Tesis Doctoral

Metaheurísticas híbridas aplicadas al problema de ruteo de arcos capacitados

Martínez, Cristian Alejandro

2011

Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Metaheurísticas híbridas aplicadas al Problema de Ruteo de Arcos Capacitados

Tesis presentada para optar por el título de Doctor de la Universidad de Buenos Aires en el área de Ciencias de la Computación

Cristian Alejandro Martínez

Directores de tesis: Dra. Irene Loiseau - Dr. Mauricio G. C. Resende
Consejero de estudios: Dra. Irene Loiseau

Lugar de trabajo: Universidad de Buenos Aires - Universidad Nacional de Salta

Buenos Aires, 2011

Metaheurísticas híbridas aplicadas al Problema de Ruteo de Arcos Capacitados

Resumen

El Problema de Ruteo de Arcos Capacitados (CARP) es un problema de optimización combinatoria que consiste en satisfacer demandas de servicios/productos sobre determinadas calles de una red vial mediante una flota homogénea de vehículos, minimizando el costo total de recorrido involucrado. Ha sido aplicado a casos reales como recolección de residuos, mantenimiento de calles, lectura de medidores eléctricos, entre otros.

CARP es un problema de optimización combinatoria de tipo NP-Hard. A tal efecto, en la literatura se han propuesto algoritmos exactos y heurísticas. Los primeros, basados en su mayoría en las técnicas Branch and Bound y Cutting Plane, obtienen soluciones óptimas sobre instancias de datos de tamaño reducido. Los segundos, en general, alcanzan soluciones cercanas a las óptimas y a bajo costo computacional.

El objetivo de esta tesis es el desarrollo de algoritmos heurísticos que contengan características salientes de metaheurísticas tales como Honey Bee Mating Optimization (HBMO), Biased Random Key Genetic Algorithm (BRKGA), Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS), entre otras. Los resultados computacionales obtenidos por los algoritmos propuestos usando diferentes instancias de la literatura, muestran que los mismos son competitivos y robustos.

Palabras claves: BRKGA, GRASP, HBMO, Metaheurísticas, Problema de Ruteo de Arcos Capacitados, Ruteo de vehículos, Scatter Search, Tabu Search, VNS

Hybrid metaheuristics for the Capacitated Arc Routing Problem

Abstract

The Capacitated Arc Routing Problem (CARP) consists in determining a set of vehicle trips minimizing total travel cost, so that each demand over certain streets of a road network be served. Our motivation to deal with this problem is related to its application in real world cases such as street sweeping, urban waste collection and electric meter, reading just to mention a few.

Since its first formulation, several exact and approximative methods for this NP-Hard problem have been proposed. The former, generally based on Branch and Bound and Cutting Plane methods, reach optimal solutions in small data instances. The latter, obtain near optimal solutions using low CPU effort.

The objective of this work consists in proposing hybrid heuristic algorithms that include most important features from metaheuristics such as Honey Bee Mating Optimization (HBMO), Biased Random Key Genetic Algorithm (BRKGA), Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS), among others. The algorithms were tested with several well-known instances from the literature. The results obtained were competitive in terms of objective function value and required computational time.

Keywords: BRKGA, Capacitated Arc Routing Problem, GRASP, HBMO, Metaheuristics, Scatter Search, Tabu Search, Vehicle routing, VNS

Agradecimientos

A Irene Loiseau y Mauricio Resende, al grupo de Optimización Combinatoria, a la Comisión de Doctorado, a los docentes del Departamento de Computación de la UBA, a los colegas del Departamento de Informática de la UNSa, a Silvia Rodriguez, Daniel Morales y Ana María Aramayo, a Ana Haedo, María Elena Buemi, a mis alumnos y a todos los que colaboraron en el desarrollo de mi doctorado.

¡¡¡ Muchas gracias !!!

A mi familia, en especial a mi abuelo Félix y a mi hermano Enrique

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este capítulo se hará una introducción al problema de ruteo de vehículos elegido, indicándose la motivación del mismo, su aplicación y las técnicas usadas para alcanzar soluciones al mismo.

