables en un sistema de inventario dadas por (Abdul, 2004) son:

- *Demanda*
- *Costos*

Los tipos de Demanda según (Abdul, 2004) son:

- ✓ *Demanda determinística y estacionaria*: esta suposición consiste en asumir una demanda constante y conocida, es decir, la demanda no cambia y puede ser fijada a priori. El modelo que se basa en esta suposición es el EOQ.
- ✓ *Demanda determinística variable en el tiempo*: esta suposición asume una demanda no constante, es decir, que la cantidad demandada varía con el tiempo. Un ejemplo de este tipo es el problema dinámico de tamaño de lote.
- ✓ *Demanda incierta*: se dice que la demanda es incierta cuando no se pueden conocer a priori los valores exactos de la demanda, pero si se conoce la distribución de la demanda. Normalmente se dispone de una serie de valores de la demanda en el pasado y, a partir de ellos, se intenta estimar la distribución de la demanda y los valores de los parámetros que caracterizan dicha distribución. En algunas

situaciones, debe considerarse demanda incierta, por ejemplo, en la salida al mercado de un nuevo producto. Este es el tipo de demanda considerada en este trabajo de título, es decir, una demanda estocástica.

- ✓ *Demanda desconocida*: Cuando tampoco es posible conocer la distribución de la demanda, se dice que la demanda es desconocida. En este caso se suele asumir una distribución a priori de la demanda, y después, una vez que se dispone de una nueva observación de la demanda, se actualiza el parámetro estimado usando la regla de Bayes.

Según (Herrera Povis, 2006) básicamente existen cuatro tipos de costos pertinentes:

- ✓ *Costo del producto:* Es la suma que se paga al proveedor por el producto recibido, o costo directo de manufactura si este se produce. Normalmente es igual al precio de adquisición.

- ✓ *Costos de adquisición:* Son aquellos costos en los que se incurre al colocar la orden de compra o si se trata de manufactura se considera como costos de preparación. Estos costos varían con cada orden de compra colocada. Los costos de adquisición incluyen costos de servicio de correo, llamadas telefónica a los proveedores, costos de mano de obra en las compras y contabilidad, costos de recepción, tiempo de computo para el mantenimiento de los registros y abastecimiento para la elaboración de la orden de compra.

- ✓ *Costos de manejo de inventario:* Los costos de llevar el inventario son costos reales, los que salen del bolsillo y se relacionan con tener el inventario disponible. Estos costos incluyen los seguros, calefacción, energía, impuestos, perdidas por robo, descomposición de productos o por rotura y los costos en los que se incurre por tener el capital ocioso en los inventarios.

- ✓ *Costos por la falta de existencia:* Los costos por falta de existencia son los que ocasiona la demanda, cuando las existencias se agotan o sea son los costos de ventas

perdidas o de pedidos no surtidos. La empresa pierde el margen de utilidad de las ventas no realizadas y la confianza del cliente.

Existen algunos supuestos, según (Sáez, 2003) en modelos de inventario que se deben considerar, dentro de los que se identifican están:

- ✓ *Ordenes repetitivas:* La decisión de ordenar es repetitiva en el sentido que es repetida en forma regular.

- ✓ *Demanda constante:* Se asume que la demanda es conocida y ocurre a tasa constante.

- ✓ *Lead Time constante:* Por lead time (L) entenderemos el tiempo transcurrido entre la emisión de una orden y la llegada de los artículos solicitados.

- ✓ *Ordenes continuas:* Se supondrá que se puede efectuar una orden en cualquier instante. En estos casos se habla de modelos de inventario con revisión continua. Si la revisión del inventario se hace a intervalos regulares se habla de modelos con revisión periódica. Tal es el caso de situaciones en la que sólo se puede efectuar órdenes cada cierto período de tiempo.

Si bien la consideración de demanda constante y lead time constante pueden ser altamente irreales y restrictivas, existen muchas situaciones en las que estas consideraciones permiten obtener buenas aproximaciones respecto de la situación real.

1.2 Modelo EOQ

Los supuestos para el modelo EOQ según (Gutierrez Exposito, 2008) son las siguientes:

- ✓ La demanda es conocida y constante, a una razón de *d* unidades por unidad de tiempo.

- ✓ La cantidad a pedir puede ser un número no entero, y no hay restricciones sobre su tamaño.

- ✓ Los costos no dependen de la cantidad de reposición, es decir, no hay descuentos dependiendo del tamaño de lote.

- ✓ Los costos no varían con el tiempo. Existe un coste de reposición, *c*, por pedido, y un coste de mantenimiento, *h*, por unidad mantenida a lo largo de cierta unidad de tiempo.

- ✓ Las reposiciones son instantáneas, es decir, el período de reposición es cero.

- ✓ No se permiten roturas.

- ✓ El horizonte de planificación es muy largo, es decir, se asume que los parámetros toman el mismo valor durante un largo período de tiempo.

Como el período de retardo es cero y la demanda es conocida, es evidente que sólo se debe realizar un pedido cuando el nivel de inventario llega a cero. Un gráfico del nivel de inventario se puede ver en la Ilustración 2.2.1.

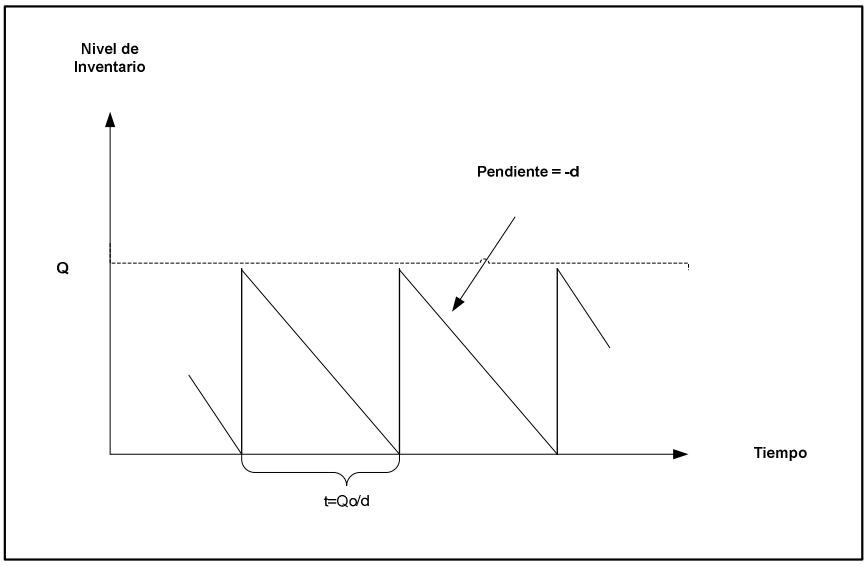


Ilustración 2.2.1 Gráfico del nivel de inventario.

Para este modelo la cantidad de reposición óptima, conocida como EOQ, (Economic Order Quantity) es:

$$Q_o = \sqrt{\frac{2dc}{h}} \quad (1)$$

Esta fórmula es uno de los primeros resultados y el más conocido de la Teoría de Inventarios. Se conoce como la fórmula de Harris (1913) o de Wilson (1934), ya que estos autores fueron los primeros que recogieron en sus respectivos trabajos dicha fórmula.

Para este trabajo de título se tomó en cuenta el modelo EOQ, ya que en el caso de DNDRP considerado como base para este problema, se optimiza la cantidad de lote que debe ordenar cada centro de distribución, y según el modelado realizado por (Miranda, 2004) la cantidad de lote para cada centro de distribución viene dado por la expresión (1) .

CAPÍTULO 3

Algoritmos Genéticos

3.1 Descripción

El algoritmo genético fue insertado por (Holland, 1995), y desde ahí se ha convertido en el método más popular para resolver problemas de optimización con múltiples óptimos locales.

AG se obtiene por la analogía entre los componentes de la configuración de un vector x (solución inicial) y la estructura genética de un cromosoma. Cada célula de todo organismo viviente contiene uno o más string de DNA, agrupados en cromosomas, los cuales pueden ser o no divididos en uno o más genes. El organismo puede tener múltiples cromosomas en cada célula.

En los AG, existe una característica muy especial que es la utilización de operadores, el más importante es el operador de cruce, ya que es el mecanismo fundamental de la búsqueda, el cual consiste en el intercambio de los valores de determinadas variables o individuos de la población. La combinación de los operadores de cruce con los de mutación, es la que se encarga de crear la descendencia a partir de la población inicial, manteniendo ciertas características de los individuos de esta población.

El operador de mutación debe encargarse de mantener el nivel de diversidad en cada población generada. Los individuos de cada población deben representarse por una serie de cromosomas, que generalmente están representados en cadenas binarias, pero pudiendo también tener otro tipo de representaciones.

Esta representación es de primordial importancia, debido a que de esta depende la obtención de la solución, por lo que esta representación debe considerar toda la variedad posible de soluciones para un determinado problema, así como las características que se desea que tenga la solución final.

Teniendo claro la representación de los individuos, es que se debe generar una población inicial, ya sea de forma aleatoria o según otro mecanismo que se desee implementar. Luego de la generación de una población inicial, el segundo paso es seleccionar a los individuos que se combinarán para generar la población siguiente.

Generalmente esta selección se hace de a pares, los cuales serían los padres que generarán nuevos hijos, estos hijos generados por las operaciones de cruce y/o mutación pueden ser insertados en una población nueva que sería la población inicial de la siguiente iteración o bien los hijos pueden ser reemplazados por los padres desde la misma población inicial, siempre y cuando los hijos sean mejores que los padres.

Inicio;

Generar aleatoriamente n números de x^i individuos de largo p;

Evaluar el fitness g para cada individuo;

Repetir:

Seleccionar n individuos y^i nuevos generados con $g(x^i)$, y agruparlos aleatoriamente en pares;

Para cada par de y^i hacer:

Generar de manera aleatoria y uniforme r_1 del rango (0,1);

If $r_1 \leq p_c$ (probabilidad de cruzamiento) entonces generar un entero k aleatorio y uniforme del rango desde 1 a (p-1) e intercambiar los (p-k) elementos por cada padre y^i a la derecha del elemento k;

Para cada individuo y^i hacer:

Para cada dígito binario en un individuo hacer:

Generar aleatoriamente r_2 uniformemente del rango (0,1);

If $r_2 \leq p_m$ (probabilidad de mutación) cambiar el actual elemento de 0 a 1 o viceversa;

Guardar los individuos para la siguiente ejecución, según el mecanismo de inserción deseado;

Calcular el fitness g;

Hasta que el criterio de finalización = verdadero;

Y_{opt} es la aproximación de la solución óptima;

Fin.

Ilustración 3.1.1 Algoritmo Genético (GA)

3.2 Elementos de un Algoritmo Genético

Los AG son capaces de ir creando soluciones para problemas del mundo real y la evolución de dichas soluciones hacia valores óptimos depende de una adecuada codificación de las mismas, es por esto que los algoritmos genéticos cuentan con varios elementos, que para la buena codificación de problemas se deben tomar en cuenta a la hora de aplicarlos.

Criterio de codificación

❖ Representación

Generalmente el tipo de representación más utilizado es la representación binaria, por la simple razón que facilita y se adapta a las operaciones de cruce y mutación, pero existen otros tipos de representaciones, ya que sólo deben ser capaces de identificar las características que llevarán a una posible solución al problema. Se pueden considerar tres tipos básicos de representaciones según (Caballero, 2002):

- ✓ *Representación Binaria*: Donde cada gen del individuo o cromosoma toma el valor 1 o 0.

0	1	1	0	0	0	0	0	1	0

Ilustración 3.2.1 Ejemplo de individuo con representación binaria

- ✓ *Representación Entera*: Donde cada gen del individuo o cromosoma puede tomar un valor entero. Se puede acotar la representación a solo enteros positivos o solo enteros negativos según sea más representativo para el problema.

8	10	-7	5	7	56	3	21	1	-5

Ilustración 3.2.2 Ejemplo de individuo con representación entera

Para la implementación de AG en el problema propuesto, se utilizó este tipo de representación, si bien la más utilizada en la literatura es la representación binaria, se trató de proponer una forma de resolución distinta para ver cómo se comportan las soluciones obtenidas con este tipo de representación.

- ✓ *Representación real*: Donde cada gen de un individuo o cromosoma toma un valor real.

8.0	-1.0	-5.7	3.0	4.7	-6.3	5.5	2.1	-1.0	4.5

Ilustración 3.2.3 Ejemplo de individuo con representación real

❖ Población Inicial

En un AG es de vital importancia la población inicial, ya que es el punto de partida para el algoritmo, esta población tiene varias formas de ser generada. Se puede generar aleatoriamente cada gen de cada individuo o cromosoma de la población, o bien implementar alguna función que genere dicha población u otro tipo de creación que se adapte a los individuos que se necesitan.

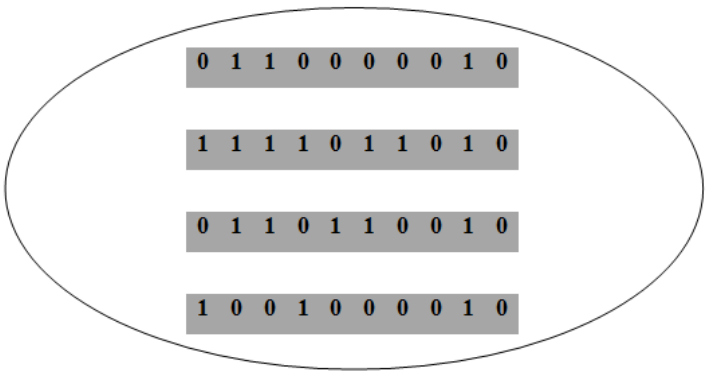


Ilustración 3.2.4 Ejemplo de población Inicial para codificación binaria

❖ Función de Evaluación

La función de evaluación, también llamada función Fitness, es la función encargada de evaluar la adaptación de los individuos de la población, es decir mide la calidad de cada solución generada. Esta función fitness depende de la representación utilizada, además del tipo de problema a resolver. En el caso de problemas de optimización, la función fitness

suele ser la función objetivo del problema, por lo tanto, si es un problema de minimización, un buen fitness es uno que arroje un bajo costo como solución.

❖ Reparación o Penalización

La penalización es utilizada cuando existen restricciones para el problema a solucionar, es decir, cuando un individuo generado no cumple con dichas restricciones, y por lo tanto se le debe sumar a la función fitness una penalización. Esta penalización depende de los valores que se obtengan de la función objetivo, para que pueda tener una real influencia en las posibles soluciones del problema.

El otro caso consiste en reparar los individuos no factibles con alguna función implementada que arregle al individuo hasta que cumpla con las restricciones.

❖ Operador de Selección

Este operador es el encargado de elegir los individuos desde la población inicial, a los cuales se les aplicará los operadores de cruce y/o mutación. Este operador debe considerar las características más positivas para que se mejoren a lo largo de las generaciones siguientes. La manera más general de seleccionar a los individuos, es elegir a los individuos de mejor fitness, o darle una mayor probabilidad de ser elegidos. Pero se debe considerar que existen individuos que pueden poseer información útil para las posibles futuras soluciones y que no necesariamente tienen un buen valor para el fitness, por lo que se debe considerar el factor de aleatoriedad para que la diversidad se mantenga durante las generaciones. Los principales mecanismos de selección son (Caballero, 2002):

- ✓ *Ruleta o Selección Proporcional*: Este método consiste en que la probabilidad de reproducción de un individuo es proporcional su valor de la función fitness. Por lo que se debe calcular esta probabilidad y luego seleccionar aleatoriamente según estos valores.

Individuo	Representación	Fitness	Probabilidad
1	2343345334	34567	10,20%
2	3342112524	54653	16,12%
3	2134121132	32543	9,60%
4	2342332452	98657	29,10%
5	2345245111	87687	25,87%
6	2223333442	30876	9,11%
	Suma total	338983	

Tabla 1 Datos de ejemplo para la selección por ruleta

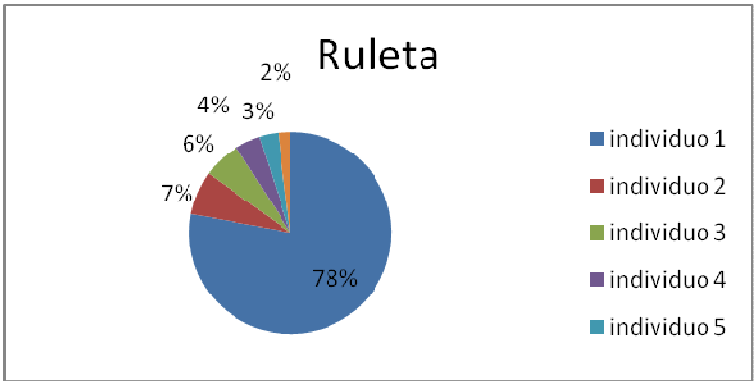


Ilustración 3.2.5 Selección por Ruleta con las probabilidades de los individuos

- ✓ *Selección por Ranking*: Este método consiste en el cálculo de la probabilidad de reproducción de un individuo, teniendo en cuenta un ordenamiento de la población según el valor de la función fitness. El tipo de selección anterior funciona mal cuando existan grandes diferencias entre los fitness de los individuos de la población. Por ejemplo si un cromosoma ocupa el 76% de la ruleta el resto de los individuos tienen muy pocas posibilidades de ser elegidos. La selección por ranking da solución a este problema.

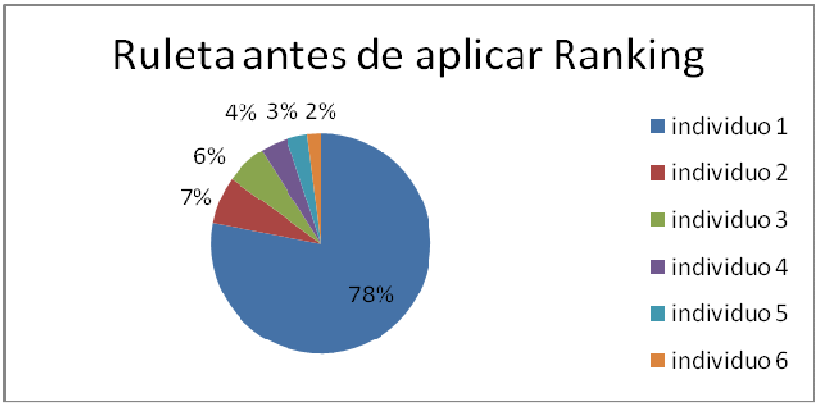


Ilustración 3.2.6 Ruleta con los individuos antes de aplicar el ranking

Luego los individuos son ordenados de acuerdo a su ranking de fitness. De esta manera si tenemos *n* individuos el individuo con peor fitness se le asignará un 1 y el que tenga el mejor fitness se le asignará la *n*, multiplicando su valor por el respectivo *n*.

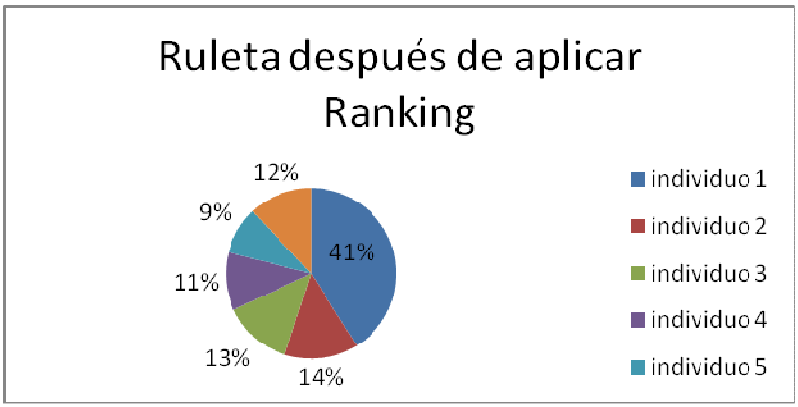


Ilustración 3.2.7 Ruleta con los individuos después de aplicar el ranking

- ✓ *Selección por Torneo*: Este método consiste en ir eligiendo pares de individuos aleatoriamente, luego se genera un número aleatorio entre 0 y 1. Si este número es menor a cierto rango, se selecciona al individuo con mejor fitness, si por el contrario, el número es mayor a tal rango se selecciona al individuo de peor fitness. Se debe considerar que el rango debe asignar mayor probabilidad a los individuos de mejor fitness.

Individuo	Representación	Fitness
1	2343345334	34567
2	3342112524	54653
3	2134121132	32543
4	2342332452	98657
5	2345245111	87687
6	2223333442	30876

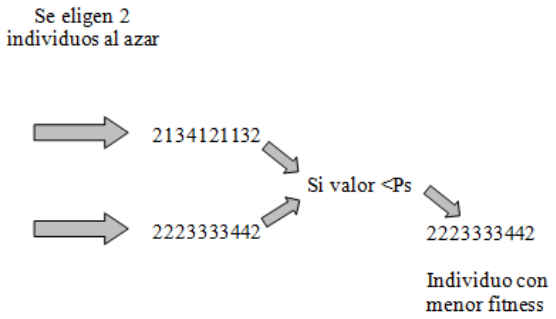


Ilustración 3.2.8 Selección por torneo según adaptación de los individuos

❖ Operador de Cruce

Este operador explora la información almacenada en la población, luego combinarla para crear o generar nuevos y mejores individuos. Los mecanismos de cruce más comunes son (Caballero, 2002):

- ✓ *Cruce de un punto*: Este método es el más sencillo, el cual consiste en generar un valor aleatorio, y desde esa posición de la cadena de los padres intercambiar todos los valores que se encuentren a su izquierda para generar a los hijos.

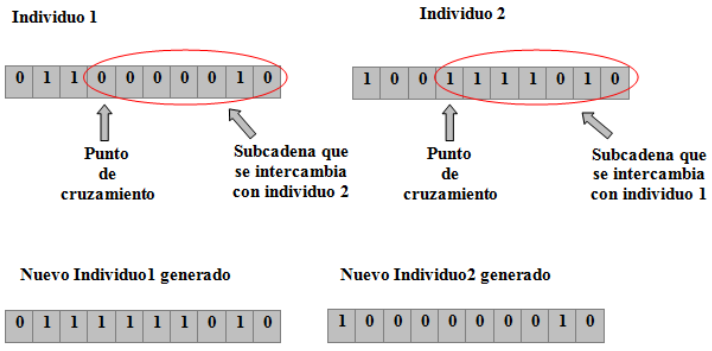


Ilustración 3.2.9 Ejemplo de Cruzamiento de un punto.

- ✓ *Cruce de n puntos*: Este método es una generalización del método anterior, el cual consiste en seleccionar varias posiciones de la cadena de los padres e intercambiar los valores a ambos lados de estas posiciones, para generar a los hijos.

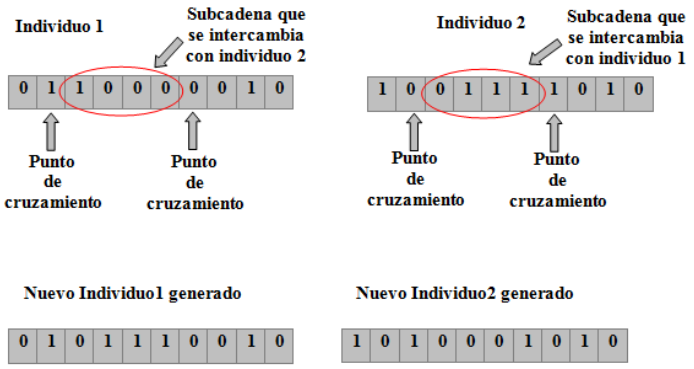


Ilustración 3.2.10 Ejemplo de cruzamiento de n puntos.

- ✓ *Cruce Uniforme*: Este método consiste en elegir aleatoriamente genes del individuo que se van a intercambiar, la cantidad de genes también se puede elegir al azar o dejarlo como un parámetro de entrada.

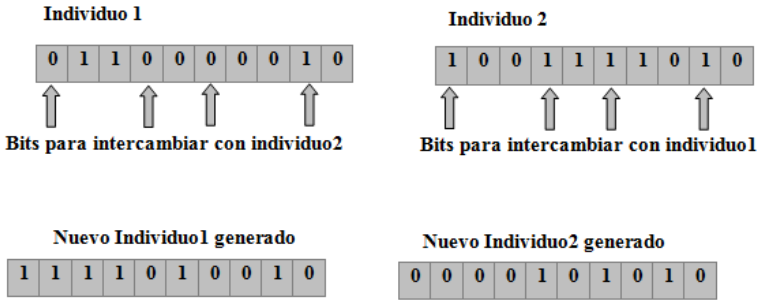


Ilustración 3.2.11 Ejemplo de cruzamiento uniforme.

