Trabajo Fin de Máster
Organización Industrial y Gestión de Empresas

Problema de Transporte MultiTerminal y Contenedores MultiSize: Dos enfoques metaheurísticos con Python.

Autora: Ana Isabel Pérez Cano
Tutor: Alejandro Escudero Santana

Dep. de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Máster
Organización Industrial y Gestión de Empresas

Problema de Transporte MultiTerminal y Contenedores MultiSize: Dos enfoques metaheurísticos con Python.

Autora:
Ana Isabel Pérez Cano

Tutor:
Alejandro Escudero Santana
Profesor Ayudante Doctor

Dep. de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Máster: Problema de Transporte MultiTerminal y Contenedores MultiSize: Dos enfoques metaheurísticos con Python.

Autora: Ana Isabel Pérez Cano

Tutor: Alejandro Escudero Santana

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Resumen

Este trabajo se centra en el problema del enrutamiento de vehículos como un sistema intermodal de transporte donde se produce movimientos de entrega y recogida de contenedores por camiones y determinadas a priori por el cliente. Este sistema intermodal de transporte se puede identificar como operaciones de *drayage* y además se verá como parte de un gran objetivo, una cadena de suministro.

Las operaciones de *drayage* se pueden considerar como uno de los elementos que ocupa mayor importancia en cuanto al total de los costes de transporte que presenta una cadena de suministro. En consecuencia, es esencial buscar que estas operaciones se realicen de la forma más eficiente posible tanto en costes como en calidad, para poder así conseguir cada vez más que las empresas puedan expandirse a mayores niveles en el mercado.

En el problema básico de recogida y entrega se busca construir un conjunto de rutas para satisfacer unas solicitudes/tareas de entrega y recogida de clientes dispersos geográficamente. Se parte de una flota de vehículos disponible, donde cada vehículo tiene una capacidad dada, un tiempo de operación, un lugar de inicio y un lugar final. En cada solicitud se especifica el tamaño del contenedor a transportar, donde se va a recoger (origen) y donde se va a entregar (destino). Estos contenedores son transportados de su origen a su destino sin transbordo en otros lugares.

El objetivo por tanto es de satisfacer todas las solicitudes por los vehículos disponibles generando una solución a coste mínimo, sin olvidar las restricciones adicionales que se definan para el problema.

En la literatura se puede encontrar una gran variedad de enfoques posibles de solución para el tipo de problema expuesto. De las posibles metodologías de resolución aplicables relativas al problema de *drayage*, se eligen dos de ellas: Recocido Simulado y Búsqueda Tabú.

La búsqueda tabú es capaz de presentar buenas soluciones para estos problemas, sin embargo, su aplicación es menos popular. En cuanto al recocido simulado, método también bastante conocido y que obtiene resultados considerablemente buenos, se diferencia con la búsqueda tabú en que éste al no verificar el vecindario completo, le permite moverse más rápido entre vecindarios y con ello, realizar mayor número de iteraciones en un tiempo dado. Estos dos enfoques de solución se basan en resolver un problema por búsqueda local, basados en una solución inicial y una estructura de vecindario.

En función de observar la aplicación en el mundo real de este trabajo, se procede simular en *Spyder*, entorno de desarrollo de código para programación científica en lenguaje *Python*, para generar soluciones en los dos casos posibles, recocido simulado y búsqueda tabú. Se añaden tres escenarios con diferentes parámetros en los dos casos, generando diversos experimentos computacionales para verificar, comparar y así poder concluir las comparaciones en el rendimiento entre métodos y el conjunto de parámetros establecido en cada escenario. Estas comparaciones se basan en el coste total (la distancia total recorrida) y el tiempo computacional que requiere ejecutarlos hasta obtener solución con los parámetros establecidos.

Abstract

This work focuses on the problem of vehicle routing as an intermodal transport system where delivery and pick-up moves of containers are produced by trucks and determined a priori. This intermodal transport system can be identified as drayage operations and will also be seen as part of a major objective, a supply chain.

Drayage operations can be considered as one of the most essential elements in terms of the total transportation costs of a supply chain. Consequently, it is crucial to ensure that these operations are carried out in the most efficient manner possible in terms of costs and quality, in order to achieve that companies can expand to higher levels in the world market.

In the pick-up and delivery basic problem, it is sought to build a set of routes to satisfy the requests/tasks (delivery and pick-up tasks) of geographically dispersed customers. It will be given a fleet of available vehicles, where vehicles have a capacity, a operating time, a starting place and a final place. Each task specifies the size of the container to be transported, where it will be picked-up (origin) and where it will be delivered (destination). These containers are transported from their origin to their destination without transshipment in other places.

The objective therefore is to satisfy all the requests for the available vehicles generating a solution at minimum cost, without forgetting the additional restrictions that are defined for the problem

In the literature you can find a wide variety of possible approaches to solve the type of problem exposed. Among the possible applicable resolution methodologies related to the drayage problem, two of them are chosen: Simulated Annealing and Tabu Search.

Tabu search can present good solutions for these problems, however, its application is less popular. As for simulated annealing, a method that is also well known and that obtains considerably satisfactory results, it differs from the tabu search in that it does not verify the entire neighborhood, so that way can move faster between neighborhoods and performs a greater number of iterations in a given time. These two solution approaches are based on solving a problem by local search, based on an initial solution and a neighborhood structure.

To observe the real-world application of this work, we proceed to simulate in Spyder, a code development environment for scientific programming in Python language, to generate solutions in the two possible cases, simulated annealing and tabu search. Three scenarios with different parameters are added in both cases, generating different computational experiments to verify, compare and thus be able to conclude the comparisons in the performance between methods and the set of parameters established in each scenario. These comparisons are based on the total cost (the total distance traveled) and the computational time required to execute them until obtaining a solution with the established parameters.

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

Este trabajo se centra en el problema del enrutamiento de vehículos como un sistema intermodal de transporte donde se produce movimientos de entrega y recogida de contenedores por camiones y determinadas a priori por el cliente. Este sistema intermodal de transporte se puede identificar como operaciones de *drayage* y además se verá como parte de un gran objetivo, una cadena de suministro.

Las operaciones de *drayage* se pueden considerar como uno de los elementos que ocupa mayor importancia en cuanto al total de los costes de transporte que presenta una cadena de suministro. En consecuencia, es esencial buscar que estas operaciones se realicen de la forma más eficiente posible tanto en costes como en calidad, para poder así conseguir cada vez más que las empresas puedan expandirse a mayores niveles en el mercado.

Con el desarrollo de este trabajo se pretende obtener un enrutamiento optimizado para el problema de *drayage* que se presenta mediante dos técnicas de resolución metaheurísticas y a través del lenguaje de programación de Python.

1.1 Antecedentes.

La cadena de suministro es una red de organizaciones que están involucradas, a través de vínculos ascendentes y descendentes, en los diferentes procesos y actividades que producen valor en forma de productos y servicios en manos del consumidor final (M. Christopher). La gestión de esta cadena se refiere a la coordinación de la producción, el inventario, la ubicación y el transporte entre las organizaciones involucradas en la cadena de suministro para lograr la mejor combinación de capacidad de respuesta y eficiencia para el mercado que se sirve (M. Hugos).

La logística es la parte de la gestión de la cadena de suministro que trata de obtener el producto adecuado, para el cliente adecuado, en la cantidad adecuada, en las condiciones adecuadas, en el lugar correcto, en el momento adecuado y al coste correcto (John J. Coyle et al).

Un sistema logístico se puede definir a grandes rasgos como un conjunto de tres elementos principales: el procesamiento de pedidos, el almacenamiento y el transporte. Los cuales están directamente respaldados por servicios de información logística que producirán y analizarán información relativa al transporte de mercancías entre el sistema. Y servicios adicionales necesarios para el proceso logístico.

El elemento de procesamiento de pedidos constituye un elemento indispensable para la iniciación y monitoreo de procesos logísticos. El procesamiento de pedidos comprende el procesamiento y la supervisión de ordenar datos desde el momento del pedido hasta la llegada de los productos al sitio del cliente y de la devolución de los documentos de envío.

El elemento de almacenamiento denota el almacenamiento de mercancías. Las funciones de recolección y empaquetamiento entran en la categoría de servicios adicionales. La recolección hace referencia a la compilación de artículos vendibles de acuerdo con los pedidos individuales de los clientes para formar paquetes y unidades de envío.

Finalmente, el elemento de transporte desplaza la mercancía desde el sitio de producción hasta la ubicación de almacenamiento y la ubicación de entrega en el sitio del cliente. Por lo tanto, es importante la diferencia espacial entre el suministro y la demanda.

El elemento de transporte es la actividad económica más importante entre los componentes de un sistema logístico. Entre un tercio y dos tercios de los gastos de los costos logísticos de las empresas se

gastan en transporte. La Figura 1 da una mejor visión de qué categorías de costos se consideran en un sistema logístico y qué parte del total representan.

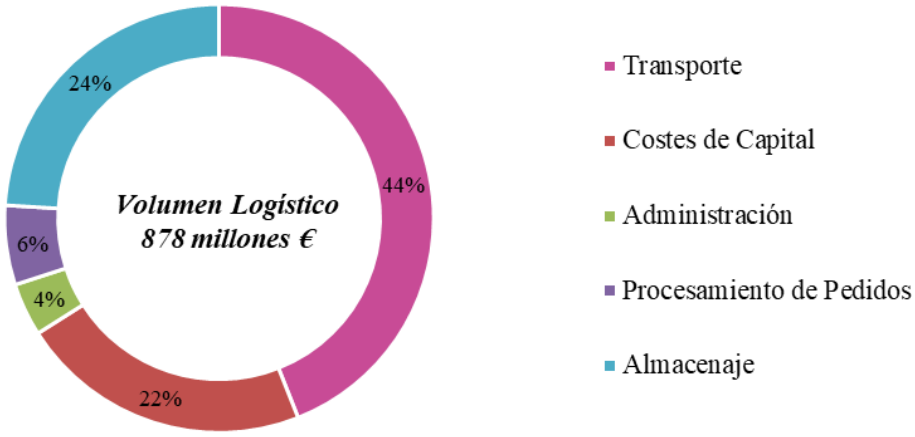


Figura 1. Volumen logístico y coste de componentes en EU28 en 2012. Fuente: Fraunhofer SCS.

En consecuencia, el servicio de transporte se podría considerar uno de los elementos claves en una cadena logística. Por lo que las empresas deben afrontar requerimientos cada vez más exigentes para poder sobrevivir tanto a nivel nacional como internacional. Es decir, un buen servicio de transporte en las actividades logísticas podría proporcionar una mejor eficiencia logística, reducir los costos de operación y promover la calidad del servicio.

La mayor parte del transporte de mercancías se realiza por carretera, mientras que los servicios ferroviarios y de agua se encuentran más desorganizados en su funcionamiento y coordinación. La Figura 2 muestra la división modal producida en dos periodos de tiempo, de cinco modos de transporte: carretera, ferrocarril, vías navegables, aire y marítimo. En 2015, la carretera representó poco más de la mitad de todas las toneladas-kilómetros realizadas en la UE-28. El transporte marítimo siguió con cerca de un tercio del total del transporte, seguido por el ferrocarril (12,3%) y las vías navegables interiores (4,3%). En términos de toneladas-kilómetros, el transporte aéreo desempeña un papel marginal a nivel de la UE, con una cuota del 0,1%.

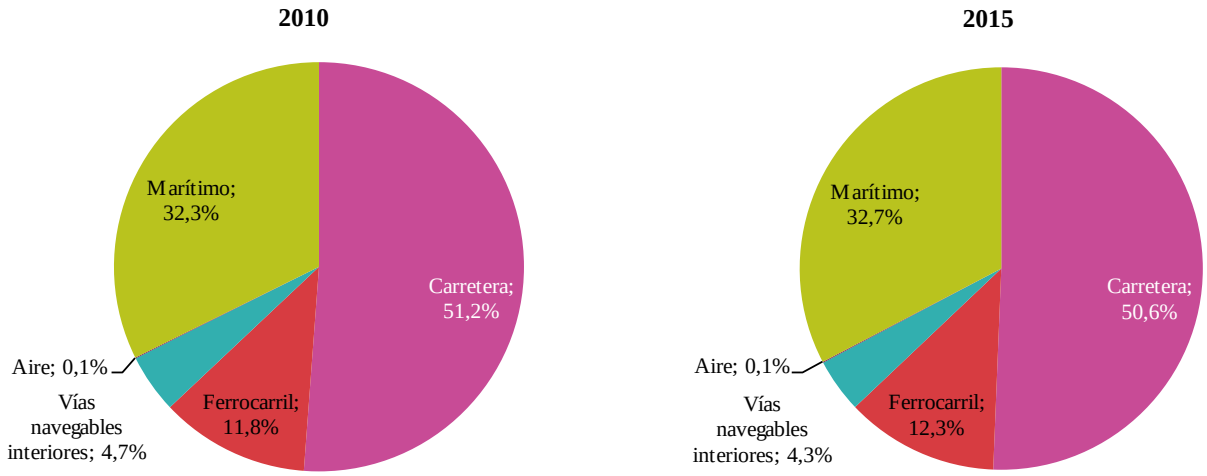


Figura 2. Transporte de mercancías en la UE-28: reparto modal (% del total de tn-km). Fuente: Eurostat, 2017.

Nota: El transporte aéreo y marítimo sólo cubre los transportes intracomunitarios (transporte desde / hacia los países de la UE) y excluye los transportes extracomunitarios.

Estos modos de transporte (aire, aguas interiores, marítimo, ferrocarril o carretera) se han visto influenciados por la evolución tecnológica: con el impulso de la contenerización a mediados de los años 1900 y un nuevo enfoque en la logística y los requerimientos de la cadena de suministro global,

se establece el escenario para el crecimiento del transporte intermodal. Las más reciente tendencias en logística en Europa se centran en la importancia del papel tan creciente que está teniendo el transporte intermodal debido a:

- Las necesidades cambiantes de los clientes y la hipercompetición de las cadenas de suministro en un mercado global;

- La necesidad de responder con fiabilidad y flexibilidad mediante una coordinación integrada e integrada a través de varios modos;

- Las limitaciones y la coordinación de la capacidad de infraestructura, incluidas las cuestiones normativas y reglamentarias.

La intermodalidad permite que al menos dos modos diferentes de transporte se utilicen de manera integrada en una cadena de transporte. La integración entre los modos se puede abordar en tres niveles: (1) infraestructura y medios de transporte "hardware" (por ejemplo, intensificar el diseño intermodal de las redes transeuropeas de transporte, mejorar el diseño y las funciones de los puntos de transferencia intermodales y armonizar las normas para los medios de transporte.). (2) las operaciones y el uso de la infraestructura (operaciones interoperables e interconectadas como la integración de las autopistas de mercancías en un contexto intermodal, desarrollo de precios y precios comunes principios, armonización de las normas de competencia y regímenes de ayudas estatales de forma intermodal.). (3) servicios y regulación (de un marco modal a un modo independiente, es decir, armonización y normalización de procedimientos e intercambio electrónico de datos, responsabilidad intermodal, investigación y demostración, benchmarking y estadísticas intermodales).

Se fija entonces como objetivo el poder desarrollar un marco para una integración óptima de diferentes modos de transporte a fin de permitir un uso eficiente y rentable de una cadena de transporte a través de servicios de puerta a puerta transparentes y orientados al cliente favoreciendo la competencia entre los operadores de transporte.

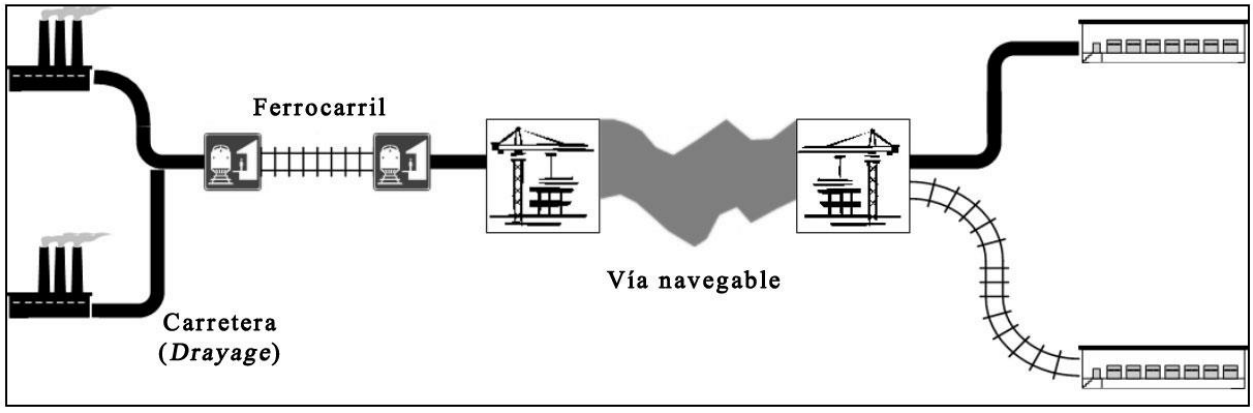


Figura 3. Ejemplo de red de transporte intermodal. Fuente: T. Bektas y T.G. Crainic.

Gracias al transporte intermodal de mercancías podemos obtener un servicio de transporte mejor diseñado, más flexible, más seguro, las rutas de los vehículos y los plazos de entrega se planifican con más consistencia, lo que, en consecuencia, puede llevar a precios más bajos.

Hay que tener en cuenta que para conseguir la intermodalidad, es necesario la estandarización de la mercancía, es decir, es necesario el uso de unidades de transporte estandarizadas que pueden transferirse fácilmente de un modo de transporte a otro sin manipular las mercancías mismas.

Una de estas unidades de transporte estandarizadas son los contenedores. Un contenedor se puede

definir como una caja metálica grande de tamaño estándar en la que se embala la mercancía para su transporte a bordo de modos de transporte especialmente configurados para ello. Están diseñados para moverse con equipos de manipulación comunes que permiten transferencias intermodales de alta velocidad en unidades económicamente grandes entre buques, vagones, chasis de camiones y barcazas utilizando un mínimo de mano de obra. El tamaño de referencia de estos contenedores es la caja de 20 pies. También se utiliza la caja de 40 pies y el "Hi Cube" contenedores que son un pie más alto al tamaño estándar de 20 pies.

El uso de esta unidad de transporte estandariza presenta ciertas ventajas:

- Unidad estándar. Un contenedor puede ser manipulado en cualquier parte del mundo ya que sus dimensiones son un estándar ISO. La rápida difusión fue facilitada por el hecho de que su iniciador McLean no patentó su invención, por lo cual todos los segmentos de la industria podían acceder a la norma.

- Flexibilidad. Un contenedor puede transportar una amplia variedad de mercancías que van desde materias primas, productos manufacturados y automóviles hasta productos congelados. Existen también contenedores especializados para el transporte de líquidos y productos alimenticios perecederos en contenedores refrigerados.

- Administración. El contenedor lleva un número de identificación único y un código de tipo de tamaño que permite la gestión del transporte no en términos de cargas, sino en términos de unidad. La gestión informatizada permite reducir los tiempos de espera considerablemente y conocer la ubicación de los contenedores en cualquier momento. Permite asignar los contenedores según la prioridad, el destino y las capacidades del transporte disponibles.

- Velocidad. Las operaciones intermodales son mínimas y rápidas, se estima que la contenerización ha reducido el tiempo en un factor del 80%.

- Almacenamiento. El contenedor limita los riesgos de daños para las mercancías dado a su diseño resistente. Con ello, se establece un embalaje de los productos que contiene más simple, menos costoso y puede ocupar menos volumen. Esto reduce los costos de seguro, ya que la carga es menos propensa a ser dañada durante el transporte.

- Seguridad. El contenido del contenedor es anónimo, ya que sólo puede abrirse en el origen, en la aduana y en el destino. Los robos, especialmente los de productos de valor, se reducen considerablemente, lo que se traduce en una reducción de las primas de seguro.

El término usado en transporte intermodal de mercancías para determinar el transporte por carretera (con camiones) de mercancías en contenedores es *drayage*. Independientemente del lugar donde vaya el contenedor, *drayage* se utiliza específicamente para rutas de camiones de distancias cortas y se usa con frecuencia para conectar los segmentos de transporte marítimo o ferrovial de línea de servicio regular en una cadena de transporte de mercancías más grande.

Las investigaciones sobre *drayage* son relativamente tempranas dado que los acontecimientos recientes han sido los que han despertado el interés de varios grupos por este tema. En cuanto a estos acontecimientos, económicamente hablando, se puede apreciar que el crecimiento del comercio globalizado ha derivado en un gran aumento de las importaciones y exportaciones en contenedores estandarizados.

Además, si se tiene en cuenta con el aumento del coste de combustible, se limitan las opciones de reducción de costos a lo largo de la cadena de suministro para las empresas. Es por ello por lo que,

aunque el *drayage* es un elemento muy pequeño en todo el concepto general de la cadena de suministro, sus costos y problemas potenciales pueden ser desproporcionadamente altos si no se tienen en cuenta.