Muchos problemas de decisión existentes en el mundo real, en particular aquellos relacionados con la producción, ruteo de vehículos, logística, planificación, entre otros, pueden ser formulados como problemas de optimización [120]. En nuestro caso, nos centraremos en los de optimización combinatoria, los cuales son una clase dentro de los problemas de optimización. En los problemas de optimización combinatoria, las soluciones a los mismos se codifican usando variables discretas y el proceso de búsqueda de soluciones consiste en explorar el espacio de soluciones (del problema) representado mediante listas, conjuntos, matrices o grafos. Debido a las soluciones alcanzadas sobre diferentes problemas reales listados en la literatura como así también el avance en el estudio teórico-práctico en diferentes áreas dentro de la Investigación de Operaciones, la optimización combinatoria ha logrado un creciente interés motivado por los logros alcanzados.

Dentro de los problemas de optimización combinatoria, se encuentran los de ruteo de vehículos y en particular, nuestro problema de estudio. El problema de Ruteo de Arcos Capacitados (CARP) consiste en atender demandas sobre ciertas calles de una red vial a través de una flota homogénea de vehículos, los cuales inician y finalizan sus recorridos en un depósito. El objetivo del problema consiste en minimizar el costo total de recorrido de manera de atender todas las demandas solicitadas sin sobrecargar la capacidad de carga de los vehículos usados. Uno de los beneficios del estudio de este problema radica en su aplicación real y concreta en organismos públicos y empresas. En ambos casos,

estas organizaciones deben obtener soluciones rápidas y con el menor uso de recursos posible. Entre las aplicaciones de CARP o de sus extensiones relacionados con servicios públicos, podemos mencionar la recolección de residuos ([28], [34]), entre otros) y las tareas relacionadas con el mantenimiento de calles en época invernal ([93], [126]). El estudio de CARP y sus extensiones está motivado entonces, por la necesidad de ofrecer mejores y variadas alternativas de soluciones a las organizaciones que deben tratar con estas problemáticas.

Dado que el problema de estudio es NP-Hard [45], no se sabe aún que sea posible obtener la solución óptima de cualquier instancia del problema en tiempo polinomial. En la literatura, pueden encontrarse algoritmos heurísticos propuestos para CARP. La baja calidad de soluciones devueltas por los algoritmos heurísticos tradicionales, motivó el uso y desarrollo de nuevas propuestas algorítmicas. Definidas en [69] como técnicas generales usadas para guiar o controlar un método heurístico y específico del problema con el objetivo de mejorar el rendimiento o robustez de los métodos subordinados, las metaheurísticas han sido aplicadas a diferentes problemas de optimización (entre ellos, CARP), gracias a su versatilidad para adaptarse a nuevos problemas, como también su sencillez de implementación, calidad de las soluciones logradas y el bajo tiempo computacional asociado.

La tesis propuesta consiste en el desarrollo de varios algoritmos híbridos basados en metaheurísticas (entre otras GRASP, Variable Neighborhood Search y Tabu Search), los cuales obtienen soluciones rápidas y de calidad al problema de Ruteo de Arcos Capacitados (CARP). Las propuestas abordan el problema de optimización elegido desde diferentes ópticas, teniendo en común tres fases: de construcción de soluciones, de mejora y de escape de óptimos locales. Las diferentes pruebas computacionales realizadas sobre los algoritmos con diferentes instancias de prueba tomadas de la literatura, muestran cómo los algoritmos alcanzan soluciones eficientes a través del balance de las estrategias de intensificación y diversificación aplicadas.

El resto de la tesis se organiza de la siguiente manera: el Capítulo 2 contiene una introducción a la Optimización Combinatoria. A su vez, el Capítulo 3 describe las diferentes metaheurísticas usadas en las propuestas para el problema de estudio. El Capítulo 4 está dedicado al estado del arte sobre CARP. Los algoritmos propuestos para CARP son descriptos en el Capítulo 5. En tanto, en el Capítulo 6 se muestran los resultados de las pruebas de los algoritmos, usando diferentes instancias de la literatura. Finalmente,

el Capítulo 7 está dedicado a las conclusiones del trabajo realizado como así también a ciertas líneas de trabajo futuro.