❖ Operador de Mutación

Este operador consiste en realizar nuevas soluciones a partir de los individuos de la población inicial, para generar una variabilidad dentro de esta población. La manera más general de realizar la mutación es la Puntual (Caballero, 2002), donde se elige al azar un gen del individuo para ser modificado.

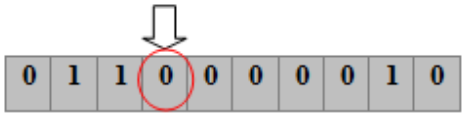


Ilustración 3.2.12 Ejemplo de individuo padre para la mutación

Si aleatoriamente se eligió el gen de la posición 3, el hijo generado queda de la siguiente manera:

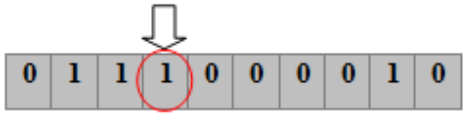


Ilustración 3.2.13 Ejemplo de individuo generado por operador de mutación.

❖ Inserción o Reemplazo de la población

Cada vez que se aplican los operadores de cruce y mutación se generan nuevos hijos, los cuales deben ser insertados o reemplazados en la población para la próxima generación. Es para esto que existen cuatro mecanismos (Caballero, 2002):

- ✓ El primer mecanismo consiste en generar individuos para la población e ir insertándolos en ella, cuando se sobrepasa de un cierto límite de individuos, se deben eliminar los individuos menos adaptados.

- ✓ El segundo mecanismo consiste en que cada vez que se genera un individuo, este se inserta en la población y se elimina el individuo peor adaptado.

- ✓ El tercer mecanismo consiste en que cada vez que se genera un individuo, este se inserta en la población y se elimina aleatoriamente un individuo, independiente si es el mejor o peor adaptado.

- ✓ El cuarto mecanismo, va insertando los individuos generados en una población temporal, la cual será la población inicial para la próxima generación. Este mecanismo no utiliza reemplazo, solo inserción.

❖ Criterio de Parada

El criterio de parada en un AG, generalmente utilizado, es el número máximo de iteraciones o un tiempo máximo de ejecución. Más eficientemente se puede realizar un estudio del número de iteraciones donde los resultados de la función fitness ya no tienen mucha variabilidad, es decir, el algoritmo a partir de cierto número de iteraciones comienza a comportarse de manera constante o converger.

Cada uno de estos elementos, clasificados por (Caballero, 2002), serán considerados a la hora de implementar el problema propuesto aplicando AG.

Para la implementación de AG, se realizará un estudio de cada uno de estos elementos, y así elegir el tipo más adecuado en la resolución del problema para poder encontrar el individuo más adaptado, es decir, el individuo que arroje la mejor solución.

Como cada elemento se clasifica en varios tipos, la elección se realizará probando cómo se comporta el algoritmo con cada uno de ellos, y así seleccionar el que arroje los mejores resultados.

CAPÍTULO 4

Descripción de DNDRP

4.1 Introducción

El problema del diseño de redes de distribución involucra cómo distribuir una serie de nodos en la red, además de las diferentes posibilidades de localización de los proveedores, clientes o centros de distribución.

Se explicarán algunos de los términos más importantes en la configuración de una red de distribución:

- ✓ Los nodos en la red de distribución se dividen en plantas (*P*), centros de distribución (*CD*) y los clientes (*C*).
- ✓ En la red, existe una planta central que envía bienes (productos) hacia los *CD*, los *CD* trasfieren estos bienes a los clientes.
- ✓ Existen decisiones de ruta, las cuales no se tomaron en cuenta en este trabajo de título.

Para lograr el diseño de la red, se debe decidir qué centros de distribución abrir de los *N* posibles centros para servir a los *M* clientes. Además de tomar en cuenta las decisiones de inventario, que implica el tamaño de los lotes de cada centro de distribución para que los costos sean mínimos y abastecer la demanda de los clientes.

Para poder garantizar el nivel de servicio de los clientes y minimizar la cantidad de instalaciones y costos de inventario, es por lo que se está tratando de buscar la mejor solución al problema. El nivel de servicio a los clientes es expresado como un mínimo nivel del stock que se debe mantener en los *CD* disponibles.

El problema implica varias decisiones estratégicas, las cuales se deben tomar para diseñar diferentes tipos de redes de distribución y poder elegir la mejor con respecto a los objetivos que se deban cumplir o satisfacer.

4.2 Formulación del Modelo

Para cualquier instalación i, se realiza una periódica revisión, con una política (Q_i, RP_i) , para abastecer una demanda que se comporta de manera estocástica, siendo Q_i la cantidad fija ordenada a los proveedores para abastecer el inventario cuando cae desde el punto de reorden RP_i. Se asume una demanda de tipo estocástica con una media D_i (unidades de producto por unidad de tiempo) y una varianza U_i, para cada planta i. Éstos últimos corresponden a la suma de las demandas medias y varianzas de los clientes asignados al centro i.

También se asume que habrá una demora de un LT_i, la cual es el tiempo que se demora una orden solicitada en llegar a la planta i (Miranda, 2004).

La evolución del nivel de inventario para cada centro de distribución i es mostrada en la ilustración 4.2.1.

Se puede notar que cuando el nivel de inventario cae desde el punto de reorden RP_i una cantidad Q_i es ordenada, la cual es recibida en un tiempo LT_i más tarde.

Una vez que la cantidad ha sido ordenada, aparece una diferencia entre el nivel de inventario que se tiene en ese momento, representado por la línea continua y el inventario que se debería tener, representado por la línea segmentada. Esta diferencia desaparece cuando la cantidad ordenada llega al centro de distribución *i*.

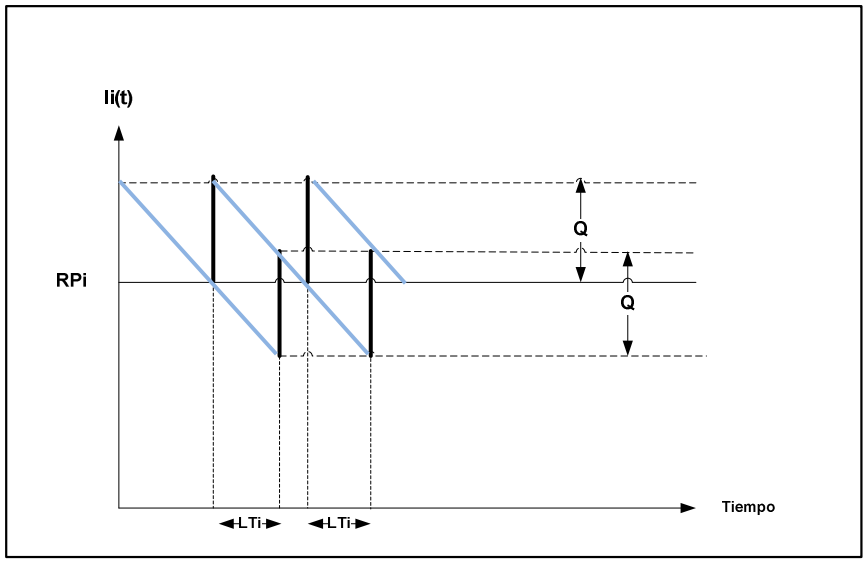


Ilustración 4.2.1 Evolución del nivel de inventario $I_i(t)$ del centro *i*

Esta política de inventario no penaliza la demanda insatisfecha. Sino que establece un nuevo punto de reorden RP_i. Una vez que la cantidad ordenada ha llegado debería cubrir la demanda durante un tiempo LT_i. La probabilidad 1-α es la de satisfacer la demanda durante el LT_i.

La restricción dada para el nivel de servicio se puede expresar de la siguiente manera:

$$\Pr ob\left(D\left(LT_i\right) \leq RP_i\right) = 1 - \alpha \quad (2)$$

Donde $D(LT_i)$ es la demanda aleatoria dado un tiempo para cada centro de distribución *i*.

Si se considera una demanda con una Distribución Normal, RP_i se puede determinar de la siguiente manera:

$$P_i = D_i \times LT_i + Z_{1-\alpha} \times SD_i \times \sqrt{LT_i} \quad (3)$$

Donde $Z_{1-\alpha}$ es el valor de la distribución Normal Estándar, la cual tiene una probabilidad acumulada de $1-\alpha$. Este parámetro es asumido fijo para toda la red, determinando un nivel de servicio uniforme para el sistema, el cual se denomina *K*.

Una cantidad fija Q_i es ordenada cuando el nivel de inventario cae por debajo del punto de reorden RP_i.

El costo de almacenamiento por unidad de producto en una unidad de tiempo para una instalación *i*, está dado por HC_i ($/unidad-día). Entonces, la tasa media del costo de holding para una instalación *i*, basada en la expresión (1), puede ser escrita como:

$$HC_i \times K \times \sqrt{LT_i} \times \sqrt{U_i} + HC_i \times \frac{Q_i}{2} \quad (4)$$

El primer término en (2) es la media de los gastos asociados con mantener el stock de seguridad de una instalación $i \times \left(K \times \sqrt{LT_i} \times \sqrt{U_i}\right)$, el segundo término en la expresión (2) es la media de los gastos incurridos en holding (almacenamiento) de la cantidad Q_i, la cual es utilizada por el inventario para cubrir la demanda entre dos órdenes sucesivas. Esto es, si OC_i es el costo de orden de la instalación *i*, RC_i es el costo de transporte, entre la

planta central y la instalación *i*, y TP_i es el lapso de tiempo entre dos órdenes consecutivas para el sitio *i*, la operación de costo durante este período está dada por:

$$RC_i \cdot Q_i + OC_i + \left(HC_i \cdot K \cdot \sqrt{LT_i} \cdot \sqrt{U_i} + HC_i \cdot \frac{Q_i}{2} \right) \cdot TP_i \qquad (5)$$

Entonces, si dividimos (3) por TP_i (el cual es igual a $\frac{Q_i}{D_i}$), la tasa del costo incurrido por el sitio *i* esta dado por la siguiente expresión:

$$\left(RC_i + \frac{OC_i}{Q_i} \right) \cdot D_i + HC_i \cdot K \cdot \sqrt{LT_i} \cdot \sqrt{U_i} + HC_i \cdot \frac{Q_i}{2} \qquad (6)$$

Ahora consideraremos las siguientes variables:

Z_i : Toma el valor 1, si la instalación *i* es abierta, y 0 en caso contrario.

Y_{ij} : Toma el valor 1, si la instalación *i* abastece al cliente *j*, y 0 en caso contrario.

Si d_j es la media de la demanda del cliente *j*, el total de la tasa del costo operacional para toda la red puede ser escrita como:

$$\sum_{i=1}^{N} \left(HC_i \cdot K \cdot \sqrt{LT_i} \cdot \sqrt{U_i} + HC_i \cdot \frac{Q_i}{2} \right) + \sum_{i=1}^{N} \sum_{j=1}^{M} \left(RC_i + \frac{OC_i}{Q_i} \right) \cdot d_j \cdot Y_{ij} \qquad (7)$$

En la última expresión, se asume la siguiente relación:

$$\sum_{j=1}^{M} d_j \cdot Y_{ij} = D_i \qquad \forall i = 1,...,N \qquad (8)$$

la cual representa la media de la demanda servida por las instalaciones a los clientes. Más aun, si TC_{ij} es el costo de transporte entre la planta i y el cliente j, el costo asociado a la red completa es:

$$\sum_{i=1}^{N}\sum_{j=1}^{M} TC_{ij} \cdot d_j \cdot Y_{ij} \qquad (9)$$

Entonces, reemplazando la expresión (7) en el segundo término de la expresión (5), la tasa del costo de la red puede ser escrita de la siguiente manera:

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\left(TC_{ij} \cdot RC_i + \frac{OC_i}{Q_i} \right) \cdot d_j \cdot Y_{ij} \qquad (10)$$

Finalmente, se consideran los siguientes parámetros:

F_i : Costo de fijo para la instalación *i.*

TH : Horizonte de planeación.

Si consideramos el costo de instalación fijo para cada planta, el costo total de sistema es:

$$\sum_{i=1}^{N} F_i \cdot Z_i + TH \cdot \sum_{i=1}^{N} HC_i \cdot K \cdot \sqrt{LT_i} \cdot \sqrt{U_i} + TH \cdot \sum_{i=1}^{N} HC_i \cdot \frac{Q_i}{2} + \sum_{i=1}^{N}\sum_{j=1}^{M}\left(TC_{ij} + RC_i + \frac{OC_i}{Q_i} \right) \cdot d_j \cdot Y_{ij} \quad (11)$$

TH permite la suma de los costos de instalaciones incurridos durante el horizonte de planeación, y la tasa de costo incurrido por toda la red. Más aún, *TH* puede ser reemplazada por un factor, en este caso se considerará *TH*=1 (Miranda, 2004).

Si la ecuación (9) se deriva y se iguala a cero se obtiene la siguiente expresión:

$$TH \cdot \frac{HC_i}{2} - TH \cdot \frac{OC_i}{Q_i^2} \cdot \sum_{j=1}^{M} d_j \cdot Y_{ij} = 0 \qquad (12)$$

De la expresión (10) se obtiene lo siguiente:

$$Q_i = \sqrt{\frac{2 \cdot OC_i \cdot D_i}{HC_i}} \qquad \forall i = 1,...,N \qquad (13)$$

Reemplazando la expresión (11) en (10) queda la siguiente función objetivo (Miranda, 2004):

$$Min \sum_{i=1}^{N} F_i \cdot Z_i + \sum_{i=1}^{N} TH \cdot HC_i \cdot Z_{1-\alpha} \cdot \sqrt{LT_i} \cdot \sqrt{U_i} + \sum_{i=1}^{N} TH \cdot \sqrt{2 \cdot HC_i \cdot OC_i} \cdot \sqrt{D_i} + \sum_{i=1}^{N} \sum_{j=1}^{M} TH \cdot (TC_{ij} + RC_i) \cdot d_j \cdot Y_{ij} \qquad (14)$$

F_i : Costo de fijo del centro de distribución *i*.

Z_i : Variable que toma el valor 1 si el centro *i* está abierto y cero en el otro caso.

TH : Horizonte de planeación.

HC_i : Costo de almacenamiento del centro de distribución *i*.

1-α : Nivel de servicio asociado al nivel de stock de seguridad de cada centro.

LT_i : Tiempo que se demora en llegar una orden para el centro de distribución *i*.

U_i : Varianza total del centro de distribución *i*.

OC_i : Costo de ordenamiento del centro de distribución *i*.

D_i : Demanda total del centro de distribución *i*.

TC_{ij} : Costo de asignación del centro de distribución *i* al cliente *j*.

RC_i : Costo de transporte de la planta central al centro de distribución *i*.

d_j : Media de la demanda del cliente *j*.

Y_{ij} : Variable que toma el valor 1 si el centro de distribución *i* sirve al cliente *j*, y cero en el otro caso.

N : Cantidad de centros de distribución de la red.

M : Cantidad de clientes de la red.

Cap_i : Capacidad de almacenamiento del centro de distribución i.

u_j : Varianza de la demanda del cliente j.

Sujeta a las siguientes restricciones (Miranda, 2004):

$$\sum_{i=1}^{N} Y_{ij} = 1 \quad \forall j = 1,...,M \qquad (15)$$

$$\sum_{j=1}^{M} d_j . Y_{ij} \leq Cap_i . Z_i \qquad \forall i = 1,...,N \qquad (16)$$

$$\sum_{j=1}^{M} d_j . Y_{ij} = D_i \qquad \forall i = 1,...,N \qquad (17)$$

$$\sum_{j=1}^{M} u_j . Y_{ij} = U_i \qquad \forall i = 1,...,N \qquad (18)$$

$$Z_i, Y_{ij} \in \{0,1\} \quad \forall i = 1,...,N \quad \forall j = 1,...,M \qquad (19)$$

La ecuación (13) asegura que cada cliente es servido por una instalación. La ecuación (14) asegura que la capacidad máxima de las instalaciones no se exceda (solo si la instalación está localizada en un lugar), la ecuación (15) calcula la media de la demanda total que es abastecida por la instalación i. La ecuación (16) calcula la varianza de la demanda total que es abastecida por la instalación i. Implícitamente se considera que la demanda es independientemente distribuida hacia los clientes. Finalmente, (17) son estados de integridad para las variables Y_{ij} y Z_i.

CAPÍTULO 5

Resolución de CFLP aplicando AG

5.1 AG aplicado al Diseño de Redes de Distribución

AG es un algoritmo, que aplicado a algún problema, transforma un conjunto de individuos o población, cada uno de los cuales tiene asociado un valor de adaptación, que se nombrará como *Fitness*, en una nueva población (la siguiente generación) utilizando una serie de operadores para encontrar al más adaptado.

Es por esto que para aplicar este algoritmo al problema de diseño de redes de distribución, se realizará el estudio de cómo este mismo algoritmo resuelve y puede encontrar al individuo más adaptado en el problema CFLP, es decir, para un caso particular del DNDRP, ya que CFLP es equivalente a una simplificación DNDRP cuando $HC=0$, para todo $i=1,...,N$, cuando la varianza de la demanda no afecta al costo del sistema total (Miranda, 2005).

Más aún, cuando se evalúa la solución obtenida por CFLP en la ecuación (12), es equivalente a solucionar secuencialmente las decisiones de inventario: el punto de reorden y la cantidad de ordenamiento para cada centro pueden ser obtenidas a partir de la configuración entregada por CFLP.

5.2 Análisis del Prototipo

La función objetivo para el prototipo, basada en el modelo CFLP dada por (Leung, 1986) es:

$$Min \sum_{i=1}^{N} F_i . Z_i + \sum_{i=1}^{N} \sum_{j=1}^{M} d_j . TC_{ij} . Y_{ij} \qquad (20)$$

$$Sujeto \quad a: \quad \sum_{j=1}^{M} d_j Y_{ij} = Cap_i . Z_i \quad \forall i = 1,...,N \qquad (21)$$

Donde:

N	:	Cantidad centros de distribución que se pueden abrir para satisfacer la demanda de los clientes.
M	:	Cantidad de clientes de la red.
F_i	:	Costo fijo de instalación para cada instalación i.
TC_{ij}	:	Costo de asignación de una instalación i a un cliente j.
Z_i	:	Toma el valor 1 si se abre la instalación i y 0 si se mantiene cerrada.
d_j	:	Demanda de un cliente j.
Cap_i	:	Capacidad máxima de una instalación i.
Y_{ij}	:	Toma el valor 1 si la instalación i sirve al cliente j, y 0 en caso contrario.

La expresión (19) permite que no se sobrepase la capacidad máxima de cada centro de distribución.

Criterio de Codificación

❖ Representación

Las soluciones del espacio de búsqueda deben ser codificadas de manera que se puedan realizar las operaciones de selección, cruzamiento y mutación, además de definir como se evaluarán estas posibles soluciones.

Es por esto que para codificar las soluciones, se realizó de la siguiente manera:

Para los clientes, se tomó una codificación entera positiva, que va desde 1 al número máximo de clientes dado como parámetro de entrada, es decir, *M*.

Ejemplo: Sea $M = 10$, los clientes enumerados serán:

Ilustración 5.2.1 Ejemplo de codificación de los Clientes

Para los centros de instalaciones, se utilizó un número entero positivo que va desde 1 al número de instalaciones dado como parámetro de entrada, es decir, *N*.

Ejemplo: Sea $N = 5$, los centros de distribución enumerados serán:

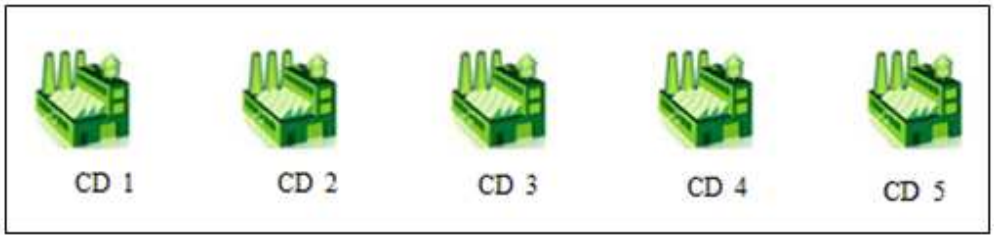


Ilustración 5.2.2 Ejemplo de codificación de los CD

(Sourirajan, 2008), (Itaim Ananias, 2005) entre otros, al aplicar AG, la codificación que utilizan es la representación de tipo binaria, pero en este trabajo de investigación se utilizó un tipo de codificación a través de representación entera positiva.

Un individuo sería representado de la siguiente manera:

Individuo o Cromosoma

1	2	3	4	5	6	7	8	9	10
5	3	1	4	4	3	1	3	4	1

Ilustración 5.2.3 Ejemplo de Individuo o cromosoma para CFLP

Donde el valor de la celda representa la instalación abierta, y el número de la posición de las columnas representa a los clientes.

Una población sería representada de la siguiente manera:

Población

	1	2	3	4	5	6	7	8	9	10
1	5	3	1	4	4	3	1	3	4	1
2	4	3	2	5	5	1	2	1	1	2
3	3	2	3	5	2	5	2	5	3	5
4	2	3	2	4	5	1	2	3	2	5
5	4	4	2	3	2	1	2	2	2	2
6	1	5	1	2	2	5	1	3	2	5

Ilustración 5.2.4 Ejemplo de Población para CFLP

Donde el número de la posición de las filas representa a cada individuo de la población, el número de la posición de las columnas representa a los clientes, y el valor de las celdas a los centros abiertos.

❖ Operadores

Para resolver el problema, se utilizan los operadores de selección, cruzamiento y mutación. El desempeño del algoritmo genético depende del valor de sus parámetros principales: *tamaño de la población, tasa de selección, tasa de mutación y tasa de cruce*.

Los valores escogidos para estos parámetros, en el CFLP son:

- *Tamaño de población* = 500 Este valor se puede ir modificando, para encontrar un valor óptimo en el tamaño de la población, esto lo determina el valor del fitness, es decir, cuando el fitness ya no tienen grandes variaciones.
- *Tasa de Selección* = 0.7 La probabilidad de selección se determina según el rango que se desea asignar a la elección de un fitness de mejor adaptación, es decir, para un problema de minimización, como en este caso, se le debe asignar un mayor rango de selección a los individuos con un fitness menor que a uno con fitness mayor.
- *Tasa de cruce* = 0.9 La probabilidad de cruce debe ser alta, ya que es el operador más importante.
- *Tasa de mutación* = 0.2 La probabilidad de mutación es baja debido a que este operador es el que le proporciona variabilidad al espacio de posibles soluciones.

❖ Selección del individuo

(Arapoglu, 2001), (Itaim Ananias, 2005) utilizan el operador de selección por ruleta, obteniendo buenos resultados. En este trabajo se realizaron pruebas para ver qué tipo de selección utilizar, obteniendo los mejores resultados con la Selección por torneo. Las pruebas realizadas para la selección por ruleta no se acercaron a la óptima solución esperada, ni para grandes tamaños de población ni para grandes números de iteraciones,

obteniendo resultados que difieren en un 56% del resultado óptimo, mientras que para la selección por torneo solo difiere en un 0,1%.

Para implementar la selección por torneo se fue seleccionando parejas de individuos al azar desde la población inicial, luego para elegir un individuo de los dos seleccionados, se debe considerar la probabilidad de selección definida.

Ejemplo: Se tienen los siguientes individuos elegidos al azar de la población inicial y con una probabilidad definida anteriormente P_s=0.7, entonces:

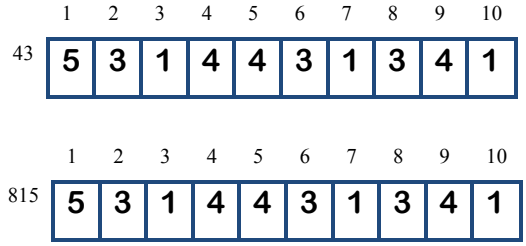


Ilustración 5.2.5 Ejemplo de Selección para CFLP

El fitness del individuo número 43 de la población es $Fitness_{43}=1.456.432,275$ y el fitness del individuo número 815 de la población es $Fitness_{815}=1.012.456,025$.

Luego se debe generar un valor aleatorio entre [0,1], si este valor es menor que P_s, se selecciona el individuo con menor fitness, es decir, el individuo número 815, por el contrario, si el valor generado es mayor a P_s, entonces se selecciona el individuo número 43.

Una vez seleccionado este individuo, se inserta en una población temporal que irá almacenando estos individuos hasta completar un número determinado de individuos. Para este problema se considera un número igual a la mitad del tamaño de la población, es decir 250 individuos.