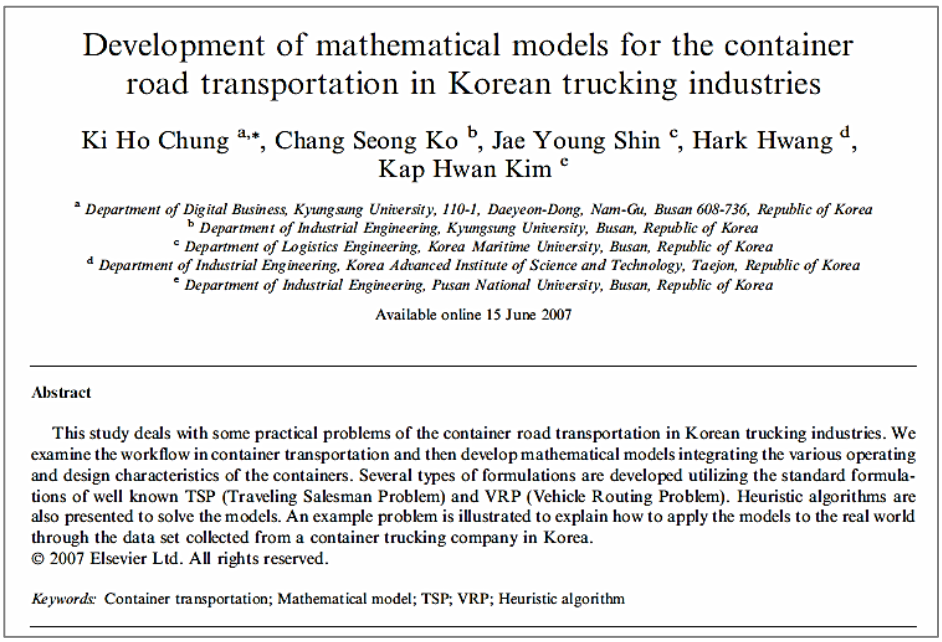
Reducir costes los costes del *drayage* es unos de los aspectos que más se buscan intercontinentalmente. En cuanto a corta distancia, esta reducción se puede ver como la reducción del tiempo de viaje por igual. De modo que la planificación eficiente de estas operaciones de *drayage* son una tarea importante para las empresas, ya que representan una cantidad considerable del costo total de un transporte intermodal (Smilowitz, 2006). Especialmente, son los movimientos de contenedores vacíos a los que se debe prestar atención, ya que son las actividades costosas que no generan ingresos.

1.2 Objetivo del trabajo.

Como se ha visto, las operaciones de transporte de mercancías requieren de una gestión eficaz, ya que una gestión eficaz nos permitiría grandes ahorros logísticos. Debido a la alta competencia en el mercado global, las empresas cada vez más se enfocan en invertir en su sistema logístico y de ahí que buscar la optimización de las operaciones de logísticas juega un papel fundamental.

La gestión eficaz del transporte se puede considerar en tres niveles de planificación: estratégica (ubicación de las localizaciones de carga y descarga), táctica (números de modo de transporte, tamaño de las cargas, etc.) y operacional (enrutamiento y la programación del modo de transporte). Teniendo en cuenta los niveles mencionados, las empresas entonces se enfocan en la búsqueda de una solución de transporte que permita obtener la óptima configuración de distribución que los lleve al objetivo deseado, realizar sus operaciones de transporte a costo mínimo.

En consecuencia, el estudio de este trabajo se centra en el problema práctico del *drayage*, es decir, el transporte por carretera (con camiones) de mercancías en contenedores y como conseguir la configuración óptima de este servicio de transporte con una serie de restricciones. Más concretamente, este estudio se basa en la investigación realizada por Ki Ho Chung, Chang Seong Ko, Jae Young Shin, Hark Hwang y Kap Hwan Kim en "Development of mathematical models for the container road transportation in Korean trucking industries" de la Figura 4:



Development of mathematical models for the container road transportation in Korean trucking industries

Ki Ho Chung [a,*], Chang Seong Ko [b], Jae Young Shin [c], Hark Hwang [d], Kap Hwan Kim [c]

[a] Department of Digital Business, Kyungsung University, 110-1, Daeyeon-Dong, Nam-Gu, Busan 608-736, Republic of Korea
[b] Department of Industrial Engineering, Kyungsung University, Busan, Republic of Korea
[c] Department of Logistics Engineering, Korea Maritime University, Busan, Republic of Korea
[d] Department of Industrial Engineering, Korea Advanced Institute of Science and Technology, Taejon, Republic of Korea
[e] Department of Industrial Engineering, Pusan National University, Busan, Republic of Korea

Available online 15 June 2007

Abstract

This study deals with some practical problems of the container road transportation in Korean trucking industries. We examine the workflow in container transportation and then develop mathematical models integrating the various operating and design characteristics of the containers. Several types of formulations are developed utilizing the standard formulations of well known TSP (Traveling Salesman Problem) and VRP (Vehicle Routing Problem). Heuristic algorithms are also presented to solve the models. An example problem is illustrated to explain how to apply the models to the real world through the data set collected from a container trucking company in Korea.

Keywords: Container transportation; Mathematical model; TSP; VRP; Heuristic algorithm

Figura 4. Portada del artículo: "Development of mathematical models for the container road transportation …".

De acuerdo con Bodin, Golden, Assad y Ball (1983), y Savelsbergh y Sol (1995), los problemas con

contenedores se pueden aproximar a un problema de enrutamiento del vehículo con recogida y entrega (Pick-up and Delivery, VRPPD). Este tipo de problemas viene de una derivación del problema de optimización combinatoria del enrutamiento del vehículo (VRP), el cual a su vez generaliza al conocido problema del viajante ambulante (TSP).

En definitiva, el problema de recogida y entrega en el caso de estudio de este trabajo consiste en el enrutamiento de un modo de transporte por carretera (una flota de camiones, sin limitación) que debe transportar carga moviendo contenedores de diferente tamaño (20 pies y 40 pies) entre terminales y un depósito inicial/final, dado un conjunto de solicitudes de recogida y entrega conocidas. De manera que el objetivo que se busca es entonces minimizar la distancia total de circulación de los camiones dentro de un límite de horas operativas para poder realizar todas las operaciones de transporte de mercancías en contenedores al menor coste.

1.3 Estructura del trabajo.

Con el desarrollo de este trabajo se pretende obtener un enrutamiento optimizado para el problema de *drayage* que se presenta mediante la aplicación de dos metaheurísticas, Búsqueda Tabú y Recocido Simulado, a través del lenguaje de programación de Python. Para ello, se sigue el esquema que muestra la Figura 5.

Figura 5. Estructura del trabajo.

Como ya se ha visto, en la sección 1 se introduce el tema de estudio de este trabajo, el problema del transporte de mercancías en contenedores junto al objetivo que se busca con el desarrollo del trabajo y su estructura. En la sección 2 se presenta y define el problema práctico de *drayage* de aplicación real a resolver en este estudio, además de una revisión literaria de investigaciones realizadas sobre el tema de estudio. En la sección 3 se expone una revisión de las metodologías de resolución disponibles hasta la fecha que se han aplicado para el tipo de problema en cuestión, además se eligen dos de ellas y se explica su adaptación al problema. En la cuarta sección se muestran los resultados de la aplicación de las dos metaheurísticas a través de Python con diferentes experimentos computacionales. En la última y quinta sección se analizan los resultados obtenidos.

2 EL PROBLEMA DE DRAYAGE MULTITERMINAL Y MULTISIZE

El estudio de este trabajo se centra en el problema práctico del *drayage*, es decir, el transporte por carretera (con camiones) de mercancías en contenedores y como conseguir la configuración óptima de este servicio de transporte con una serie de restricciones. Más concretamente, este estudio se basa en la investigación realizada por Ki Ho Chung, Chang Seong Ko, Jae Young Shin, Hark Hwang y Kap Hwan Kim en "Development of mathematical models for the container road transportation in Korean trucking industries".

De acuerdo con Bodin, Golden, Assad y Ball (1983), y Savelsbergh y Sol (1995), los problemas con contenedores se pueden aproximar a un problema de enrutamiento de vehículos con recogida y entrega (Pick-up and Delivery, VRPPD). Este tipo de problemas viene de una derivación del problema de optimización combinatoria del enrutamiento del vehículo (VRP), el cual a su vez generaliza al conocido problema del viajante ambulante (TSP). Para más información detallada relativa a los problemas de optimización combinatoria y sus problemas más conocidos como son el TSP y el VRP, veasé el Anexo I.

En el problema básico de recogida y entrega se busca construir un conjunto de rutas para satisfacer unas solicitudes/tareas de entrega y recogida de clientes dispersos geográficamente. Se parte de una flota de vehículos disponible, donde cada vehículo tiene una capacidad dada, un tiempo de operación, un lugar de inicio y un lugar final. En cada solicitud se especifica el tamaño del contenedor a transportar, donde se va a recoger (origen) y donde se va a entregar (destino). Estos contenedores son transportados de su origen a su destino sin transbordo en otros lugares.

El objetivo por tanto es de satisfacer todas las demandas de los clientes por los vehículos disponibles generando una solución a coste mínimo, sin olvidar las restricciones que se definan para el problema. Estas restricciones son las características que puede presentar diferentes variaciones de este problema. En la siguiente subsección se enumeran las más populares.

2.1 Características del problema.

2.1.1 Solicitudes de transporte.

Una característica clave de estos problemas de enrutamiento es la forma en que las solicitudes de transporte están disponibles. Se pueden dar dos situaciones: una situación estática, todas las solicitudes se conocen a priori; y una situación dinámica, puede haber solicitudes por determinar a priori, es decir, los datos se conocen una vez empezado el enrutamiento. Por consiguiente, cuando se da una situación dinámica y entra una nueva solicitud, al menos una de las rutas configuradas debe modificarse para incluir dicha solicitud.

Otro concepto importante en los problemas de enrutamiento son los tipos de unidades de transporte estandarizadas (contenedores) que se emplean en el enrutamiento. Como ya se mencionaba en una sección anterior, un contenedor se puede definir como una caja metálica grande de tamaño estándar en la que se embala la mercancía para su transporte a bordo de modos de transporte especialmente configurados para ello. Están diseñados para moverse con equipos de manipulación comunes que permiten transferencias intermodales de alta velocidad en unidades económicamente grandes entre buques, vagones, chasis de camiones y barcazas utilizando un mínimo de mano de obra. El tamaño de referencia de estos contenedores es la caja de 20 pies. También se utiliza la caja de 40 pies y los contenedores *Hi Cube* que son un pie más alto al tamaño estándar de 20 pies.

En último lugar, se debe definir los posibles movimientos de contenedores que pueden realizar los

vehículos. Éstos se componen de tractor y chasis, y se pueden fijar tres tipos: vehículo con chasis de contenedor de 20 pies, con chasis de contenedor de 40 pies y con chasis combinado, el cual permite el transporte de dos contenedores de 20 pies o un contenedor de 40 pies. Con esto, se puede determinar la cantidad de contenedores que se pueden transportar por vehículo y de qué tamaño.

2.1.2 Ventanas temporales.

Se pueden dar dos situaciones: si hay limitaciones de tiempo, el vehículo tiene que visitar a un cliente dentro de un cierto marco de tiempo, lo que supone una complicación del problema considerable respecto a no tener limitaciones de tiempo. El vehículo puede llegar antes de que se abra la ventana de tiempo, pero el cliente no será servido hasta que las ventanas de tiempo se abran. Y además no se permite llegar después de que la ventana de tiempo se haya cerrado. El problema de encontrar una configuración factible es NP-duro, en consecuencia, puede ser difícil construir.

En el caso de que no hay limitaciones de tiempo, supone que obtener una configuración factible es menos relevante: asignar arbitrariamente solicitudes de transporte a vehículos, ordenar arbitrariamente las solicitudes de transporte asignadas a un vehículo y procesar cada solicitud de transporte por separado; y

También se debe considerar las limitaciones temporales para los vehículos. Estos no suelen estar disponibles durante todo el día, están sujetos a horas operativas de servicio. Un vehículo puede tener una o varias ventanas temporales que indican el periodo o los períodos en los que está disponible.

2.1.3 Función objetivo.

La función objetivo de estos problemas puede ser muy variada según el enfoque del autor. A continuación, se mencionan los más comunes:

- Minimizar el número de vehículos. Minimizar el número de vehículos para atender todas las solicitudes es por lo general el objetivo principal de muchas investigaciones, ya que se puede considerar que la contratación de vehículos y conductores es uno de los aspectos más caros.
- Maximizar los beneficios. Esta función únicamente se emplea cuando se tiene la posibilidad de rechazar una petición de transporte cuando se contempla como desfavorable transportar la carga correspondiente.
- Minimizar el tiempo de viaje. Minimizar el tiempo de viaje total efectuado por el vehículo.
- Minimizar la longitud de la ruta. Minimizar la distancia de viaje total efectuada por el vehículo.
- Minimizar los inconvenientes del cliente. El inconveniente del cliente se mide en términos de desviación del tiempo de recogida (es decir, la diferencia entre el tiempo de recogida real y el deseado), el tiempo de entrega (la diferencia entre el tiempo de suministro deseado y el tiempo de suministro real) y el tiempo de viaje excesivo.

2.2 Formulación.

Siguiendo la formulación expuesta por K.H. Chung, C. S. Ko, J. Y. Shin, H. Hwang y K. H. Kim (2007) se introduce un modelo base. Posteriormente, se anotan algunas variaciones respecto del modelo de Chung et al. (2007) para la definición del problema de este trabajo.

2.2.1 Modelo Base.

Este modelo sería el resultado de aplicar la variante del problema de enrutamiento de vehículos con recogida y entrega (VRPPD) y la variante de considerar chasís combinado para responder a las

solicitudes de los clientes. Se establecen entonces los siguientes supuestos:

- Hay tres tipos de vehículos disponibles para completar las solicitudes de transporte: vehículos de 20 pies, de 40 pies y vehículos de chasis combinados.
- El tiempo operativo diario de cada vehículo está limitado con un marco temporal y es igual para todos los vehículos.

Para *n* solicitud de movimiento de contenedor se generan dos nodos: su punto de recogida (nodo i) y su punto de entrega (nodo *n*+*i*). Y se define la siguiente anotación:

$N_1 = \{1, 2, \ldots, n\}$: conjunto de nodos de recogida.

$N_2 = \{n + 1, n + 2, \ldots, 2n\}$ conjunto de nodos de entrega.

$N = N_1 + N_2 + \{0\}$

K_1: conjunto de contenedores 20 pies.

K_2: conjunto de contenedores con chasis combinado.

K_3: conjunto de contenedores 40 pies.

$K = K_1 + K_2 + K_3$

s_j: solicitud de recogida o entrega para el nodo j, 1 si se carga un contenedor de 20 pies en el nodo j, 2 si se carga un contenedor de 40 pies en el nodo j, -1 si se descarga un contenedor de 20 pies en el nodo j, y -2 si un contenedor de 40 pies se descarga en el nodo j.

T: tiempo operativo máximo de un vehículo por día.

t_{ij}: tiempo total de carga y descarga para el nodo *i* y el tiempo de viaje requerido para moverse desde el nodo *i* al nodo *j*.

c_{ij}^k: coste del transporte desde el nodo *i* al nodo j por el vehículo *k*.

x_{ij}^k: 1 si el vehículo *k* viaja del nodo *i* al nodo *j*, 0 en caso contrario.

t_i^k: tiempo de llegada del vehículo *k* del nodo *i*.

w_i^k: tamaño de contenedores que hay en el vehículo *k* cuando éste abandona el nodo *i*.

El problema se formula de la siguiente manera:

$$Minimizar \sum_{i \in N} \sum_{j \in N} \sum_{k \in K} c_{ij}^k \cdot x_{ij}^k \tag{10}$$

Sujeto a:

$$\sum_{j \in N} \sum_{k \in K} x_{ji}^k = 1 \qquad i \in N \tag{11}$$

$$\sum_{j \in N} x_{ij}^k = \sum_{j \in N} x_{ji}^k \qquad k \in K, \qquad i \in N \tag{12}$$

$$\sum_{j \in N} x_{0j}^k \leq 1 \qquad k \in K \tag{13}$$

$$\sum_{j \in N} x_{ij}^k = 1 \rightarrow t_i^k + t_{i,n+i} \leq t_{n+i}^k \qquad i \in N, \qquad k \in K \tag{14}$$

$$x_{ij}^k = 1 \rightarrow t_i^k + t_{ij} \leq t_j^k \qquad i \in N, j \in N, j \neq n+i, k \in K \tag{15}$$

$$t_0^k = 0 \qquad k \in K \tag{16}$$

$$t_i^k + t_{i0} \leq T \qquad i \in N_2, \qquad k \in K \tag{17}$$

$$x_{ij}^k = 1 \rightarrow w_i^k + s_j \leq w_j^k \qquad i \in N, \qquad j \in N, \qquad k \in K \tag{18}$$

$$w_0^k = 0 \qquad k \in K \tag{19}$$

$$w_i^k \begin{cases} \leq 1 & k \in K_1, \quad i \in N \\ \leq 2 & k \in K_2, \quad i \in N \\ = 0 \text{ ó } 2 & k \in K_3, \quad i \in N \end{cases} \tag{20}$$

$$x_{ij}^k \in \{0,1\} \qquad i \in N, \qquad j \in N, \qquad k \in K \tag{21}$$

$$t_i^k \geq 0, \; w_i^k \geq 0 \qquad i \in N, \qquad k \in K \tag{22}$$

La función objetivo (10) trata de minimizar el coste total de los tres tipos de vehículos por el servicio de transporte de las solicitudes de movimiento de contenedores. La restricción (11) significa que el nodo *i* debe ser visitado por un vehículo. La restricción (12) implica una regla de conservación de flujo. La restricción (13) implica la disponibilidad de vehículos para transportar contenedores. La restricción (14) representa la relación de precedencia que obliga al nodo *i* a ser visitado antes del nodo *n* + *i*. La restricción (15) cumple los requisitos de compatibilidad entre rutas y horarios. Las restricciones (16) y (17) describen la limitación del tiempo en funcionamiento del vehículo *k*. La restricción (18) expresa los requisitos de compatibilidad entre rutas y cargas/descargas del vehículo, mientras que las limitaciones (19) y (20) son las restricciones de capacidad. La restricción (21) fuerza a que la variable x_{ij}^k solo puedan tomar valores binarios. La restricción (22) define los valores que puede tomar las variables t_i^k y w_i^k.

2.3 Variación del modelo base.

Una de las problemáticas básicas es buscar asginar las solicitudes de entrega a solicitudes de recogida, ya que los movimientos vacíos no producen ningún beneficio. Con esto, el problema de estudio se basa en encontrar un conjunto de rutas de vehículos que satisfagan todas las solicitudes de movimiento de contenedores (recordar que para cada solicitud de movimiento de contenedor se generan dos nodos: su punto de recogida y su punto de entrega) y que minimicen el coste total. Este coste es el que conlleva el uso del vehículo, ya sea determinado en dinero, distancia, tiempo, rutas, etc. En nuestro caso, se define en distancia, por lo que se busca minimizar la distancia total realizada por los vehículos.

En nuestro caso, se sigue manteniendo la limitación temporal de operación de los vehículos, que se fija igual para todos los vehículos. Los vehículos realizan movimientos de contenedores desde un origen a un destino determinado dentro de un margen de tiempo operativo, pero con la variación de que debe incluirse el tiempo necesario de salida desde el depósito al origen de la primera solicitud asignada y su vuelta desde el destino de la última solicitud al depósito.

Respecto al inicio y fin de cada ruta, se incluye un único depósito desde el cual salen y se recogen todos los vehículos dentro de su margen temporal de operación, por lo que una solicitud solo se asignará al vehículo si este es capaz de ir, realizarla y volver al depósito.

En cuanto a los tipos de movimientos, en nuestro problema se establece un tipo de vehículo, pero con la misma posibilidad que el modelo base: el vehículo puede realizar solicitudes/tareas tanto de 20 pies como de 40 pies por igual, esto es, el vehículo puede transportar tanto un contenedor de 20 pies, un contenedor de 40 pies o dos contenedores de 20 pies.

2.4 Revisión Literaria.

Algunas de las investigaciones que se han llevado a cabo respecto al transporte intermodal de mercancías en contenedores se comparan en la Tabla 1. En la segunda columna se indica si los modelos de estudio usan carga homogénea (CH), es decir, solo sólo puede transportarse tamaños de contenedores homogéneos (40 pies ó 20 pies), o carga a medias (CM), admite llevar un contenedor de 20 pies, dos contenedores de 20 pies o un contenedor de 40 pies. La tercera columna indica si las solicitudes son incompletas, es decir, si se permite solicitudes con datos desconocidos de antemano. La cuarta (y quinta) columna indica si hay uno o varios depósitos de contenedores (terminales/nodos clientes). La última columna indica si se consideran ventanas temporales en el modelo.