Capítulo 2

Optimización Combinatoria

En este capítulo se realizará una introducción a la Optimización Combinatoria, definiendo conceptos importantes que servirán de base para el capítulo siguiente. Asimismo, se explicarán algunos problemas clásicos de optimización.

2.1. Introducción

Muchos problemas de optimización consisten en obtener un conjunto de valores (o configuración de variables del problema) que satisfagan condiciones intrínsecas del problema y a la vez cumplan con determinados objetivos. Los problemas que cumplen estas condiciones se agrupan en dos clases: aquellos cuyas soluciones deben codificarse con variables continuas y aquellas cuyas soluciones deben codificarse con variables discretas. Los primeros son problemas de *optimización continua* y los segundos, problemas de *optimización combinatoria*. Más formalmente:

Definición 1: Un problema de Optimización Combinatoria P=(X,D,f,R) se define como [64]:

- Un conjunto de variables discretas X=$\{x_1, x_2, \ldots, x_n\}$
- Un dominio D = $\{D_1, D_2, \ldots, D_n\}$ sobre el conjunto X, siendo D_i el conjunto de posibles valores que puede tomar x_i

- Un conjunto de restricciones R = $\{R_1, R_2, \ldots, R_m\}$ definidas sobre el problema P, tal que

$$R_i \subseteq D_1 \times D_2 \times \ldots \times D_n$$

- Una función objetivo f que debe optimizarse donde

$$f : D_1 \times D_2 \times \ldots \times D_n \rightarrow \Re^+$$

Dado un problema P, el espacio de soluciones S contiene todos los valores posibles para X sujeto a las restricciones R

$$S = \{s = \{(x_1, v_1), (x_2, v_2), \ldots, (x_n, v_n)\} | v_i \in D_i, \text{ s cumple las restricciones R}\}$$

El objetivo en los problemas de optimización consiste en encontrar una solución $s*$ $\in S$ tal que el valor de la función objetivo es un óptimo global.

A continuación, definiremos a un óptimo global $s*$ para un problema P=(X,D,f,R).

Definición 2: Un óptimo global $s*$ en el espacio de soluciones S es aquel que cumple:

$$\forall s \in S, f(s*) \geq (\leq) f(s)$$

Un óptimo global $s*$ es una solución (no necesariamente única) tal que cumple con todas las restricciones del problema y no existe otra solución s en el espacio de soluciones del problema, cuyo valor de la función objetivo sea mejor.

Como se verá más adelante, existen técnicas que aplicadas a un problema de optimización (en general), sólo alcanzan óptimos locales. Antes de dar una definición de los mismos, es necesario definir que es una estructura de vecindario.

Definición 3: Una estructura de vecindario es una función N: S $\rightarrow 2^s$ la cual asigna a toda solución $s \in$ S, un conjunto de vecinos N(s) $\subseteq S$. N(s) también es conocido como la *vecindad (o vecindario) de s.*

Una definición de solución óptima local de un problema P, es la siguiente:

Definición 4: Una solución óptima local (mínimo o máximo local, según la función objetivo del problema) respecto a un vecindario N es una solución tal que

$$\forall s \in N(\bar{s}) : f(s) \geq (\leq) f(\bar{s})$$

Análogamente, $\bar{s}$ es óptimo local estricto si $f(s) < (>) f(\bar{s}), \forall s \in N(\bar{s})$.

Si S es un conjunto finito numerable, entonces la Definición 1 corresponde a un problema de *Optimización Combinatoria*. Por otra parte, si $S = \Re^n$, el problema es de *Optimización Continua*.

Los problemas de Optimización Combinatoria han surgido de la práctica como también de disciplinas que hacen uso de métodos númericos como la inteligencia artificial, la investigación de operaciones, la bioinformática y el comercio electrónico [69]. Los problemas más conocidos se relacionan con la planeación, la logística, el diseño de hardware, el secuenciamiento de cadenas de ADN, la planificación horaria, entre otros. En estos problemas, en general se deben encontrar agrupamientos, ordenamientos o asignaciones óptimas de un conjunto discreto y finito de objetos que satisfagan determinadas condiciones o restricciones.