❖ Cruzamiento

El operador de cruce utilizado en este problema, aplica la técnica de cruce básico, es decir, seleccionando un punto al azar de la cadena.

Ejemplo: Se seleccionan al azar dos individuos de la población temporal.

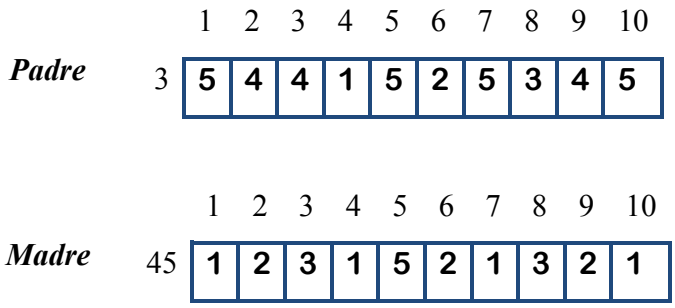


Ilustración 5.2.6 Individuos seleccionados para el cruzamiento

Teniendo en cuenta el valor de probabilidad de cruce definido $P_c=0,9$, se debe generar un valor aleatorio entre [0,1] , si el valor generado es menor que P_c se realiza el cruzamiento de la siguiente manera:

Se debe generar un valor aleatorio *r* entre [1,*M*] , este valor cumple la tarea de ser el pivote o punto desde donde se realiza el cruzamiento, es decir, a partir de ese pivote se intercambian cada uno de los genes de los individuos.

Suponiendo que el valor generado es *r=5*, entonces:

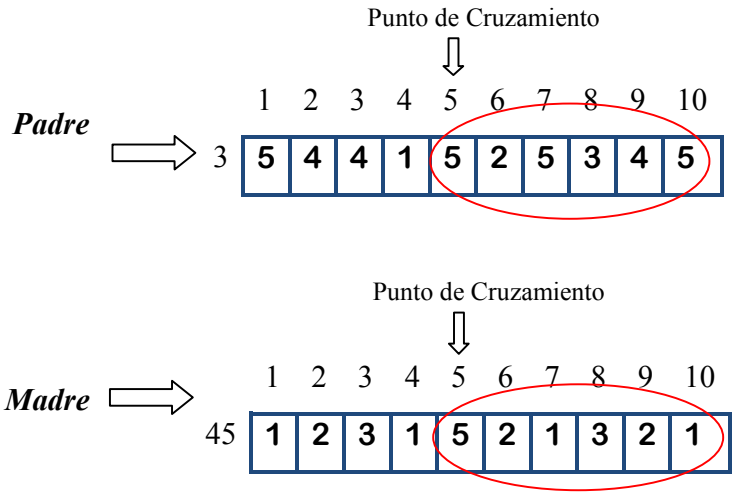


Ilustración 5.2.7 Intercambio de los genes entre los individuos

Las celdas desde ese pivote hacia la izquierda son intercambiadas, por lo que los dos individuos generados por el cruzamiento quedarían de la siguiente manera:

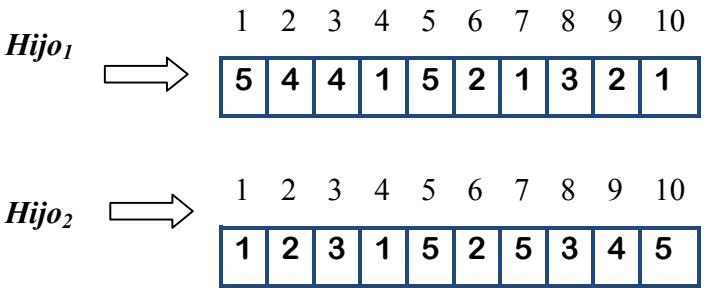


Ilustración 5.2.8 Individuos generados por el cruzamiento

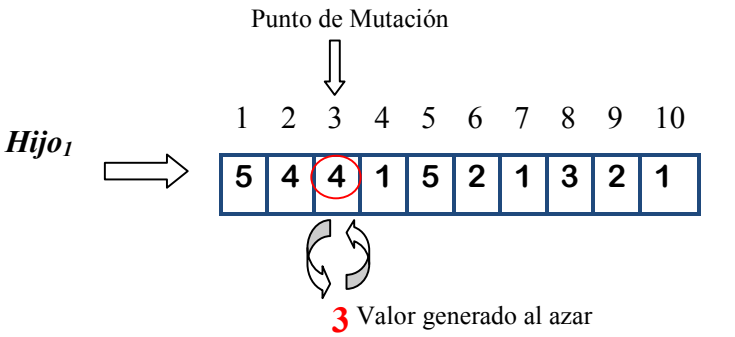
Estos nuevos individuos generados son insertados en la población temporal, si existen individuos que no cumplen con la probabilidad de cruce, estos pasan a la siguiente operación, es decir, mutación.

❖ Mutación

El operador de mutación utiliza la técnica de mutación puntual, en donde cada gen del individuo tiene una probabilidad de mutarse o no, que es calculada en cada pasada del operador de mutación, si se cumple tal probabilidad se introducirá un nuevo gen aleatorio.

Ejemplo: Se tienen los dos individuos generados por la operación de cruce anterior, y suponiendo que cumple con la probabilidad de mutación $P_M=0,2$, entonces:

Se debe generar un valor aleatorio r en $[1,M]$, para cada hijo, cuyo valor cumple con la tarea de pivote en donde se realiza la mutación.



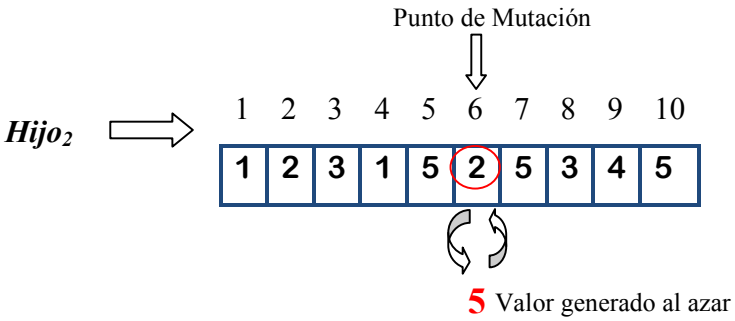


Ilustración 5.2.9 Ejemplo de mutación de los individuos seleccionados

El valor que toma la celda de dicha posición es generado aleatoriamente, según las restricciones de capacidad dadas. Suponiendo que los nuevos valores para las celdas seleccionadas son 3 y 5 respectivamente, entonces, los nuevos individuos quedarían de la siguiente manera:

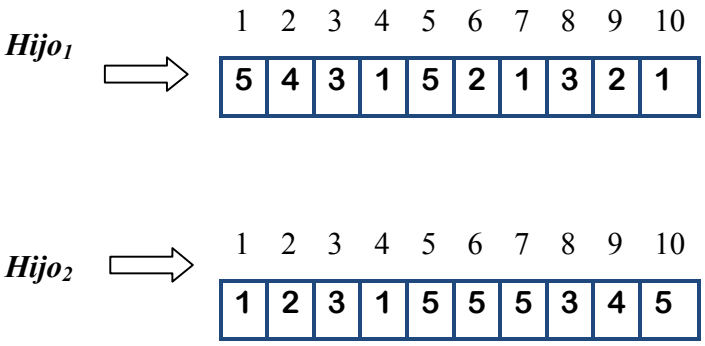


Ilustración 5.2.10 Ejemplo de individuos generados por la mutación

Estos nuevos individuos generados son insertados en la población temporal. Las operaciones de cruce y mutación se realizan en los individuos seleccionados hasta que se complete la población temporal. El tamaño de esta población debe ser igual al tamaño de la población inicial, es decir, deben completar los 500 individuos.

❖ Función Objetivo

La función objetivo de CFLP (Leung, 1986) está dada por:

$$Min\sum_{i=1}^{N} F_i \cdot Z_i + \sum_{i=1}^{N}\sum_{j=1}^{M} d_j \cdot TC_{ij} \cdot Y_{ij}$$

Para definir la función fitness, que es la encargada de ver el nivel de adaptación de cada individuo de la población, se tomó en cuenta la función objetivo del CFLP, por lo tanto, el valor arrojado por la función fitness calculada para cada individuo es la misma utilizada para minimizar el problema.

❖ Criterio de Parada

El criterio de parada de un algoritmo genético determina cuando se considera resuelto el problema sin que sea necesario disponer, en una situación intermedia, de información de lo cerca que se está de solucionarlo. Los criterios más corrientes se refieren a un límite en el número de iteraciones, movimientos, operaciones elementales o tiempo de cómputo total.

Para determinar el criterio de parada, es decir, número de generaciones del algoritmo para este trabajo de título, es que se experimentó con las iteraciones, es decir, el algoritmo se detiene cuando la mejora en el resultado del fitness sea nula o muy pequeña con el transcurso de las generaciones.

❖ Criterio de tratamientos para individuos no factibles

El criterio de tratamiento de individuos no factibles, consiste en la forma en que se tratará a los individuos que no cumplan con las restricciones del problema, es decir, en este caso, con las restricciones de capacidad de los centros de distribución, y que cada cliente sea abastecido sólo por un centro de distribución. En este algoritmo genético se utilizó una función que arregla los individuos defectuosos, para que cumplan con las restricciones del problema.

❖ Datos de Entrada

Se utiliza para los casos de prueba, los archivos de datos que provee la OR Library.

Los datos de entrada son los siguientes:

N	:	Número de centros de distribución.
M	:	Número de clientes.
T	:	Tamaño de la Población.
Iter	:	Número de iteraciones.
P_s	:	Probabilidad de Selección.
P_c	:	Probabilidad de Cruzamiento.
P_m	:	Probabilidad de Mutación.

costoFijo[N] = Vector de largo N que almacena los costos fijos de instalación correspondientes a cada uno de los centros de distribución.

*costoAsignacion[M]*x*[N]* = Matriz de dimensión MxN que almacena el costo de asignación que existe entre un cliente y todos los centros de distribución.

capacidad[N] = Vector de largo N que almacena la capacidad máxima de almacenamiento correspondiente a cada uno de los centros de distribución.

demanda[M] = Vector de largo M que almacena la demanda correspondiente a los clientes que debe ser cubierta por el centro de distribución respectivo.

Este conjunto de datos tiene como objetivo ser utilizado como valor de entrada para calcular el costo total obtenido a través del modelo preliminar propuesto.

❖ Datos de Procesamiento

Para iniciar el programa se utilizan estructuras, una para la población de individuos inicial y temporal, y otra para el fitness calculado de cada individuo de cada población.

poblacion[T].fitness = Estructura de vectores de largo *T*, el cual almacena el fitness de cada población calculada para cada iteración, teniendo en cuenta todas las variables de la función objetivo.

poblacion[T]×[M].inicial = Estructura de matrices de dimensión *TxM*, la cual los individuos para cada población. Para la inicialización de esta matriz será llenada de forma aleatoria, tomando en cuenta el número de instalaciones con la que se cuenta para servir a los M clientes, además de las restricciones de capacidad.

poblacion[T]×[M].temporal = Estructura de matrices de dimensión *TxM*, la cual almacenará los individuos para cada población, donde la población para la siguiente iteración es derivada de la población inicial, a la cual se le aplican las operaciones de selección, cruzamiento y mutación, siempre y cuando se cumpla con las probabilidades exigida.

❖ Datos de Salida

Los resultados finales que se obtienen son mostrados un vector de un largo *M*, el cual representará las instalaciones abiertas que sirven a los *M* clientes. Además se muestra el fitness o resultado de la función objetivo generado por la población resultante.

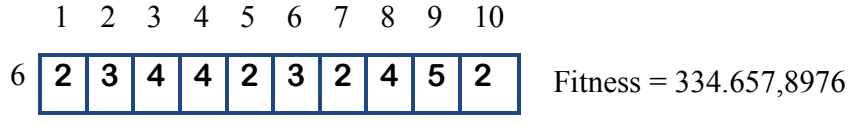


Ilustración 5.2.11 Ejemplo de una solución para CFLP

Donde la fila representa al individuo número 6, que es elegido de la población por tener el menor fitness, y las columnas son los 10 clientes, con sus respectivos centros de distribución que los abastecen representados por las celdas del vector.

Gráficamente la solución se puede representar de la siguiente manera:

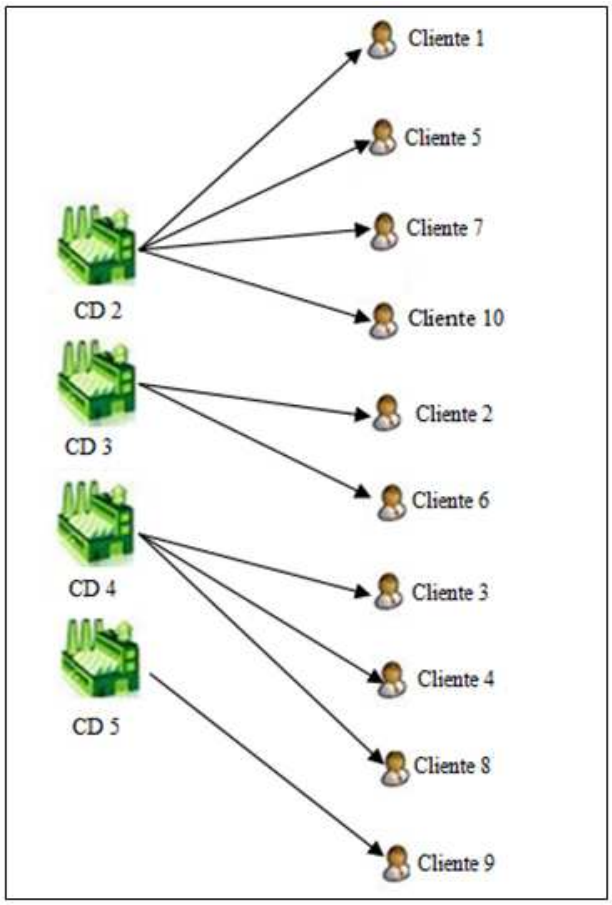


Ilustración 5.2.12 Ejemplo de la distribución de centros para CFLP

5.3 Algoritmo para CFLP

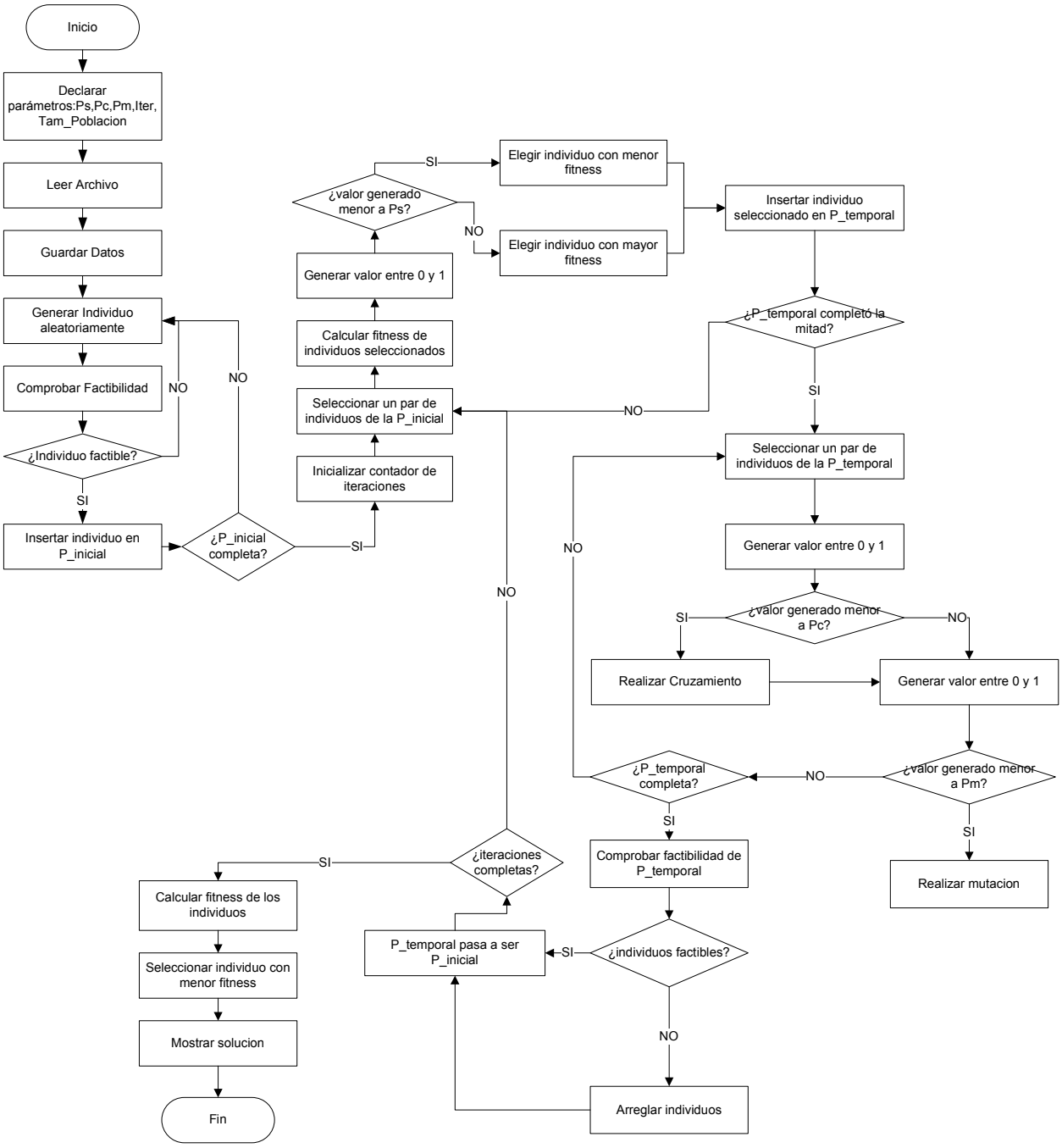


Ilustración 5.3.1 Diagrama de Flujo del Algoritmo Preliminar para CFLP

5.4 Discusión de Resultados para CFLP

Los resultados obtenidos son para la aplicación de AG a una simplificación del problema propuesto, es decir, CFLP. La implementación del algoritmo preliminar arrojó los siguientes resultados, según los siguientes parámetros de entrada:

Tamaño de la población = 1000 individuos.
Archivo de datos (OR-Library) = cap61.
Número de iteraciones = 100.

El gráfico N°1 muestra una ejecución del algoritmo imprimiendo el valor del mejor fitness en cada iteración, hasta finalizar las 100 definidas como parámetro de entrada. Se puede observar que el mejor fitness fue obtenido cerca de la iteración número 60, pero como el criterio de término aun no se ha cumplido, el programa sigue iterando hasta completar las 100.

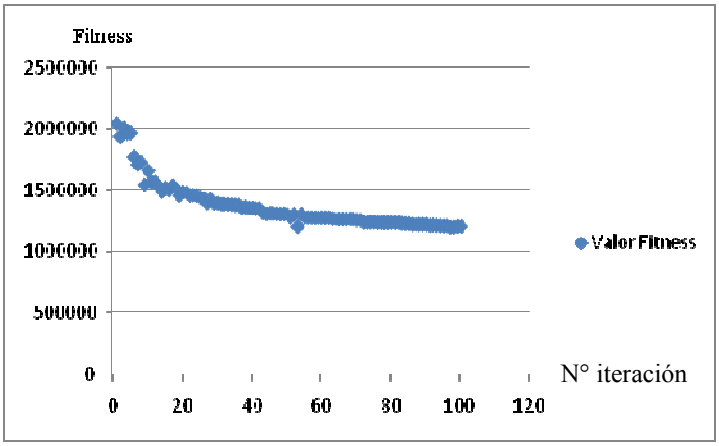


Ilustración 5.4.1 Gráfico N°1

Las pruebas realizadas tenían como objetivo ver el comportamiento de la curva de resultados de la función fitness, ya que puede ocurrir que los fitness encontrados fueran de manera muy irregular, es decir muy altos o muy bajos, pero con el gráfico N°1 se puede

observar que la curva se comporta de manera regular, es decir sin sobresaltos muy bruscos, esto hace que el algoritmo preliminar funciona de la manera esperada en lo que respecta a los valores esperados de la función fitness, además de mostrar que el algoritmo empieza a converger.

Los siguientes dos gráficos muestran cómo se comporta el algoritmo ejecutándolo 10 veces, para cada una de estas ejecuciones se modificó el parámetro de entrada, cantidad de iteraciones, para ver si al modificar en número de iteraciones, el algoritmo convergía a una solución mejor o se comporta igual independiente de este parámetro.

Las pruebas fueron realizadas para una red de 16 centros de distribución y 50 clientes, dados como parámetro de entrada según archivo cap61 de la OR-Library.

Tamaño de la Población = 1000 individuos.

Tabla 2 Resultados para una población de tamaño igual a 1000 individuos

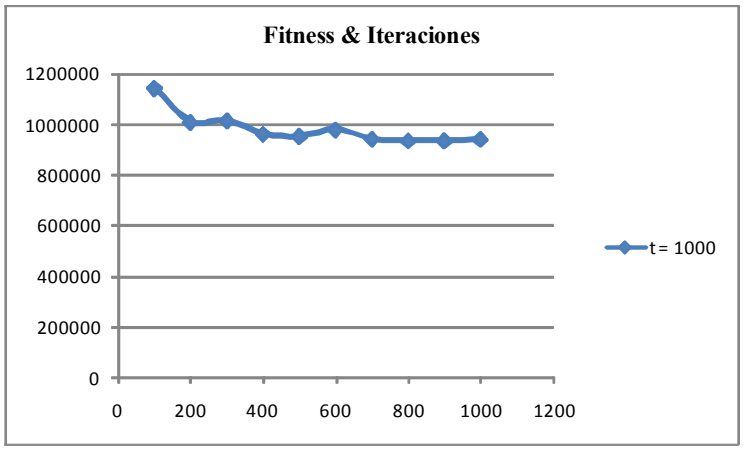


Ilustración 5.4.2 Gráfico N°2

Tamaño de la Población = 2000 individuos.

Tabla 3 Resultados para una población de tamaño igual a 2000 individuos

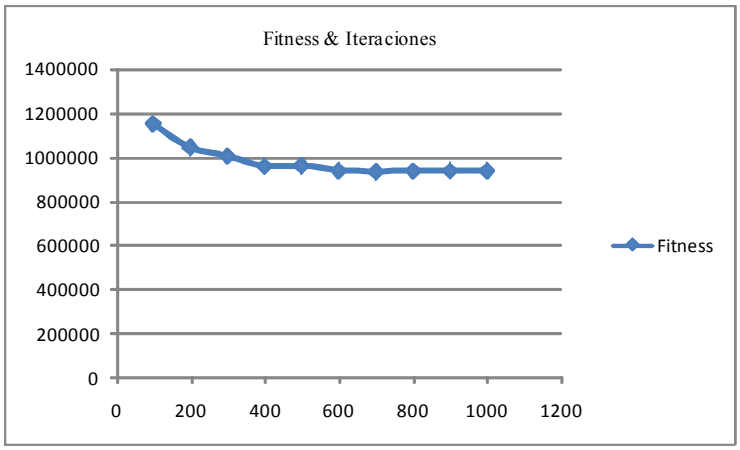


Ilustración 5.4.3 Gráfico N°3

Se puede apreciar que si bien el mínimo de gráfico N°2 fue obtenido en la ejecución con el número de iteración igual a 800 y en el gráfico N°3 en la iteración igual a 700, las siguientes ejecuciones arrojan resultados similares, es decir, existen pequeñas variaciones y el algoritmo empieza a converger, por lo que se puede deducir que al aumentar el número de iteraciones, el algoritmo llega a un punto donde las soluciones no tienen mayores diferencias unas de otras, y es ese el punto de término que se debe escoger.

Además en las tablas 2 y 3, se observa el tiempo que se demora en obtener el resultado, el tiempo es medido en segundos, y va en incremento debido al número de iteraciones que se requieren para cada ejecución.

Valor con AG	Valor OR-Library	% de diferencia	Tamaño de Población (individuos)
933.568,9	932.615,750	0,1	1000
933.422,575	932.615,750	0.09	2000
933.001,575	932.615,750	0.04	3000

Tabla 4 Tabla de comparación de resultados

En la tabla 4 se puede ver que los valores obtenidos utilizando AG difieren del óptimo (OR Library) hasta aproximadamente 0,1 %.

CAPÍTULO 6

Resolución de DNDRP aplicando AG

6.1 AG aplicado a DNDRP

A continuación se aplica los algoritmos genéticos al modelo de DNDRP tomando como base el prototipo realizado para CFLP, aplicando los cambios necesarios para lograr su implementación.