	TIPO DE CARGA *(CH / CM)*	**¿SOLICITUDES INCOMPLETAS?** *(SÍ / NO)*	**DEPÓSITOS** *(1 / MÚLTIPLE)*	**TERMINALES** *(1 / MÚLTIPLE)*	**VENTANAS TEMPORALES** *(SÍ / NO)*
WANG Y REGAN 2002 **[10]** JULA ET AL. 2005 **[13]**	CH	No	1	Múltiple	No
SMILOWITZ 2006 **[16]** BRAEKERS ET AL. 2011 **[26]** ESCUDERO ET AL. 2013 **[34]**	CH	Sí	1	Múltiple	Sí
CHUNG ET AL. 2007 **[18]**	CH y CM	No	Múltiple	Múltiple	Sí
IMAI ET AL. 2007 **[17]**	CH	No	1	1	No
NAMBOOTHIRI Y ERERA 2008 **[20]** REINHARDT ET AL. 2012 **[27]** PAZOUR Y NEUBERT 2013 **[52]**	CH	No	1	Múltiple	Sí
CHEUNG ET AL. 2008 **[21]**	CM	No	Múltiple	Múltiple	Sí
CARIS Y JANSSENS 2009 **[22]**	CH	No	1	1	Sí
ZHANG ET AL. 2010 **[23]** NOSSACK AND PESCH 2013 **[32]**	CH	Sí	Múltiple	Múltiple	Sí
ZHANG ET AL. 2011 **[24]**	CH	Sí	1	1	Sí
VIDOVIC ET AL. 2011 **[25]**	CM	No	1	1	No
BRAEKERS ET AL. 2013 **[31]**	CH	Sí	1	Múltiple	Sí
WANG Y YUN 2013 **[33]**	CH	Sí	1	Múltiple	Sí
SCHÖNBERGER ET AL. 2013 **[30]**	CM	Sí	1	1	No
STERZIK Y KOPFER 2013 **[50]**	CH	No	Múltiple	Múltiple	Sí
LAI ET AL. 2013 **[51]**	CH y CM	No	1	Múltiple	Sí
ZHANG, YUN Y KOPFER 2015 **[53]**	CM	Sí	1	Múltiple	No
FUNKE Y KOPFER 2016 **[54]** VIDOVIC ET AL. 2017 **[55]**	CM	No	1	Múltiple	Sí
TRABAJO ACTUAL	CM	No	1	Múltiple	No

Tabla 1. Comparación de diferentes investigaciones.

Wang y Regan (2002) y Jula et al. (2005) modelaron el problema de recogida y entrega como un problema del vendedor ambulante múltiple con ventanas del tiempo. Y los resuelven usando un método basado en particiones de ventanas de tiempo y un método híbrido de programación dinámica y algoritmos genéticos, respectivamente.

Smilowitz (2006) modeló el problema de *drayage* como un problema de enrutamiento de recursos múltiples con tareas flexibles. El enfoque de solución desarrollado para solucionar el problema incluyó la generación de columnas incrustada en un marco de *Branch and Bound.*

Imai et al. (2007) abordaron un problema de enrutamiento de vehículos que surge cuando se recoge y se entrega la carga completa del contenedor desde / hacia un terminal intermodal. Los viajes de entrega y recogida pueden combinarse en una ruta para ahorrar costes. Se desarrolló una heurística de relajación basada en la Lagrange para resolver el problema.

Chung et al. (2007) construyó varios modelos matemáticos de problemas de transporte de contenedores con diferentes casos: caso de una sola mercancía y el caso de múltiple mercancía.

Cheung et al. (2008) estudiaron el problema de *drayage* transfronterizos. Desarrollaron un modelo de decisión de atributo para el problema e implementaron un algoritmo de etiquetado adaptativo para resolverlo.

Namboothiri y Erera (2008) estudiaron el problema de *drayage* dado un sistema de cita de acceso al puerto, lo que impone una limitación en el número de camiones que pueden acceder al puerto durante cada intervalo de tiempo.

Caris y Janssens (2009) extendieron el problema en Imai et al. (2007) al considerar las ventanas de tiempo en las ubicaciones de los clientes. Se propuso una heurística de inserción de dos fases para construir una solución inicial, que se mejoró con una heurística de búsqueda local.

Zhang et al. (2010) y (2011) extendieron también este problema a un entorno multi-depot con multi-terminal y un único depósito y un único terminal, respectivamente. Resolvieron el problema utilizando el algoritmo de búsqueda tabú reactivo y la heurística basada en particiones de ventanas, respectivamente.

Zhang et al. (2011) examinaron un problema de *drayage* con una terminal y un depósito de área local. Los contenedores vacíos se consideraron como recursos de transporte aparte de los camiones. Se desarrolló un algoritmo basado en la búsqueda tabú reactiva para resolver el problema.

Braekers et al. (2011) estudian el problema del enrutamiento de vehículos de carga completa en operaciones de drayage. De antemano, se desconoce el origen o el destino de las solicitudes de transporte de contenedores vacíos. Se presenta un algoritmo de dos fases que utiliza un recocido determinista para resolver el problema bi-objetivo, minimizando el número de vehículos utilizados y minimizando la distancia total recorrida.

Vidovic et al. (2011) estudian el problema de distribución de contenedores de diferentes tamaños. Formulan el problema emparejando nodos de clientes en los procesos de distribución y recolección de contenedores con IP de coincidencia múltiple y para los problemas de tamaños mayores proponemos un enfoque heurístico basado en utilidades coincidentes.

Escudero et al. (2013) proponen un enfoque de optimización dinámica para considerar la posición de los vehículos para así poder replanificar, utilizando una heurística de inserción para resolverla.

Braekers et al. (2013) estudian un problema de enrutamiento de camiones de carga completa para el transporte de contenedores cargados y vacíos en operaciones de *drayage*. Se proponen dos enfoques de solución: un enfoque secuencial y un enfoque integrado. Para ambos enfoques, se presenta un algoritmo de recocido determinar de dos fases y de una fase.

Wang y Yun (2013) investigan un problema de transporte de contenedores por camión y tren. Los

contenedores se clasifican en cuatro tipos según la dirección (contenedores entrantes y contenedores salientes) y estado del contenedor (contenedores llenos o contenedores vacíos). Desarrollan un modelo matemático a través de un modelo de gráfico y se propone una búsqueda de tabú híbrido.

Nossack y Pesch (2013) consideran los contenedores vacíos como recursos de transporte y son proporcionados por la compañía de camiones. El problema se formula como problema de recogida y entrega con carga llena y ventanas temporales (FTPDPTW), y se resuelve mediante un enfoque de solución heurística en dos etapas.

Schönberger, Buer y Kopfer (2013) consideran una generalización del problema de recogida y entrega con contenedores de 20 pies y 40 pies con localización de entrega y recogida a priori desconocida. El modelo se valida mediante un Solver de programación lineal Mixto-Entero.

Sterzik y Kopfer (2013) definen una formulación matemática integral que considera el enrutamiento y la programación de vehículos y el reposicionamiento de contenedores vacíos simultáneamente. El problema se resuelve con una eficiente heurística de Búsqueda Tabú.

Lai, Crainic, Di Francesco y Zuddas (2013) abordan un nuevo problema de enrutamiento, donde las cargas del contenedor deben enviarse desde un puerto para los importadores y de los exportadores al puerto en camiones que llevan uno o dos contenedores, sin separar camiones y contenedores durante el servicio al cliente. Determinan la solución inicial mediante una variante del algoritmo Clarke-and-Wright y lo mejora mediante una secuencia de las fases de búsqueda local.

Pazour y Neubert (2013) consideran el problema de drayage entre ciudades con las limitaciones operativas, incluida una gran cantidad de cargas por programa de conductores, tiempos de inicio de los controladores, ubicaciones de inicio y fin del conductor, patrones de tráfico por hora, ventanas de tiempo de carga y horas requeridas de servicio del conductor.

Zhang, Yun y Kopfer (2015) investigan un transporte de camiones de contenedores de varios tamaños (MSCTT), problema en el cual un camión puede transportar un contenedor de 40 pies o dos contenedores de 20 pies. Esta el problema considera órdenes de transporte fijas y flexibles. Tres procedimientos de búsqueda de árbol y un algoritmo de búsqueda tabu reactivo mejorado (IRTS) se diseñan para resolver el problema de MSCTT.

Funke y Kopfer (2016) presentan el problema de transporte de contenedores de varios tamaños (mICT), los camiones pueden transportar hasta dos contenedores de 20 pies o uno de 40 pies a la vez a lo largo de las rutas con varias camionetas y lugares de entrega. Una programación lineal de enteros mixtos se presenta utilizando dos funciones alternativas objetivo: minimización de la distancia total de viaje y minimización del tiempo total de operación de los camiones.

Vidovic, Popovic, Ratkovic y Radivojevic (2017) abordan el problema del enrutamiento de vehículos en operaciones de drayage, donde los vehículos pueden transportar contenedores de diferentes tamaños con ventanas de tiempo. Se modela como un múltiplo problema de coincidencia y formulado como un modelo de programa lineal entero mixto (MILP). Para su resolución se emplea una heurística de búsqueda de vecindario variable.

3 METODOLOGÍA

En la literatura se puede encontrar una gran variedad de enfoques posibles de solución para el tipo de problema expuesto en la sección anterior. Con esta idea, se presenta una revisión de las posibles metodologías de resolución aplicables relativas al problema de *drayage*. Partiendo de esta revisión, se eligen dos de ellas: Recocido Simulado y Búsqueda Tabú.

La búsqueda tabú es capaz de presentar buenas soluciones para estos problemas, sin embargo, su aplicación es menos popular. En cuanto al recocido simulado, método también bastante conocido y que obtiene resultados considerablemente buenos, se diferencia con la búsqueda tabú en que éste al no verificar el vecindario completo, le permite moverse más rápido entre vecindarios y con ello, realizar mayor número de iteraciones en un tiempo dado. Estos dos enfoques de solución se basan en resolver un problema por búsqueda local, basados en una solución inicial y una estructura de vecindario.

3.1 Revisión de metodologías de resolución.

En la literatura abundan las metodologías de resolución para este tipo de problemas. La Figura 6 clasifica estas metodologías según Ganesh et al. (2007a):

1. Técnicas Exactas
2. Heurística.
3. Metaheurística.
4. Enfoques interactivos.
5. Enfoques híbridos.

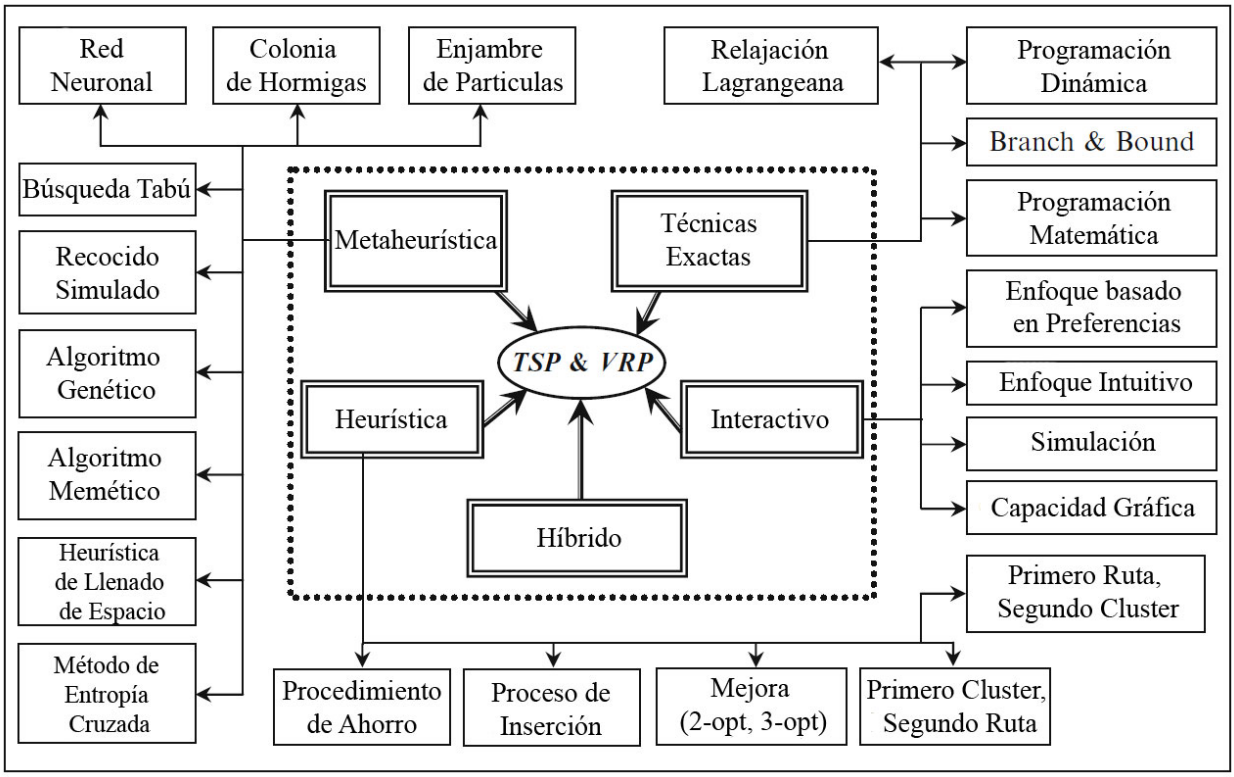


Figura 6. Metodologías de resolución. Fuente: S.P. Anbuudayasankar y K. Ganesh Sanjay Mohapatra.

3.1.1 Técnicas Exactas.

Los métodos clásicos de resolución exacta (es decir, enumerativa, *Branch and Bound*, programación dinámica, programación lineal, integrada, etc.) permiten encontrar soluciones exactas óptimas, pero son demasiado complejos y lentos de resolver cuando se aplican a problemas del mundo real (por ejemplo: problemas grandes o fuertemente limitados, problemas multimodales o variables en el tiempo, etc.). En los mejores algoritmos para VRP es más complicado ver que puedan abordar casos que impliquen más de 100 vértices, mientras que problemas de TSP con cientos e incluso miles de vértices se resuelven más fácilmente a la optimalidad en la actualidad.

Los algoritmos *Branch and Bound* (Radharamanan y Choi, 1986; Laporte et al., 1987 y Ralphs, 2003), algoritmos *Branch and Cut* (Bard et al., 1998, Lysgaard et al., 2004 y Fukasawa et al., 2006), Programación dinámica (Magnanti, 1981 y Kolen et al., 1987) y el procedimiento de relajación Lagrangeana (Kallehauge et al., 2001, Magnanti, 1981; Stewart y Golden, 1984) son algunos de los enfoques exactos que se han utilizado como metodlogías de resolución.

Hasta finales de la década de 1980, los métodos exactos más eficaces fueron los algoritmos *Branch and Bound* basados en las relajaciones de combinatorias elementales. Más recientemente, se han propuesto enfoques basados en relajaciones lagrangianas o en el procedimiento de limitación aditiva, los cuales aumentan el tamaño del problema que puede resolverse a óptimo.

El algoritmo *Branch and Cut* se considera actualmente el mejor enfoque exacto disponible para la resolución del VRP. La investigación en esta área ha sido motivada por la aparición y el éxito de la combinatoria poliédrica como metodología de resolución de problemas combinatorios complejos, como el caso del TSP.

3.1.2 Heurística.

En los problemas combinatorios, los procedimientos heurísticos son ampliamente utilizados. Las heurísticas limitan su exploración del espacio de búsqueda, pero apuntan a una buena solución en un tiempo razonablemente corto con la posibilidad de renunciar a encontrar una solución óptima global del problema. En problemas de gran tamaño es difícil que un algoritmo heurístico encuentre una solución óptima global.

Se definen tres categorías básicas de heurística según Laporte (1992):

1. Heurística constructiva: Estos algoritmos comienzan desde una solución vacía e iterativamente construyen rutas insertando uno o más nodos en cada iteración, hasta que todos los nodos son enrutados. Los algoritmos de construcción de rutas se especifican por tres componentes principales: un criterio de inicialización, un criterio de selección (qué nodos se eligen para su inserción en la actual iteración) y un criterio de inserción (dónde ubicar a los nodos elegidos en las rutas actuales).

   a) El vecino más cercano: comienza con un nodo aleatorio, encontrando el más cercano al último nodo seleccionado en cada iteración y completa la ruta con todos los nodos. Requiere alta capacidad de tiempo computacional (Rosenkrantz et al., 1977).

   b) Procedimiento de ahorro: se basa en el concepto de ahorro, en la estimación de una reducción de costos obtenida al atender a dos clientes secuencialmente en la misma ruta, en lugar de en dos separadas. Puede haber rutas subóptimas (Clarke y Wright, 1964, Dror y Trudeau, 1986, Altinkemer y Gavish, 1991 y Reimann et al., 2004). Doyuran y Çatay (2011) proponen una mejora robusta a la formulación de Clarke-Wright.

2. Heurística de dos fases: se basan en la descomposición del proceso de VRP en dos subproblemas separados: (1) clustering: determinar una partición de nodos en subconjuntos, cada uno correspondiente a una ruta, y (2) enrutamiento: determina la secuencia de nodos en

cada ruta.

Las heurísticas bifásicas no emplean un enfoque unificado para combinar las fases de clustering y enrutamiento; cada uno utiliza un bucle recursivo para la implementación del método bifásico y obtener los mejores resultados.

a) Primero Cluster, Segundo Ruta: agrupar los nodos y determinar las rutas factibles para cada grupo es el principio de este enfoque. Es difícil utilizar este procedimiento cuando los vehículos tienen diferentes capacidades (Renaud y Boctor, 2002). Algunos de los métodos son algoritmos de barrido (*Sweep*) (Gillett y Miller, 1974 y el algoritmo de Fisher y Jaikumar, 1981). El algoritmo comienza con un nodo/cliente arbitrario y luego asigna secuencialmente a los clientes restantes al vehículo actual considerando los mismos en orden creciente del ángulo polar con respecto al depósito y al cliente inicial. Tan pronto como el cliente actual no puede ser asignado de manera factible al vehículo actual, una nueva ruta se inicializa con él. Una vez que todos los clientes son asignados a vehículos, cada ruta se define por separado resolviendo un TSP.

b) Primero Ruta, Segundo Cluster: comienza con un enrutamiento construido de todos los nodos y va subdividiendo agrupaciones menores y factibles (Beasley, 1983, Mole et al., 1983 y Hachicha et al., 2000).

3. Heurística de mejora de búsqueda local: estos algoritmos se utilizan a menudo para mejorar las soluciones iniciales generadas por otras heurísticas. A partir de una solución dada, un método de búsqueda local aplica modificaciones simples, tales como intercambios de arco o movimientos de clientes, para obtener soluciones vecinas de posiblemente mejores costos. Si se encuentra una solución de mejora, entonces se convierte en la solución actual y el proceso itera; de lo contrario se ha identificado un mínimo local. Una característica importante del desarrollo del vecindario es la forma en que se realizan las inserciones: se podría utilizar inserción aleatoria o inserción en la mejor posición en la ruta objetivo; alternativamente, se podría utilizar esquemas de inserción que implican una reoptimización parcial de la ruta objetivo (Gendreau et al. 1997).

   a) Procedimiento de inserción: Este procedimiento construye una solución determinando la inserción del nodo más barato de la ruta. Puede haber subrutas. Algunos ejemplos de tipos de inserción son la inserción más cercana, inserción más barata, inserción más lejana, inserción rápida y el algoritmo de inserción de "convex-hull" (Chung y Norback, 1991, Gendreau et al., 1992, y Foisy y Potvin, 1993).

   b) Procedimiento de mejora: Dada una ruta, el algoritmo examina todas las rutas que le son vecinas y trata de encontrar una ruta más corta. Comenzando con una pequeña ruta inicial, elegida arbitrariamente o por algún otro método, si no hay una ruta vecina que sea más corta que la original, el proceso se detiene. Esto modifica las rutas paso a paso y mantiene la viabilidad de la solución. Requiere un tiempo computacional largo (Psaraftis 1983; Sule et al. 1991; Mak y Morton 1993; Bianchi et al. 2005). Algunos ejemplos son los métodos 2-opt y 3-opt (Lin 1965, Alfa et al., 1991), el algoritmo Lin-Kernighan (Lin y Kernighan 1973, Papadimitriou 1992, Helsgaun 2000) y Or-opt (Or 1976; Taillard et al. 1997).

3.1.3 Metaheurística.

Las metaheurísticas son estrategias de alto nivel para explorar espacios de búsqueda usando diferentes métodos, es decir, se enfocan en la exploración profunda de las regiones más prometedoras del espacio de soluciones mediante métodos que combinan típicamente reglas sofisticadas de búsqueda de vecindario, estructuras de memoria y recombinación de soluciones.