Una forma de resolver los problemas de optimización combinatoria es dada una instancia del problema, buscar todas las soluciones en el espacio de soluciones. Esto es, enumerar todas las soluciones posibles y luego elegir la mejor. A pesar que esta forma de resolver parece lógica y simple, en la mayoría de los casos, esta propuesta se vuelve rápidamente impracticable porque el número posible de soluciones crece de manera exponencial según el tamaño de la instancia. En algunos problemas de optimización combinatoria, el estudio y análisis de la estructura del problema y las características de las instancias del problema, permiten desarrollar algoritmos que alcanzan soluciones óptimas en menor tiempo computacional que la búsqueda exhaustiva. En otros problemas, estos mismos algoritmos no logran un rendimiento mucho mejor que la búsqueda exhaustiva [31].

Dada la importancia práctica de los problemas de optimización combinatoria, se han desarrollado diferentes algoritmos con el objeto de obtener soluciones a los mismos. Estos algoritmos se clasifican en *algoritmos exactos* y *algoritmos heurísticos*.

Los algoritmos exactos garantizan obtener la solución óptima a un problema de optimización en un tiempo computacional dependiente de la instancia del problema, siendo la misma de tamaño finito. Sin embargo, para muchos problemas de tipo NP-Hard, los

algoritmos exactos necesitan un tiempo exponencial para alcanzar las soluciones óptimas. Incluso para instancias pequeñas, éstos algoritmos podrían consumir demasiado tiempo computacional lo cual puede resultar poco práctico. Consideremos el Problema de Asignación Cuadrática (QAP). Actualmente, los algoritmos exactos de la literatura han alcanzado soluciones óptimas sobre algunas instancias QAP de dimensión no superior a 90 [81]. Al respecto, Fischetti et al. [41] han reportado resultados óptimos para varias instancias de la literatura. Para sus pruebas, utilizaron el software Cplex versión 12.2 y un procesador Quad-Core Intel Xeon 3.2Ghz con 16Gb de RAM. Por ejemplo, para la instancia *Tai64c* (de dimensión 64), el resultado óptimo fue alcanzado en aproximadamente 5 horas.

Los algoritmos heurísticos obtienen soluciones cercanas a la solución óptima, pero a bajo costo computacional. Se clasifican en algoritmos constructivos y algoritmos de búsqueda local. Los algoritmos constructivos generan soluciones (a un problema) agregando elementos en forma iterativa a una solución parcial, hasta que la misma es completada. En cambio, los algoritmos de búsqueda local examinan el vecindario de una solución inicial con el objetivo de alcanzar el óptimo local. En el Capítulo 3, ambos algoritmos serán explicados con mayor detalle.

A continuación, daremos una introducción a algunos problemas clásicos de optimización combinatoria.

2.2. Problemas de Optimización Combinatoria

2.2.1. Problema de Asignación Cuadrática

El Problema de Asignación Cuadrática (Quadratic Assignment Problem o QAP) [3] es un problema importante tanto a nivel teórico como práctico. Dado un conjunto N={1,2, ..., n} y las matrices F=$\{f_{ij}\}$, D=$\{d_{ij}\}$, C=$\{c_{ij}\}$ de orden n, QAP consiste en encontrar una permutación φ del conjunto N tal que sea mínima la función objetivo

$$f(\varphi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)} + \sum_{i=1}^{n} c_{i\varphi(i)}$$

Donde:

- f_{ij} es el flujo entre instalaciones (facilities) i y j
- d_{ij} es la distancia entre lugares (locations) i y j
- $\varphi(i)$ es el lugar asignado a la instalación i
- $c_{i\varphi(i)}$ es el costo de asignar la instalación i a algún lugar

Introducido originalmente en los '50 como un modelo matemático para la ubicación de actividades económicas indivisibles (o no-separables), QAP puede describirse como el problema de asignar un conjunto de instalaciones/servicios (facilities) a un conjunto de lugares/ubicaciones (locations), conocidas las distancias entre lugares y los flujos entre instalaciones. El objetivo del problema es ubicar cada instalación en exactamente un lugar tal que la función objetivo (antes indicada) es mínima.