Criterio de Codificación

❖ **Representación**

El criterio de codificación fue la misma utilizada para el prototipo, CFLP. Para los clientes, la codificación que se tomó fue un número entero positivo que va desde 1 al número máximo de clientes dado como parámetro de entrada, es decir, *M*.

Para los centros de instalaciones, se utilizó un número entero positivo que va desde 1 al número de instalaciones dado como parámetro de entrada, es decir, *N*.

❖ Operadores

Para resolver el problema, se utilizan los operadores de selección, cruzamiento y mutación. El desempeño del algoritmo genético depende del valor de sus parámetros principales: tamaño de la población, tasa de selección, tasa de mutación y tasa de cruce. Estos operadores fueron implementados de la misma forma que se implemento para el prototipo, no sufrieron ningún cambio.

❖ Función Objetivo

La función objetivo para el modelo DNDRP dada por (Miranda, 2004) es:

$$Min\sum_{i=1}^{N} F_i \cdot Z_i + \sum_{i=1}^{N} TH \cdot HC_i \cdot Z_{1-\alpha} \cdot \sqrt{LT_i} \cdot \sqrt{U_i} + \sum_{i=1}^{N} TH \cdot \sqrt{2 \cdot HC_i \cdot OC_i} \cdot \sqrt{D_i} + \sum_{i=1}^{N}\sum_{j=1}^{M} TH \cdot (TC_{ij} + RC_i) \cdot d_j \cdot Y_{ij}$$

F_i : Costo de fijo del centro de distribución *i*.

Z_i : Variable que toma el valor 1 si el centro *i* está abierto y cero en el otro caso.

TH : Horizonte de planeación.

HC_i : Costo de almacenamiento del centro de distribución *i*.

1-α : Nivel de servicio asociado al nivel de stock de seguridad de cada centro de distribución

LT_i : Tiempo que se demora en llegar una orden para el centro de distribución *i*.

U_i : Varianza total del centro de distribución *i*.

OC_i : Costo de ordenamiento del centro de distribución *i*.

D_i : Demanda total del centro de distribución *i*.

TC_{ij} : Costo de asignación del centro de distribución *i* al cliente *j*.

RC_i : Costo de transporte de la planta central al centro de distribución *i*.

d_j : Media de la demanda del cliente *j*.

Y_{ij} : Variable que toma el valor 1 si el centro de distribución *i* sirve al cliente *j*, y cero en el otro caso.

N : Cantidad de centros de distribución de la red.

M : Cantidad de clientes de la red.

Cap_i : Capacidad de almacenamiento del centro de distribución i.

u_j : Varianza de la demanda del cliente j.

Para aplicar los AG se simplificó la expresión de la siguiente manera:

$$C_{ij} = TH \cdot \left(RC_i + TC_{ij}\right) d_j, \quad CL_i = TH \cdot \sqrt{2 \cdot HC_i \cdot OC_i}, \quad CS_i = TH \cdot HC_i \cdot Z_{1-\alpha} \cdot \sqrt{LT_i}$$

Por lo que la función objetivo descrita por (Miranda, 2004) queda dada por:

$$Min \sum_{i=1}^{N} F_i \cdot Z_i + \sum_{i=1}^{N} CS_i \cdot \sqrt{U_i} + \sum_{i=1}^{N} CL_i \cdot \sqrt{D_i} + \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij} \cdot Y_{ij} \qquad (22)$$

Para definir la función fitness, que es la encargada de ver el nivel de adaptación de cada individuo de la población, se tomó en cuenta la función objetivo del DNDRP simplificada, por lo tanto, el valor arrojado por la función fitness calculada para cada individuo fue la misma utilizada para minimizar el problema, es decir, la expresión(20).

❖ Criterio de Parada

Para determinar el criterio de parada, es decir, número de generaciones del algoritmo para este trabajo de investigación, es que se experimentó con las iteraciones, es decir, el algoritmo se detiene cuando éste converja hacia un valor.

❖ Criterio de Tratamiento de Individuos no Factibles

En este algoritmo genético se utilizó una función que arregla los individuos defectuosos, para que cumplan con las restricciones del problema (Miranda, 2004).

❖ Datos de Entrada

Se utiliza para los casos de prueba, archivos de datos que se utilizaron para el paper de (Miranda P., 2004).

Los datos de entrada serán los siguientes:

N	:	Número de centros de distribución.
M	:	Número de clientes.
T	:	Tamaño de la Población.
$Iter$	:	Número de iteraciones.
P_s	:	Probabilidad de Selección.
P_c	:	Probabilidad de Cruzamiento.
P_m	:	Probabilidad de Mutación.

Se utilizan las siguientes estructuras para el almacenamiento de los datos ingresados a través del archivo utilizado en el paper de (Miranda, 2004).

Datos con respecto a los clientes:

cliente[M].demanda = Estructura de vectores de largo *M*, el cual almacena la demanda de cada cliente.
cliente[M].varianza = Estructura de vectores de largo *M*, el cual almacena la varianza de cada cliente.

Datos con respecto a los centros de distribución:

instalación[N].costoFijo = Estructura de vectores de largo *N*, el cual almacena el costo fijo de cada centro de distribución.
instalación[N].CS = Estructura de vectores de largo *N*, el cual almacena el costo de stock de seguridad de cada centro de distribución.

instalación[N].CL = Estructura de vectores de largo *N*, el cual almacena el costo de almacenamiento y ordenamiento de cada centro de distribución.
instalación[N].capacidad = Estructura de vectores de largo *N*, el cual almacena la capacidad de cada centro de distribución.
instalación[M][N].asignacion = Estructura de matrices de dimensión Mx*N*, el cual almacena el costo asignación entre cada cliente y cada centro de distribución.

Este conjunto de datos tiene como objetivo ser utilizado como valor de entrada para calcular el costo total obtenido a través del modelo preliminar propuesto.

❖ Datos de Procesamiento

Una vez procesados los datos de entrada se necesita una estructura que almacene las demandas y varianzas totales para cada centro de distribución, de cada individuo generado:

instalacion[T][N].demandaTotal = Estructura de matrices de dimensión *TxN,* el cual almacena la demanda total para cada centro de cada individuo generado.
instalacion[T][N].varianzaTotal = Estructura de matrices de dimensión *TxN*, el cual almacena la varianza total para cada centro de cada individuo generado.

También se utilizaron estructuras para la población de individuos, y para el fitness calculado de cada individuo de cada población.

poblacion[T].fitness = Estructura de vectores de largo *T*, el cual almacena el fitness de cada población calculada para cada iteración, teniendo en cuenta todas las variables de la función objetivo.

poblacion[T]×[M].inicial = Estructura de matrices de dimensión *TxM*, la cual almacena las instalaciones abiertas que sirven a los *M* clientes para cada población. Para la inicialización de esta matriz será llenada de forma aleatoria, tomando en cuenta el número

de instalaciones con la que se cuenta para servir a los *M* clientes, además de las restricciones de capacidad.

poblacion[T]×[M].temporal = Estructura de matrices de dimensión *TxM*, la cual almacena las instalaciones abiertas que sirven a los *M* clientes para cada población, donde la población para la siguiente iteración es derivada de la población inicial, a la cual se le aplican las operaciones de selección, cruzamiento y mutación, siempre y cuando se cumpla con las probabilidades exigida.

❖ Datos de Salida

Los resultados finales que se van a obtener de la investigación son guardados en un vector de largo *M*, el cual representa las instalaciones abiertas que sirven a los *M* clientes. También se muestra el fitness o resultado de la función objetivo generado por la población resultante.

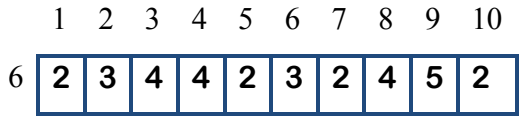


Ilustración 6.1.1 Ejemplo de solución para DNDRP

Donde la fila representa al individuo número 6 que es elegido de la población por tener el menor fitness, y las columnas son los 10 clientes, con sus respectivos centros de distribución que los abastecen representados por las celdas del vector.

6.2 Algoritmo para DNDRP

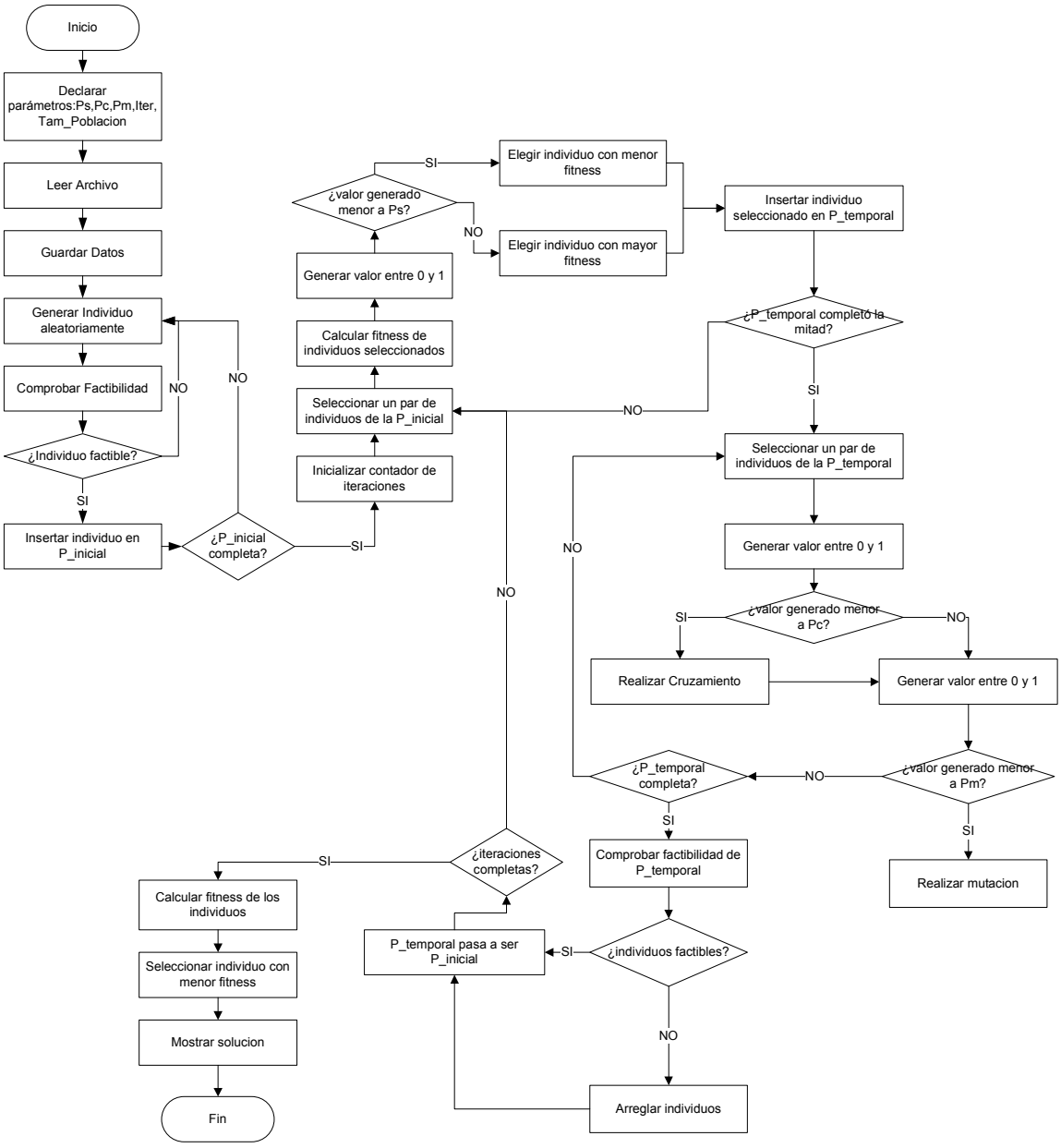


Ilustración 6.2.1 Diagrama de Flujo de Algoritmo para DNDRP

6.3 Discusión de Resultados para DNDRP

La implementación de la solución tanto para el prototipo como para DNDRP fue desarrollado con el lenguaje de programación C, siendo ejecutado en un computador con procesador Core Duo de 1.5GHz – 1.5GHz, con memoria de 2GB y un disco duro de 140 GB.

Las tablas 5, 6, 7 y 8 muestran los resultados obtenidos en el modelo planteado en (Miranda, 2004) para DNDRP y poder compararlos con los resultados obtenidos aplicando AG. Los resultados fueron obtenidos con Lingo, Relajación Lagrangeana y AG, los cuales son comparados con los obtenidos por SDND (*Sequential Distribution Network Design*) para ver cómo cambia el diseño y resultados al incorporar risk pooling, es decir, como cambia la red y los costos asociados a ella.

SDND es el problema CFLP con un enfoque secuencial, el cual consiste en evaluar la configuración entregada por CFLP en la función objetivo de DNDRP, donde sólo se consideran los costos de transporte e instalación, esto hace que la solución entregada sea la cota inferior para comparar con los demás resultados obtenidos.

Para la obtención de los resultados en las tablas 5, 6, 7 y 8 se utilizó un nivel de servicio del 50, 75, 90 y 97,5% respectivamente.

Resumen de Soluciones	SDND	DNDRP - LINGO		DNDRP - LR		DNDRP - AG	
	Costo Total	Costo Total	% Save	Costo Total	% Save	Costo Total	% Save
Caso X_-50HC	600891	600891	0,00%	605891	-0,83%	605716	0,80%
Caso X_-25HC	624418	624418	0,00%	624521	-0,02%	625370	-0,14%
CasoX	644260	639400	0,75%	644260	0,00%	641784	0,39%
CasoX_+25HC	661725	653912	1,18%	661725	0,00%	656315	0,82%
CasoX_+50HC	677493	667009	1,55%	677297	0,03%	669454	1,20%

Tabla 5 Resultados obtenidos considerando un nivel de servicio del 50%

Resumen de Soluciones	SDND	DNDRP - LINGO		DNDRP - LR		DNDRP - AG	
	Costo Total	Costo Total	% Save	Costo Total	% Save	Costo Total	% Save
Caso X_-50VC	719798	702188	2,45%	719552	0,03%	685035	5,07%
Caso X_-25VC	757348	733394	3,16%	754136	0,42%	720337	5,14%
CasoX	795126	764802	3,81%	776477	2,35%	768528	3,46%
CasoX_+25VC	832805	803743	3,49%	809202	2,83%	811317	2,65%
CasoX_+50VC	870593	833453	4,27%	843010	3,17%	852687	2,10%

Tabla 6 Resultados obtenidos considerando un nivel de servicio del 75%

Resumen de Soluciones	SDND	DNDRP - LINGO		DNDRP - LR		DNDRP - AG	
	Costo Total	Costo Total	% Save	Costo Total	% Save	Costo Total	% Save
Caso X_-50VC	787831	767490	2,58%	783668	0,53%	743304	5,99%
Caso X_-25VC	859201	818078	4,79%	832812	3,07%	802420	7,08%
CasoX	931003	880998	5,37%	898893	3,45%	882627	5,48%
CasoX_+25VC	1002618	937329	6,51%	961110	4,14%	971854	3,17%
CasoX_+50VC	1074440	993807	7,50%	1025370	4,57%	1055849	1,76%

Tabla 7 Resultados obtenidos considerando un nivel de servicio del 90%

Resumen de Soluciones	SDND	DNDRP - LINGO		DNDRP - LR		DNDRP - AG	
	Costo Total	Costo Total	% Save	Costo Total	% Save	Costo Total	% Save
Caso X_-50VC	863872	828161	4,13%	860043	0,44%	809453	6,68%
Caso X_-25VC	973042	914062	6,06%	935273	3,88%	910895	6,82%
CasoX	1082874	1000429	7,61%	1032900	4,61%	1028371	5,30%
CasoX_+25VC	1192421	1086592	8,88%	1130880	5,16%	1119760	6,49%
CasoX_+50VC	1302283	1172978	9,93%	1218510	6,43%	1213294	7,33%

Tabla 8 Resultados obtenidos considerando un nivel de servicio del 97,5%

En la tabla 5 se puede ver que los resultados obtenidos con AG son muy cercanos a los obtenidos con SDND, pero no mejores a los obtenidos con LINGO, es decir, se acercan a la cota inferior pero no mejoran los resultados como lo hace LINGO.

Cuando se utiliza un nivel de servicio del 75%, como lo muestra la tabla 6, los resultados obtenidos mejoran considerablemente, ya que para todos los casos de prueba se obtuvieron mejores resultados que SDND, y para algunos de ellos, hasta resultaron mejores que utilizando LINGO y LR.

En la tabla 7 y 8, también se obtuvieron mejoras a los resultados obtenidos por SDND y en algunos casos mejores que DNDRP-LINGO, por lo tanto los resultados obtenidos aplicando AG a DNDRP fueron cercanos, y más aún, a veces mejores que lo esperado.

El tiempo de respuesta para los resultados obtenidos aplicando AG para los 4 casos de nivel de servicio, fue considerando un tradeoff entre la cantidad de iteraciones y el tamaño de la población, arrojando como resultado un tiempo de aproximadamente 3 segundos, tomando una población de 100 individuos y 30 iteraciones para lograr su resolución.

La ilustración 6.3.1 muestra la media de la reducción de costos cuando se aplica un coeficiente de variación aplicado a la varianza de la demanda tomando en cuenta los niveles de servicios (75%,90%, 97,5%) para el problema DNDRP aplicando AG. Se debe notar que cuando existe un incremento en la varianza de la demanda se produce un incremento en los costos, como también incrementan los costos cuando aumenta el nivel de servicio entregado a los clientes.

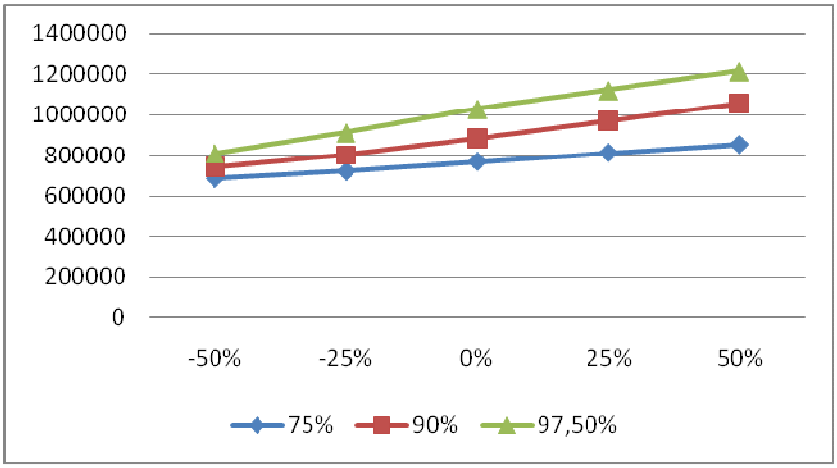


Ilustración 6.3.1 Media en la reducción de los costos de DNDRP aplicando AG

En la ilustración 6.3.2 se muestra cómo se comporta la curva de los resultados obtenidos, cuando el tamaño de la población se mantiene constante e igual a 100 individuos, y solo se va modificando el número de iteraciones. Se puede notar que la curva converge en pocas iteraciones.

En la ilustración 6.3.3 se demuestra que al converger en pocas iteraciones, el tiempo transcurrido para obtener la solución final no sobrepasa 1[seg], es decir se obtiene un buen tiempo de respuesta.

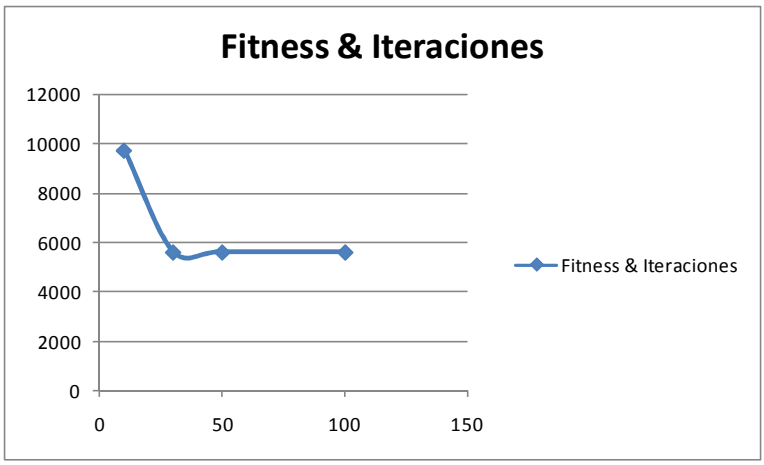


Ilustración 6.3.2 Gráfico "Fitness & Iteraciones" considerando 100 individuos

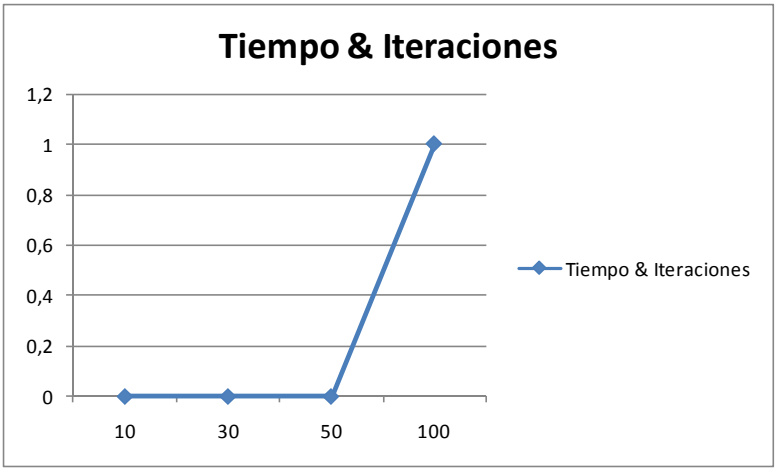


Ilustración 6.3.3 Gráfico "Tiempo & Iteraciones" considerando 100 individuos

Al aumentar el tamaño de la población a 500 individuos, para aumentar las posibles soluciones, es que en la ilustración 6.3.4 se muestra cómo se comporta la curva de los resultados.

Se puede ver que la curva ya no muestra grandes diferencias en comparación a la población igual a 100 individuos, ya que la probabilidad de seleccionar individuos con mejor fitness desde una población con más individuos, si bien aumenta, existe una probabilidad de que los individuos se repitan con mayor periodicidad.

En la ilustración 6.3.5 se demuestra que al converger rápidamente, el tiempo transcurrido para obtener la solución final no sobrepasa los 2[seg].

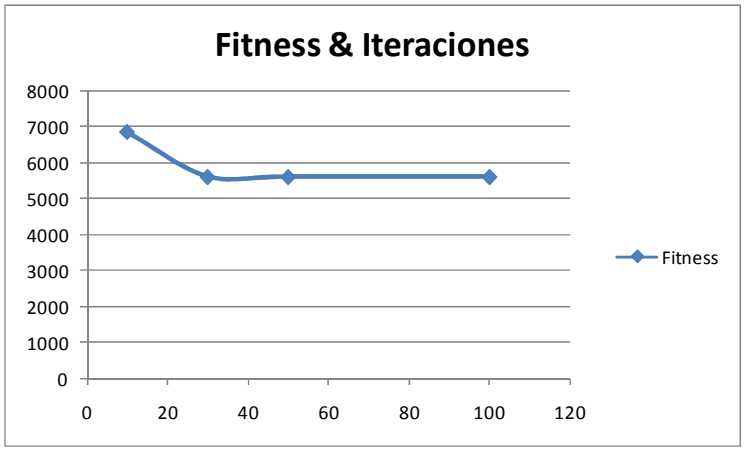


Ilustración 6.3.4 Gráfico "Fitness & Iteraciones" considerando 500 individuos

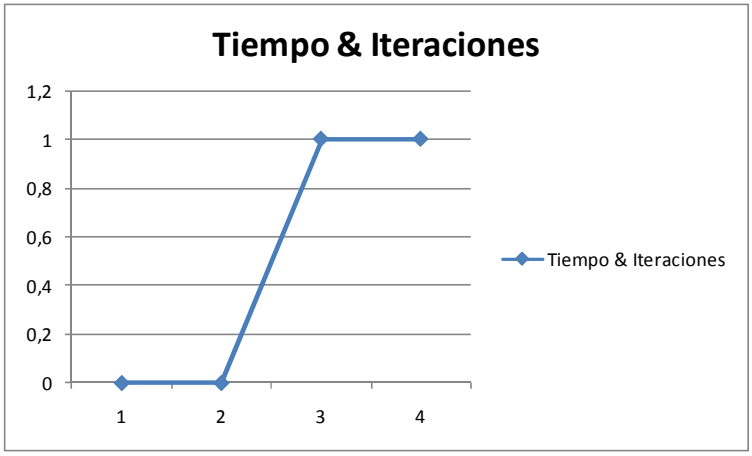


Ilustración 6.3.5 Gráfico "Tiempo & Iteraciones" considerando 500 individuos

Ahora considerando fijo el número de iteraciones, uno de los parámetros de entrada más importante, se grafica en la ilustración 6.3.6 como se comportan los resultados obtenidos cuando se va aumentando el tamaño de la población.