La calidad de las soluciones producidas generalmente es mucho mayor que las obtenidas por heurística

clásica. Sin embargo, el aumento del tiempo de cálculo es un gran problema. Aun así, los métodos de resolución más prometedores y eficaces son metaheurísticas (Gendreau et al., 2002).

Siguiendo la clasificación de Blum y Roli (2013), las metaheurísticas se clasifican sin memoria y basados en la memoria.

3.1.3.1 Metaheurística sin memoria.

Las dos metaheurísticas más importantes sin memoria son el Recocido Simulado (Kirkpatrick et al., 1983) y Umbral de Aceptación (TA) (Dueck y Scheuer 1990).

- Recocido Simulado (SA):

El recocido se conoce como el templado de ciertas aleaciones de metal, vidrio o cristal. Se calientan por encima de su punto de fusión, manteniendo su temperatura, y luego enfriándola muy lentamente hasta que se solidifica en una perfecta estructura cristalina. Este proceso produce materiales de alta calidad.

La simulación del proceso de recocido puede describirse como un proceso de generación de soluciones de un problema de optimización combinatoria, en donde se vayan obteniendo, conforme el proceso avanza, mejores soluciones a las anteriores. Con este propósito, se puede observar una analogía entre el sistema físico y un problema de optimización combinatoria en donde: los estados de los materiales físicos corresponden a soluciones del problema, la energía de un estado al coste de una solución y la temperatura a un parámetro de control.

SA se compone básicamente de dos procesos estocásticos: un proceso para la generación de soluciones y el otro para la aceptación de soluciones. La generación de temperatura es responsable de la correlación entre las soluciones generadas y la solución original.

En definitiva, SA es un algoritmo de descenso modificado por movimientos de ascenso al azar para escapar de mínimos locales que no son mínimos globales.

- Umbral de Aceptación (TA):

TA es una modificación de la SA. Específicamente, deja fuera el elemento estocástico aceptando soluciones peores mediante la introducción de un umbral determinista. Durante la optimización del proceso, el nivel umbral se reduce gradualmente como la temperatura en SA (Tarantilis et al., 2005).

Tarantilis et al. (2002a) presentó una variante de un algoritmo TA, llamado *Backtracking Adaptable Threshold Accepting* (BATA), en el que la estructura de vecindad es definida de manera similar a la de Osman (1993). Tarantilis et al. (2002b) desarrollada otra variante de TA, denominada Límite de Umbral Basado en la Lista (LBTA). LBTA expande el algoritmo estándar TA mediante la introducción de una lista de valores umbral. Tanto BATA como LBTA se han utilizado en numerosas operaciones de distribución de mercancías (Tarantilis y Kiranoudis 2001b, 2002a, b, c, e).

3.1.3.2 Metaheurística con memoria.

Las metaheurísticas basadas en la memoria exploran el espacio de soluciones a través de una o varias listas de soluciones guardadas dentro de una memoria limitada.

El termino memoria se utilizó explícitamente en los algoritmos de Búsqueda Tabú (TS) (Glover 1989) y los Algoritmos Adaptativos Basados en Memoria (AMBA) (Rochat y Taillard 1995). Sin embargo, otras metaheurísticas como el Algoritmo Genético (Holanda 1975), ACO (Dorigo 1992), Optimización de Enjambre de Partículas (PSO) (Kennedy y Eberhart 1995) y Algoritmo Memético (Moscato 1989) utilizan mecánica y estructuras parecida a recuerdos.

- Búsqueda tabú (TS):

TS utiliza exploración y memoria flexible para guiar la búsqueda en el proceso de solución. Mediante

la exploración, determina una búsqueda basada en las propiedades de la solución actual y el historial de búsqueda haciendo repetidamente movimientos de una solución a otros que se encuentran en su vecindario. TS selecciona el mejor movimiento basado en la función de evaluación. Esta función elige una solución que produzca la mayor mejoría (o la menor no mejoría) en la función objetivo en cada iteración.

Mediante la memoria, utiliza estructuras de memoria a corto y largo plazo para mantener un historial selectivo de búsqueda. Estas memorias están diseñadas para hacer un seguimiento de la solución, como alguno de sus atributos que sea de interés, visitados en el proceso de búsqueda. La memoria a corto plazo (lista tabú) registra los movimientos que se produzcan. La memoria a largo plazo evita que se vuelva a soluciones ya visitadas, así como, repetición y bucles.

- Algoritmos Adaptativos Basados en Memoria (AMBA):

Un algoritmo adaptativo es un algoritmo que cambia su comportamiento en el momento en que se ejecuta, basado en información disponible y en un mecanismo de recompensa (o criterio) definido a priori. Tal información podría ser la historia de los datos recibidos recientemente, información sobre los recursos computacionales disponibles u otra información adquirida en tiempo de ejecución relacionada con el entorno en el que opera.

El primer AMBA fue presentado por Rochat y Taillard (1995). La lógica adaptativa de la memoria constituye una de las herramientas más potentes para la intensificación del proceso de búsqueda (Tarantilis et al., 2005).

- Algoritmo Genético (GA):

Los algoritmos genéticos son algoritmos de búsqueda probabilística inspirados en la teoría de Darwin de la evolución. Tienen la capacidad de producir soluciones optimizadas incluso cuando las dimensiones del problema aumentan. En consecuencia, han sido exitosamente aplicado a una amplia variedad de problemas.

Los algoritmos genéticos operan sobre una población de soluciones representadas por alguna codificación. Cada miembro de la población consiste en una serie de genes, cada uno de los cuales es una unidad de información. Las nuevas soluciones se obtienen combinando genes de diferentes miembros de la población (cruce) para producir descendientes o alterando miembros existentes de la población (mutación). Una simulación de la «selección natural» se realiza evaluando primero la calidad de cada solución y luego seleccionando a los más aptos para sobrevivir a la siguiente generación.

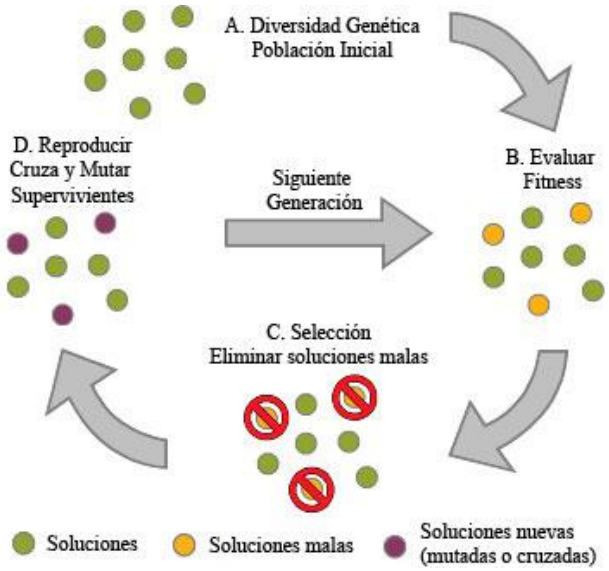


Figura 7. Ejemplo gráfico del procedimiento del algoritmo genético.

– Optimización de colonias de hormigas (ACO):

ACO es una técnica probabilística para resolver problemas computacionales que se basada en el comportamiento de las hormigas para buscar un camino entre su nido y una fuente de alimento. Algunas especies de hormigas vagan al azar y hasta que encuentran comida no vuelven a su nido dejando un rastro de feromona. Si otras hormigas encuentran ese rastro, es más probable que sigan ese camino en vez de ir al azar.

Para aplicar ACO, el problema de optimización se transforma en el problema de encontrar la mejor trayectoria en un gráfico ponderado. Las hormigas artificiales construyen de forma incremental soluciones moviéndose sobre el gráfico. El proceso de construcción de la solución es estocástico y está sesgado por feromona, es decir, un conjunto de parámetros asociados a componentes gráficos (nodos o bordes) cuyos valores son modificados por las hormigas artificiales mientras se ejecuta.

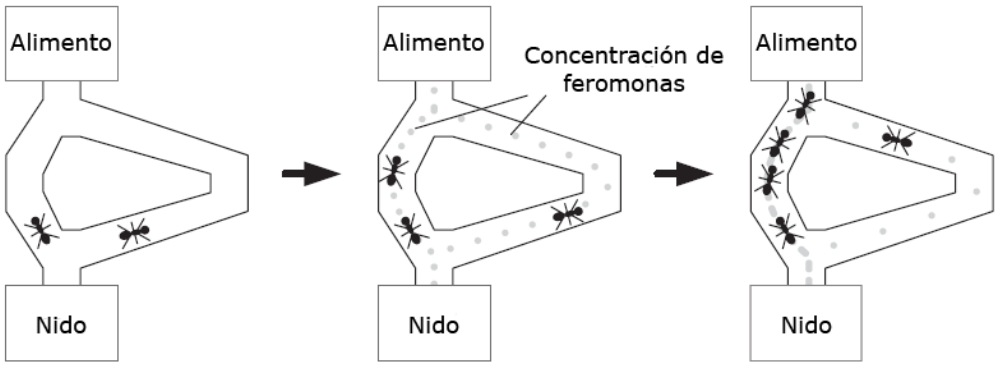


Figura 8. Ejemplo gráfico de una colonia de hormigas. Fuente: S. Ciruela (2008).

– Optimización de Enjambre de Partículas (PSO):

La optimización de enjambres de partículas es una técnica de optimización estocástica en el comportamiento social del acopio de aves o la escolarización de peces. El sistema se inicializa con una población de soluciones aleatorias y busca una óptima mediante la actualización de generaciones.

PSO optimiza un problema al tener una población de soluciones candidatas (partículas), moviéndolas en el espacio de búsqueda. Los movimientos de las partículas son guiados por las mejores posiciones encontradas en el espacio de búsqueda, la cual se actualiza a medida que las partículas encuentran mejores posiciones.

Una partícula tiene una posición actual, velocidad y un registro de posiciones pasadas. La optimización consiste en la ejecución del algoritmo para un número dado de iteraciones. En cada iteración se actualiza la velocidad de cada partícula y la posición hasta la mejor solución.

La Figura 9 muestra un ejemplo de PSO con el objetivo de búsqueda de encontrar la estrella. En cada iteración se encuentra la partícula más cerca de la estrella para así mover el resto de partículas en esa dirección.

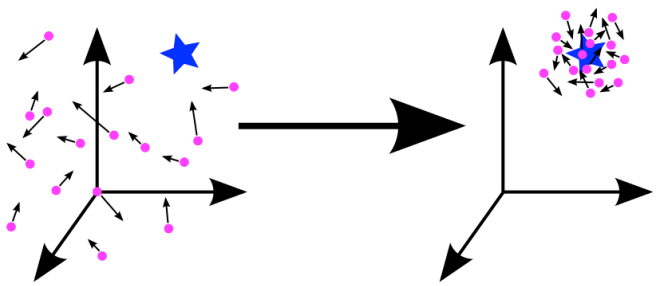


Figura 9. Ejemplo gráfico del procedimiento de PSO. Fuente: J. Becker.

– Algoritmo de Memético

El concepto de algoritmo memético se introdujo por primera vez por Moscato y Norman (1992) para describir algoritmos evolutivos en los que predomina la búsqueda local. El concepto de "meme" se establece como una unidad de información que se reproduce a medida que las personas intercambian ideas. Un meme se diferencia de un gen en cómo se pasa entre los individuos: cada individuo adapta el meme como ve mejor, los genes pasan sin alteraciones.

La principal ventaja de los algoritmos meméticos es que el espacio de posibles soluciones se reduce al subespacio de óptimo local. En otras palabras, considere una función variable f (x) descrita por la curva de la Figura 10. La solución inicial producida por la mutación de una solución existente está fuera del espacio de optimo local, pero usando la información disponible sobre la altura de la curva, podemos seguir un camino hacia arriba hasta que un mejor optimo local.

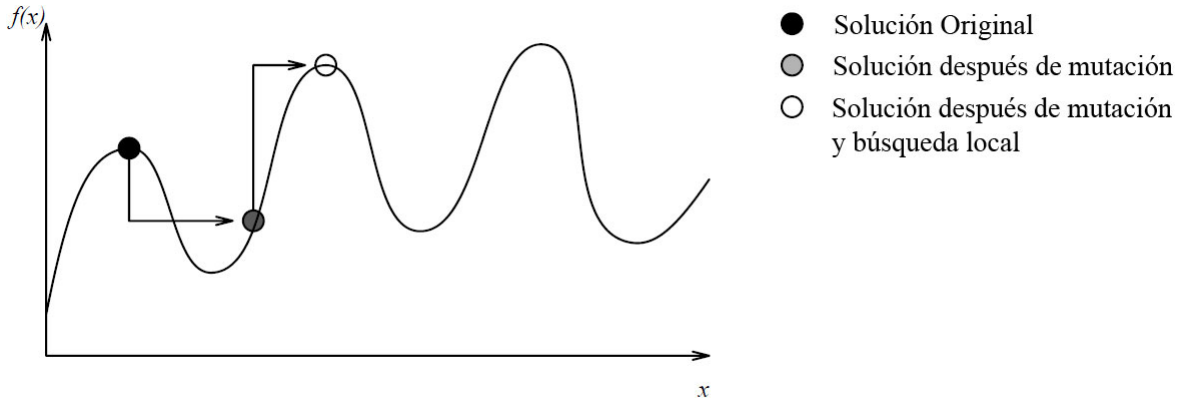


Figura 10. Ejemplo de una operación memética. Fuente: E. K. Burke, J. P. Newall and R. F. Weare.

3.1.4 Enfoques interactivos.

Son enfoques simples que pueden adaptarse a cualquier aplicación. Pueden basarse en la intuición, la simulación, la preferencia o algún tipo de gráficos para ayudar a la toma de decisiones y su objetivo es integrar la visión y la experiencia del usuario, así como el poder y la precisión de la heurística en un entorno interactivo.

Un enfoque interactivo nos permite encontrar una solución por medio de un procedimiento de ensayo y error: la máquina es superior en lo que concierne cálculos y tareas exhaustivas de búsqueda, y el usuario humano es superior al juzgar situaciones difusas. A través de la interacción, el usuario es capaz de controlar el proceso de solución mediante la selección de parámetros iniciales, seleccionando algoritmos y ajustando soluciones. De esta manera, el usuario guía las heurísticas hacia partes prometedoras de la solución espacio.

3.1.5 Enfoques híbridos.

Algunas investigaciones también han abordado el acercamiento híbrido, es decir, conseguir una combinación de dos o más enfoques/metodologías con el objetivo de que estas combinaciones consigan un mejor desempeño de lo que se obtiene por separado. Muchos ejemplos de enfoques híbridos mejoran el potencial de obtener buenas soluciones a bajo tiempo computacional (Laporte et al., 2000).

3.2 Recocido Simulado (SA).

Como se hablaba en la sección anterior, el recocido se conoce como el templado de ciertas aleaciones de metal, vidrio o cristal. Se calientan por encima de su punto de fusión, manteniendo su temperatura, y luego enfriándola muy lentamente hasta que se solidifica en una perfecta estructura cristalina. Este proceso produce materiales de alta calidad.

La simulación del proceso de recocido puede describirse como un proceso de generación de soluciones de un problema de optimización combinatoria, en donde se vayan obteniendo, conforme el proceso avanza, mejores soluciones a las anteriores. Con este propósito, se puede observar una analogía entre el sistema físico y un problema de optimización combinatoria en donde: los estados físicos de los materiales corresponden a soluciones del problema, la energía de un estado al coste de una solución y la temperatura a un parámetro de control.

Siguiendo la termodinámica estadística, P_α, corresponde a la probabilidad de que un sistema físico esté en el estado α con energía E_α a la temperatura T, la cual satisface la distribución de Boltzmann:

$$P_\alpha = \frac{1}{Z} e^{\frac{-E_\alpha}{k_B} T}$$

donde k_B es la constante de Boltzmann, T es la temperatura absoluta y Z es la función de partición que viene definida por:

$$Z = \sum_{\beta} e^{\frac{-E_\beta}{k_B} T}$$

donde la suma de todos los estados β será la energía E_β a la temperatura T.

A la altura T, la distribución de Boltzmann muestra preferencia uniforme para todos los estados, independientemente de la energía. Cuando T se aproxima a cero, sólo los estados con energía mínima tienen probabilidad de ocurrencia no nula.

En SA, se omite la constante k_B. A T alto, el sistema ignora pequeños cambios en la energía y se aproxima al equilibrio térmico rápidamente, es decir, realiza una búsqueda aproximada del espacio de los estados globales y encuentra un buen mínimo. A medida que se baja T, el sistema responde a pequeños cambios en la energía, y realiza una búsqueda en el vecindario del mínimo ya determinado y encuentra un mínimo mejor. En $T = 0$, cualquier cambio en los estados del sistema no conduce a un aumento en la energía, y, por lo tanto, el sistema debe alcanzar el equilibrio si $T = 0$.

T es un parámetro de control llamado temperatura computacional, que controla la magnitud de las perturbaciones de la función energética $E(x)$. La probabilidad de un cambio de estado se determina por la distribución de Boltzmann de la diferencia de energía en los dos estados:

$$P = e^{-\frac{\Delta E}{T}}$$

La probabilidad de movimientos ascendentes en la función de energía ($\Delta E > 0$) es grande a T alto y baja a T bajo. SA permite movimientos ascendentes de forma controlada: intenta mejorar la búsqueda local codiciosa ocasionalmente tomando un riesgo y aceptando una solución peor.

El recocido simulado estándar se conoce también como el recocido de Boltzmann. Se puede definir en la Figura 11 el siguiente esquema para el proceso de SA según Kirkpatrick, Gelatt y Vecchi:

Inicio

Generar x(0) inicial aleatorio.

Fijar un *T* suficientemente alto.

Búsqueda

Repetir:

- Aplicar perturbaciones aleatorias al estado x = x + *Δx;*
- Evaluar *ΔE(x) = E(x + Δx) - E(x):*

  Si *ΔE(x) < 0*, mantener el nuevo estado;

  De otra manera, aceptar el nuevo estado con probabilidad $P = e^{-\frac{\Delta E}{T}}$;

Hasta que el número de transiciones aceptadas sobrepase un nivel límite.

Fijar T = T – *ΔT;*

Hasta que T sea suficientemente pequeño.

Figura 11. Esquema general del SA.

Cómo se formule el proceso de enfriamiento de *T* es uno de los parámetros más importante para la eficiencia de SA. Si *T* se reduce demasiado rápidamente, puede ocurrir una convergencia prematura a un mínimo local. En contraste, si es demasiado lento, el algoritmo es muy lento para converger.

Geman y Geman (1984), basándose en un análisis de cadena de Markov sobre SA, han demostrado que *T* debe ser disminuido de acuerdo con:

$$T(t) \geq \frac{T_0}{ln(1+t)} \qquad t = 1, 2, \ldots$$

para asegurar una convergencia al mínimo global con probabilidad uno, donde T_0 es una temperatura inicial suficientemente grande.

Para garantizar que se converge al mínimo global con probabilidad uno, *T(t)* necesita disminuir logarítmicamente con el tiempo. Esto es prácticamente demasiado lento, por ello en la práctica se suele aplicar un proceso heurístico de enfriamiento rápido para mejorar la velocidad de convergencia.

3.2.1 Variantes del Recocido Simulado.

El recocido simulado estándar es un método de búsqueda estocástico y la convergencia al óptimo global es demasiado lenta, de manera que muchos métodos se han propuesto para poder acelerar la búsqueda en SA. Algunos de estos métodos son:

- Recocido de Cauchy: cambia la distribución de Boltzmann por la distribución de Cauchy, la cual permite un acceso más fácil a probar mínimos locales en la búsqueda del mínimo global.

$$T_k = \frac{T_0}{(1+k)}$$

Para controlar el número de iteraciones (M), se puede realizar una modificación al esquema de Cauchy de tal manera:

$$T_{k+1} = \frac{T_k}{(1 + \beta \cdot T_k)}$$

Siendo $\beta = \frac{T_o - T_f}{M \cdot T_o \cdot T_f}$.

En este estudio se ha escogido esta secuencia de enfriamiento rápido para el algoritmo de recocido simulado, pero con la modificaicón de que β es igual a α. En otras palabras, dado un coeficiente de enfriamiento α, se calcula la siguiente temperatura a partir del esquema de Cauchy modificado.

- *Reannealing* simulado (*Simulated Reannealing*): siempre que realizamos una búsqueda multidimensional en un problema físico no lineal del mundo real, inevitablemente se debe lidiar con distintas sensibilidades que cambian el espacio de búsqueda. En cualquier tiempo de recocido dado, parece razonable intentar "estirar" el intervalo sobre el cual se están buscando los parámetros. Se ha demostrado que es ventajoso lograr esto con un resellado periódico del tiempo de recocido (*Reannealing*). Por lo que, el *Reannealing* permite adaptarse a las sensibilidades cambiantes del espacio de parámetros multidimensionales.
- SA generalizada: generaliza el recocido de Cauchy y el recocido de Boltzmann en un único marco que resulta ser más rápido que ambos por separado.
- SA con valor global conocido: es el mismo que el de la Figura 11 excepto que en cada iteración se genera un punto aleatorio uniforme sobre una esfera cuyo radio depende de la diferencia entre el valor actual de función *E(x(t))* y el valor óptimo E^*. *T* también depende de esta diferencia.