QAP tiene otras aplicaciones tales como diseño de circuitos eléctricos, asignación de procesadores en ambientes distribuidos, ordenamiento de datos interrelacionados sobre cintas magnéticas, balanceo de turbinas, entre otros. Es conocido que el problema es NP-Hard [109] y en general, se ha probado la dificultad que presenta alcanzar soluciones óptimas para las instancias del problema. QAPLIB [17], es una librería que posee instancias del problema de diferentes tamaños y complejidades.

2.2.2. Problema del Viajante

El Problema del Viajante (Travelling Salesman Problem o TSP) ([104], [121]) es uno de los problemas de optimización combinatoria más estudiados. De manera general, dado un grafo ponderado, conexo y dirigido, TSP consiste en encontrar el camino cíclico más corto tal que todo nodo del grafo sea visitado exactamente una vez. A los efectos de definir formalmente TSP, daremos unas definiciones previas.

Definición 5: Camino y ciclo Hamiltoniano
Sea G=(V,E,w) un grafo ponderado, conexo y dirigido donde V=$\{v_1, v_2, \ldots, v_n\}$ es el conjunto de n=|V| vértices, E $\subseteq V \times V$ el conjunto de aristas (arcos), y una función w: E $\rightarrow \Re^+$ que asigna toda arista $e \in$ E un peso w(e).
Un *camino* en G es una lista $(u_1, u_2, \ldots, u_k)$ de vértices $u_i \in$ V $\forall i = 1, \ldots, k$, tal que cualquier par (u_i, u_{i+1}) $\forall i = 1, \ldots, k-1$, es una arista de G.
Un camino cíclico en G es un camino tal que el primer y el último vértice coinciden.

Un *ciclo hamiltoniano* (o circuito) es un camino cíclico en G que visita todos los vértices del grafo exactamente una vez (excepto el vértice inicial); es decir, p=$(u_1, u_2, \ldots, u_n, u_1)$ es un ciclo hamiltoniano en G sii n=|V| y $\{u_1, u_2, \ldots, u_n\}$=V.

Definición 6: Peso (costo) de un camino
Dado un grafo G ponderado, conexo y dirigido, y un camino p=$(u_1, u_2, \ldots, u_k)$ en G, el peso de un camino w(p) se define como $w(p) = \sum_{i=1}^{k-1} w((u_i, u_{i+1}))$.

Puede definirse entonces [69]: Dado un grafo G ponderado, conexo y dirigido, el Problema del viajante (TSP) consiste en encontrar un ciclo hamiltoniano en G con peso mínimo.

El interés en este problema de tipo NP-Hard [45] se debe entre otros, a que es un problema conceptualmente simple de explicar y entender, didácticamente fácil de enseñar en cursos de grado, con aplicaciones reales concretas (generalmente relacionadas con problemas de ruteo de vehículos) y continuamente se proponen diferentes variaciones al problema (Múltiple TSP, CVRP, VRPTW, etc). Recientemente [9], se han alcanzado soluciones óptimas a instancias TSP con más de 85000 nodos.

2.2.3. Problema de la Mochila Múltiple

El Problema de la Mochila Múltiple (Multiple Knapsack Problem o MKP)[15] puede definirse así:

Maximizar $\sum_{j \in N} p_j \cdot x_j$
sujeto a

$$\sum_{j \in N} w_{ij} \cdot x_j \leq c_i, \forall i \in M$$
$$x_j \in \{0, 1\}, \forall j \in N$$

Donde:

- n es el número de elementos (items)
- m es el número de mochilas (knapsack) disponibles
- N={1,2,..., n} y M={1,2,..., m}
- $p_j \geq 0$ es el beneficio del elemento j-ésimo, $\forall j \in N$

- $w_{ij} \geq 0$ son los pesos del elemento j-ésimo en las mochilas, $\forall i \in M, \forall j \in N$
- $c_i \geq 0$ es la capacidad de la mochila i-ésima, $\forall i \in M$

El objetivo de este problema NP-Hard [45] es encontrar un subconjunto de elementos que maximice el beneficio total, tal que los elementos seleccionados no sobrepasen la capacidad de carga de las mochilas.