Los resultados convergen a la solución final para tamaños de la población cercanos a los 100 individuos, demorándose 0 [seg] para lograr éstos resultados. El comportamiento del tiempo para lograr la solución se muestra en la figura 6.3.7.

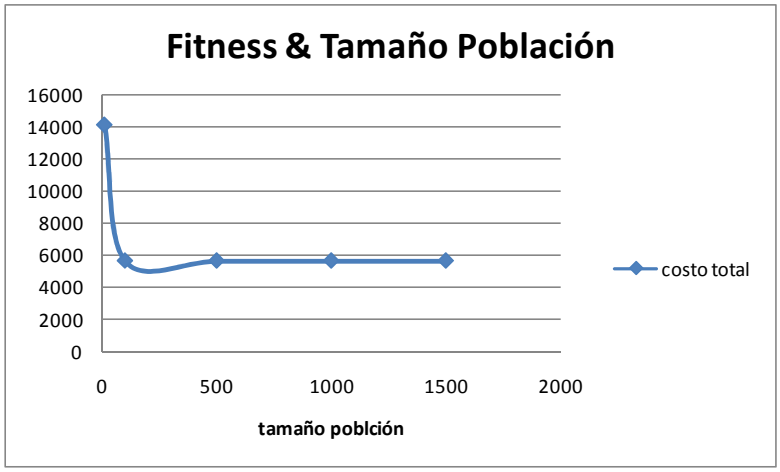


Ilustración 6.3.6 Gráfico "Fitness & Población" considerando 30 iteraciones

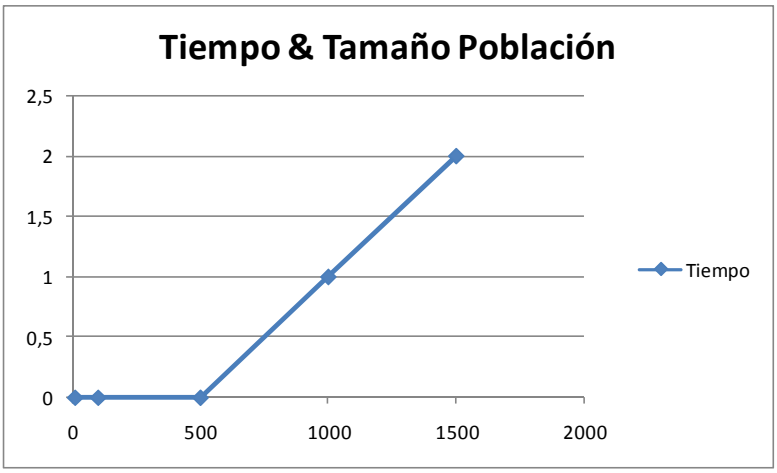


Ilustración 6.3.7 Gráfico "Tiempo & Población" considerando 30 iteraciones

Si se disminuye el parámetro de entrada a 10 iteraciones, la curva presenta un cambio respecto a la convergencia de la solución final, ya que existe una demora en encontrar la solución. La curva converge lentamente para grandes tamaños de la población teniendo así una demora de más de 1 segundos en comparación a las 30 iteraciones consideradas en el gráfico de la ilustración 6.3.8. Por lo que se puede concluir que al

aumentar el tamaño de la población no hace que la curva converja más rápido como lo hace el número de iteraciones.

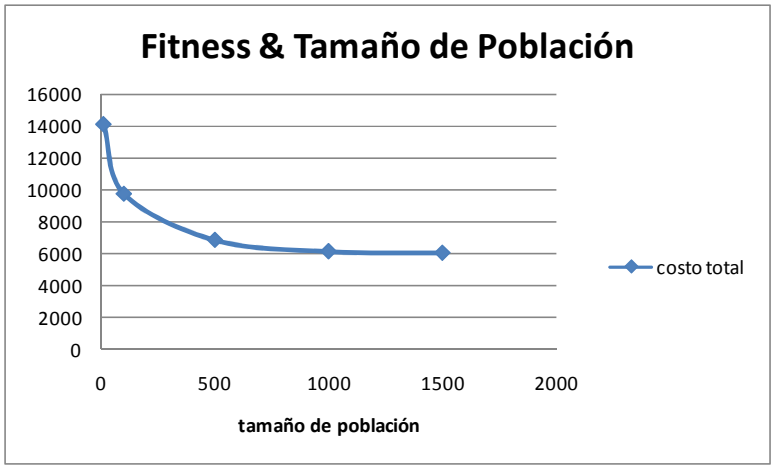


Ilustración 6.3.8 Gráfico "Fitness & Población" considerando 10 iteraciones

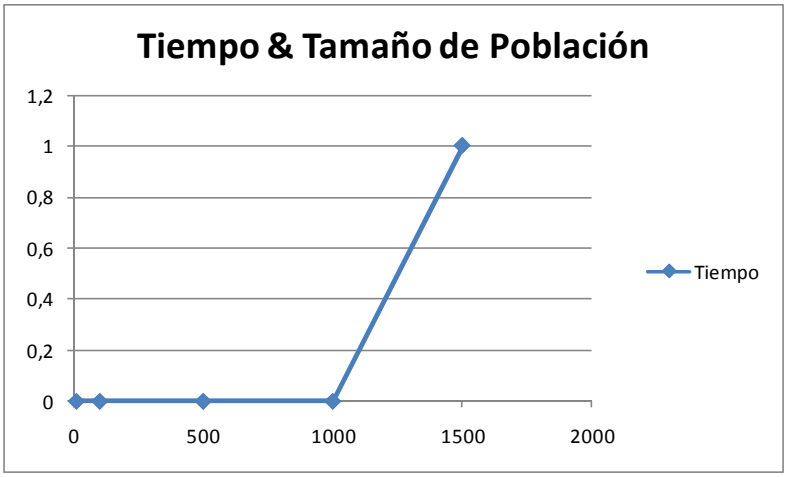


Ilustración 6.3.9 Gráfico "Tiempo & Población" considerando 10 iteraciones

Con los gráficos anteriores se realizó un tradeoff de la cantidad de iteraciones versus el tamaño de la población, donde se pudo concluir que el parámetro de la cantidad de iteraciones tiene mayor incidencia que el tamaño de la población, ya que al modificar el número de iteraciones se puede obtener de forma más rápida la solución final demorándose

un tiempo promedio bajo, mientras que al modificar el tamaño de la población solo se gasta tiempo adicional y no se logra el objetivo de manera eficiente.

Para los parámetros de entrada, Probabilidad de Selección, Probabilidad de Mutación y Probabilidad de Cruzamiento, se tomaron muestras para realizar el análisis de los resultados, pero no arrojaron grandes cambios en el comportamiento del algoritmo, ya que no afectan al tiempo ni al costo de la solución obtenida, siempre y cuando se consideren valores para las probabilidades dentro de un rango considerado como los valores comúnmente usados en la literatura.

Los resultados obtenidos son considerados buenos, ya que en la mayoría de los casos fue mejor que los obtenidos por los otros métodos de resolución, estos resultados fueron obtenidos utilizando el caso de 10 instalaciones y 20 clientes. Si bien fueron buenos resultados, estos se podrían mejorar cambiando la representación en alguno de los elementos de los AG, ya sea en la forma en que se genera la población inicial, o cambiar la función que repara los individuos no factibles, o alguna otra modificación que entregue mejores resultados.

En casi la totalidad de los casos analizados para esta instancia (10 instalaciones y 20 clientes) los AG convergieron a un mismo valor cada vez que se ejecutaba el algoritmo, pero se debe tener en cuenta que este tipo de metaheurística no siempre arroja las mismas salidas para las mismas entradas.

6.4 AG aplicado a DNDRP considerando el tamaño de lote

Hasta esta parte del proyecto se ha cumplido con el objetivo de este trabajo de investigación, pero se seguió complementando este estudio en base al paper de (Miranda, 2005), ya que propone mejoras al modelo que son de gran importancia a considerar.

La primera restricción a considerar es el tamaño del lote que debe ordenar cada centro de distribución, y la segunda restricción tiene relación con la capacidad de inventario de cada centro, esto tiene importancia ya que el hecho de reducir la capacidad del inventario no implica necesariamente que aumente el número de centros de distribución abiertos. En efecto, si se reduce el tamaño del lote, esto permite una óptima asignación de los clientes hacia los diferentes centros, reduciendo el costo del sistema total (Miranda, 2005).

En el modelo de localización estudiado en (Miranda, 2004), el tamaño del lote a ordenar para cada centro de distribución esta dado por el modelo EOQ, el cual entrega el Q (tamaño de lote) óptimo que se debe considerar en la función objetivo dado por:

$$Min \sum_{i=1}^{N} F_i \cdot Z_i + TH \sum_{i=1}^{N} \left(OC_i \cdot \frac{D_i}{Q_i} + HC_i \cdot \frac{Q_i}{2} \right) + \sum_{i=1}^{N} TH \cdot HC_i \cdot Z_{1-\alpha} \cdot \sqrt{U_i} \cdot \sqrt{LT_i} + \sum_{i=1}^{N} \sum_{j=1}^{M} TH \cdot (RC_i + TC_{ij}) \cdot d_j \quad (23)$$

F_i : Costo de fijo del centro de distribución *i*.

Q_i : Cantidad de lote ordenada por el centro de distribución *i*.

Z_i : Variable que toma el valor 1 si el centro *i* está abierto y cero en el otro caso.

TH : Horizonte de planeación.

HC_i : Costo de almacenamiento del centro de distribución *i*.

1-α : Nivel de servicio asociado al nivel de stock de seguridad de cada centro de distribución

LT_i : Tiempo que se demora en llegar una orden para el centro de distribución *i*.

U_i : Varianza total del centro de distribución *i*.

OC_i : Costo de ordenamiento del centro de distribución *i*.

D_i : Demanda total del centro de distribución *i*.

TC_{ij} : Costo de asignación del centro de distribución *i* al cliente *j*.

RC_i : Costo de transporte de la planta central al centro de distribución *i*.

d_j : Media de la demanda del cliente *j*.

Y_{ij} : Variable que toma el valor 1 si el centro de distribución *i* sirve al cliente *j*, y cero en el otro caso.

N : Cantidad de centros de distribución de la red.

M : Cantidad de clientes de la red.

Cap_i : Capacidad de almacenamiento del centro de distribución *i*.

u_j : Varianza de la demanda del cliente *j*.

Donde Q_i según (Miranda, 2004) está dado por:

$$Q_i = \sqrt{\frac{2 \cdot OC_i \cdot D_i}{HC_i}} \qquad \forall i = 1,...,N \qquad (24)$$

Pero si existiera cualquier restricción que dependiera del tamaño del lote asociado a los centros de distribución, este procedimiento no sería válido. Por lo que en el paper de (Miranda, 2005), se incorpora dos nuevas restricciones que dependen de Q_i.

La primera restricción tiene relación al límite de la cantidad máxima ordenada. Donde se asume una capacidad homogénea $Q_{máx}$ para cada centro de distribución.

Por lo tanto, la siguiente restricción que se debe incluir en el modelo está dada por:

$$0 \le Q_i \le Q_{\max} \qquad \forall i = 1,...,N \qquad (25)$$

La segunda restricción establece el nivel máximo de inventario para cada centro de distribución. Esta restricción fija una probabilidad máxima, β, que indica la capacidad de inventario *Icap* en niveles peak. Estos niveles peak sólo ocurren cuando los pedidos comienzan a llegar a cada centro. Este nivel de inventario corresponde al punto de reorden menos la demanda estocástica durante el lead time más el tamaño de lote ordenado $RP_i - SD(LT_i) + Q_i$. Por lo tanto, la restricción para la capacidad de inventario puede ser escrita de la siguiente manera:

$$P_r\left(RP_i - SD(LT_i) + Q_i \le ICap\right) = 1 - \beta \qquad (26)$$

Esta restricción puede ser reformulada como una restricción determinista y no lineal dada por:

$$Q_i + \left(Z_{1-\alpha} - Z_{1-\beta}\right).\sqrt{LT_i}.\sqrt{U_i} \le ICap \qquad \forall i = 1,...,N \qquad (27)$$

Finalmente, para asegurar que esta restricción sea sólo para los centros de distribución abiertos, la expresión puede ser escrita como:

$$Q_i + \left(Z_{1-\alpha} - Z_{1-\beta}\right).\sqrt{LT_i}.\sqrt{U_i} \le ICap.Z_i \qquad \forall i = 1,...,N \qquad (28)$$

Por lo tanto, si se toman en cuenta las siguientes simplificaciones:

$$C_{ij} = TH\ .\left(RC_i + TC_{ij}\right).d_j \qquad CS_i = TH\ .HC_i.K\ .\sqrt{LT_i}$$

La expresión (25) queda reducida a:

$$Min\sum_{i=1}^{N} F_i.Z_i + \sum_{i=1}^{N}\left(OC_i.\frac{D_i}{Q_i} + HC_i.\frac{Q_i}{2}\right) + \sum_{i=1}^{N}.CS_i..\sqrt{U_i} + \sum_{i=1}^{N}\sum_{j=1}^{M} C_{ij}.Y_{ij} \qquad (29)$$

Con las siguientes restricciones:

$$\sum_{i=1}^{N} Y_{ij} = 1 \qquad \forall j = 1,...,M \qquad (30)$$

$$\sum_{j=1}^{M} d_j.Y_{ij} = D_i \qquad \forall i = 1,...,N \qquad (31)$$

$$\sum_{j=1}^{M} u_j.Y_{ij} = U_i \qquad \forall i = 1,...,N \qquad (32)$$

$$0 \le Q_i \le Q_{\max} \qquad \forall i = 1,...,N \qquad (33)$$

$$Q_i + \left(Z_{1-\alpha} - Z_{1-\beta}\right).\sqrt{LT_i}.\sqrt{U_i} \le ICap.Z_i \qquad \forall i = 1,...,N \qquad (34)$$

1-α : Nivel de servicio asociado al nivel de stock de seguridad de cada centro.

1-β : Nivel de servicio asociado a la capacidad de inventario que no debe exceder cada centro.

LT_i : Tiempo que se demora en llegar una orden para el centro de distribución *i*.

Z_i : Variable que toma el valor 1 si el centro *i* está abierto y cero en el otro caso.

U_i : Varianza total del centro de distribución *i*.

D_i : Demanda total del centro de distribución *i*.

d_j : Demanda del cliente *j*.

Y_{ij} : Variable que toma el valor 1 si el centro de distribución *i* sirve al cliente *j*, y cero en el otro caso.

N : Cantidad de centros de distribución de la red.

M : Cantidad de clientes de la red.

Cap_i : Capacidad de almacenamiento del centro de distribución *i*.

u_j : Varianza del cliente *j*.

Q_{imax} : Lote máximo que puede tener el centro de distribución *i*.

Q_i : Lote óptimo del centro de distribución *i*.

La expresión (28) restringe que cada cliente sea servido por solo un centro de distribución para abastecer su demanda, mientras que (29) y (30) realizan los cálculos para la demanda y varianza total de cada centro de distribución. La restricción (31) es con respecto a la nueva variable Q_i, ya que esta variable no puede exceder el $Q_{máx}$ de cada centro ni tampoco puede tomar el valor cero. (32) es la nueva restricción de capacidad de inventario.

Notar, que para la elección de Q_i para cada centro, se debe tomar en cuenta la siguiente expresión:

$$Q_i = Min\left[Q_{i\max}, \sqrt{\frac{2 \cdot OC_i \cdot D_i}{HC_i}}, ICap_i - \left(Z_{1-\alpha} - Z_{1-\beta}\right) \cdot \sqrt{LT_i} \cdot \sqrt{U_i}\right] \quad (35)$$

La expresión (35) permite que se utilice un tamaño óptimo de lote para cada centro, además de asegurar que no se exceda el tamaño de lote máximo.

Criterio de Codificación

❖ Representación

El criterio de codificación fue la misma utilizada para el prototipo, DNDRP. Para los clientes, la codificación que se consideró fue un número entero positivo que va desde 1 al número máximo de clientes dado como parámetro de entrada menos uno, es decir, *M*-1.

Para los centros de instalaciones, se utilizó un número entero positivo que va desde 1 al número de instalaciones dado como parámetro de entrada menos uno, es decir, *N*-1.

❖ Operadores

Para resolver el problema, se utilizaron los operadores de selección, cruzamiento y mutación. El desempeño del algoritmo genético depende del valor de sus parámetros principales: tamaño de la población, tasa de selección, tasa de mutación y tasa de cruce. Estos operadores fueron implementados de la misma forma que se hizo para el prototipo, no tuvo ningún cambio.

❖ Función Objetivo

Se debe tomar en cuenta la siguiente expresión basada en (Miranda, 2006) para simplificar la ecuación:

$$C_{ij} = TH \cdot \left(RC_i + TC_{ij}\right) d_j$$

Por lo tanto según (Miranda, 2006), el modelo para DNDRP considerando la variable *Q* es:

$$Min \sum_{i=1}^{N} F_i \cdot Z_i + \sum_{i=1}^{N} \left(OC_i \cdot \frac{D_i}{Q_i} + HC_i \cdot \frac{Q_i}{2} \right) + \sum_{i=1}^{N} \cdot TH \cdot HC_i \cdot Z_{1-\alpha} \cdot \sqrt{LT_i} \cdot \sqrt{U_i} + \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij} \cdot Y_{ij}$$

Para definir la función fitness, que es la encargada de ver el nivel de adaptación de cada individuo de la población, se tomó en cuenta la función objetivo del DNDRP simplificada, por lo tanto, el valor arrojado por la función fitness calculada para cada individuo fue la misma utilizada para minimizar el problema.

❖ Criterio de Parada

Para determinar el criterio de parada, es decir, número de generaciones del algoritmo para este trabajo de investigación, es que se experimentó con las iteraciones, es decir, el algoritmo se detiene cuando este converja hacia un valor.

❖ Criterio de tratamiento de individuos no factibles.

En este algoritmo genético se utilizó una función que arregla los individuos defectuosos, para que cumplan con las restricciones del problema.

❖ Datos de Entrada

Se utilizó para los casos de prueba, archivos de datos que se utilizaron para el paper de (Miranda, 2006).

Los datos de entrada fueron los siguientes:

N : Número de centros de distribución.
M : Número de clientes.
T : Tamaño de la Población.
Iter : Número de iteraciones del problema.
P_s : Probabilidad de Selección.
P_c : Probabilidad de Cruzamiento.
P_m : Probabilidad de Mutación.
1-α : Nivel de servicio asociado al nivel de stock de seguridad de cada centro.
1-β : Nivel de servicio asociado a la capacidad de inventario que no debe exceder cada centro.
TH : Horizonte de Planeación.

Se utilizaron las siguientes estructuras para el almacenamiento de los datos ingresados a través del archivo:

Datos con respecto a los clientes:

cliente[M].demanda = Estructura de vectores de largo *M*, el cual almacena la demanda de cada cliente.

cliente[M].varianza = Estructura de vectores de largo *M*, el cual almacena la varianza de cada cliente.

Datos con respecto a los centros de distribución:

instalación[N].costoFijo = Estructura de vectores de largo *N*, el cual almacena el costo fijo de cada centro de distribución.

instalación[N].costoOrdenamiento = Estructura de vectores de largo *N*, el cual almacena el costo de ordenamiento de cada centro de distribución.

instalación[N].costoAlmacenamiento = Estructura de vectores de largo *N*, el cual almacena el costo de almacenamiento de cada centro de distribución.

instalación[N].leadTime = Estructura de vectores de largo *N*, el cual almacena el lead time de cada centro de distribución.

instalación[N].Qmax = Estructura de vectores de largo *N*, el cual almacena el lote máximo que puede tener cada centro de distribución.

instalación[N].capacidad = Estructura de vectores de largo *N*, el cual almacena la capacidad de cada centro de distribución.

instalación[M][N].costoAsignacion = Estructura de matrices de dimensión M*x*N, el cual almacena el costo asignación entre cada cliente y cada centro de distribución.

Este conjunto de datos tiene como objetivo ser utilizado como valor de entrada para calcular el costo total obtenido a través del modelo preliminar propuesto.

❖ Datos de Procesamiento

Una vez procesados los datos de entrada se necesita una estructura que almacene las demandas y varianzas totales para cada centro de distribución, de cada individuo generado:

instalación[T][N].demandaTotal = Estructura de matrices de dimensión *TxN,* el cual almacena la demanda total para cada centro de cada individuo generado.
instalación[T][N].varianzaTotal = Estructura de matrices de dimensión *TxN,* el cual almacena la varianza total para cada centro de cada individuo generado.
También se necesita almacenar el cálculo del *Q* óptimo y el *Q* que se despeja de la restricción (32) para luego ser utilizados en la función objetivo:

instalación[T][N].Qopt = Estructura de matrices de dimensión *TxN,* el cual almacena el cálculo para obtener el *Q* óptimo o *Q* de Wilson para cada centro de cada individuo generado.
instalación[T][N].Qr = Estructura de matrices de dimensión *TxN,* el cual almacena el cálculo para obtener el *Q* deseado desde una de las restricciones para cada individuo generado.

El *Q* utilizado en la función objetivo debe ser el mínimo entre los tres *Q* de cada centro por lo que se almacena el *Q* mínimo de cada centro:

instalación[T][N].Qminimo = Estructura de matrices de dimensión *TxN,* el cual almacena el *Q* mínimo entre el *Q* de Wilson, *Q* máximo y el *Q* de la restricción, ya que es este valor el cual se ocupa para la función objetivo y los cálculos necesarios.

Se guarda el valor de la restricción calculado para cada centro con el fin de realizar la reparación de los individuos que no cumplan con la factibilidad:

instalación[T][N].restriccion1 = Estructura de matrices de dimensión *TxN,* el cual almacena el cálculo de la restricción para cada centro de cada individuo generado.
instalación[T][N].restriccion2 = Estructura de matrices de dimensión *TxN,* el cual almacena el cálculo de la restricción para cada centro de cada individuo generado.

También se utiliza estructuras para la población de individuos, y para el fitness calculado de cada individuo de cada población.

población[T].fitness = Estructura de vectores de largo *T*, el cual almacena el fitness de cada población calculada para cada iteración, teniendo en cuenta todas las variables de la función objetivo.
población[T][M].inicial = Estructura de matrices de dimensión *TxM*, la cual almacena las instalaciones abiertas que sirven a los *M* clientes para cada población. Para la inicialización de esta matriz será llenada de forma aleatoria, tomando en cuenta el número de instalaciones con la que se cuenta para servir a los *M* clientes, además de las restricciones de capacidad.

población[T][M].inicial = Estructura de matrices de dimensión *TxM*, la cual almacena las instalaciones abiertas que sirven a los *M* clientes para cada población, donde la población para la siguiente iteración es derivada de la población inicial, a la cual se le aplican las operaciones de selección, cruzamiento y mutación, siempre y cuando se cumpla con las probabilidades exigida.

❖ Datos de Salida

Los resultados finales que se van a obtener de la investigación son guardados en un vector de largo *M*, el cual representa las instalaciones abiertas que sirven a los *M* clientes. También se muestra el fitness o resultado de la función objetivo generado por la población resultante, además del tamaño de lote para cada centro de distribución abierto.

Ejemplo:

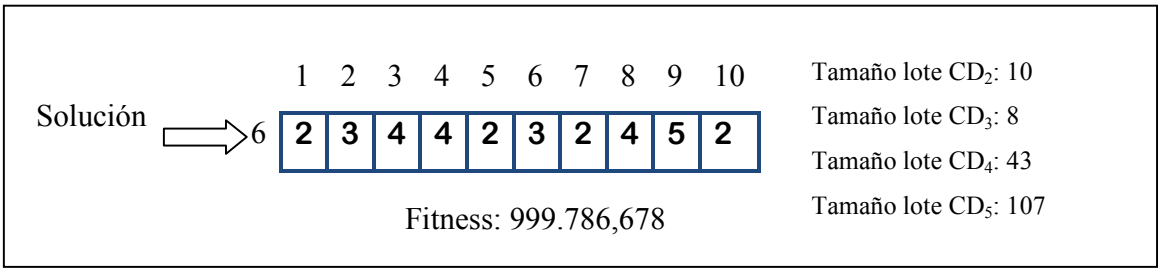


Ilustración 6.4.1 Solución para DNDRP considerando *Q*

Donde la fila representa que de la población fue elegido el individuo número 6, y las columnas son los 10 clientes, con sus respectivos centros de distribución que los abastecen representados por las celdas del vector, el fitness de la solución arrojada y el tamaño de lote para cada centro abierto.