3.3 Búsqueda Tabú (TS).

TS utiliza exploración y memoria flexible para guiar la búsqueda en el proceso de solución. Mediante la exploración, determina una búsqueda basada en las propiedades de la solución actual y el historial de búsqueda haciendo repetidamente movimientos de una solución a otros que se encuentran en su vecindario. TS selecciona el mejor movimiento basado en la función de evaluación. Esta función elige una solución que produzca la mayor mejoría (o la menor no mejoría) en la función objetivo en cada iteración.

Mediante la memoria, utiliza estructuras de memoria a corto y largo plazo para mantener un historial selectivo de búsqueda. Estas memorias están diseñadas para hacer un seguimiento de la solución, como alguno de sus atributos que sea de interés, visitados en el proceso de búsqueda. La memoria a corto plazo (lista tabú) registra movimientos que se produzcan recientemente. La memoria a largo plazo evita que se vuelva a soluciones ya visitadas, así como, repetición y bucles.

En definitiva, se evitan movimientos de vuelta a soluciones previamente visitadas mediante el uso de memorias que registran la historia reciente de la búsqueda de soluciones.

- Búsqueda del espacio y estructura del vecindario:

Los dos primeros elementos básicos de cualquier TS son la definición del espacio de búsqueda y la estructura de vecindad.

El espacio de búsqueda de LS o TS es simplemente el espacio de todas las posibles soluciones que se pueden considerar durante la búsqueda. Un espacio de búsqueda mejorado podría ser un conjunto de soluciones factible, eliminado las no factibles. Pero es importante apuntar que no siempre es una buena

idea restringir el espacio de búsqueda a soluciones factibles; en muchos casos, es deseable que la búsqueda se mueva a soluciones no factibles y a veces necesarias para conseguir un mejor óptimo, no visto de la otra manera.

Por otra parte, se encuentra la estructura del vecindario. A cada iteración de LS o TS, se pueden aplicar diferentes transformaciones locales a la solución actual, denotada como *S*. Estas transformaciones definen un conjunto de soluciones vecinas en el espacio de búsqueda, denotado *N(S)* (vecindario de *S*). En definitiva:

N(S) = {subconjunto del espacio de búsqueda definido por soluciones obtenidas aplicando una sola transformación local a *S*}.

En general, se puede encontrar más posibles y atractivas estructuras de vecinos que definiciones de espacio de búsqueda. Esto viene del hecho de que puede haber varias estructuras de vecindad plausibles para una definición dada del espacio de búsqueda.

Con esto, se puede decir que elegir un espacio de búsqueda y una estructura del vecindario es uno de los pasos más críticos en el proceso de la búsqueda tabú.

- Tabús:

Los tabús son los elementos diferenciadores de TS respecto a una simple búsqueda local. Como se menciona anteriormente, los tabús se usan para prevenir bucles de movimientos que no mejoran y que se alejan del óptimo local.

El objetivo por tanto es que cuando ocurre este bucle, se necesita algo para evitar que la búsqueda retome movimientos peores hacia atrás. Esto se consigue declarando tabú (prohibir) movimientos que invierten el efecto de movimientos recientes. Estos movimientos se almacenan en una memoria a corto plazo de la búsqueda (lista tabú) y por lo general sólo se registra una cantidad limitada de información de soluciones pasadas.

- Criterio de aspiración:

Puede ocurrir que una búsqueda tabú prohíba movimientos atractivos, incluso cuando no hay peligro de bucle, o pueden conducir a un estancamiento en la búsqueda. Por ello, es necesario utilizar métodos algorítmicos que permitan cancelar tabús. Estos son llamados criterios de aspiración. El criterio de aspiración más simple y más utilizado consiste en permitir un movimiento, incluso si es tabú, si se obtiene una solución mejor que la hasta ahora conocida.

- Criterio de terminación:

En teoría, la búsqueda podría durar para siempre, a menos que se conozca el valor óptimo del problema de antemano. De modo que, en la práctica, la búsqueda tiene que ser detenida en algún momento. Los criterios de detención más comunes son: después de un número fijo de iteraciones (o una cantidad fija de tiempo de CPU); después de un cierto número de iteraciones sin una mejora en el valor objetivo de la función (criterio más empleado); o cuando el objetivo alcanza un valor umbral preestablecido.

- Intensificación y Diversificación:

El objetivo de la intensificación de la búsqueda es explorar más a fondo las partes del espacio de búsqueda que parecen prometedoras con el fin de asegurar que se encuentran las mejores soluciones.

En general, la intensificación se basa en una memoria a medio plazo, como una memoria reciente, que registra el número de iteraciones consecutivas que varios componentes de la solución se han encontrado en la solución actual sin interrupción.

La intensificación se utiliza en muchas implementaciones TS, pero no siempre es necesario. Esto se debe a que hay muchas situaciones en las que la búsqueda realizada por el proceso de búsqueda normal es suficientemente completa y por tanto no se necesita una intensificación.

En cuanto a la diversificación, este se define como un mecanismo algorítmico que intenta aliviar el TS forzando la búsqueda en áreas previamente inexploradas del espacio de búsqueda. Se basa en una memoria a largo plazo, como una memoria de frecuencia, en la que se registra el número total de iteraciones (desde el inicio de la búsqueda) que varios componentes de la solución se han encontrado presentes en la solución actual o han estado involucrados en los movimientos seleccionados.

Hay dos técnicas principales de diversificación: La primera, denominada diversificación de reinicio, fuerza algunos componentes raramente usados en la solución actual (o la solución mejor conocida hasta ahora) y reiniciando la búsqueda desde este punto. La segunda técnica de diversificación, la diversificación continua, integra consideraciones de diversificación directamente en el proceso regular de búsqueda. Esto se logra mediante el sesgo de la evaluación de posibles movimientos añadiendo al objetivo un término relacionado con las frecuencias de los componentes.

En último lugar, se concreta un esquema general del proceso TS con los elementos expuestos anteriormente en la Figura 12. Además, se define la siguiente notación:

S, la solución actual.
S^*, la solución mejor conocida.
f^*, valor de S^*.
$N(S)$, vecindario de S.
$\tilde{N}(S)$, el subconjunto "admisible" de $N(S)$ (es decir, no tabú o permitido por aspiración).

Inicio

Elegir/construir una solución inicial S_0;

Fijar $S = S_0$; $f^* = f(S_0)$; $S^* = S_0$; $T = \emptyset$;

Búsqueda

Mientras el criterio de terminación no sea satisfecho, hacer:

- Seleccionar S in $\underset{S' \in \tilde{N}(S)}{argmin}[f(S')]$;
- Si $f(S) < f^*$, entonces fijar $f^* = f(S)$ y $S^* = S$;
- Grabar como tabú el movimiento actual en T (eliminar el más viejo si es necesario);

Terminar.

Figura 12. Esquema general de TS.

3.4 Adaptación al problema de estudio.

3.4.1 Construcción de la solución inicial.

Para la construcción de una solución inicial se aplican permutaciones aleatorias. Este procedimiento consiste en variar aleatoriamente (al azar) el orden de procesamiento de las tareas/solicitudes conocidas a priori de los clientes. En consecuencia, cada vez que se ejecute el algoritmo, partirá de una solución inicial diferente.

Paso 1: Generar conjunto de tareas que realizar. (*Datos conocidos a priori*)

Paso 2: Aleatorizar conjunto → Solución Inicial.

	Posibles Soluciones
Tarea 1	123, 132, 213, 231, 312, 321
Tarea 2	
Tarea 3	

Tabla 2. Ejemplo de posibles soluciones con un conjunto de 3 tareas.

3.4.2 Construcción de la función objetivo.

Se definen dos funciones para la elaboración de la función objetivo:

- Función Distancia Tarea.

Paso 1: Generar tarea *i*.

Paso 2: Calcular distancia de tarea *i*, tal que la distancia corresponde al sumatorio del tiempo de realizar la tarea *i* (tiempo de desplazamiento desde el origen a su destino), y su tiempo de carga y descarga.

- Función Distancia Cadena.

Paso 1: Generar una solución inicial. (*Apartado 4.3.1*)

Paso 2: Evaluar primera tarea de la solución inicial. C = 0; CAP = 40;

2.1 Tarea de capacidad 40:

C = 0;

2.1.1 Primera tarea, se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida.

2.1.2 Si la tarea actual *i* corresponde en origen igual al destino de la tarea anterior *i*-1,

Si el vehículo actual puede realizar dicha tarea en su tiempo operativo, realizar tarea, calcular la Función Distancia Tarea y añadirla al contador de Distancia Total Recorrida.

Si no, el vehículo actual vuelve al depósito y se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida.

2.1.3 Si no,

Si el vehículo actual puede realizar dicha tarea en su tiempo operativo, realizar tarea, calcular la Función Distancia Tarea y añadirla al contador de Distancia Total Recorrida, además incluir el tiempo vacío de desplazamiento de la tarea anterior a la actual.

Si no, el vehículo actual vuelve al depósito y se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida.

2.2 Tarea de capacidad 20:

C = C + 20;

C < CAP

2.2.1 Primera tarea, se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida.

2.2.2 Si la tarea actual *i* corresponde en origen igual al destino de la tarea anterior *i*-1,

Si el vehículo actual puede realizar dicha tarea en su tiempo operativo, realizar tarea, calcular la Función Distancia Tarea y añadirla al contador de Distancia Total Recorrida.

Si no, el vehículo actual vuelve al depósito y se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida.

2.2.3 Si no,

Si el vehículo actual puede realizar dicha tarea en su tiempo operativo, realizar tarea, calcular la Función Distancia Tarea y añadirla al contador de Distancia Total Recorrida, además incluir el tiempo vacío de desplazamiento de la tarea anterior a la actual.

Si no, el vehículo actual vuelve al depósito y se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida.

C = CAP

2.2.4 Si la tarea actual *i* corresponde en origen y en destino igual a la tarea anterior *i*-1,

Si el vehículo actual puede realizar la tarea actual en su tiempo operativo, realizar tarea, PERO NO calcular la Función Distancia Tarea. Tampoco se añade nuevo tiempo al contador de Distancia Total Recorrida (se contabiliza una vez, en la tarea anterior). C = 0

Si no, el vehículo actual vuelve al depósito y se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida. C = 20

2.2.5 Si no,

2.2.5.1 Si la tarea actual *i* corresponde en origen igual al destino de la tarea anterior *i*-1,

Si el vehículo actual puede realizar dicha tarea en su tiempo operativo, realizar tarea, calcular la Función Distancia Tarea y añadirla al contador de Distancia Total Recorrida. C = 0

Si no, el vehículo actual vuelve al depósito y se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida. C = 20

2.2.5.2 Si no,

Si el vehículo actual puede realizar dicha tarea en su tiempo operativo, realizar tarea, calcular la Función Distancia Tarea y añadirla al contador de Distancia Total Recorrida, además incluir el tiempo vacío de desplazamiento de la tarea anterior a la actual. C = 0

Si no, el vehículo actual vuelve al depósito y se introduce nuevo vehículo desde depósito que realiza la tarea actual. Se calcula la Función Distancia Tarea y se añade al contador de Distancia Total Recorrida. C = 20

Paso 3: Repetir el paso 2 hasta evaluar el resto de tareas sucesivas del conjunto de la solución inicial.

Paso 4: Obtener el contador final de la Distancia Total Recorrida.

3.4.3 Construcción del vecindario.

Para la construcción de los vecindarios, en ambos casos de Recocido Simulado y Búsqueda Tabú, se implementa permutaciones entre un elemento *i* y el elemento más inmediato *i*+1. Es decir, una vez generada una solución inicial, el vecindario de ésta se crea aplicando permutaciones de un elemento por el elemento más inmediato.

También se tiene en cuenta la repetición de cada elemento: hay *n* elementos donde el primer elemento se repite n_1 veces, el segundo n_2 veces… hasta el último elemento, que se repite n_k veces. Debido a esto, las permutaciones se aplican entre los *n* elementos, esto es que la permutación se hará entre elementos diferentes, evitando permutar elementos iguales que no producirían ningún cambio en la estructura del vecindario.

Para aclarar mejor la idea, se muestra en la Tabla 3 un ejemplo de cuál es la permutación considerada y cuál es una permutación mala que no produce ningún cambio. Como se aprecia, una permutación entre el elemento 1 y el elemento inmediato, que es otro 1, no produce variación ninguna en la estructura por lo que el vecindario sería el mismo que la solución inicial. Por lo que se amplía la permutación hasta el siguiente elemento que sea diferente, en este ejemplo, el elemento 2.

Conjunto de Tareas {1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4} ↓ **Solución Inicial Generada Aleatoria** {1, 1, 1, 1, 2, 2, 1, 1, 2, 3, 3, 4, 3}	**Permutación Mala** NO {1, 1, 1, 1, 2, 2, 1, 1, 2, 3, 3, 4, 3} **Permutación Buena** SÍ {1, 1, 1, 1, 2, 2, 1, 1, 2, 3, 3, 4, 3}

Tabla 3. Ejemplo de permutación con repetición para la construcción del vecindario.

3.4.4 Códigos.

A través del lenguaje de programación *Python* se escribe el código de resolución para las metaheurísticas. En el Anexo II se encuentran los códigos de Recocido Simulado, Búsqueda Tabú y el código referente a la función objetivo.

Tanto los algoritmos metaheurísticos como la función objetivo se escriben y se implementan en *Python* 2.7. con la ayuda del software *Spyder*, entorno de desarrollo integrado multiplataforma de código abierto para programación científica en lenguaje *Python*.

4 ANÁLISIS DE RESULTADOS

En función de observar la aplicación real de lo desarrollado anteriormente, se procede a simular en el entorno de programación *Spyder* para generar soluciones en los dos casos posibles, recocido simulado y búsqueda tabú.

Se añaden diez escenarios con diferentes parámetros en los dos casos, generando diversos experimentos computacionales para así poder validar, verificar y comparar los dos métodos de resolución utilizados en este estudio.

Estos métodos son comparados en base al coste total (la distancia total recorrida) y el tiempo computacional que requiere ejecutarlos hasta obtener solución con los parámetros establecidos.

4.1 Experimentos metaheurísticos.

Los datos que se asumen a priori para este problema son para cada solicitud de movimiento de contenedor, el cual genera dos nodos: su punto de recogida (punto de inicio) y su punto de entrega (punto de finalización). Entonces para cada solicitud se tiene:

- El identificador de la solicitud (T). Se basa en un identificador del tipo de movimiento de contenedor, ya sea de 20 o de 40 pies, seguido del número (asignación por orden de entrada) de la solicitud a la que corresponde ese movimiento: Por ejemplo, i40_2 sería un movimiento de contenedor de 40 pies de la solicitud nº2).
- El origen de la solicitud de movimiento, donde se inicia la actividad (O).
- El destino de la solicitud de movimiento, donde se finaliza la actividad (D).
- El tiempo necesario para moverse desde el origen al destino de la actividad (DD1).
- El tiempo de carga del contenedor necesario para iniciar la actividad de la solicitud de movimiento (TC).
- El tiempo de descarga del contenedor necesario para finalizar la actividad de la solicitud de movimiento (TD).
- El tiempo necesario para moverse desde el depósito hasta el origen de la actividad (SI).
- El tiempo necesario para moverse desde el destino de la actividad hasta el depósito (SV).
- El número de contenedores de 40 pies que deben moverse en total de una misma solicitud (R40).
- El número de contenedores de 20 pies que deben moverse en total de una misma solicitud (R20).

Para comparar las dos metaheurísticas, se determinan varios parámetros para la búsqueda tabú y recocido simulado. La selección de parámetros ejerce una influencia significativa en las soluciones y el tiempo de ejecución en ambos algoritmos.

Para la búsqueda tabú, los parámetros a determinar son la longitud de la lista tabú(*tabu_size*) y el criterio de detención, que sería el número fijo de iteraciones del algoritmo para buscar solución (*n_iteration*). A mayor número de iteraciones, mayor oportunidad de buscar en áreas mayores, pero también mayor tiempo computacional.

Debido al gran tiempo computacional que se requiere para manejar un problema de estas dimensiones, se propone reducir la búsqueda en el vecindario de modo que en cuanto encuentre una mejor solución en comparación a la solución mejor conocida, se para el proceso de búsqueda en ese vecindario y pasa a otro vecindario. En otras palabras, no se va a buscar en todo el vecindario y de este conjunto sacar la solución óptima, si no que en cuanto encuentre una mejor a la actual, se produce otra iteración (otro vecindario).

Además, el parámetro *n_iteration* de la búsqueda tabú no se va a predefinir a priori. Dado al objetivo de su aplicación en Python. se estudia que requiera un tiempo de computacional medianamente aplicado a una jornada de trabajo normal, en consecuencia, el tiempo requerido en encontrar solución esté entorno a las 6 horas. Con esta idea se cambia el parámetro de número de iteraciones por tiempo (horas) de computación.

La metaheurística se ejecutará dentro de un límite temporal, de manera que la última iteración a realizar esté en el máximo de 6 horas y posteriormente lo que tarde esta iteración en resolverse podrá suponer un suplemento al tiempo. Es decir, que podrá tardar más de 6 horas debido al tiempo necesario para finalizar la última iteración.

Dado a que se está estudiando la respuesta del algoritmo en encontrar solución basado en el tiempo se ha optado por ceder ese margen de tiempo en resolver la última iteración, pero para una aplicación real podría estudiarse la idea de que el algoritmo se detuviera a las 6 horas completamente y devolviera la mejor solución que tuviera.

Este límite temporal se establece para tres casos diferentes donde se estudie la respuesta del algoritmo en torno a las 2 horas, en 4 horas y por último en 6 horas como se mencionaba anteriormente. El otro parámetro *tabu_size* se mantiene fijo para los tres casos y es igual a la longitud de todas las solicitudes/tareas del problema (*len(points)*).

En cuanto al recocido simulado, se determinan la temperatura máxima inicial (*max_temp*), la temperatura mínima final (*min_temp*), el coeficiente de enfriamiento (*alpha*) y el número de iteraciones por cada nivel de temperatura (L). El algoritmo de recocido simulado selecciona aleatoriamente un vecino en cada iteración, por lo que puede dar como resultado diferentes soluciones usando igual conjunto de parámetros en cada ejecución del algoritmo. Es por ello que cuantas más repeticiones ejecutamos, más preciso es en la obtención del valor óptimo, es decir, más se elimina el efecto de aleatoriedad en el resultado. Sin embargo, más repeticiones requieren el aumento en el tiempo de cálculo.

Debido a gran tiempo computacional que se requiere para manejar un problema de estas dimensiones, en esta ocasión, se propone rebajar la vecindad, en lugar de explorar todas las vecindades de i y j, se fija aleatoriamente i y explora todos los j.

Para esta metaheurística se han fijado unos parámetros que obtienen solución en torno a los tres casos que se establecen para la metaheurística búsqueda tabú, es decir, se establecen los mismos parámetros para los tres casos exceptuando el parámetro L que variara para encontrar solución aproximadamente en 2 horas, en 4 horas y, por último, 6 horas.

En conclusión, se establecen tres escenarios con parámetros diferentes para cada método de resolución y para cada escenario se realizan diez experimentos computacionales. En la Tabla 4 se muestran los escenarios metaheurísticos que se realizan en este trabajo para estudiar la influencia de diferentes combinaciones de parámetros en la obtención de la solución óptima y el tiempo computacional.

Recocido Simulado			Búsqueda Tabú		
Escenario 1	**Escenario 2**	**Escenario 3**	**Escenario 1**	**Escenario 2**	**Escenario 3**
max_temp: 500	max_temp: 500	max_temp: 500	tabu_size: len(points)	tabu_size: len(points)	tabu_size: len(points)
min_temp: 0.2	min_temp: 0.2	min_temp: 0.2	tiempo: 6 h	tiempo: 4 h	tiempo: 2 h
L: 700	L: 1000	L: 1500			
α: 0.2	α: 0.2	α: 0.2			

Tabla 4. Escenarios metaheurísticos bajo estudio.

4.2 Resultados computacionales.

Para cada escenario se ensayan 10 experimentos y de estos experimentos se obtienen varios resultados: el tiempo computacional requerido hasta que le algoritmo obtiene solución, la distancia total recorrida por todos los vehículos y las solicitudes que debe realizar cada camión, además con ello, el número total de vehículos que se emplean para realizar todas las solicitudes.

Según lo definido en el apartado anterior, los datos empleados en este estudio se muestran en la Tabla 5. Los resultados del coste total (distancia total recorrida) y el tiempo (en segundos) computacional empleado en los diferentes escenarios de los dos casos metaheurísticos se muestra en la Tabla 6 para Recocido Simulado y en la Tabla 7 para Búsqueda Tabú. Además, en el caso de la búsqueda tabú, se muestra el número de iteraciones realizadas. El resultado de solicitudes por camiones se puede encontrar en el Anexo V.