Propuesto por primera vez en los '50 para problemas de asignación de presupuestos [100], MKP ha sido aplicado a problemas de carga de bultos en aviones [118], problemas de corte [111], selección de proyectos [43], entre otros.

Capítulo 3

Metaheurísticas

En este capítulo, detallaremos las técnicas heurísticas para luego dar un panorama general de las metaheurísticas. Después, describiremos las usadas en las propuestas realizadas para CARP.
De cada una de ellas, se detallarán los principales aspectos con el objeto de entender el funcionamiento y el aporte que proveen en la obtención de soluciones a problemas de optimización.

3.1. Introducción

Una técnica sencilla y lógica que puede usarse para resolver un problema de optimización, consiste en examinar todas las soluciones factibles del problema, evaluarlas usando la función objetivo y luego elegir la mejor. Esta técnica llamada de *enumeración completa* (también conocida como fuerza bruta), a pesar de que es aplicada en diferentes problemas computacionales, puede resultar impracticable incluso en instancias de tamaño medio debido a la gran cantidad de soluciones posibles [121].

Dentro de la clasificación de algoritmos exactos, Branch and Bound ([94], [123]) es una técnica eficiente de enumeración completa de soluciones, la cual construye un árbol de soluciones del problema y que, para evitar una búsqueda exhaustiva en el mismo para obtener la solución óptima, reduce esta tarea a través de podas. Esta técnica como otras tales como Branch and Price ([12], [27]) y Branch and Cut ([97], [101], [123]), han sido aplicadas exitosamente a diferentes problemas de optimización combinatoria. Sin

embargo, a medida que aumenta el tamaño de las instancias de los problemas, el tiempo computacional para alcanzar soluciones óptimas crece en forma exponencial.

Dada la dificultad subyacente en los problemas de optimización de tipo NP-Hard, la investigación en técnicas heurísticas ha cobrado relevancia en los últimos 40 años. Una *heurística* es una técnica (consistente de una regla o un conjunto de reglas) que busca (y posiblemente encuentra) *buenas soluciones*, a un costo computacional razonable. En cierto sentido, una heurística es *aproximada* ya que alcanza buenas soluciones a bajo esfuerzo computacional, pero no asegura la optimalidad [120]. Las heurísticas son reglas, criterios, métodos o principios para decidir entre diferentes cursos de acción, cuál es el más efectivo a los efectos de cumplir con un objetivo (como por ejemplo, lograr una solución factible para un problema o mejorar una solución).

A pesar que los algoritmos heurísticos no aseguren la obtención de soluciones óptimas, puedan no precisar la brecha entre soluciones devueltas y la óptima, o incluso, no aseguren la factibilidad de las mismas, éstas pueden alcanzarse en un tiempo razonable y a igual calidad que las obtenidas por métodos exactos.

Los algoritmos heurísticos se clasifican en algoritmos *constructivos* y algoritmos de *búsqueda local*. A continuación, estos serán definidos.

Definición 7: Algoritmo constructivo [31]
Un algoritmo constructivo construye una solución a un problema de optimización combinatoria de manera incremental. Paso a paso, agrega componentes (o elementos) a la solución parcial hasta obtener una completa.

La forma en que los elementos son agregados a la solución en construcción, puede ser al azar o bien mediante la aplicación de alguna regla.

En el Cuadro 3.1 se muestra un pseudo-código de un algoritmo constructivo goloso.

Cuadro 3.1: Algoritmo constructivo goloso.

```
procedure greedyConstruction(problemData)

solution={}
while solution is not completed do
  element = greedySelection(solution,problemData)
  solution = solution + {element}
end while
return solution
end procedure
```

Por ejemplo, un algoritmo constructivo para el Problema del Viajante (TSP), construye una solución agregando nodos del grafo (es decir, ciudades) de manera iterativa hasta que todos estén incluidos en la misma. Dicha inserción puede ser en forma aleatoria (tal como propone Inserción Aleatoria o Random insertion), la que menos incrementa el costo del recorrido total (Inserción más Económica o Cheapest Insertion), aquella cuya distancia a algún nodo en la solución sea máxima (Inserción más Distante o Farthest Insertion), entre otras alternativas posibles.