6.5 Algoritmo para DNDRP considerando *Q*

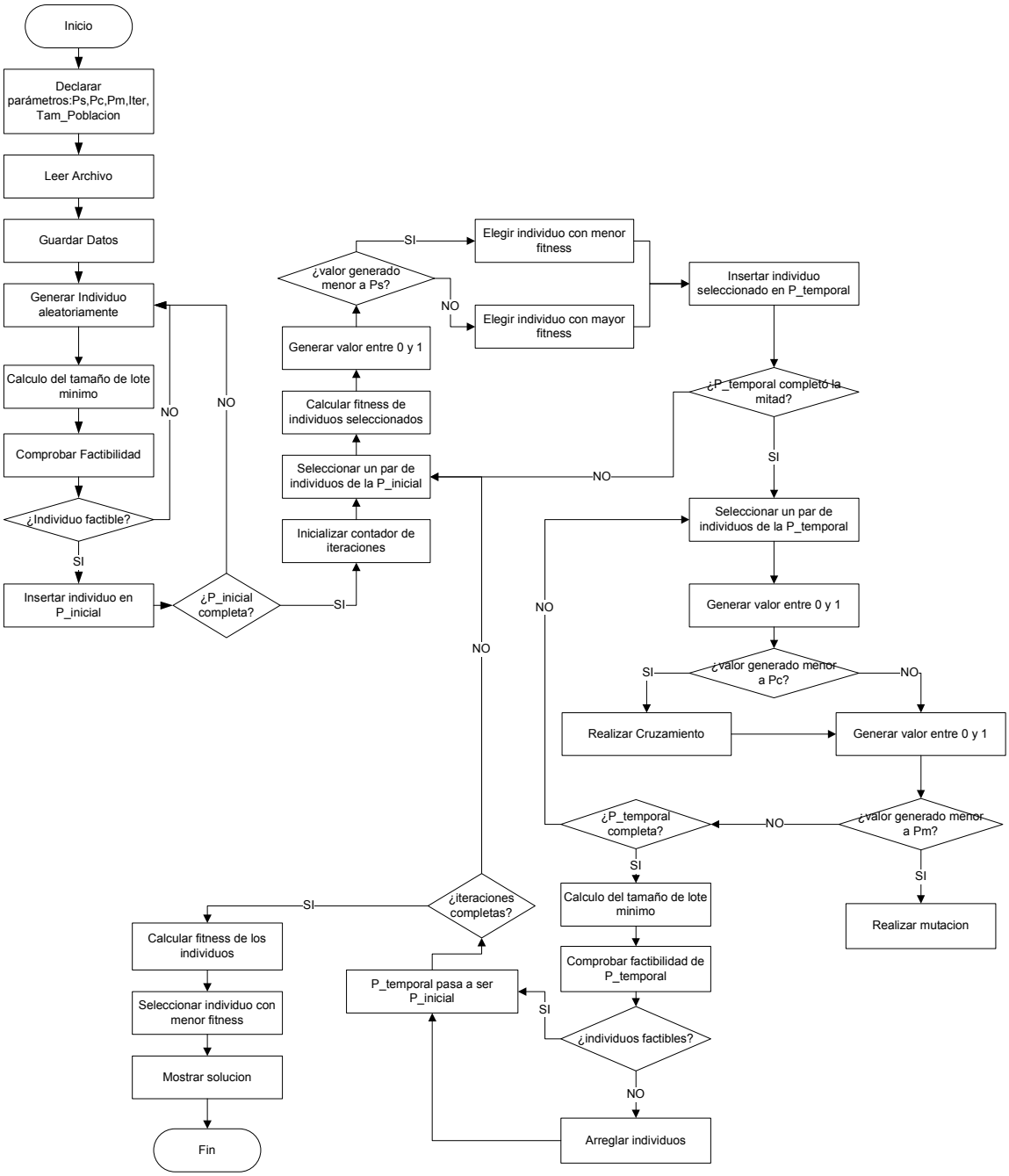


Ilustración 6.5.1 Diagrama de Flujo de algoritmo para DNDRP considerando *Q*

6.6 Discusión de resultados para DNDRP considerando *Q*

Para realizar un estudio y comparación de resultados, se utilizaron archivos de entrada de datos, los cuales se crearon para 16 casos diferentes, modificando los diferentes tipos de variables que se consideran como entrada, como por ejemplo, el costo fijo, costo de ordenamiento, que tamaño de lote entre otros.

Para cada uno de los casos de prueba, se realizó la búsqueda de las soluciones mediante un algoritmo de búsqueda exhaustiva y con AG, obteniendo los mismos resultados en ambos casos, es decir, la aplicación de AG convergió a los resultados óptimos obtenidos con la búsqueda exhaustiva.

El tiempo de respuesta promedio obtenido para los casos de prueba fueron de 4 segundos, tomando una población de 100 individuos y un número de iteraciones de 30. La elección del tamaño de población y el número de iteraciones se realizó generando un tradeoff con estas dos variables en relación al tiempo y solución obtenida.

CAPÍTULO 7

Conclusiones

En la primera parte de este informe, se realizó el estudio acerca de varios temas de interés para la investigación, en el caso de las metaheurísticas, se pudo concluir que si bien son de gran importancia a la hora de resolver problemas específicos, en nuestro caso el problema se resolverá aplicando la metaheurística, Algoritmos Genéticos.

Con respecto a los modelos de localización encontrados en la literatura, se hizo mayor énfasis en los problemas que involucran todos los tipos de decisiones, tácticas y estratégicas, ya que se desea solucionar un problema lo más cercano a la realidad posible, se tomó como base el DNDRP, ya que es el problema para el diseño de redes de distribución considerando políticas de control de inventario para satisfacer una demanda de tipo estocástica por parte de los clientes.

Se logró obtener una formulación al problema para comenzar a desarrollar un diseño de la red que satisfaga todas las restricciones, además de minimizar los costos.

Las pruebas realizadas al algoritmo preliminar determinaron que el algoritmo converge en algún número de iteraciones, es decir, la función fitness ya no arroja notables variaciones entre una cantidad de iteraciones y otra, además que para un tamaño pequeño de la población los resultados se alejan más del óptimo, mientras que al aumentar el tamaño de la población existen más individuos que se pueden adaptar mejor al problema, arrojando mejores soluciones.

Se debe tomar en cuenta que SDND sólo entrega una cota inferior para la solución óptima, lo cual implica una red más costosa (Miranda, 2004), por lo se debe tomar en cuenta cuándo una solución es mejor que la más simple solución entregada por SDND, es

decir, cual es el efecto causado por la demanda variable, la magnitud de los costos de almacenamiento o el nivel de servicio.

Debido a la integración de los problemas de inventario con los problemas de localización, es que nace el efecto risk poolling. Tal efecto influye en el diseño de la red de distribución debido a que al considerar las decisiones de inventario, es decir, disminuir los costos de stock de seguridad, la red tendería a la centralización, esto quiere decir que habría una disminución en la cantidad de centros de distribución abiertos.

Luego de analizados los resultados obtenidos aplicando los algoritmos genéticos, se puede concluir que éstos se encuentran cercanos o mejores que las soluciones obtenidas bajo otros métodos de resolución, además de obtener tiempos de respuestas razonables entre 0 a 3 segundos aproximadamente.

Se puede decir a nivel general que, los costos disminuyen debido a que al ser AG, un método de resolución aproximada, éste encuentra configuraciones para la red que cumplen con servir a todos los clientes, además de considerar todas las restricciones establecidas, pero no siempre eligiendo una buena configuración de la red, es decir , para unos casos de prueba, la configuración arrojada como solución, fue abrir un solo centro, ya que cumplía con todas las restricciones y era la de menor costo, pero puede que con otro método de resolución se encontrara una mejor red, con mejores costos y mejor configuración.

Para dar mejoras a los resultados obtenidos, en relación al tamaño de lote que deben ordenar los centros y cumplir con las restricciones de inventario, es que se tomaron en cuenta las observaciones realizadas por (Miranda, 2005) al modelo estudiado en este trabajo de investigación, obteniendo resultados esperados, es decir los mismos que los obtenidos aplicando métodos de resolución exactos.

Los AG tienen la potencialidad de contar con una representación de individuos o cromosomas que no depende de la función objetivo. Esto es debido, a que sólo modificando

la función fitness y utilizando la misma representación dada, se puede desarrollar AG para optimizar diferentes funciones objetivos bajo las mismas restricciones (Sourirajan, 2008).

En el futuro, se podría explotar más el uso de esta información utilizando otras metaheurísticas y comparar cual se comporta mejor para este tipo de problemas. También se puede extender el uso de los AG a problemas de diseños de redes de distribución con demanda estocástica para múltiples productos.

Anexos

Heurísticas

El problema de diseño de redes de distribución, es un problema del tipo combinatorial, por lo que se hace necesario el estudio de métodos heurísticos, para encontrar un diseño óptimo de la red de distribución con las respectivas restricciones, donde optimizar se refiere a encontrar la mejor solución posible para un determinado problema. Si bien en la literatura es posible encontrar una gran cantidad de problemas de optimización que además son relativamente fáciles de resolver, la clase de problema que se desea resolver en este trabajo es de mayor complejidad, ya que no se puede garantizar que el problema sea resuelto de manera óptima en un tiempo razonable.

Una de las razones por la cual elegir un método heurístico es debido a que el problema propuesto puede demorar demasiado tiempo para lograr una solución con algún método exacto de resolución, por lo que es preferible un método más flexible.

El problema DNDRP tomado como base para este trabajo, es una extensión del clásico problema de localización de instalaciones con restricciones de capacidad, el cual ya es de tipo NP-hard. Esto significa que si se usa este modelo para resolver grandes instancias, cualquier paquete comercial para resolver este tipo de problema se demora demasiado tiempo en resolver ese tipo de instancias (Miranda, 2004).

Los problemas NP (Non-Deterministic Polinomial Time), son aquellos problemas que con algoritmos determinísticos, es decir, para una determinada entrada, se obtenga siempre la misma salida no se puedan obtener resultados en tiempo razonable (De Jong K.A., 1989). Por lo tanto, el problema en estudio, es decir No Determinista consiste en ingresar alguna entrada, y dada esta entrada se pueden obtener diferentes salidas, demorándose un tiempo polinomial (tiempo considerado razonable) en encontrar una solución. Es por este motivo, que un método de resolución exacto en este caso no es factible para un tiempo considerado como razonable, teniendo que acudir a las heurísticas.

Una heurística para los problemas de complejidad NP, es un algoritmo en tiempo polinomial que obtiene soluciones óptimas o cercanas al óptimo para algunas entradas, pero no para otras. El estudio de heurísticas involucra, por una parte el diseño de un algoritmo y por otro lado, la manera de cómo evaluar la calidad de la heurística (Feige, 2005).

Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución (Martí Cunquero, 2003).

Métodos Heurísticos

❖ Métodos en Descomposición

El problema original se descompone en subproblemas más sencillos de resolver, teniendo en cuenta, aunque sea de manera general, que ambos pertenecen al mismo problema, un ejemplo de este método es el algoritmo devorador, Greedy (Martí Cunquero, 2003).

❖ Métodos Inductivos

La idea de estos métodos inductivos es generalizar de versiones pequeñas o más sencillas al caso más completo. Propiedades o técnicas identificadas en estos casos más fáciles, pueden ser aplicadas al problema completo (Martí Cunquero, 2003).

❖ Métodos de Reducción

Consiste en identificar propiedades que se cumplen mayoritariamente por las buenas soluciones e introducirlas como restricciones del problema. El objeto es restringir el espacio de las soluciones simplificando el problema. El riesgo obvio es dejar fuera las soluciones óptimas del problema original (Martí Cunquero, 2003).

❖ Métodos Constructivos

Consisten en construir literalmente paso a paso una solución del problema. Usualmente son métodos deterministas y suelen estar basados en la mejor elección de cada iteración. Estos métodos han sido muy utilizados en problemas clásicos como el del viajante, el algoritmo Grasp (Greedy Randomized Adaptive Search Procedures) es un ejemplo muy característico de este método combinado con el método de Búsqueda Local (Martí Cunquero, 2003).

❖ Métodos de Búsqueda Local

A diferencia de los métodos anteriores, los procedimientos de búsqueda local comienzan con una solución del problema y la mejoran progresivamente. El procedimiento realiza en cada paso un movimiento de una solución a otra con mejor valor. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore (Martí Cunquero, 2003).

Las medidas de calidad de un algoritmo, para cualquier método heurístico, se pueden realizar, según (Martí Cunquero, 2003), mediante diversos procedimientos como:

a) **Comparación con la solución óptima**: Aunque normalmente se recurre al algoritmo aproximado por no existir un método exacto para obtener el óptimo, o por ser éste computacionalmente muy costoso, en ocasiones puede que dispongamos de un procedimiento que proporcione el óptimo para un conjunto limitado de ejemplos (usualmente de tamaño reducido). Este conjunto de ejemplos puede servir para medir la calidad del método heurístico. Normalmente se mide, para cada uno de los ejemplos, la desviación porcentual de la solución heurística frente a la óptima, calculando posteriormente el promedio de dichas desviaciones. Si llamamos C_h al coste de la solución del algoritmo heurístico y C_{opt} al coste de la solución óptima de un ejemplo dado, en un problema de minimización la desviación porcentual viene dada por la expresión: $$\frac{C_h \cdot C_{opt}}{C_{opt}} \cdot 100$$

b) **Comparación con una cota**: En ocasiones el óptimo del problema no está disponible ni siquiera para un conjunto limitado de ejemplos. Un método alternativo de evaluación consiste en comparar el valor de la solución que proporciona el heurístico con una cota del problema (inferior si es un problema de minimización y superior si es de maximización). Obviamente la eficiencia de esta medida dependerá de la cota (cercanía de ésta al óptimo), por lo que, de alguna manera, tendremos que tener información de lo buena que es dicha cota. En caso contrario la comparación propuesta no tiene demasiado interés.

c) **Comparación con un método exacto truncado**: Un método enumerativo como el de Ramificación y Acotación explora una gran cantidad de soluciones, aunque sea únicamente una fracción del total, por lo que los problemas de grandes dimensiones pueden resultar computacionalmente inabordables con estos métodos. Sin embargo, podemos establecer un límite de iteraciones (o de tiempo) máximo de ejecución para el algoritmo exacto. También podemos saturar un nodo en un problema de

maximización cuando su cota inferior sea menor o igual que la cota superior global más un cierto α (análogamente para el caso de minimizar). De esta forma se garantiza que el valor de la mejor solución proporcionada por el procedimiento no dista más de α del valor óptimo del problema. En cualquier caso, la mejor solución encontrada con estos procedimientos truncados proporciona una cota con la que contrastar el heurístico.

d) **Comparación con otros heurísticos**: Este es uno de los métodos más empleados en problemas difíciles sobre los que se ha trabajado durante tiempo y para los que se conocen algunos buenos heurísticos. Al igual que ocurre con la comparación con las cotas, la conclusión de dicha comparación está en función del heurístico escogido.

e) **Análisis del peor caso**: Uno de los métodos que durante un tiempo tuvo bastante aceptación es analizar el comportamiento en el peor caso del algoritmo heurístico; esto es, considerar los ejemplos que sean más desfavorables para el algoritmo y acotar analíticamente la máxima desviación respecto del óptimo del problema. Lo mejor de este método es que acota el resultado del algoritmo para cualquier ejemplo; sin embargo, por esto mismo, los resultados no suelen ser representativos del comportamiento medio del algoritmo. Además, el análisis puede ser muy complicado para los heurísticos más sofisticados.

Aquellos algoritmos que, para cualquier ejemplo, producen soluciones cuyo coste no se aleja de un porcentaje ε del coste de la solución óptima, se llaman Algoritmos ε-Aproximados. Esto es; en un problema de minimización se tiene que cumplir para un ε > 0 que: $c_h \leq (1+\varepsilon) \cdot c_{opt}$

El procedimiento para medir la metaheurística elegida en este trabajo, será a través de la comparación con una cota. Esta cota será extraída de la investigación realizada por (Miranda, 2004), donde se obtuvieron resultados al problema mediante un enfoque basado en relajación de LaGrange y el método del sub-gradiente.

Como se mencionó inicialmente, este trabajo de investigación se enfocará en los métodos de búsqueda local, por lo que se pretende dar una clara explicación del funcionamiento del algoritmo para llevar a cabo la resolución de una búsqueda local. El problema original se descompone en subproblemas más sencillos de resolver, teniendo en cuenta, aunque sea de manera general, que ambos pertenecen al mismo problema, un ejemplo de este método es el algoritmo devorador, Greedy.

Algoritmo de Búsqueda Local (LS)

El algoritmo de búsqueda local, tiene como principal objetivo ir explorando la vecindad de un punto inicial hasta encontrar la mejor solución, después de haber encontrado una mejor solución se puede explotar esa vecindad para iniciar una nueva búsqueda. Este algoritmo parte con una solución inicial, que generalmente es aleatoria (x_0), la cual se va comparando iterativamente con los vecinos hasta obtener una mejor solución. Si la solución encontrada es mejor que la solución inicial, el nuevo punto de inicio de búsqueda es la nueva solución (y), sino, se sigue probando con otro vecino.

Inicio;
Elegir aleatoriamente una solución x_0;
Inicializar $x := x_0$;
Repetir:
    Generar una nueva solución y desde los vecinos de x;
    Si $f(y) \geq f(x)$ entonces $x := y$;
Hasta:
    $f(y) < f(x)$ para todo y en el vecindario de x;
Fin;

Ilustración 8.1 Algoritmo de Búsqueda Local (LS)

Pero este tipo de algoritmo posee varias desventajas, las que se numerarán a continuación:

- ✓ Por definición, algunos algoritmos terminan en un máximo local, y generalmente no se produce la suficiente información para saber si ese máximo local cae en un máximo global.
- ✓ La obtención de un máximo local depende de la solución inicial, donde la elección de esta solución no posee directrices para escoger la correcta.
- ✓ Es típicamente imposible obtener un gran salto en el tiempo de computación.

Estas desventajas, han producido un efecto de seguir investigando, otro tipo de solución a nuestro problema, por lo que se realizará un estudio de las metaheurísticas más conocidas (Fouskakis D., 2002).

Las metaheurísticas nacieron para dar mejoras en las soluciones obtenidas con los métodos heurísticos tradicionales, los cuales llegaban a una solución óptima, pero con algunas desventajas, las cuales fueron guiadas a través de la incorporación de un nuevo término, Metaheurístico, el cual se encarga de solucionar problemas de índole más general, aunque dejando de lado la eficiencia de esta solución al no ser tan específicas como las heurísticas.

Una definición clara dada en (Velez, 2007):

"Los procedimientos Metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Los Metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos."

La metaheurísticas incluyen entre otras (Glover, 1977): Simulated Annealing (enfriamiento simulado), Tabú Search (búsqueda tabú), Iterated Local Search (búsqueda local iterativa), Variable Neighborhood Search (búsqueda local con vecindario variable), Evolutionary Algorithms (algoritmos evolutivos), Genetics Algorithms (algoritmos genéticos),etc.

❖ Enfriamiento Simulado (SA)

Es un método se optimización discreto, que fue desarrollado cerca de los 1980s. Es un método de optimización combinatoria de propósito general que busca el mínimo de una función de muchas variables. El enfriamiento simulado es una técnica robusta, con capacidad para obtener el mínimo absoluto en vez del mínimo local (Alvarez de los Mozos, 2003).

Es una técnica de búsqueda local estocástica, la cual aproxima el máximo de la función objetivo $f: S \rightarrow R$ sobre un conjunto finito S. Este opera iterativamente eligiendo un elemento y desde el vecindario $N(x)$ en la actual solución x; el candidato y debe ser aceptado o rechazado como la nueva solución. La solución encontrada puede ser aceptada con una probabilidad positiva incluso si y es peor que x, lo cual es permitido por el algoritmo para no caer en un máximo local.

Inicio;

Elegir una solución inicial x_0;

Seleccionar la temperatura inicial y final $T_0, T_f > 0$

Inicializar $x:=x_0$ y $T:=T_0$;

Repetir:

Repetir:

Elegir una nueva solución y desde el vecindario de x;

Si $f(y) \geq f(x)$ entonces $x:=y$;

Sino elegir u aleatoriamente uniforme del rango $(0,1)$

Si $u < \exp\left[\frac{f(y)-f(x)}{T}\right]$ entonces $x:=y$, sino $x:=x$;

Hasta contador = n_{iter};

Decrementar T de acuerdo a la temperatura listada;

Hasta criterio de finalización = verdadero;

x es la aproximación de la solución óptima.

Fin;

Ilustración 8.2 Algoritmo Enfriamiento Simulado (SA)

$$\text{Donde } p(x,y,T) := \begin{cases} 1 & Si\ f(y) \geq f(x) \\ \exp\left[\frac{f(y)-f(x)}{T}\right] & Si\ f(y) < f(x) \end{cases}$$

$p(x,y,T)$: probabilidad que y pase a ser un candidato dada la solución actual x. Esta probabilidad es controlada por la temperatura dada $T>0$ (Fouskakis D., 2002).

❖ Búsqueda Tabú (TS)

Es el procedimiento de más alto nivel para la resolución de problemas de optimización, propuesto para escapar de la trampa del óptimo local. Originalmente fue propuesto (Fouskakis D., 2002) como una herramienta de optimización aplicable a problemas de cobertura no lineales, pero luego fue presentado con mayores detalles años después.

Este método de optimización estocástica, es utilizado para resolver varios tipos de problemas, como: problemas de horario de empleados, problemas de máxima satisfacción, dignación de caminos para las telecomunicaciones, problemas de probabilidad logística, entre otros.

El algoritmo TS se divide en tres partes: búsqueda preliminar, intensificación y diversificación. La búsqueda preliminar, la parte más importante del algoritmo, funciona de la siguiente manera: Partiendo desde una solución inicial específica, TS examina todos los vecinos e identifica el que tiene el más alto valor para la función objetivo. Moverse hacia una nueva solución no significa que se obtenga la mejor solución, pero TS se mueve a la nueva configuración de todas formas; esto inhabilita al algoritmo para que siga funcionando la búsqueda sin llegar a bloquearse por la ausencia de posibles movimientos, y por lo tanto escapando de un óptimo local.

Si no existen posibles movimientos, TS elige el que menos degrada a la función objetivo, devolviéndose a otra solución y dejando la solución que provocó el óptimo local como visitado, por lo que es olvidado, siguiendo con la próxima búsqueda. Esto se puede realizar gracias a una lista que va almacenando los vecinos visitados, esta lista contiene s elementos, los cuales son los movimientos tabú. El parámetro s es llamado “tamaño de la lista tabú”.

La segunda etapa, intensificación, comienza con la mejor solución encontrada, luego el algoritmo comienza nuevamente con la búsqueda. El usuario debe definir la cantidad de iteraciones que pueda tener el algoritmo, si se cumple con la cantidad máxima de iteraciones, éste pasa a la siguiente etapa, y si cumple la cantidad máxima de iteraciones pero no se ha encontrado una solución, igualmente pasa a la siguiente etapa. La intensificación nos provee de una manera simple de que se enfoque en la búsqueda alrededor de la mejor solución actual.

Por último, la etapa de diversificación, comienza nuevamente limpiando la lista tabú, inicializándola con los movimientos más frecuentes. Luego es elegido aleatoriamente un nuevo estado para volver a iniciar la búsqueda con un específico número de iteraciones. La diversificación provee una manera simple de explotar regiones que habían sido poco visitadas. Al finalizar la tercera etapa, la mejor solución encontrada debe ser registrada (Fouskakis D., 2002).

Inicio;

Aleatoriamente elegir una solución inicial i_{start}, inicializar $i= i_{start}$ y evaluar la función objetivo $f(i)$;

Inicializar $\alpha=low$; Determinar $s= Tamaño$, largo de la lista; Inicializar $Move=0$ y $i_{max}=i_{start}$;

Repetir:

Búsqueda Preliminar

Agregar i a la lista tabú s;

Inicializar $s=s-1$. Si $s=0$ entonces inicializar $s= Tamaño$; $Move=Move+1$, $i_{nbhd} =i$ y $c_{nbhd} =low$, un número pequeño.