T	O	D	DD1	TC	TD	SI	SV	R40	R20
0	1	2	50	5	5	30	30	30	8
1	1	5	30	5	5	30	0	1	3
2	1	8	35	5	5	30	10	48	15
3	2	1	50	5	5	30	30	8	24
4	2	4	40	5	5	30	20	36	13
5	2	5	30	5	5	30	0	47	3
6	3	5	30	5	5	30	0	25	5
7	3	6	35	5	5	30	5	1	26
8	3	8	25	5	5	30	10	13	13
9	4	2	40	5	5	20	30	29	6
10	4	5	20	5	5	20	0	17	28
11	4	7	25	5	5	20	5	11	10
12	5	1	30	5	5	0	30	26	15
13	5	2	30	5	5	0	30	8	20
14	5	3	30	5	5	0	30	48	3
15	5	4	20	5	5	0	20	28	6
16	5	6	5	5	5	0	5	23	13
17	5	7	5	5	5	0	5	46	17
18	5	8	10	5	5	0	10	3	18
19	6	3	35	5	5	5	30	41	5
20	6	5	5	5	5	5	0	9	9
21	6	8	15	5	5	5	10	45	28
22	7	4	25	5	5	5	20	7	15
23	7	5	5	5	5	5	0	38	6
24	8	1	35	5	5	10	30	45	29
25	8	3	25	5	5	10	30	4	29
26	8	5	10	5	5	10	0	10	22
27	8	6	15	5	5	10	5	32	22

Tabla 5. Datos del problema.

Recocido Simulado

Escenario 1	*Escenario 2*	*Escenario 3*
Tiempo Total: 38645 Tiempo de Ejecución: 7225.09899998	Tiempo Total: 37555 Tiempo de Ejecución: 17214.1389999	Tiempo Total: 37035 Tiempo de Ejecución: 20029.0949998
Tiempo Total: 39030 Tiempo de Ejecución: 8180.84099984	Tiempo Total: 38190 Tiempo de Ejecución: 17450.4720001	Tiempo Total: 37055 Tiempo de Ejecución: 25265.737
Tiempo Total: 39690 Tiempo de Ejecución: 8953.49499989	Tiempo Total: 37550 Tiempo de Ejecución: 17831.4990001	Tiempo Total: 36135 Tiempo de Ejecución: 26471.9979999
Tiempo Total: 39225 Tiempo de Ejecución: 8422.51699996	Tiempo Total: 37485 Tiempo de Ejecución: 15250.0020001	Tiempo Total: 36425 Tiempo de Ejecución: 25477.095
Tiempo Total: 38640 Tiempo de Ejecución: 8427.26699996	Tiempo Total: 37915 Tiempo de Ejecución: 15979.1719999	Tiempo Total: 36675 Tiempo de Ejecución: 26200.303
Tiempo Total: 38415 Tiempo de Ejecución: 9436.06500006	Tiempo Total: 38065 Tiempo de Ejecución: 16293.342	Tiempo Total: 36955 Tiempo de Ejecución: 23349.3660002
Tiempo Total: 38805 Tiempo de Ejecución: 9144.27900004	Tiempo Total: 38165 Tiempo de Ejecución: 15499.744	Tiempo Total: 36890 Tiempo de Ejecución: 24844.6590002
Tiempo Total: 38685 Tiempo de Ejecución: 9094.80900002	Tiempo Total: 37865 Tiempo de Ejecución: 15983.9790001	Tiempo Total: 36895 Tiempo de Ejecución: 27828.619
Tiempo Total: 38545 Tiempo de Ejecución: 7696.34200001	Tiempo Total: 38520 Tiempo de Ejecución: 16290.631	Tiempo Total: 36815 Tiempo de Ejecución: 24216.4430001
Tiempo Total: 38635 Tiempo de Ejecución: 7355.39199996	Tiempo Total: 37260 Tiempo de Ejecución: 10875.2130001	Tiempo Total: 37025 Tiempo de Ejecución: 23505.9630001

Tabla 6. Resultados computacionales de los diez experimentos del algoritmo Recocido Simulado.

Búsqueda Tabú

Escenario 1	*Escenario 2*	*Escenario 3*
Iteraciones: 1402 Tiempo Total: 42010 Tiempo de Ejecución: 7722.38200021	Iteraciones: 1389 Tiempo Total: 41975 Tiempo de Ejecución: 17394.1329999	Iteraciones: 1413 Tiempo Total: 42985 Tiempo de Ejecución: 22102.5550001
Iteraciones: 1536 Tiempo Total: 41450 Tiempo de Ejecución: 7336.37699986	Iteraciones: 1437 Tiempo Total: 42470 Tiempo de Ejecución: 23258.9230001	Iteraciones: 1393 Tiempo Total: 41785 Tiempo de Ejecución: 21630.4480002
Iteraciones: 1454 Tiempo Total: 42010 Tiempo de Ejecución: 7493.25499988	Iteraciones: 1336 Tiempo Total: 43545 Tiempo de Ejecución: 15745.03	Iteraciones: 1486 Tiempo Total: 41000 Tiempo de Ejecución: 24370.6999998
Iteraciones: 1334 Tiempo Total: 44050 Tiempo de Ejecución: 15082.1110001	Iteraciones: 1492 Tiempo Total: 42065 Tiempo de Ejecución: 16816.9039998	Iteraciones: 1790 Tiempo Total: 39145 Tiempo de Ejecución: 28482.2220001
Iteraciones: 1506 Tiempo Total: 41580 Tiempo de Ejecución: 7239.75500011	Iteraciones: 1302 Tiempo Total: 43030 Tiempo de Ejecución: 20472.8469999	Iteraciones: 1308 Tiempo Total: 42930 Tiempo de Ejecución: 22517.543
Iteraciones: 1371 Tiempo Total: 42630 Tiempo de Ejecución: 7546.39299989	Iteraciones: 1491 Tiempo Total: 41375 Tiempo de Ejecución: 18957.9419999	Iteraciones: 1478 Tiempo Total: 41105 Tiempo de Ejecución: 22461.8380001
Iteraciones: 1410 Tiempo Total: 42515 Tiempo de Ejecución: 7534.84500003	Iteraciones: 1289 Tiempo Total: 44420 Tiempo de Ejecución: 15790.7850001	Iteraciones: 1253 Tiempo Total: 45225 Tiempo de Ejecución: 21678.3859999
Iteraciones: 1353 Tiempo Total: 43395 Tiempo de Ejecución: 7510.64100003	Iteraciones: 1389 Tiempo Total: 42710 Tiempo de Ejecución: 15856.2320001	Iteraciones: 1419 Tiempo Total: 42365 Tiempo de Ejecución: 34064.24
Iteraciones: 1445 Tiempo Total: 42725 Tiempo de Ejecución: 7253.14999986	Iteraciones: 1381 Tiempo Total: 42240 Tiempo de Ejecución: 16075.372	Iteraciones: 1438 Tiempo Total: 42000 Tiempo de Ejecución: 22921.325
Iteraciones: 1358 Tiempo Total: 42780 Tiempo de Ejecución: 7274.04200006	Iteraciones: 1443 Tiempo Total: 42930 Tiempo de Ejecución: 25620.0569999	Iteraciones: 1440 Tiempo Total: 42085 Tiempo de Ejecución: 22633.9549999

Tabla 7. Resultados computacionales de los diez experimentos del algoritmo Búsqueda Tabú.

4.3 Estudio Estadístico.

En este apartado se expone una comparación en términos de media para los resultados de tiempo computacional que se obtiene para cada escenario de 2 horas, 4 horas y 6 horas y se comparan con los resultados de costes totales (distancia total recorrida) de los experimentos calculando su media, desviación estándar máxima y minima. Para búsqueda tabú, además, se añade la comparación respecto al número de iteraciones.

Este estudio estadístico es necesario para poder concluir las comparaciones en el rendimiento entre métodos y el conjunto de parámetros establecido en cada escenario.

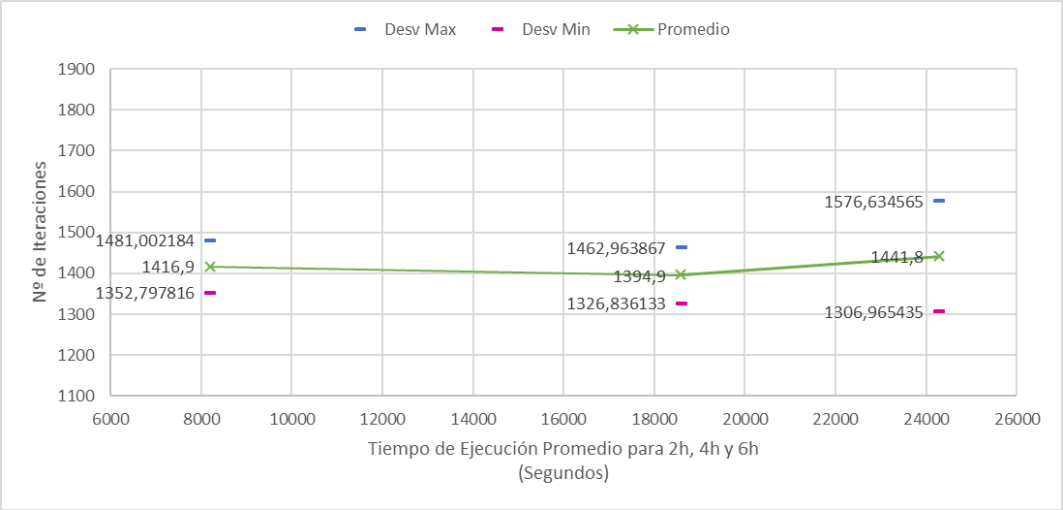


Figura 13. Comparación de tiempo de ejecución promedio y nº de iteraciones de los experimentos de TS.

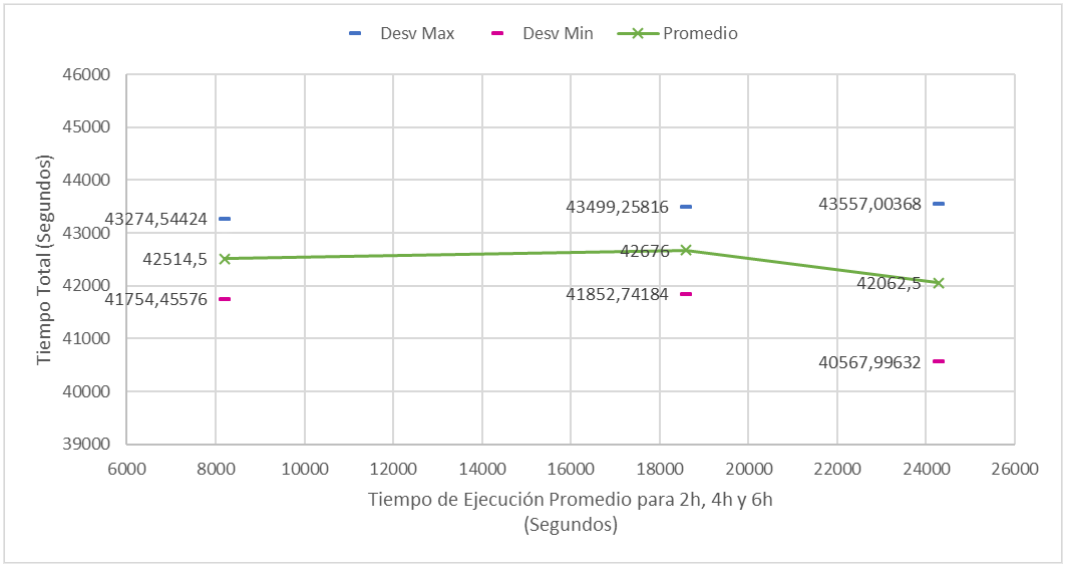


Figura 14. Comparación de tiempo de ejecución promedio y tiempo total de los experimentos de TS.

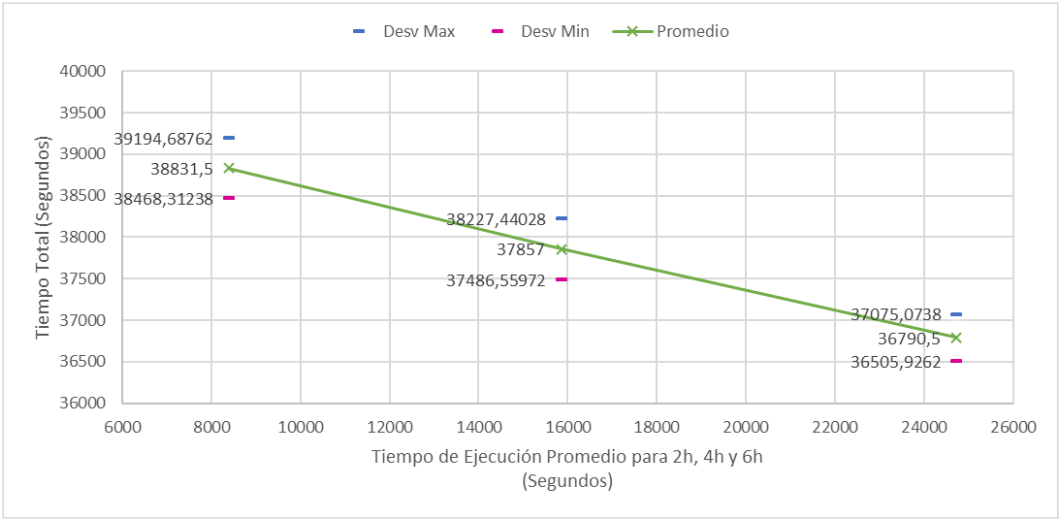


Figura 15. Comparación de tiempo de ejecución promedio y tiempo total recorrido de los experimentos de SA.

5 CONCLUSIONES

El objetivo en este problema es enrutar todas las solicitudes de entrega y recogida a vehículos disponibles considerando las restricciones de tiempo de operación del vehículo y de tamaño de los contenedores en el menor tiempo de ejecución posible para reducir costes.

Con la ayuda de dos algoritmos metaheurísticos se ha testeado el problema para conseguir soluciones con diferentes parámetros. Sin embargo, estos algoritmos no son capaces de producir soluciones en tiempos razonables si tenemos en cuenta todo el espacio de búsqueda en el vecindario. Debido al gran tiempo computacional requerido para resolver un problema de estos tamaños, se ha procedido a implementar una búsqueda de vecindario "más corta" y diferente para cada algoritmo metaheurístico para así acelerar los cálculos. Estas construcciones de vecindarios son:

- Para el algoritmo de recocido simulado se propone rebajar la vecindad, en lugar de explorar todas las vecindades de *i* y *j*, se fija aleatoriamente *i* y explora todos los *j*.
- Para la búsqueda tabú se propone reducir la búsqueda en el vecindario de modo que en cuanto encuentre una mejor solución en comparación a la solución mejor conocida, se para el proceso de búsqueda en ese vecindario y pasa a otro vecindario.

Con esta idea se han simulado 60 experimentos en total, 30 para cada algoritmo, de los cuales hay 10 para cada escenario de parámetros diferentes. De estos experimentos se obtiene el Tiempo Total (Coste Total) y Tiempo de Ejecución. En el caso de la búsqueda tabú, además se obtiene el número de iteraciones. Esta información está recogida en las Tablas 6 y 7, y los análisis respectivos a cada caso en las Figuras 13, 14 y 15.

En cuanto al algoritmo de búsqueda tabú se puede concluir de los experimentos que el tiempo de ejecución no influye en gran medida a obtener mejores soluciones (tiempo total), es decir, a mayor tiempo de computación no implica que se vaya a obtener mejores resultados. Esto quiere decir que llega bastante rápido (dentro de las dos horas) a los mejores resultados posibles y a partir de ahí no encuentra gran variación en un óptimo mejor.

Tiempo de Ejecución			
Escenario	Desv Max	Desv Min	Promedio
1	10498,37626	5900,213942	8199,2951
2	21903,3728	15294,2722	18598,8225
3	28066,46758	20506,17482	24286,3212
Iteraciones			
Escenario	Desv Max	Desv Min	Promedio
1	1481,002184	1352,797816	1416,9
2	1462,963867	1326,836133	1394,9
3	1576,634565	1306,965435	1441,8
Tiempo Total			
Escenario	Desv Max	Desv Min	Promedio
1	43274,54424	41754,45576	42514,5
2	43499,25816	41852,74184	42676
3	43557,00368	40567,99632	42062,5

Tabla 8. Resumen estadístico de los rtdos computacionales de los experimentos del algoritmo TS.

Hay que mencionar que se parte siempre de una solución inicial aleatoria y que no se busca en todo el vecindario, pero realizando diez experimentos, todos indican la misma conclusión de que llega rápidamente a las mejores soluciones posibles. Esto se confirma mediante la observación de los

experimentos de como evolucionaban rápidamente a altas iteraciones y se detenían produciendo más tiempo computacional en las últimas iteraciones, resultándole difícil encontrar otra solución mejor a la actual.

En cuanto al algoritmo de recocido simulado se puede concluir de los experimentos que el tiempo de ejecución si influye con relación a obtener mejores resultados. A mayor tiempo de ejecución (en nuestro caso se optó por aumenta el parámetro *L*, número de iteraciones por temperatura), se obtienen mejores resultados.

Tiempo Total			
Escenario	Desv Max	Desv Min	Promedio
1	39194,68762	38468,31238	38831,5
2	38227,44028	37486,55972	37857
3	37075,0738	36505,9262	36790,5
Tiempo de Ejecución			
Escenario	Desv Max	Desv Min	Promedio
1	9130,638681	7656,582519	8393,6106
2	17710,79385	14022,84475	15866,8193
3	26754,15602	22683,69958	24718,9278

Tabla 9. Resumen estadístico de los rtdos computacionales de los experimentos del algoritmo SA.

Finalmente, comparando los dos algoritmos se puede observar que el algoritmo de recocido simulado produce mejores soluciones en cuanto a tiempo total, con diferencia al algoritmo de búsqueda tabú a igualdad aproximada de tiempo computacional.

5.1 Investigación futura.

- Estructura del vecindario: Este trabajo propone dos estructuras de vecindario para acelerar los cálculos, pero pueden no ser las óptimas. Futuras investigación pueden abordar la aplicación de otras construcciones de vecindario que sean más adecuadas.
- Metaheurísticas: La aplicación de otros algoritmos metaheurísticas pueden producir cambios interesantes que lleguen a optimizar la resolución de estos problemas de forma significante. En consecuencia, futuras investigaciones podrían centrarse en la aplicación de otras metaheurísticas.
- Solución inicial: La generación de la solución inicial puede ser mejorada para que encuentre una solución inicial más adecuada y así compenzar el algoritmo con una solución que muestre una buena dirección de optimización, lo cual podría ahorrar mucho tiempo computacional.
- Perspectiva dinámica: La mayoría de los problemas de recogida y entrega de aplicación en el mundo real tienen demanda sensible al tiempo. Actualmente se ha empezado a investigar sobre el problema dinámico. De la importancia práctica, parece ser un área de investigación con mucho potencial.

Referencias

[1] RAFF, S., 1983. Routing and scheduling of vehicles and crews: The state of the art. En: *Computers & Operations Research*. Reino Unido: Elsevier, Vol. 10, Iss. 2; 63-211. ISSN 0305-0548.

[2] INGBER, L., 1989. Very fast simulated re-annealing. En: *Mathematical and Computer Modelling.* Reino Unido: Elsevier, Vol. 12, Iss. 8; 967-973. ISSN 0895-7177.

[3] SARUWATARI, Y., HIRABAYASHI, R. y NISHIDA, N., 1992. Node duplication lower bounds for the capacitated arc routing problema. En: *Journal of the Operations Research Society of Japan*. Japón: The Operations Research Society of Japan, Vol. 35, Iss. 2; 119-133. ISSN 0453-4514.

[4] SAVELSBERGH, M. W. P. y SOL, M., (1995). The General Pickup and Delivery Problem. En: *Transportation Science*. Estados Unidos: INFORMS, Vol. 29, Iss. 1; 17-29. ISSN 1526-5447.

[5] BURKE, E.K., NEWALL, J.P. y WEARE, R.F., (1996). A memetic algorithm for university exam timetabling. En: *Lecture Notes in Computer Science: Practice and Theory of Automated Timetabling.* Springer, Vol. 1153. ISBN 978-3-540-61794-5.

[6] EUROPEAN COMMISSION, 1997. *Intermodality and intermodal freight transport in the European Union. A systems approach to freight transport. Strategies and actions to enhance efficiency, services and sustainability* [en línea]. Luxemburgo: Oficina de Publicaciones Oficiales de las Comunidades Europeas [consulta: 12 de octubre de 2017]. Disponible en: http://cordis.europa.eu/pub/transport/docs/intermodal_freight_transport_en.pdf

[7] CHAO, I. M., GOLDEN, B. L. y WASIL, E. A., 1998. A New Algorithm for the Site-Dependent Vehicle Routing Problem. En: *Advances in Computational and Stochastic Optimization, Logic Programming and Heuristic Search*. Estados Unidos: Springer, Vol. 9; 301-312. ISBN 978-1-4419-5023-9.