Definición 8: Algoritmo de búsqueda local [31]
Un algoritmo de búsqueda local se basa en la exploración iterativa del vecindario de soluciones, para de esta manera, mejorar la calidad de la solución actual mediante cambios locales definidos por una estructura de vecindario.

Los algoritmos de búsqueda local iterativamente modifican soluciones (obteniendo de esta manera *soluciones vecinas*) a un problema de optimización, mediante cambios (es decir, mediante movimientos de elementos de la solución). Una estructura de vecindario define implícitamente los posibles cambios a aplicar a una solución. Los cambios son previamente evaluados usando alguna función de optimización, para de esta manera conducir el proceso de búsqueda.

En el Cuadro 3.2 se muestra un pseudo-código de un algoritmo de búsqueda local.

Cuadro 3.2: Algoritmo de búsqueda local.

```
procedure localSearch(prevSolution)

best=prevSolution
while improvement is posible do
  neighbor = getFromNeighborhood(best)
  if f(neighbor) < f(best) then
    best = neighbor
  end if
end while
return best
end procedure
```

Se han propuesto diferentes algoritmos de búsqueda local para el Problema del Viajante. Croes [23] propuso el algoritmo 2-OPT, el cual dada una solución inicial, elimina dos aristas y luego reconecta los 4 nodos involucrados de forma distinta para de esta manera obtener una nueva solución. En el caso del algoritmo 3-OPT propuesto por Lin ([82], [83]), la obtención de una nueva solución a partir de una inicial es a través de la eliminación de tres aristas y la posterior reconexión de forma distinta (a la inicial) y al menor costo posible.

Hace poco más de 20 años, ha surgido en la literatura una nueva clase de algoritmo que guía a métodos heurísticos subordinados, conocido como *metaheurísticas*. Su aparición se debió a dos situaciones habituales al usar heurísticas: la falta de diversidad de soluciones provistas por los algoritmos constructivos y la baja calidad de los óptimos locales devueltos por los algoritmos de búsqueda local. Las metaheurísticas mejoran a las técnicas heurísticas debido a que integran diferentes estrategias de búsqueda para explorar el espacio de soluciones de manera más eficiente y efectiva. Su aplicación a diferentes problemas de optimización combinatoria, ha sido extensa y exitosa ([31], [50], [64], [106], entre otros).

Trataremos de definir una metaheurística de manera formal.

Definición 8: Metaheurística [120]
Es un proceso iterativo maestro que guía y modifica la operación de heurísticas subordinadas, para producir soluciones de alta calidad a cualquier problema de optimización.

Existen diferentes características que poseen las metaheurísticas:

- Son estrategias que con pocos cambios, son aplicables a una gran variedad de problemas de optimización.
- Para conducir la búsqueda a soluciones de alta calidad, hacen uso de información específica del problema y/o aprenden a lo largo de su ejecución.
- Pueden subordinar heurísticas constructivas y/o de búsqueda local.
- Incorporan mecanismos para evitar estancamiento en óptimos locales.

En la literatura, se han propuesto variadas y exitosas metaheurísticas tales como Tabu Search (TS), Variable Neighborhood Search (VNS), Ant Colony Optimization (ACO), Iterated Local Search (ILS), Greedy Randomized Adaptive Search Procedure (GRASP), Simulated Annealing (SA), entre otras ([50], [64], [69], [85], [90], [106]).

A continuación, haremos una clasificación de las mismas.

3.2. Clasificación

En base a las características de cada metaheurística, es posible obtener diferentes clasificaciones. Listaremos algunas de ellas:

- Origen. Según la fuente de inspiración de la metaheurística, se clasifican en *inspiradas en la naturaleza* y *no inspiradas en la naturaleza*. En el primero grupo se encuentran Algoritmos Genéticos [52], Ant Colony Optimization [31], Honey-bee Mating Optimization [1], entre otras. En el segundo, podemos mencionar a Tabu Search [50], Iterated Local Search [85], Guided Local Search [121], entre otras.