Para cada vecino j de i hacer:

Si $f(j) > \alpha$ hacer:

Si $f(j) \geq c_{nbhd}$ entonces inicializar $i_{nbhd}=j$ y $c_{nbhd}=f(j)$;

Si $f(j) \leq \alpha$ hacer:

Si j está en la lista tabú ir al próximo vecino;

Sino si j no es tabú y $f(j) \geq c_{nbhd}$ entonces $i_{nbhd}=j$ y $c_{nbhd}=f(j)$;

Inicializar $\alpha=\min(\alpha,c_{nbhd})$ y $i=i_{nbhd}$;

Si $f(i) \geq f(i_{max})$ entonces $i_{max}=i$;

Sino $Move \neq maxmoves$ volver al inicio de la Búsqueda preliminar.

Intensificación

Repetir:

Inicializar $i=i_{max}$ y limpiar la lista tabú;

Repetir:

Hacer la *Búsqueda preliminar*;

Hasta encontrar una mejor solución que i_{max} . Si no se encuentran mejoras, entonces ir a *Diversificación;*

Hasta n_{impr} replicaciones;

Diversificación

Limpiar la lista tabú e colocar los s movimientos más frecuentes;

Aleatoriamente elegir una solución inicial i;

Evaluar $f(i)$;

Repetir:

Hacer la *Búsqueda preliminar*;

Hasta que hayan ocurrido n_{div} repeticiones;

Hasta que el algoritmo completo hay sido repetido rep veces; I_{max} es la aproximación a la óptima solución;

Fin

Ilustración 8.3 Algoritmo Búsqueda Tabú (TS)

Código Fuente

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define c 100                    //número de clientes
#define w 40                     //número de instalaciones

#define ps 0.9                   //probabilidad de selección
#define pc 0.9                   //probabilidad de cruzamiento
#define pm 0.2                   //probabilidad de mutación

#define t 100                    //tamaño de la población (cantidad de individuos)
#define iter 50                  //número de iteraciones  (condición de término)

#define k 1.6448                 //nivel de servicio para pm_2004
#define alfa 1.6448              //nivel de servicio para pm_2006
#define beta 1.6448              //nivel de servicio para pm_2006
#define TH 1                     //horizonte de planeación
```

ESTRUCTURA NECESARIAS PARA EL PROGRAMA

```
struct datosCliente{
    int demanda;                 //media de la demanda de cada cliente
    int varianza;                //varianza de la demanda de cada cliente
    }cliente[c];

struct datosinstalacion{
    int cf;                      //costo fijo
    int oc;                      //costo de ordenamiento
    int hc;                      //costo de almacenamiento
    int capacidad;               //capacidad máxima
    int lt;                      //lead time
    double qmax;                 //tamaño de lote máximo
    }instalacion[w];

struct datosAsignacion{
    int TC;                      //costo de asignación
    }asignacion[c][w];

struct datosInstalacion2{
    int d;                       //demanda total
    int u;                       //varianza total
    double qminimo;              //tamaño de lote mínimo entre los tres qmax,qwilson y qrestriccion
    double qopt;                 //tamaño de lote calculado por la formula se q de Wilson
    double qr;                   //tamaño de lote despejado por una de las restricciones del PM_2006
    int restriccion;             //valor de la restricción de PM_2006
    }instalacion2[t][w];
```

```
struct funcionobjetivo{
    double fitness;                    //fitness total de cada individuo generado
    double s1,s2,s3,s4;
    double probabilidad;               //probabilidad para selección por ruleta
    double probabilidad_acumulada;     //probabilidad acumulada para selección por ruleta
    }fitnesspoblacion[t];

struct poblacionindividuos{
    int poblacion_inicial;             //población inicial de individuos generados
    int poblacion_siguiente;           //población temporal de individuos generados
    }poblacion[t][c];
```

DECLARACIÓN DE FUNCIONES UTILIZADAS

```
int guardarDatos_PM_2004(struct datosCliente cliente[c],struct datosInstalacion instalacion[w], struct datosAsignacion asignacion[c][w],char* nombre_archivo);
int guardarDatos_PM_2006(struct datosCliente cliente[c],struct datosInstalacion instalacion[w], struct datosAsignacion asignacion[c][w],char* nombre_archivo);

int generarIndividuos_PM_2004(int r,struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c], struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]);
int generarIndividuos_PM_2006(int r,struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c], struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]);

int comprobar_restriccion_PM_2004(int r,struct datosInstalacion instalacion[w],struct poblacionIndividuos poblacion[t][c],struct datosInstalacion2 instalacion2[t][w]);
int comprobar_restriccion_PM_2006(int r,struct datosInstalacion instalacion[w],struct poblacionIndividuos poblacion[t][c],struct datosInstalacion2 instalacion2[t][w]);

double calcular_fitness_PM_2004(struct datosCliente cliente[c], struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w],struct datosAsignacion asignacion[c][w],struct poblacionIndividuos poblacion[t][c],struct funcionObjetivo fitnessPoblacion[t]);
double calcular_fitness_PM_2006(struct datosCliente cliente[c], struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w],struct datosAsignacion asignacion[c][w],struct poblacionIndividuos poblacion[t][c],struct funcionObjetivo fitnessPoblacion[t]);

int seleccionar_individuos_ruleta(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos poblacion[t][c]);
int seleccionar_individuos_torneo(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos poblacion[t][c]);

int llenar_poblacion_siguiente(struct poblacionIndividuos poblacion[t][c]);

int intercambiar_PM_2004(struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c], struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]);
int intercambiar_PM_2006(struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c], struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]);

double menor_fitness_PM_2004(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos poblacion[t][c],char* nombre_archivo);
double menor_fitness_PM_2006(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos poblacion[t][c],char* nombre_archivo);

double calcular_probabilidad(struct funcionObjetivo fitnessPoblacion[t]);
```

FUNCION PRINCIPAL

```
int main(int argc, char *argv[])
{
   //srand(8);                                    //usar una semilla específica
   srand (time(null));                            //usar una semilla aleatoria
   int i,j,a,r,eleccion1,eleccion2;

                        MENÚ PARA ELEGIR EL TIPO DE SELECCION DE INDIVIDUOS
 if(argc != 3){
   printf("Debe ingresar los nombres de archivos de la siguiente manera: archivoDatos.txt
archivoResultados.txt\n ");
 }
   printf ("Desea seleccion por torneo o ruleta\n");
   printf("ruleta: presione 1\n");                //"1" elegir selección por ruleta
   printf("torneo: presione 2\n");                // "2" elegir selección por torneo
   scanf("%d",&eleccion1);

                        MENÚ PARA ELEGIR QUE MODELO DESEA EJECUTAR
   printf ("Que modelo desea:\n");
   printf("Miranda 2004: presione 1\n");          // "1" MODELO PABLO MIRANDA 2004
   printf("Miranda 2006: presione 2\n");          // "2" MODELO PABLO MIRANDA 2006
   scanf("%d",&eleccion2);


                        GUARDAR LOS DATOS DEL ARCHIVO DE ENTRADA
if(eleccion2 == 1){
guardarDatos_PM_2004(cliente,instalacion,asignacion,argv[1]);
}
else if(eleccion2 == 2){
guardarDatos_PM_2006(cliente,instalacion,asignacion,argv[1]);
}/

                        GENERAR INDIVIDUOS PARA LA POBLACION INICIAL
 for(r=0;r<t;r++){
   if(eleccion2 == 1){
     generarIndividuos_PM_2004(r,poblacion,cliente,instalacion,instalacion2);
    }
   else if(eleccion2 == 2){
     generarIndividuos_PM_2006(r,poblacion,cliente,instalacion,instalacion2);
    }
}

                        CALCULAR EL FITNESS DE LA POBLACION INICIAL
 if(eleccion2 == 1){
   calcular_fitness_PM_2004(cliente,instalacion,instalacion2,asignacion,poblacion,fitnessPoblacion);
   }
 else if(eleccion2 == 2){
   calcular_fitness_PM_2006(cliente,instalacion,instalacion2,asignacion,poblacion,fitnessPoblacion);
   }

                        CALCULAR LA PROBABILIDAD DE LA POBLACION INICIAL
calcular_probabilidad(fitnessPoblacion);     //necesaria para formar la ruleta

for(a=0;a<ITER;a++){     //comienza el ciclo según número máximo de iteraciones
```

SELECCIONAR INDIVIDUOS (RULETA O TORNEO)

```
if(eleccion1 == 1){
    seleccionar_individuos_ruleta(fitnessPoblacion,poblacion);
    }
  else if(eleccion1 == 2){
    seleccionar_individuos_torneo(fitnessPoblacion,poblacion);
   }
```

LLENAR POBLACION TEMPORAL APLICANDO OPERADORES

```
llenar_poblacion_siguiente(poblacion);
```

INTERCAMBIAR POBLACIÓN DE TEMPORAL A INICIAL

```
if(eleccion2 == 1){
    intercambiar_PM_2004(poblacion,cliente,instalacion,instalacion2);
}
else if(eleccion2 == 2){
    intercambiar_PM_2006(poblacion,cliente,instalacion,instalacion2);
}
```

CALCULAR EL FITNESS DE LA POBLACIÓN TEMPORAL

```
if(eleccion2 == 1){
   calcular_fitness_PM_2004(cliente,instalacion,instalacion2,asignacion,poblacion,fitnessPoblacion);
 else if(eleccion2 == 2){
     calcular_fitness_PM_2006(cliente,instalacion,instalacion2,asignacion,poblacion,fitnessPoblacion);
```

CALCULAR LA PROBABILIDAD DE LA POBLACION TEMPORAL

```
calcular_probabilidad(fitnessPoblacion);
}//Fin for de las iteraciones
```

MOSTRAR EL MENOR FITNESS

```
if(eleccion2 == 1){
   menor_fitness_PM_2004(fitnessPoblacion,poblacion,argv[2]);
}
else if(eleccion2 == 2){
     menor_fitness_PM_2006(fitnessPoblacion,poblacion,argv[2]);
}
}//fin Main
```

FUNCION QUE GUARDA LOS DATOS PARA PM_2004

```
int guardarDatos_PM_2004(struct datosCliente cliente[c],struct datosInstalacion instalacion[w], struct
datosAsignacion asignacion[c][w],char* nombre_archivo){
 int i,j;
 FILE *archivo;
 char variable[10000],*t1,s2[4] = " \n\t";
 double dato[10000],v_d;
 archivo = fopen(nombre_archivo,"r");

 if(archivo == NULL){
    printf ("\nEl contenido del archivo de prueba esta vacio \n\n");
}
else{
    i=0;
    while (feof(archivo) == 0){
        fgets(variable,1000,archivo);
     for(t1=strtok(variable,s2); t1!=NULL; t1=strtok(NULL, s2)){
```

```
        if(t1!=NULL && t1!=" "){
           v_d=atof(t1);
           dato[i]=v_d;
            i++;
         }
       }
    }
  }

int posicion=3;                                  //inicializador de la posición para leer los datos
for(i=0;i<c;i++){
  cliente[i].demanda=dato[posicion];             //almacena la media de las demandas
  cliente[i].varianza=dato[posicion+1];          //almacena la varianza de las demandas
  posicion=posicion+2;
  }

 posicion=posicion+1;                            // inicializador de la posición para leer los datos
 for(j=0;j<w;j++){
    instalacion[j].cf=dato[posicion];            //almacena costo fijo
    instalacion[j].oc=dato[posicion+1];          //almacena costo de ordenamiento
    instalacion[j].hc=dato[posicion+2];          //almacena costo de almacenamiento
    instalacion[j].capacidad=dato[posicion+3];   //almacena capacidad
    instalacion[j].lt=dato[posicion+4];          //almacena lead time
    posicion=posicion+5;
}

posicion=posicion+1;
for(i=0;i<c;i++){
  for(j=0;j<w;j++){
    asignacion[i][j].TC=dato[posicion];          //almacena costo de asignación o transporte
     posicion++;
  }
 }
}//fin Guardar datos
```

FUNCION PARA GUARDAR LOS DATOS PARA PM 2006

```
int guardarDatos_PM_2006(struct datosCliente cliente[c],struct datosInstalacion instalacion[w], struct
datosAsignacion asignacion[c][w],char* nombre_archivo){
 int i,j;
 FILE *archivo;
 char variable[10000],*t1,s2[4] = " \n\t";
 double dato[10000],v_d;
 archivo = fopen(nombre_archivo,"r");
 if(archivo == NULL){
    printf ("\nEl contenido del archivo de prueba esta vacio \n\n");
}
else{
   i=0;
   while (feof(archivo) == 0){
        fgets(variable,1000,archivo);
           for(t1=strtok(variable,s2); t1!=NULL; t1=strtok(NULL, s2)){
              if(t1!=NULL && t1!=" "){
                 v_d=atof(t1);
                 dato[i]=v_d;
                 i++;
```

```
                }
            }
        }
    }

int posicion=3;                                  //inicializador de la posición para leer los datos
for(i=0;i<c;i++){
    cliente[i].demanda=dato[posicion];           //almacena la media de la demandas
    cliente[i].varianza=dato[posicion+1];        //almacena la varianza de las demandas
    posicion=posicion+2;
}

 posicion=posicion+1;
 for(j=0;j<w;j++){
    instalacion[j].cf=dato[posicion];            //almacena costo fijo
    instalacion[j].oc=dato[posicion+1];          //almacena costo de ordenamiento
    instalacion[j].hc=dato[posicion+2];          //almacena costo de almacenamiento
    instalacion[j].capacidad=dato[posicion+3];   //almacena capacidad
    instalacion[j].lt=dato[posicion+4];          //almacena lead time
    instalacion[j].qmax=dato[posicion+5];        //almacena el tamaño de lote
    posicion=posicion+6;
}

 posicion=posicion+1;
 for(i=0;i<c;i++){
    for(j=0;j<w;j++){
       asignacion[i][j].TC=dato[posicion];       //almacena costo de asignación o transporte
       posicion++;
    }
  }
}//fin Guardar datos
```

FUNCIÓN PARA GENERAR INDIVIDUOS PARA PM_2004

```
generarIndividuos_PM_2004(int r, struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c],
struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]){
int i,j,sumaD,sumaU,respuesta;

for(i=0;i<c;i++){
     poblacion[r][i].poblacion_inicial= (rand() % (w));
}
for(j=0;j<w;j++){
    sumaD=0;
    sumaU=0;
    for(i=0;i<c;i++){
       if(j == poblacion[r][i].poblacion_inicial){
          sumaD = sumaD+ cliente[i].demanda;          //cálculo de la media de la demanda total
          sumaU = sumaU + cliente[i].varianza;       //cálculo de la varianza de la demanda total
          }
       }
   instalacion2[r][j].D = sumaD;                     //almacena la media de la demanda total
   instalacion2[r][j].U = sumaU;                     //almacena la varianza de la demanda total
}/
respuesta = comprobar_restriccion_PM_2004(r,instalacion,poblacion,instalacion2);   //comprueba la
factibilidad
```

```
if(respuesta == 0){   //si la respuesta es 0 se debe generar individuo hasta que cumpla con las restricciones
   generarIndividuos_PM_2004(r,poblacion,cliente,instalacion,instalacion2);
  }
}//Fin Generar individuos
```

FUNCIÓN PARA GENERAR INDIVIDUOSEN PM_2006

```
generarIndividuos_PM_2006(int r, struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c],
struct datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]){
int i,j,sumaD,sumaU,respuesta;
for(i=0;i<c;i++){
    poblacion[r][i].poblacion_inicial= (rand() % (w));
}//fin for
for(j=0;j<w;j++){
   sumaD=0;
   sumaU=0;
   for(i=0;i<c;i++){
      if(j == poblacion[r][i].poblacion_inicial){
        sumaD = sumaD + cliente[i].demanda;            //cálculo de la media de la demanda total
         sumaU = sumaU + cliente[i].varianza;          //cálculo de la varianza de la demanda total
        }
    }
  instalacion2[r][j].D = sumaD;                        //almacena la media de la demanda total
  instalacion2[r][j].U = sumaU;                        //almacena la varianza de la demanda total

instalacion2[r][j].Qopt=sqrt((2.0*(double)instalacion[j].OC*(double)instalacion2[r][j].D)/(double)instalacion[
j].HC);                                                //almacena q calculado con formula de q de Wilson

instalacion2[r][j].Qr=(double)instalacion[j].capacidad-((alfa-
beta)*(sqrt((double)instalacion[j].LT*instalacion2[r][j].U)));      //almacena el Q despejado de la restricción

 if(instalacion2[r][j].D == 0){                        //si el centro no es abierto deja las variables en cero
    instalacion2[r][j].Qopt=0.0;
    instalacion2[r][j].Qr=0.0;
    instalacion2[r][j].Qminimo=0.0;
    }
 else{
     if(instalacion2[r][j].Qopt > 0.0){          //elige el Q mínimo para hacer los cálculos para en fitness y
restricción
       instalacion2[r][j].Qminimo= instalacion2[r][j].Qopt;
       }
      else {
          if(instalacion[j].Qmax <= instalacion2[r][j].Qr){
            instalacion2[r][j].Qminimo= instalacion[j].Qmax;
            }
           else{
                if(instalacion2[r][j].Qr > 0.0)
                  instalacion2[r][j].Qminimo= instalacion2[r][j].Qr;
                  }
              }
          }
       }
for(j=0;j<w;j++){ //cálculo de la restricción tomando el menor Q mínimo
instalacion2[r][j].restriccion=(double)instalacion2[r][j].Qminimo+(2*k*(sqrt((double)instalacion[j].LT*(doub
le)instalacion2[r][j].U)));
 }
```

```
respuesta = comprobar_restriccion_PM_2006(r,instalacion,poblacion,instalacion2);  //función que comprueba la factibilidad de los individuos generados

if(respuesta == 0){
     //si la respuesta es 0 se debe generar individuo hasta que cumpla con las restricciones
     generarIndividuos_PM_2006(r,poblacion,cliente,instalacion,instalacion2);
   }
}//Fin Generar individuos
```

FUNCIÓN PARA COMPROBAR QUE INDIVIDUO SEA FACTIBLE EN PM_2004

```
int comprobar_restriccion_PM_2004(int r,struct datosInstalacion instalacion[w],struct poblacionIndividuos poblacion[t][c],struct datosInstalacion2 instalacion2[t][w]){
int j,i,flag;
flag =1;                              //valor de respuesta si el individuo es exitoso
  for(j=0;j<w;j++){
    if((double)instalacion[j].capacidad < instalacion2[r][j].D){
      flag = 0;                       //valor de respuesta para individuo NO factible
     }
   }
return(flag);
}//Fin comprobar restricciones
```

FUNCIÓN PARA COMPROBAR QUE INDIVIDUO SEA FACTIBLE EN PM_2006

```
int comprobar_restriccion_PM_2006(int r,struct datosInstalacion instalacion[w],struct poblacionIndividuos poblacion[t][c],struct datosInstalacion2 instalacion2[t][w]){
int j,i,flag;
flag =1;                              //valor de respuesta si el individuo es exitoso
 for(j=0;j<w;j++){
    if((double)instalacion[j].capacidad < instalacion2[r][j].restriccion){
      flag = 0;                       //valor de respuesta para individuo NO
     }
   }
return(flag);
}//Fin comprobar restricciones
```

FUNCIÓN PARA CALCULAR EL FITNESS_PARA PM_2004

```
double calcular_fitness_PM_2004(struct datosCliente cliente[c], struct datosInstalacion instalacion[w], struct datosInstalacion2 instalacion2[t][w],struct datosAsignacion asignacion[c][w],struct poblacionIndividuos poblacion[t][c],struct funcionObjetivo fitnessPoblacion[t]){
int i,j,r,x,q,dem,matrizInstalacion[t][c][w];
double suma1,suma2,suma3,suma4,var,Sumatoria1[t],Sumatoria2[t], Sumatoria3[t], Sumatoria4[t];
for(r=0;r<t;r++){          // matriz de asignación de instalaciones
   for(i=0;i<c;i++){
      int n=poblacion[r][i].poblacion_inicial;
      matrizInstalacion[r][i][n]=1;
      }
}

int ia[t][w];              // matriz para ver si la instalación está cerrada o abierta
for(r=0;r<t;r++){
   suma1 =0.0;
   for(j=0;j<w;j++){
      int sum=0;
```

```
      for(i=0;i<c;i++){
         sum = sum + matrizInstalacion[r][i][j];
      }
       ia[r][j]=sum;
   }
}

for(r=0;r<t;r++){                          //cálculo de los costos fijos para las instalaciones abiertas
   suma1 =0.0;
   for(j=0;j<w;j++){
      if(ia[r][j] != 0){
        suma1 = suma1 + (double)instalacion[j].CF;
        }
    }
  Sumatoria1[r]= suma1;
  fitnessPoblacion[r].S1=Sumatoria1[r];      //se guarda la suma de los costos fijos como S1
}// fin sumatoria de costos fijos

for(r=0;r<t;r++){                          //cálculo de los costos de asignación
   suma2=0.0;
   for (i=0;i<c;i++){
      x = poblacion[r][i].poblacion_inicial;
      suma2 = suma2 + ((double)asignacion[i][x].TC);
    }
   Sumatoria2[r]=(double)TH*suma2;
   fitnessPoblacion[r].S2=Sumatoria2[r];     //se guarda la suma de los costos de asignación como S2
}//fin sumatoria de costos de asignación

for(r=0;r<t;r++){                          //cálculo de los costos de almacenamiento
  suma3 = 0.0;
  for(q=0;q<w;q++){

suma3=suma3+(sqrt(2*(double)instalacion[q].HC*(double)instalacion[q].OC*(double)instalacion2[r][q].D));
    }
   Sumatoria3[r] = (double)TH*suma3;
   fitnessPoblacion[r].S3=Sumatoria3[r];     //se guarda la suma de los costos de almacenamiento como S3
}// fin sumatoria de los costos de almacenamiento

double mult[w];
for(r=0;r<t;r++){                          //cálculo de los costos de stock de seguridad
   suma4 = 0.0;
   for(q=0;q<w;q++){
      mult[q]=
(((double)instalacion[q].HC)*(sqrt((double)instalacion2[r][q].U))*((double)(sqrt(instalacion[q].LT)))*K);
      suma4 = suma4 + (mult[q]);
   }
   Sumatoria4[r] = (double)TH*suma4;
   fitnessPoblacion[r].S4=Sumatoria4[r];     //se guarda la suma de los costos de stock de seguridad como S4
}// fin sumatoria de stock de seguridad

for(r=0;r<t;r++){                          //cálculo del fitness total de cada individuo
  fitnessPoblacion[r].fitness = Sumatoria1[r]+Sumatoria2[r]+Sumatoria3[r]+Sumatoria4[r];
  }
}//Fin calcular fitness
```

FUNCIÓN PARA CALCULAR EL FITNESS PARA PM 2006

```
double calcular_fitness_PM_2006(struct datosCliente cliente[c], struct datosInstalacion instalacion[w], struct datosInstalacion2 instalacion2[t][w],struct datosAsignacion asignacion[c][w],struct poblacionIndividuos poblacion[t][c],struct funcionObjetivo fitnessPoblacion[t]){
int i,j,r,x,q,dem,matrizInstalacion[t][c][w];
 double suma1,suma2,suma3,suma4,var,Sumatoria1[t],Sumatoria2[t], Sumatoria3[t], Sumatoria4[t];

for(r=0;r<t;r++){                        // matriz de asignación de instalaciones
  for(i=0;i<c;i++){
      int n=poblacion[r][i].poblacion_inicial;
      matrizInstalacion[r][i][n]=1;
      }
}

int ia[t][w];
for(r=0;r<t;r++){                        // matriz para ver si la instalación está cerrada o abierta
   suma1 =0.0;
   for(j=0;j<w;j++){
      int sum=0;
      for(i=0;i<c;i++){
         sum = sum + matrizInstalacion[r][i][j];
      }
       ia[r][j]=sum;
   }
}

for(r=0;r<t;r++){                              //se guarda la suma de los costos fijos como S1
   suma1 =0.0;
   for(j=0;j<w;j++){
      if(ia[r][j] != 0){
      suma1 = suma1 + (double)instalacion[j].CF;
     }
   }
   Sumatoria1[r]= suma1;
   fitnessPoblacion[r].S1=Sumatoria1[r];
}// fin sumatoria de costos fijos

for(r=0;r<t;r++){                              //cálculo de los costos de asignación
   suma2=0.0;
   for (i=0;i<c;i++){
      x = poblacion[r][i].poblacion_inicial;
      suma2 = suma2 + ((double)asignacion[i][x].TC);
    }
    Sumatoria2[r]=(double)TH*suma2;
    fitnessPoblacion[r].S2=Sumatoria2[r];    //se guarda la suma de los costos de asignación como S2
}// fin sumatoria de costos de asignación

for(r=0;r<t;r++){                              //cálculo de los costos de almacenamiento
   suma3 = 0.0;
   for(q=0;q<w;q++){
      if(instalacion[q].Qmax < instalacion2[r][q].Qopt){         //Se toma el Qmax como el Qminimo
        suma3                                  =                                  suma3                                  +
((((double)instalacion[q].OC*(double)instalacion2[r][q].D)/(double)instalacion[q].Qmax)+(((double)instalaci
on[q].HC*(double)instalacion[q].Qmax)/2));
      }
```