[8] DEWITT, W., CLINGER, J., UNIVERSITY OF MARYLAND y LOUIS BERGER GROUP, 1999. Intermodal Freight Transportation [en línea]. En: *Committee on Intermodal Freight Transport, TRB* [consulta: 5 de octubre de 2017]. Disponible en: http://onlinepubs.trb.org/onlinepubs/millennium/00061.pdf

[9] CARRETO, C. y BAKER, B., 2001. An Improved GRASP Interactive Approach for the Vehicle Routing Problem with Backhauls. En: *4th Metaheuristics International Conference: 16 a 20 de julio.* Portugal.

[10] WANG, X. y REGAN, A. C., 2002. Local truckload pickup and delivery with hard time window constraints. En: *Transportation Research, Part B: Methodological*. Elsevier, Vol. 36, Iss. 2; 97-112. ISSN 0191-2615.

[11] ASTOLFI, A, 2002. Optimization, an introduction [en línea]. Reino Unido: Imperial College London. Actualizado: septiembre 2006 [consulta: 26 de septiembre de 2017]. Disponible en: https://www.researchgate.net/file.PostFileLoader.html?id=5825d06093553b7b954cf2c9&assetKey=AS%3A427239255875586%401478873184535

[12] GENDREAU, M., 2003. An Introduction to Tabu Search. En: *Handbook of Metaheuristics*. Estados Unidos: Springer, Vol. 57; 37-54. ISBN 978-1-4020-7263-5.

[13] JULA, H., DESSOUKY, M., IOANNOU, P. y CHASSIAKOS, A., 2005. Container movement by trucks in metropolitan networks: modeling and optimization. En: *Transportation Research, Part E: Logistics and Transportation Review*. Elsevier, Vol. 41, Iss. 3; 235–259. ISSN 1366-5545.

[14] GARCÍA, J. P., 2006. Modelado mediante Optimización Combinatoria [en línea]. Universidad Politécnica de Valencia [consulta: 25 de septiembre de 2017]. Disponible en: http://personales.upv.es/jpgarcia/LinkedDocuments/MCOIOptimizacionCombinatoria.pdf

[15] PETERSEN, H. L., 2006. Heuristic Solution Approaches to the Double TSP with Multiple Stacks. En: *Optimization Days (CORS 2006): 8 a 10 de mayo/Third International Workshop on Freight Transportation and Logistics (Odysseus 2006): 23 a 26 de mayo.* Canada/España.

[16] SMILOWITZ. K., 2006. Multi-resource routing with flexible tasks: an application in drayage operations. En: *IIE Transactions*. Taylor & Francis, Vol. 38, Iss. 7; 577-590. ISSN 0740-817X.

[17] IMAI, A., NISHIMURA, E. y CURRENT, J., 2007. A lagrangian relaxation-based heuristic for the vehicle routing with full container load. En: *European Journal of Operational Research*. Elsevier, Vol. 176, Iss. 1; 87–105. ISSN 0377-2217.

[18] CHUNG K. H., KO C. S., SHIN, J. Y., HWANG, H. y KIM, K. H., 2007. Development of mathematical models for the container road transportation in Korean trucking industries. En: *Computers & Industrial Engineering*. Elsevier, Vol. 53, Iss. 2; 252-262. ISSN 0360-8352.

[19] SREENIVAS, M. y SRINIVAS, T., 2008. The role of transportation in logistics chain [en línea]. En: *Journal of Mathematics and Mathematical Sciences* [consulta: 17 de septiembre de 2017]. Disponible en: http://www.siam.org/journals/plagiary/1814.pdf

[20] NAMBOOTHIRI, R. y ERERA, A. L., 2008. Planning local container drayage operations given a port access appointment system. En: *Transportation Research, Part E: Logistics and Transportation Review*. Elsevier, Vol. 44, Iss. 2; 185-202, ISSN 1366-5545.

[21] CHEUNG, R. K., SHI, N., POWELL, W. B. y SIMAO, H. P., 2008. An attribute–decision model for cross-border drayage problema. En: *Transportation Research, Part E: Logistics and Transportation Review*. Elsevier, Vol. 44, Iss. 2; 217-234. ISSN 1366-5545.

[22] CARIS, A. y JANSSENS, G., 2009. A local search heuristic for the pre- and end-haulage of intermodal container terminals. En: *Computers & Operations Research*. Elsevier, Vol. 36, Iss. 10; 2763-2772. ISSN 0305-0548.

[23] ZHANG, R., YUN, W. Y. y KOPFER, H., 2010. Heuristic-based truck scheduling for inland container transportation. En: *OR Spectrum*. Springer, Vol. 32, Iss. 3; 787-808. ISSN 0171-6468.

[24] ZHANG, G., SMILOWITZ, K. y ERERA, A., 2011. Dynamic planning for urban drayage operations. En: *Transportation Research, Part E: Logistics and Transportation Review*. Elsevier, Vol. 47, Iss. 5; 764-777. ISSN 1366-5545.

[25] VIDOVIC, M., RADIVOJEVIC, G. y RAKOVIC, B., 2011. Vehicle routing in containers pickup up and delivery processes. En: *Procedia, Social and Behavioral Sciences*. Elsevier, Vol. 20; 335-343. ISSN 1877-0428.

[26] BRAEKERS, K., CARIS, A. y JANSSENS, G.K., 2011. A deterministic annealing algorithm for abi-objective full truckload vehicle routing problem in drayage operations. En: *Procedia, Social and Behavioral Sciences*. Elsevier, Vol. 20; 344-353. ISSN 1877-0428.

[27] REINHARDT, L., SPOORENDONK, S. y PISINGER, D., 2012. Solving vehicle routing with full container load and time windows. En: *Computational Logistics.* Alemania: Springer, Vol. 7555; 120-128. ISBN 978-3-642-33586-0.

[28] MIHIR SHAH, M., 2012. Artificial Intelligence: Vehicle Routing Problem and Multi Agent System. En: *National Conference on Development of Reliable Information Systems, Techniques and Related Issues: 16 a 17 de marzo.* India.

[29] RAJAN, K, 2013. Chapter 2, Genetic algorithm and its adaptiveness [en línea]. En: *Adaptive techniques in genetic algorithm and its applications.* Mahatma Gandhi University [consulta: 23 de septiembre de 2017]. Disponible en: http://hdl.handle.net/10603/21381

[30] SCHÖNBERGER, J., BUER, T. y KOPFER, H., 2013. A Model for the Coordination of 20-foot and 40-foot Container Movements in the Hinterland of a Container Terminal. En: *Computational Logistics*. Springer, Vol. 8197; 113-127. ISBN 978-3-642-41018-5.

[31] BRAEKERS, K., CARIS, A. y JANSSENS, G., 2013. Integrated planning of loaded and empty container movements. En: *OR Spectrum*. Springer, Vol. 35, Iss. 2; 457-478. ISSN 0171-6468.

[32] NOSSACK, J. y PESCH, E., 2013. A truck scheduling problem arising in intermodal container transportation. En: *European Journal of Operational Research*. Elsevier, Vol. 230, Iss. 3; 666-680. ISSN 0377-2217.

[33] WANG, W.F. y YUN, W.Y., 2013. Scheduling for inland container truck and train transportation. En: *International Journal of Production Economics*. Elsevier, Vol. 143, Iss. 2; 349-356. ISSN 0925-5273.

[34] ESCUDERO, A., MUÑUZURI, J., GUADIX, J. y ARANGO, C., 2013. Dynamic approach to solve the daily drayage problem with transit time uncertainty. En: *Computers in Industry*. Elsevier, Vol. 64, Iss. 2; 165-175. ISSN 0166-3615.

[35] GLEISSNER, H. y FEMERLING, J. C., 2013. The Principles of Logistics. En: *Logistics. Basics, Exercises, Case Studies*. Suiza: Springer; 3-18. ISBN 978-3-319-01768-6.

[36] TIWARI, M.K., KUMAR, R.A., MOHAPATRA, P., YEW, W.K. y BENYOUCEF, L., 2014. Route Selection and Consolidation in International Intermodal Freight Transportation. En: *Applications of Multi-Criteria and Game Theory Approaches*. Reino Unido: Springer; 181-223. ISBN 978-1-4471-5294-1.

[37] XUE, Z., LIN, W., MIAO, L. y ZHANG, C., 2014. Local container drayage problem with tractor and trailer operating in separable mode. En: *Flexible Services and Manufacturing Journal*. Springer, Vol. 27, Iss.2-3;431-450. ISSN 1936-6582.

[38] ANBUUDAYASANKAR, S. P., GANESH, K. y MOHAPATRA, S., 2014. *Models for Practical Routing Problems in Logistics*. Suiza: Springer. ISBN 978-3-319-05034-8.

[39] ECORYS, FRAUNHOFER, TCI, PROGNOS y AUEB-RC/TRANSLOG, 2015. *Fact-finding studies in support of the development of an EU strategy for freight transport logistics, Lot 1: Analysis of the EU logistics sector* [en línea]. Bruselas: European Commission, Mobility and Transport DG [consulta: 12 de octubre de 2017]. Disponible en: https://ec.europa.eu/transport/sites/transport/files/themes/strategies/studies/doc/2015-01-freight-logistics-lot1-logistics-sector.pdf

[40] KARIMI, E., MALEKI, H. R. y AKBARI, R., 2015. Tabu Search Algorithm to Solve the Intermodal Terminal Location Problem. En*: Journal of Mathematical Extension*. Vol. 9, Iss. 1; 75-89. ISSN 1735-8299.

[41] DU, K. y SWAMY, M. N. S., 2016. Simulated Annealing. En: *Search and Optimization by Metaheuristics*. Birkhäuser Basel; 29-36. ISBN 978-3-319-41192-7.

[42] Adaptive Algorithm. Wikipedia: la enciclopedia libre. 17 marzo 2017, 17:00 [consulta: 23 de septiembre de 2017]. Disponible en: https://en.wikipedia.org/wiki/Adaptive_algorithm

[43] Ant colony optimization. Scholarpedia. 28 marzo 2007, 16:15 [consulta: 23 de septiembre de 2017]. Disponible en: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

[44] Freight transport statistics - modal split. Eurosatat, Statistics Explained. 2 junio 2017, 11:10 [consulta: 16 de septiembre de 2017]. Disponble en: http://ec.europa.eu/eurostat/statistics-explained/index.php/Freight_transport_statistics_-_modal_split

[45] Drayage. Wikipedia: la enciclopedia libre. 1 octubre 2017, 15:23 [consulta: 5 de octubre de 2017]. Disponible en: https://en.wikipedia.org/wiki/Drayage

[46] Freight transported in containers - statistics on unitisation. Eurosatat, Statistics Explained. 18 octubre 2017, 10:37 [consulta: 16 de septiembre de 2017]. Disponble en: http://ec.europa.eu/eurostat/statistics-explained/index.php/Freight_transported_in_containers_-_statistics_on_unitisation

[47] Intermodal Transportation and Containerization. The geography of transport system [consulta: 16 de septiembre de 2017]. Disponble en: https://people.hofstra.edu/geotrans/eng/ch3en/conc3en/ch3c6en.html

[48] What is Logistics and Supply Chain Management Definition. SuplyChainOpz [consulta: 5 de octubre de 2017]. Disponible en: http://www.supplychainopz.com/2012/04/what-is-logistics-and-supply-chain-management.html

[49] VRP Flavors. Networking and Emerging Optimization [consulta: 19 de septiembre de 2017]. Disponible en: http://neo.lcc.uma.es/vrp/

[50] STERZIK, S. y KOPFER, H. 2013. A Tabu Search Heuristic for the Inland Container Transportation Problem. En: *Computers & Operations Research*. Vol. 40, Iss. 4; 953-962. ISSN 0305-0548.

[51] LAI, M., CRAINIC, T. G., DI FRANCESCO, M. y ZUDDAS, P, 2013. An heuristic search for the routing of heterogeneous trucks with single and double container loads. En: *Transportation Research, Part E: Logistics and Transportation Review*. Vol. 56; 108-118. ISSN 1366-5545.

[52] PAZOUR, J. A. y NEUBERT, L. C., 2013. Routing and Scheduling of Cross-Town Drayage Operations at J.B. Hunt Transport. En: *Interfaces*. Vol. 43, Iss. 2; 117-129. ISSN: 0092-2102.

[53] ZHANG, R., YUN, W.Y. y KOPFER, H, 2015. Multi-size container transportation by truck: modeling and optimization. En: *Flexible Services and Manufacturing Journal*. Vol. 27, Iss. 2-3; 403-430. ISSN: 1936-6582.

[54] FUNKE, J. y KOPFER, H., 2016. A model for a multi-size inland container transportation problem. En: *Transportation Research, Part E: Logistics and Transportation Review*. Vol. 89; 70-85. ISSN 1366-5545.

[55] VIDOVIC, M., POPOVIC, D., RATKOVIC, B. y RADIVOJEVIC, G., 2017. Generalized mixed integer and VNS heuristic approach to solving the multisize containers drayage problema. En: *International Transactions in Operational Research*. Vol. 24, Iss. 3; 583-614. ISSN: 1475-3995.

[56] HAN, L., 2016. *Metaheuristic algorithms for the vehicle routing problem with time window and skill set constraints* [en línea]. Dalhousie University [consulta: 24 de octubre de 2017]. Disponible en: https://dalspace.library.dal.ca/bitstream/handle/10222/72600/Han-Lu-MSc-IENG-Dec-2016.pdf?sequence=5

Anexo I: Optimización combinatoria y clase de problemas

La optimización es el acto de conseguir el mejor resultado posible bajo unas circunstancias dadas. Hoy en día, los ingenieros se encuentran con la tesitura de tomar decisiones constantemente y estas decisiones se pueden resumir en buscar minimizar el esfuerzo o maximizar el beneficio. Si definimos que el esfuerzo o el beneficio sigue una función con ciertas variables, se puede definir que la optimización es el proceso de encontrar las condiciones que dan el máximo o mínimo valor de una función.

Para los problemas de optimización no existe únicamente una metodología disponible de resolución si no que se han ido desarrollando una serie de métodos para resolver diferentes tipos de problemas a lo largo de la historia.

Se pueden dividir en aquellos problemas de optimización con variables continuas (un número infinito de soluciones factibles) o aquellos con variables discretas, llamados combinatorios (un número finito de soluciones factibles).

Optimización combinatoria.

La optimización combinatoria es una rama de la optimización de las matemáticas aplicadas fuertemente relacionada con la investigación operativa, la teoría algorítmica y la teoría de la complejidad computacional.

Los algoritmos de optimización combinatoria resuelven problemas que se creen difíciles en general, por la vía de explorar el, habitualmente grande, espacio de soluciones del citado problema. Los buenos algoritmos de optimización combinatoria lo logran por la vía de reducir el tamaño efectivo del espacio a buscar y explorando de modo eficiente dicho espacio.

Mediante el estudio de la teoría de la complejidad computacional es posible comprender la importancia de la optimización combinatoria. La teoría de la complejidad computacional es una rama de la teoría de la computación que se centra en la clasificación de los problemas computacionales de acuerdo con su dificultad inherente (si su solución requiere de una cantidad significativa de recursos computacionales, sin importar el algoritmo utilizado) y en la relación entre dichas clases de complejidad. Los algoritmos de optimización combinatoria se relacionan comúnmente con problemas de clase de complejidad NP-Hard (NP-Complejo).

Clases de complejidad.

La clase de complejidad P es el conjunto de problemas de decisión que pueden ser resueltos por una máquina determinista en tiempo polinomial, es decir, sus soluciones pueden ser encontradas en tiempo polinomial. Esta clase en general está compuesta de problemas relativamente "fáciles" para los cuales existen algoritmos eficientes. La clase P contiene todos aquellos problemas que han sido resueltos con algoritmos bien definidos y constructivos.

La clase de complejidad NP es el conjunto de problemas de decisión que pueden resolverse por una máquina no determinista en tiempo polinomial, es decir, sus soluciones pueden ser chequeadas de modo eficiente en tiempo polinomial. La clase NP incluye las versiones de problemas de decisión de

prácticamente todos los problemas de optimización combinatoria ampliamente estudiados.

- P es un subconjunto de NP.
- Se dice que un problema P es NP-Complejo si todos los miembros del NP polinomialmente se reducen a este problema.
- Un problema P se dice que es NP-Completo si P ∈ NP, y P es NP-Complejo.

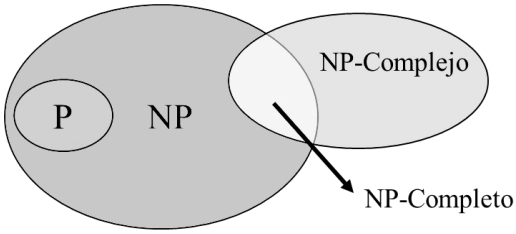


Figura 16. Clases de Complejidad.

Clases de problemas.

Algunos problemas clásicos de optimización combinatoria clasificados según su aplicación:

- Rutas: los problemas de rutas tratan de establecer un tour a recorrer para dar un determinado servicio donde se minimice la distancia recorrida. Un problema clásico es el del vendedor ambulante (TSP).
- Secuenciación: los problemas de secuenciación son, junto con los de rutas, la aplicación más popular en optimización combinatoria. Consisten en determinar el orden en el que se debe realizar una serie de tareas de un para minimizar su duración, teniendo en cuenta unas restricciones en los recursos.
- Corte y Empaquetado: son los problemas de corte son los ligados a reducir el consumo de materia prima que se vende o consume troceada.
- Horarios: hace referencia a los problemas de horarios son un problema clásico de optimización de combinatoria que tiene una gran cantidad de aplicaciones: diseño de horarios, repartos de guardias, calendarios de exámenes, repartir cargas de trabajo entre operarios con calendarios laborales, etc.
- Asignación: consisten en los problemas en los que se asignan recursos a tareas o tareas a agrupaciones. El equilibrado de líneas es un problema muy conocido en este grupo.

Travelling Salesman Problem (TSP).

El TSP se puede definir como el caso general o de partida para enunciar otros problemas combinatorios más complejos como el ruteo de vehículos y la programación de tareas dependientes del tiempo. En el caso del TSP se dispone de un solo vendedor que tiene que visitar todas las ciudades una vez en una sola ruta y se busca minimizar los costos. La métrica de coste puede definirse en términos de distancia, tiempo, etc. Y no suele haber un depósito.

La resolución del TSP se considera NP-compleja, es decir, la dificultad para encontrar la solución óptima aumenta exponencialmente, ya que el número de clientes también aumenta.

Las aplicaciones más populares de TSP son: distribución regular de bienes o recursos, de la ruta más corta de servicio al cliente, de autobuses, etc., También se puede encontrar en áreas como: aplicaciones en cristalografía, optimización de diagramas de cadenas o control de robots industrial.

Variaciones del TSP.

Teniendo en cuenta la clasificación de Davendra (2010), distingue:

- sTSP (Problema del vendedor ambulante simétrico): donde la distancia entre un par de ciudades es la misma tome el sentido que se tome. En este caso la matriz de coste sería simétrica.
- aTSP (Problema del vendedor ambulante asimétrico): donde la distancia puede ser diferente a la ida que a la vuelta e incluso no existir una de ellas. En este caso la matriz de costes no es simétrica.
- mTSP (Problema del vendedor ambulante múltiple): Dado un conjunto de nodos donde hay m vendedores ubicados en un único nodo (depósito). Los nodos restantes (ciudades) que se van a visitar son nodos intermedios. Luego, consiste en encontrar rutas para los vendedores, que empiezan y terminan en el depósito, de tal manera que cada nodo intermedio se visita exactamente una vez y el coste total se minimiza.

El mTSP es generalmente tratado como un problema de enrutamiento del vehículo relajado donde no hay restricciones en la capacidad. Por lo tanto, las formulaciones y los métodos de solución para el VRP son igualmente válidos para el mTSP si se asigna una capacidad a los vendedores.

Vehicle Routing Problem (VRP).

Este problema puede verse como la confluencia de dos conocidos problemas de optimización combinatoria. El primero, como se mencionaba anteriormente, el del viajero ambulante múltiple (mTSP) considerando la capacidad de cada vehículo como infinita (Applegate et al., 2006) y el de empaquetamiento en compartimentos (BPP, Bin Packing Problem) (Martello y Toth, 1990). Este último considera el problema de empaquetar objetos de diferentes volúmenes en un número dado de contenedores con un volumen especifico con el objetivo de minimizar el número de contenedores utilizados.

El VRP es uno de los problemas de distribución logística más conocidos y estudiados en el campo de la optimización combinatoria debido a su importancia en la cadena de suministro como ya se mencionaba en la sección anterior.

Variaciones del VRP.

Son muchas las variables a considerar para el problema VRP entre ellas está la información sobre el cliente, esta incluye información de la ubicación de los clientes y su demanda respectiva; las ventanas temporales en las cuales los clientes pueden ser servidos; tiempos de carga y descarga que requieren el producto; la flota de vehículos disponible para satisfacer la demanda; o las posibles relaciones precedentes entre clientes, es decir, cuando un cliente debe ser visitado antes que otro.

Otra variable que considerar es la información que se tiene del vehículo:

- Capacidad del vehículo.
- Depósito inicial donde se encuentra el vehículo cuando empieza la ruta y el deposito final donde debe dejarse cuando acaba la ruta.
- Los compartimientos, si el vehículo tiene diferentes compartimientos, donde puede llevar diferentes mercancías.
- Subconjuntos de arcos que el vehículo puede recorrer.

– Coste que conlleva el uso del vehículo, ya sea por distancia, tiempo, ruta, etc.

El objetivo de la función del problema VRP puede ser muy diferente según el enfoque que se quiera aplicar, los más comunes que se pueden encontrar son:

– Minimizar el coste del transporte global basado en la distancia total recorrida como los costes fijos asociados al uso de vehículos y conductores.

– Minimizar el número de vehículos necesario para satisfacer toda la demanda de los clientes.

– Conseguir la menor variación en el tiempo de recorrido y carga del vehículo.

– Minimizar las sanciones por un servicio de baja calidad.

Seguidamente se presenta un resumen en la Figura 14 de las variaciones más generalizadas del problema de enrutamiento del vehículo:

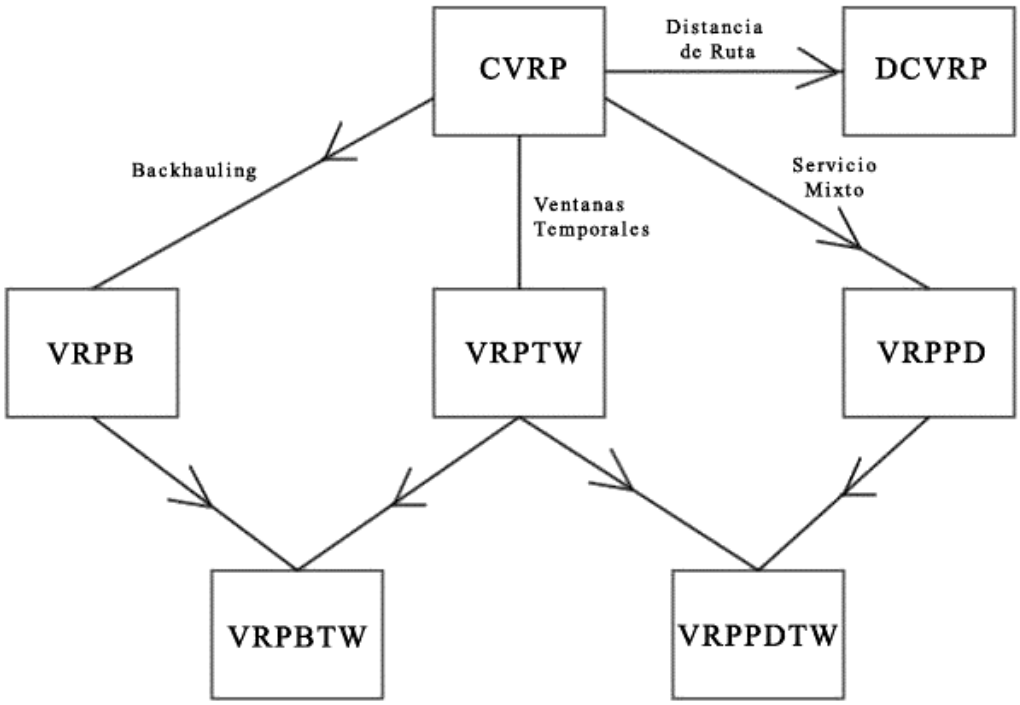


Figura 17. Mapa de los subtipos de VRP. Fuente: Toth y Vigo, 2002.

– Distance-Constrained Capacitated Vehicle Routing Problem (DCVRP)

DCVRP es una variante del CVRP donde se impone aparte de la capacidad del vehículo, también una restricción de distancia máxima.

– Vehicle Routing Problem with Time Windows (VRPTW) (Cordeau et al., 2002)

El VRPTW es un VRP con la variación que el servicio a un cliente solo puede comenzar y terminar dentro de la ventana de tiempo definida por el cliente.

– Vehicle Routing Problem with Pickup and Delivery (VRPPD) (Righini, 2000)

El VRPPD es un VRP en el que se contempla la posibilidad de que los clientes devuelvan algunos productos, es decir, haya recogida y entrega. Así que en VRPPD es necesario tener en cuenta que los bienes que se recojan deben encajar en el vehículo además de los que ya se encuentran en el vehículo para su entrega.

– Vehicle Routing Problem with Backhauls (VRPB) (Ralphs, Hartman y Galati, 2001); Jacobs-Blecha y Goetschalckx, 1992)

El VRPB es similar a VRPPD con la restricción que en el caso de VRPB todas las entregas para cada ruta deben ser completadas antes de que se realicen recogidas.

Otras variantes no mostradas anteriormente y que se han investigado son:

- Fleet Size and Mix Vehicle Routing Problem (FSMVRP):

Se considera igual que el CVRP, pero con una flota heterogénea de vehículos, es decir, los costos y capacidades son diferentes para cada vehículo.

- Split Delivery Vehicle Routing Problem (SDVRP) (Dror, Laporte y Trudeau, 1994; Archetti, Mansini y Speranza, 2001)

SDVRP es una relajación de la VRP en la que se permite que el mismo cliente puede ser servido por diferentes vehículos si se reduce los costos generales. Esta relajación es muy importante si los tamaños de los pedidos de los clientes son tan grandes como la capacidad de un vehículo.

- Stochastic Vehicle Routing Problem (SVRP) (Laporte y Louveaux, 1998)

El VRP estocástico es un VRP donde uno o varios componentes del problema son aleatorios. Tres tipos diferentes de SVRP: Clientes estocásticos, Demandas estocásticas, Tiempos estocásticos. En SVRP, se hacen dos etapas para obtener una solución. Se determina una primera solución antes de conocer las realizaciones de las variables aleatorias. En una segunda etapa, se puede tomar un recurso o acción correctiva cuando se conocen los valores de las variables aleatorias.

- Site-Dependent Vehicle Routing Problem (SDVRP)

Una flota fija de vehículos se utiliza para atender a un conjunto de clientes, pero existen dependencias de compatibilidad entre los sitios de los clientes y los tipos de vehículos. Algunos clientes con demandas extremadamente grandes pueden requerir vehículos grandes, mientras que algunos clientes localizados en áreas congestionadas pueden requerir vehículos pequeños o medianos. Otros clientes pueden ser atendidos por cualquier tipo de vehículo. El objetivo es seleccionar cuidadosamente un tipo de vehículo permisible para cada cliente.

- Arc Routing Problem (ARP)

Es uno de los problemas de enrutamiento que se centra en los arcos en una red. Este problema incluye el conocido "problema del cartero chino" (CPP). CPP es un problema de cubrir todos los arcos en una red mientras que minimiza el coste total de la distancia recorrida.

- VRP with Satellite Facilities (VRPSF) (Bard et al., 1997)

Se centra el uso de instalaciones satelitales para reabastecer vehículos durante una ruta. Cuando es posible, el reabastecimiento por satélite permite a los conductores continuar entregando hasta el cierre de su turno sin necesariamente regresar al depósito central.

- Multi-Depot VRP (MDVRP) (Hjorring, 1995)

Un MDVRP requiere la asignación de clientes a múltiples depósitos. Se define una flota de vehículos a cada depósito respectivamente. Cada vehículo proviene de un depósito, realiza el servicio a los clientes asignados y vuelve al mismo depósito al finalizar.

- Periodic VRP (PVRP) (Baptista, Oliveira y Zúquete, 2002)

En un VRP clásico típicamente el período de planificación es un solo día. En el caso del problema del enrutamiento del vehículo periódico (VRP), el período de planificación a m días.

Código de Búsqueda Tabú.

```
while time() < t_end:
        iteracion=iteracion+1
        neighbourhood=[]
        swap = []
        neighbourhood_value=[]
        for i in range(len(current_solution)):
            for j in range (i+1,len(current_solution)):
                ti=min(int(current_solution[i][4:]), int(current_solution[j][4:]))
                tj=max(int(current_solution[i][4:]), int(current_solution[j][4:]))
                if tabu_list[ti][tj] == 0 and current_solution[i] != current_solution[j]:
                    neighbour=current_solution[0:i]        +        [current_solution[j]]        +
current_solution[i+1:j] + [current_solution[i]] + current_solution[j+1:]
                    neighbourhood.append(neighbour)
                    neighbourhood_value.append(total_distanceNew(neighbour, dict_xy))
                    swap.append([int(current_solution[i][4:]), int(current_solution[j][4:])])
                    current_solution_value,          current_solution=min(zip(neighbourhood_value,
neighbourhood))
                    if current_solution_value < best_solution_value:
                        best_solution=current_solution[:]
                        best_solution_value=current_solution_value
                        break
            else:
                if time() < t_end:
                    continue  # executed if the loop ended normally (no break)
            break  # executed if 'continue' was skipped (break)
        current_solution_value,    swap,    current_solution=min(zip(neighbourhood_value,    swap,
neighbourhood))
        tabu_list[min(swap)][max(swap)]=tabu_size+1
        for i in range(len(tabu_list)-1):
            for j in range(i+1,len(tabu_list)):
                if tabu_list[i][j]>0:
                    tabu_list[i][j]=tabu_list[i][j]-1
return best_solution, best_solution_value
```

Código de Recocido Simulado.

```
while temp > min_temp:
        for i in range(L):
            neighbourhood=[]
            i = random.randint(0, len(current_solution)-1)
            for j in range (i+1,len(current_solution)):
                if current_solution[i] != current_solution[j]:
                        neighbour=current_solution[0:i]      +      [current_solution[j]]      +
current_solution[i+1:j] + [current_solution[i]] + current_solution[j+1:]
                        neighbourhood.append(neighbour)
            order=range(len(neighbourhood))
            random.shuffle(order)
            for i in range(len(neighbourhood)):
                new_solution_value=total_distanceNew(neighbourhood[order[i]], dict_xy)
                delta = new_solution_value - current_solution_value
                if delta<0:
                    current_solution_value = new_solution_value
                    current_solution = copy.copy(neighbourhood[order[i]])
                    best_solution_value = current_solution_value
                    best_solution = copy.copy(neighbourhood[order[i]])
                    break
                else:
                    a=random.random()
                    if a < math.exp(-delta/temp):
                        current_solution_value = new_solution_value
                        current_solution = copy.copy(neighbourhood[order[i]])
                        break
        temp = temp / (1+alpha * temp)
return best_solution
```

Código de la Función Objetivo.

```
for i in range(0, len(tareas_solucion)):
       if es_i40:
            if dict_xy[tareas_solucion[i-1]][2]==dict_xy[tareas_solucion[i]][1]:
                if totalCamion + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][7] < 480:
                    if i == len(tareas_solucion)-1:
                        totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
                        tareasCamiones[numeroCamion].append(solution[i])
                        totalDistancia = totalDistancia + dict_xy[tareas_solucion[i]][7] +
distance(dict_xy[tareas_solucion[i]])
                        else:
                        totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
                        tareasCamiones[numeroCamion].append(solution[i])
                        totalDistancia = totalDistancia + distance(dict_xy[tareas_solucion[i]])
                        C= 0
                else:
                    if i == len(tareas_solucion)-1:
                        totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                        numeroCamion = numeroCamion + 1
                        tareasCamiones[numeroCamion] = []
                        totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                        totalDistancia = totalDistancia + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6] + dict_xy[tareas_solucion[i]][7]
                        tareasCamiones[numeroCamion].append(solution[i])
                    else:
                        totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                        numeroCamion = numeroCamion + 1
                        tareasCamiones[numeroCamion] = []
                        totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                        totalDistancia = totalDistancia + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                        tareasCamiones[numeroCamion].append(solution[i])
                        C = 0
            elif i == 0:
                if i == len(tareas_solucion)-1:
                    totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6] + dict_xy[tareas_solucion[i]][7]
                    tareasCamiones[numeroCamion].append(solution[i])
                    totalDistancia = totalCamion
                else:
                    totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                    tareasCamiones[numeroCamion].append(solution[i])
                    totalDistancia = totalCamion
            else:
                valorMatriz = matrizdistancia[dict_xy[tareas_solucion[i-
1]][2]][dict_xy[tareas_solucion[i]][1]]
                distanciaPuntos = valorMatriz if valorMatriz > 0 else
matrizdistancia[dict_xy[tareas_solucion[i]][1]][dict_xy[tareas_solucion[i-1]][2]]

                if  totalCamion + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][7] + distanciaPuntos < 480:
                    if i == len(tareas_solucion)-1:
                        totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]]) +
distanciaPuntos
                        tareasCamiones[numeroCamion].append(solution[i])
                        totalDistancia = totalDistancia + dict_xy[tareas_solucion[i]][7] +
distance(dict_xy[tareas_solucion[i]]) + distanciaPuntos
                    else:
                        totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]]) +
distanciaPuntos
                        tareasCamiones[numeroCamion].append(solution[i])
                        totalDistancia = totalDistancia + distance(dict_xy[tareas_solucion[i]]) +
distanciaPuntos
                        C = 0
                else:

                    if i == len(tareas_solucion)-1:
                        totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                        numeroCamion = numeroCamion + 1
                        tareasCamiones[numeroCamion] = []
                        totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
```

```
                        totalDistancia = totalDistancia + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6] + dict_xy[tareas_solucion[i]][7]
                        tareasCamiones[numeroCamion].append(solution[i])
                    else:
                        totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                        numeroCamion = numeroCamion + 1
                        tareasCamiones[numeroCamion] = []
                        totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                        totalDistancia = totalDistancia + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                        tareasCamiones[numeroCamion].append(solution[i])
                        C = 0

        else:
            C = C + 20
            if C < CAP:
                if dict_xy[tareas_solucion[i-1]][2]==dict_xy[tareas_solucion[i]][1]:
                    if totalCamion + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][7] < 480:
                        if i == len(tareas_solucion)-1:
                            totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
                            tareasCamiones[numeroCamion].append(solution[i])
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i]][7] +
distance(dict_xy[tareas_solucion[i]])
                        else:
                            totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
                            tareasCamiones[numeroCamion].append(solution[i])
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]])
                    else:
                        if i == len(tareas_solucion)-1:
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                            numeroCamion = numeroCamion + 1
                            tareasCamiones[numeroCamion] = []
                            totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6] +
dict_xy[tareas_solucion[i]][7]
                            tareasCamiones[numeroCamion].append(solution[i])
                        else:
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                            numeroCamion = numeroCamion + 1
                            tareasCamiones[numeroCamion] = []
                            totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6]
                            tareasCamiones[numeroCamion].append(solution[i])
                elif i == 0:
                    if i == len(tareas_solucion)-1:
                        totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6] + dict_xy[tareas_solucion[i]][7]
                        tareasCamiones[numeroCamion].append(solution[i])
                        totalDistancia = totalCamion
                    else:
                        totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                        tareasCamiones[numeroCamion].append(solution[i])
                        totalDistancia = totalCamion
                else:
                    valorMatriz = matrizdistancia[dict_xy[tareas_solucion[i-
1]][2]][dict_xy[tareas_solucion[i]][1]]
                    distanciaPuntos = valorMatriz if valorMatriz > 0 else
matrizdistancia[dict_xy[tareas_solucion[i]][1]][dict_xy[tareas_solucion[i-1]][2]]
                    if  totalCamion + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][7] + distanciaPuntos < 480:
                        if i == len(tareas_solucion)-1:
                            totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]]) +
distanciaPuntos
                            tareasCamiones[numeroCamion].append(solution[i])
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i]][7] +
distance(dict_xy[tareas_solucion[i]]) + distanciaPuntos
                        else:
                            totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]]) +
distanciaPuntos
                            tareasCamiones[numeroCamion].append(solution[i])
```

```
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + distanciaPuntos
                    else:
                        if i == len(tareas_solucion)-1:
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                            numeroCamion = numeroCamion + 1
                            tareasCamiones[numeroCamion] = []
                            totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6] +
dict_xy[tareas_solucion[i]][7]
                            tareasCamiones[numeroCamion].append(solution[i])
                        else:
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                            numeroCamion = numeroCamion + 1
                            tareasCamiones[numeroCamion] = []
                            totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6]
                            tareasCamiones[numeroCamion].append(solution[i])
            else:

                if dict_xy[tareas_solucion[i-1]][1]==dict_xy[tareas_solucion[i]][1] and
dict_xy[tareas_solucion[i-1]][2]==dict_xy[tareas_solucion[i]][2]:
                    if totalCamion + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][7] < 480:
                        if i == len(tareas_solucion)-1:
                            totalCamion = totalCamion
                            tareasCamiones[numeroCamion].append(solution[i])
                            totalDistancia = totalDistancia
                        else:
                            totalCamion = totalCamion
                            tareasCamiones[numeroCamion].append(solution[i])
                            totalDistancia = totalDistancia
                            C = 0
                    else:
                        if i == len(tareas_solucion)-1:
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                            numeroCamion = numeroCamion + 1
                            tareasCamiones[numeroCamion] = []
                            totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6] +
dict_xy[tareas_solucion[i]][7]
                            tareasCamiones[numeroCamion].append(solution[i])
                        else:
                            totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-1]][7]
                            numeroCamion = numeroCamion + 1
                            tareasCamiones[numeroCamion] = []
                            totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                            totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6]
                            tareasCamiones[numeroCamion].append(solution[i])
                            C = 20
                else:
                    if dict_xy[tareas_solucion[i-1]][2]==dict_xy[tareas_solucion[i]][1]:
                        if totalCamion + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][7] < 480:
                            if i == len(tareas_solucion)-1:
                                totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
                                tareasCamiones[numeroCamion].append(solution[i])
                                totalDistancia = totalDistancia + dict_xy[tareas_solucion[i]][7]
+ distance(dict_xy[tareas_solucion[i]])
                            else:
                                totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
                                tareasCamiones[numeroCamion].append(solution[i])
                                totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]])
                                C = 0
                        else:
                            if i == len(tareas_solucion)-1:
                                totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-
1]][7]
                                numeroCamion = numeroCamion + 1
```

```
                                tareasCamiones[numeroCamion] = []
                                totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                                totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6] +
dict_xy[tareas_solucion[i]][7]
                                tareasCamiones[numeroCamion].append(solution[i])
                            else:
                                totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-
1]][7]
                                numeroCamion = numeroCamion + 1
                                tareasCamiones[numeroCamion] = []
                                totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                                totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6]
                                tareasCamiones[numeroCamion].append(solution[i])
                                C = 20
                    else:
                        valorMatriz = matrizdistancia[dict_xy[tareas_solucion[i-
1]][2]][dict_xy[tareas_solucion[i]][1]]
                        distanciaPuntos = valorMatriz if valorMatriz > 0 else
matrizdistancia[dict_xy[tareas_solucion[i]][1]][dict_xy[tareas_solucion[i-1]][2]]
                        if  totalCamion + distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][7] + distanciaPuntos < 480:
                            if i == len(tareas_solucion)-1:
                                totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
+ distanciaPuntos
                                tareasCamiones[numeroCamion].append(solution[i])
                                totalDistancia = totalDistancia + dict_xy[tareas_solucion[i]][7]
+ distance(dict_xy[tareas_solucion[i]]) + distanciaPuntos
                            else:
                                totalCamion = totalCamion + distance(dict_xy[tareas_solucion[i]])
+ distanciaPuntos
                                tareasCamiones[numeroCamion].append(solution[i])
                                totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + distanciaPuntos
                                C = 0
                        else:
                            if i == len(tareas_solucion)-1:
                                totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-
1]][7]
                                numeroCamion = numeroCamion + 1
                                tareasCamiones[numeroCamion] = []
                                totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                                totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6] +
dict_xy[tareas_solucion[i]][7]
                                tareasCamiones[numeroCamion].append(solution[i])
                            else:
                                totalDistancia = totalDistancia + dict_xy[tareas_solucion[i-
1]][7]
                                numeroCamion = numeroCamion + 1
                                tareasCamiones[numeroCamion] = []
                                totalCamion = distance(dict_xy[tareas_solucion[i]]) +
dict_xy[tareas_solucion[i]][6]
                                totalDistancia = totalDistancia +
distance(dict_xy[tareas_solucion[i]]) + dict_xy[tareas_solucion[i]][6]
                                tareasCamiones[numeroCamion].append(solution[i])
                                C = 20

return totalDistancia;
```