```
      else{
         if(instalacion2[r][q].Qopt == 0.0){
           suma3 = suma3 + 0.0;
         }
         else{                          //Se toma el Qwilson como el Qminimo
           suma3                                =                                suma3                                +
((((double)instalacion[q].OC*(double)instalacion2[r][q].D)/instalacion2[r][q].Qopt)+(((double)instalacion[q].
HC*instalacion2[r][q].Qopt)/2));
         }
       }
   }
   Sumatoria3[r] = (double)TH*suma3;
   fitnessPoblacion[r].S3=Sumatoria3[r];          //se guarda la suma de los costos de almacenamiento
como S3
}// fin sumatoria de los costos de almacenamiento

double mult[w];                                   //cálculo de los costos de stock de seguridad
for(r=0;r<t;r++){
   suma4 = 0.0;
   for(q=0;q<w;q++){

mult[q]=(((double)instalacion[q].HC)*(sqrt((double)instalacion2[r][q].U))*((double)(sqrt(instalacion[q].LT)))
*K);
      suma4 = suma4 + (mult[q]);
   }//fin for
   Sumatoria4[r] = (double)TH*suma4;
   fitnessPoblacion[r].S4=Sumatoria4[r];     //se guarda la suma de los costos de stock de seguridad como
}// fin sumatoria de stock de seguridad
for(r=0;r<t;r++){                                 //cálculo del fitness total
  fitnessPoblacion[r].fitness = Sumatoria1[r]+Sumatoria2[r]+Sumatoria3[r]+Sumatoria4[r];
  }
}
}//Fin calcular fitness
```

FUNCIÓN PARA CALCULAR PROBABILIDAD

```
double calcular_probabilidad(struct funcionObjetivo fitnessPoblacion[t]){
double sumaTotal,sumaAcumulada;
int i;
sumaTotal=0.0;
sumaAcumulada=0.0;

for(i=0;i<t;i++){
  sumaTotal= sumaTotal + fitnessPoblacion[i].fitness;          //calcula la suma total de la población para la
ruleta
  }

for(i=0;i<t;i++){
   fitnessPoblacion[i].probabilidad=((fitnessPoblacion[i].fitness*100.0)/sumaTotal)/100.0;          //guarda la
probabilidad de cada individuo
   sumaAcumulada=sumaAcumulada+fitnessPoblacion[i].probabilidad;          //calcula la suma acumulada
de para cada individuo
   fitnessPoblacion[i].probabilidad_acumulada=sumaAcumulada;          //guarda la probabilidad
acumulada para cada individuo
   }
}//Fin calcular probabilidad
```

FUNCION PARA SELECCIONAR INDIVIDUOS POR RULETA

```
int seleccionar_individuos_ruleta(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos poblacion[t][c]){
double r;
int i,j,indice,e;
for (i=0;i<t/2;i++){                    //generar valor aleatorio para hacer girar la ruleta
   r=(rand() % (100));                  //valor aleatorio entre 1 a 100
   r=r/100;
   for(j=0;j<t-1;j++){
     if((r>fitnessPoblacion[j].probabilidad_acumulada)                                    &&
(r<fitnessPoblacion[j+1].probabilidad_acumulada)){      //en que sección de la ruleta se encuentra el valor generado
         indice=j+1;
         for(e=0;e<c;e++){
            poblacion[i][e].poblacion_siguiente = poblacion[indice][e].poblacion_inicial;      //se guarda el individuo que se encuentra en la sección elegida por valor generado
          }
        }
   }
}
}//Fin seleccionar individuos
```

FUNCIÓN PARA SELECCIONAR INDIVIDUOS POR TORNEO

```
int seleccionar_individuos_torneo(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos poblacion[t][c]){
 int i,j,r,x1,x2,s,x,minimo,maximo,numeros[t],fitness_x1,fitness_x2;
 float p;
for (i=0;i<t;i++){                      //elige aleatoriamente padres sin repetirlos
   numeros[i] = (rand() %(t));
   if (i > 0){
     for(x=0; x<i; x++){
         if(numeros[i] == numeros[x]){
            i--;
            break;
         }
       }
    }
}
s=0;
for(i=0;i<t/2;i++){                     //de los padres escogidos se eligen dos según la probabilidad de selección
    x1=numeros[s];
   fitness_x1=fitnessPoblacion[x1].fitness;          //guarda fitness del primer individuo seleccionado
   x2=numeros[s+1];
   fitness_x2=fitnessPoblacion[x2].fitness;          //guarda fitness del segundo individuo seleccionado
    s=s+2;
   if(fitness_x1 < fitness_x2){
     minimo=x1;                         //según valor fitness asigna individuo a mínimo o máximo
     maximo=x2;
    }
   else{
    minimo=x2;
    maximo=x1;
    }
p=(rand() % (99));      //se genera aleatoriamente un valor entre 0 a 99  para probabilidad de selección
```

```
p=p/100;
if(p<Ps){            //si prob de seleccion es menor que ps se elige el fitness mas chico sino el mayor
  for(j=0;j<c;j++){
    poblacion[i][j].poblacion_siguiente = poblacion[minimo][j].poblacion_inicial;
     }
}
else {
   for(j=0;j<c;j++){
     poblacion[i][j].poblacion_siguiente= poblacion[maximo][j].poblacion_inicial;
    }
 }
}
}//Fin seleccionar individuos
```

FUNCIÓN PARA LLENAR LA MATRIZ CON INDIVIDUOS

```
int llenar_poblacion_siguiente(struct poblacionIndividuos poblacion[t][c]){
float p1,p2,s;
int i,j,x,y,r,cont,padre1[c], padre2[c], hijo1[c], hijo2[c],numeros[t/2], mut1,
mut2,punto_cruzamiento_poblacion,punto_mutacion_poblacion,temp1,temp2;
for (i=0;i<t/2;i++){          //elige padres para hacer las operaciones de cruzamiento y mutación
  numeros[i] = (rand() %(t/2));
  if (i > 0){
    for(x=0; x<i; x++){
        if(numeros[i] == numeros[x]){
          i--;
          break;  }
    }
   }
}
cont=0,s=0,r = t/2;
while(cont<t/4){             //ciclo para ver si realiza las operaciones a los padres seleccionados
   p1=(rand() % (99));      //se genera aleatoriamente un valor entre 1 a 99 para probabilidad de cruzamiento
   p1=p1/100;
   if(p1<=Pc){
      x=numeros[s];          //padre 1 seleccionado
      y=numeros[s+1];        //padre 2 seleccionado
      for (j=0;j<c;j ++){    //almacena los padres seleccionados para realizar el cruzamiento
          padre1[j]=poblacion[x][j].poblacion_siguiente;
       }
       for (j=0;j<c;j ++){
          padre2[j]=poblacion[y][j].poblacion_siguiente;
       }
 punto_cruzamiento_poblacion = (rand() % (c-2));    //se genera el pivote el azar para el cruce
for (j=0;j<c;j ++){
   hijo1[j]=  padre1[j];                            //se copia el padre2 como hijo1 para el intercambio
   hijo2[j]=  padre2[j];                            //se copia padre2 como hijo2 para el intercambio
}
if((punto_cruzamiento_poblacion==0)||(punto_cruzamiento_poblacion==1)){
   punto_cruzamiento_poblacion = punto_cruzamiento_poblacion +2;
} /* si el punto elegido es el comienzo del arreglo  se le sumará 2 ya que si no intercambiarían todos los
genes quedando el mismo individuo*/

 for(i=punto_cruzamiento_poblacion;i<c;i++){//a partir desde el pivote se realiza el intercambio
     temp1=hijo1[i];
     temp2=hijo2[i];
```

```
     hijo1[i]=temp2;
     hijo2[i]=temp1;
}

p2=(rand() % (99));                                    //valor generado para la probabilidad de mutar
p2=p2/100;
if(p2<=Pm){
   punto_mutacion_poblacion = (rand() % (c));          //se genera aleatoriamente el punto de mutación
   mut1= (rand() % (w));                               //se genera aleatoriamente el gen que se va a mutar
   hijo1[punto_mutacion_poblacion]=mut1;
   mut2= (rand() % (w));                               //se genera aleatoriamente el gen que se va a mutar
   hijo2[punto_mutacion_poblacion]=mut2;

for (j=0;j<c;j++){
     poblacion[r][j].poblacion_siguiente = hijo1[j];    //se guarda hijo1 generado por la mutación
}
for (j=0;j<c;j ++){
    poblacion[r+1][j].poblacion_siguiente = hijo2[j];   //se guarda hijo2 generado por la mutación
}
}//fin if mutacion
else{
     for (j=0;j<c;j++){
          poblacion[r][j].poblacion_siguiente = hijo1[j]; //se guarda hijo1 generado solo por el cruzamiento
     }
     for (j=0;j<c;j ++){
          poblacion[r+1][j].poblacion_siguiente = hijo2[j];//se guarda hijo1 generado solo por el cruzamiento
      }
 }//fin else
 cont++;
 r=r+2;
 s=s+2;
     }//fin if de cruzamiento
   } //fin while de llenado  se llena hasta que se cumple con el tamaño de la población
}//fin llenado de población siguiente realizando las operaciones de cruzamiento y mutación
```

FUNCION PARA INTERCAMBIAR POBLACION_PM_2004

```
int intercambiar_PM_2004(struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c], struct
datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]){
 int r,i,j,respuesta,sumaD,sumaU;

 for(r=0;r<t;r++){  //la población temporal se convierte en la población inicial de la siguiente iteración
    for(i=0;i<c;i++){
       poblacion[r][i].poblacion_inicial = poblacion[r][i].poblacion_siguiente;
   }
}

for(r=0;r<t;r++){
    for(j=0;j<w;j++){
        sumaD=0;
        sumaU=0;
        for(i=0;i<c;i++){
            if(j == poblacion[r][i].poblacion_inicial){
              sumaD = sumaD + cliente[i].demanda;//calcula la demanda total de los individuos generados
              sumaU = sumaU + cliente[i].varianza;//calcula la varianza total de los individuos generados
            }
```

```
        }
      instalacion2[r][j].D = sumaD;//guarda la demanda total de los individuos generados
      instalacion2[r][j].U = sumaU;//calcula la demanda total de los individuos generados
     }
}
for (r=0;r<t;r++){
   respuesta = comprobar_restriccion_PM_2004(r,instalacion,poblacion,instalacion2);//comprueba
factibilidad de cada individuo generado
   if(respuesta == 0){
     //si respuesta es 0 el individuo se debe arreglar
     generarIndividuos_PM_2004(r,poblacion,cliente,instalacion,instalacion2);
   }
  }
}//Fin intercambio de población siguiente a población inicial
```

FUNCION PARA INTERCAMBIAR POBLACION PARA PM 2006

```
int intercambiar_PM_2006(struct poblacionIndividuos poblacion[t][c],struct datosCliente cliente[c], struct
datosInstalacion instalacion[w],struct datosInstalacion2 instalacion2[t][w]){
int r,i,j,respuesta,sumaD,sumaU;

 for(r=0;r<t;r++){//la población temporal se convierte en la población inicial de la siguiente iteración
   for(i=0;i<c;i++){
     poblacion[r][i].poblacion_inicial = poblacion[r][i].poblacion_siguiente;
   }
 }

for(r=0;r<t;r++){
    for(j=0;j<w;j++){
       sumaD=0;
       sumaU=0;
       for(i=0;i<c;i++){
           if(j == poblacion[r][i].poblacion_inicial){
             sumaD = sumaD + cliente[i].demanda;      //calcula la demanda total de los individuos generados
             sumaU = sumaU + cliente[i].varianza;     //calcula la varianza total de los individuos generados
            }
       }
       instalacion2[r][j].D = sumaD;                  //guarda la demanda total de los individuos generados
       instalacion2[r][j].U = sumaU;                  //calcula la demanda total de los individuos generados
      instalacion2[r][j].Qopt                                                                          =
(double)sqrt((2*(double)instalacion[j].OC*(double)instalacion2[r][j].D)/(double)instalacion[j].HC);

      if(instalacion2[r][j].D == 0){
        instalacion2[r][j].Qopt=0.0;
        instalacion2[r][j].Qr=0.0;
        instalacion2[r][j].Qminimo=0.0;
      }
      else{
        if(instalacion2[r][j].Qopt > 0.0){
          instalacion2[r][j].Qminimo= instalacion2[r][j].Qopt;
         }
        else {
          if(instalacion[j].Qmax <= instalacion2[r][j].Qr){
            instalacion2[r][j].Qminimo= instalacion[j].Qmax;
          }
          else{
```

```
                if(instalacion2[r][j].Qr > 0.0)
              instalacion2[r][j].Qminimo= instalacion2[r][j].Qr; }
        }
       }
     }
   for(j=0;j<w;j++){
       instalacion2[r][j].restriccion = (double)instalacion2[r][j].Qminimo +
(2*K*(sqrt((double)instalacion[j].LT*(double)instalacion2[r][j].U)));
     }
}
for (r=0;r<t;r++){
    respuesta = comprobar_restriccion_PM_2006(r,instalacion,poblacion,instalacion2);
    if(respuesta == 0){ //si respuesta es 0 el individuo se debe arreglar
       generarIndividuos_PM_2006(r,poblacion,cliente,instalacion,instalacion2);
    }
  }
}//Fin intercambio de población siguiente a población inicial
```

FUNCION PARA ENCONTRAR EL MENOR FITNESS PARA PM_2004

```
double menor_fitness_PM_2004(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos
poblacion[t][c],char* nombre_archivo){
int j,i,r,g;
double temp_fitness[t],temp;
FILE *fichero;
fichero = fopen(nombre_archivo, "w" );
for(j=0;j<t;j++){              //se utiliza un temporal para ordenar los fitness
    temp_fitness[j]=fitnessPoblacion[j].fitness;
 }
for (i=1; i<t; i++){           //ordenamiento del los fitness de menor a mayor
    for (j=0;j<t-1;j++){
       if (temp_fitness[j] > temp_fitness[j+1]){
           temp = temp_fitness[j];
           temp_fitness[j] = temp_fitness[j+1];
           temp_fitness[j+1] = temp; }
    }
}

for (j=0;j<t;j++){             //mostrar la población con el menor fitness
  if (temp_fitness[0] == fitnessPoblacion[j].fitness){
      fprintf(fichero,"\nLa poblacion con el menor fitness es %d con un fitness de %f \n",j,temp_fitness[0]);
      fprintf(fichero,"Sumatoria1 %f\n",fitnessPoblacion[j].S1);
      fprintf(fichero,"Sumatoria2 %f\n",fitnessPoblacion[j].S2);
      fprintf(fichero,"Sumatoria3 %f\n",fitnessPoblacion[j].S3);
      fprintf(fichero,"Sumatoria4 %f\n",fitnessPoblacion[j].S4);

for(i=0;i<w;i++){
        fprintf(fichero,"\ninstalacion %d\n",i);
          for(g=0;g<c;g++){
             if(i == poblacion[j][g].poblacion_inicial){
               fprintf(fichero," %d ",g);
              }
            }
          }
     break;  }
    r=j+1;
```

```
  }
 fclose(fichero);
}//Fin mostrar el menor fitness
```

FUNCION PARA ENCONTRAR EL MENOR FITNESS PARA PM_2006

```
double menor_fitness_PM_2006(struct funcionObjetivo fitnessPoblacion[t],struct poblacionIndividuos poblacion[t][c],char* nombre_archivo){
int j,i,r,g;
double temp_fitness[t],temp;
FILE *fichero;
fichero = fopen(nombre_archivo, "w" );

for(j=0;j<t;j++){              //se utiliza un temporal para ordenar los fitness
  temp_fitness[j]=fitnessPoblacion[j].fitness;
 }

for (i=1; i<t; i++){           //ordenamiento del los fitness de menor a mayor
    for (j=0;j<t-1;j++){
      if (temp_fitness[j] > temp_fitness[j+1]){
          temp = temp_fitness[j];
          temp_fitness[j] = temp_fitness[j+1];
          temp_fitness[j+1] = temp;
        }
    }
}

for (j=0;j<t;j++){             //mostrar la poblacion con el menor fitness
  if (temp_fitness[0] == fitnessPoblacion[j].fitness){
     fprintf(fichero,"\nLa poblacion con el menor fitness es %d con un fitness de %f \n",j,temp_fitness[0]);
     fprintf(fichero,"Sumatoria1 %f\n",fitnessPoblacion[j].S1);
     fprintf(fichero,"Sumatoria2 %f\n",fitnessPoblacion[j].S2);
     fprintf(fichero,"Sumatoria3 %f\n",fitnessPoblacion[j].S3);
     fprintf(fichero,"Sumatoria4 %f\n",fitnessPoblacion[j].S4);

     for(i=0;i<w;i++){
        fprintf(fichero,"\ninstalacion %d\n",i);
          for(g=0;g<c;g++){
             if(i == poblacion[j][g].poblacion_inicial){
                fprintf(fichero," %d ",g);
              }
            }
          }
     break;
 }
    r=j+1;
 }/
 fclose(fichero);
}//Fin del la búsqueda del menor fitness
```

Referencias

Abdul, B., Betancor, J. 2004. Sistemas de distribución: avances en la gestión de inventarios. *Serie de Tesis doctorales. Ciencias y Tecnologias/24.* Universidad de La Laguna : s.n., 2004.

Alvarez de los Mozos, E., Zubillaga, F. 2003. Programación de operaciones en planta en entornos cambiantes. . *V Congreso de Ingeniería de Organización.* Valladolid, Burgos : s.n., 2003.

Arapoglu, R. A., Norman, B. A., Smith, A. E. 2001. A Genetic Algorithm Approach to Input/Output Station Location in Facilities Design. *Evolutionary Computation, IEEE Transactions* . [Department of Industrial Engineering,University of Pittsburgh]. Pittsburgh, Pennsylvania, USA : s.n., 2001.

Bramel, J. 2000. The logic of logistic: Theory, Algorithms and Applications for logistics Management. New York, United Stated : Springer-Verlag, 2000. págs. 203-216.

Caballero, R., Gomez, T., Luque, M., Molina, J., Torrico, A. 2002. A Genetic Algorithm to solve an Integer Goal Programming Model for the Higher Education. *EU/ME European Chapter on Metaheuristics.* Paris, Francia : s.n., 2002.

Cachon, G. 2000. Supply chain inventory management and value of shared information. *Management Science.* Pennsylvania, The University of Pennsylvania, United Stated : s.n., 2000. Vol. 8, 46, págs. 1032-1048.

Chen, F., Drezner, Z., Ryan, J, Simcgi-Levi, D. 2000. Quantifying the bullwhip effect in a sample supply chain: the impact of forecasting, lead times, and information. *Management Science.* 2000. Vol. 3, 46, págs. 436-443.

Daskin, M.S. 2003. Facility Location in Supply Chain Design. *Department of Industrial Engineering and Management Sciences.* Northwestern University,Evanston, Illinois. : s.n., 2003.

De Jong K.A., Spears W. M. 1989. Using Genetic Algorithms to Solve NP-complete Problems. *Proceeding of the third International Conference of Genetic Algorithms.* George Mason University, United Stated : s.n., 1989.

Feige, U. 2005. Rigorous Analysis of Heuristics for NP-hard Problems. *Manuscript corresponding to invited presentations in SODA.* Vancouver : s.n., 2005.

Fouskakis D., Draper D. 2002. Stochastic Optimization : a review. *International Statistical Review.* 2002. Vol. 3, 70, págs. 315-349.

Glover, F. 1977. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Science.* United Stated, University of Colorado : s.n., 1977. Vol. 1, 8, págs. 156-166.

Gutierrez Exposito, J. M. 2008. Métodos eficientes para algunas variantes del Modelo EOQ. *Departamento de Estadística, Investigación de Operaciones y Computación.* 2008.

Gutiérrez, V., Vidalb, C.J. 2008. Modelos de Gestión de Inventarios en Cadenas de Abastecimiento: Revisión de la Literatura. *Rev. Fac. Ing. Univ. Antioquia* . [Universidad de Antioquia, Departamento de Ingeniería Industrial. A.A. 1226,Universidad del Valle, Escuela de Ingeniería Industrial y Estadística]. Medellín, Colombia : s.n., Marzo de 2008. 43, págs. 133-149.

Hernández R., Fernández C., Baptista P. 1998. *Metodología de la investigación.* 4ta Edición. DF., México : McGraw Hill, 1998.

Hernández, R., Fernandez C., Baptista P. 1998. *Metodología de la investigación.* 2da Edición. DF., México : McGraw Hill, 1998.

Herrera Povis, A. R. 2006. Sistemas de Inventarios. *Tesis.* Lima, Peru : s.n., 2006.

Holland, J.H. 1995. Hidden order: how adaptation builds complexity. Massachusetts, United Stated : Addison-Wesley, 1995.

Itaim Ananias, P. 2005. Resolucion del problema de Set Covering utilizando un Algoritmo Genetico. 2005.

Lee, H. 2000. The value of information sharing in a two level suppliy chain. *Management Science.* Ciudad de Mexico, Mexico : s.n., 2000. Vol. 5, 43, págs. 626-643.

Leung, J. M. Y., Magnanti, T. L. 1986. Valid Inequalities and Facets of the Capacitated Plant Location Problem. Springer Berlin 1986. Vol. 44, 1-3, págs. 271-291.

Martí Cunquero, R. 2003. Procedimientos Heurísticos en Optimización Combinatoria. Valencia, Departament di Estadistica i Investigacion Operativa, Universitat de València, España : s.n., 2003. Vol. 1, 1, págs. 3-62.

Miranda, P., Garrido, R. 2005. A Simultaneous inventory control and facility location model with stochasti constraint of inventory capacity. *Networks and Spatial Economics Forthcoming.* Santiago, Chile : s.n., 2005.

Miranda, P., Garrido, R. 2004. Incorporating inventory control decisions into strategic distribution network design model with stochastic demand. [Trasportation Research Part E]. Chile : s.n., May de 2004. Vol. 3, 40, págs. 183-207.

Miranda, P., Garrido, R. 2006. Valid inequalities for Lagrangian relaxation in an inventory problem with stochastic capacity. *Transportation Research Part E.* Chile : s.n., 2006. 44, págs. 47-65.

Puchinger J., Raidl G. 2005. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification. *In Proceeding og the International Work Conference on the Interplay Between Natural and Artificial Computation.* s.l. : Springer, 2005. Vol. 3, 562, págs. 41-53.

Sáez, Eduardo. 2003. Fundamentos de Investigación de Operaciones: Teoría de Inventario. *Universidad Técnica Federici Santa María.* Valaparaíso, Chile : s.n., 2003.

Scaparra, M.P., Scutellà, M.G. 2001. Facilities, Locations, Customers: Building Blocks of Locations Model. A Survey. *Technical Report: TR-01-08.* Pisa, Università Di Pisa, Italia : s.n., 2001.

Shu, J., Teo, C., Shen, Z. 2005. Stochastic transportation-inventory network design problem. *Operations Research.* 2005. 53, págs. 48-60.

Sourirajan, K., Ozsen, L., Uzsoy,R. 2008. A genetic algorithm for a single product network design model with lead time and safety stock considerations. New York, USA : European Journal of Operational Research, 2008.

Sourirajan, K., Ozsen, L., Uzsoy,R. 2006. A single-product network design model with lead time and safety stock considerations. New York, USA : s.n., 2006. 39, págs. 411-424.

Velez, M.C., Montoya, J. A. 2007. Metaheurísticos: Una Alternativa para la solución de Problemas Combinatorios en Administración de Operaciones. *Revista EIA, ISSN 1794-1237.* Medellín, Escuela de Ingeniería de Antioquía, Colombia : s.n., Diciembre de 2007. 8, págs. 99-115.

Zhang, J., Che, B., Ye, Y. 2005. A Multi-Exchange Local Search Algorithm for the Capacitated Facility Location Problem. *Mathematics of Operations Research.* 2005. Vol. 2, 30, págs. 389-403.