- Manejo de soluciones. Según el número de soluciones que manejan al mismo tiempo durante el proceso de búsqueda, se clasifican en *basadas en poblaciones* como por ejemplo Algoritmos Genéticos, Honey-bee Mating Optimization y Biased Random Key Genetic Algorithm [35], y las *basadas en búsqueda simple* como Iterated Local Search, Variable Neighborhood Search [64], Simulated Annealing [73] y Tabu Search.

- Uso de memoria. Otra forma de clasificar metaheurísticas se relaciona con el uso (o no) de memoria para registrar el proceso de búsqueda. Algunas metaheurísticas que no incorporan mecanismos de memoria son Variable Neighborhood Search, Simulated Annealing y GRASP [37]. En cambio, Tabu Search y Ant Colony Optimization incorporan mecanismos de memoria de corto y largo plazo.

- Estructura de vecindario. En general, las metaheurísticas usan una única estructura de vecindario (Simulated Annealing, Iterated Local Search, Ant Colony Optimization, entre otros). En cambio, Variable Neighborhood Search (VNS) usa diferentes estructuras de vecindarios permitiendo así diversificar la búsqueda de soluciones y evitar estancamiento en óptimos locales mediante mecanismos eficientes.

- Movimientos. La forma en que construyen una solución o que exploran el vecindario de una solución, puede ser de manera aleatoria o determinística. Ant Colony Optimization y GRASP construyen soluciones aplicando reglas probabilísticas. Simulated Annealing acepta una solución vecina si es de mejor calidad (que la mejor solución hasta el momento) o si se cumple una condición probabilística. Iterated Local Search elige un vecino (de una solución) en forma aleatoria. En tanto, Tabu Search (en su versión simple) construye soluciones usando un memoria de corto plazo que evita incluir elementos (a la misma) que fueron recientemente utilizados.

A continuación, describiremos las metaheurísticas usadas en nuestras propuestas para CARP.

3.3. Honey-Bee Mating Optimization

La metaheurística Honey-bee Mating Optimization (HBMO)[1] está inspirada en la conducta social de las abejas, hormigas y avispas, y el proceso reproductivo de las abejas.

Los insectos antes nombrados, se caracterizan por el trabajo cooperativo de los insectos adultos en el cuidado de las crías y la construcción del nido, la convivencia armónica entre individuos de diferentes generaciones y la división jerárquica de trabajo. A pesar de ser una novel metaheurística, HBMO se ha aplicado a problemas de particionamiento [75], agrupamiento [36], automatización [24], ingeniería [2], entre otros.

Una colonia de abejas se forma por una abeja reina, miles de zánganos y abejas obreras, las cuales conviven en una misma colmena. La reina es el miembro más importante debido a que de ella depende la supervivencia de la colonia. Durante su vuelo, los zánganos la persiguen con el propósito de reproducirse con ella. En cada copulación, el esperma del zángano es almacenado en el cuerpo de la abeja reina (espermateca). La única función que tienen los zánganos en la colonia es la copulación con la reina. Luego de esto, mueren. Una vez finalizado su vuelo, la abeja reina retorna a la colmena y utiliza el esperma almacenado en su cuerpo para fertilizar los huevos. Las crías se forman de huevos fertilizados y no fertilizados. Las abejas obreras son las encargadas del cuidado de las crías. De los huevos fertilizados, surgirá una potencial reina o nuevas obreras mientras que de los no fertilizados, surgirán nuevos zánganos. Básicamente, un algoritmo HBMO para cualquier problema de optimización, debe contener las siguientes tareas [62]:

- El algoritmo comienza con el vuelo reproductivo de la abeja reina (mejor solución). Un zángano (elegido al azar) copula con la reina si lo considera agradable (buena solución) o si se cumple una condición probabilística (condición similar a la adoptada en Simulated Annealing [32]).
- Generación de nuevas crías (soluciones) mediante el cruzamiento genético entre zánganos y la abeja reina.
- Uso de abejas obreras (algoritmos de búsqueda local) para el cuidado (mejora) de las crías.
- Reemplazo de la abeja reina (mejor solución) por alguna cría (si existe) más fuerte.

Los principales pasos en HBMO son ilustrados en la Fig. 3.1 [36].

A continuación, describiremos las principales características de HBMO.

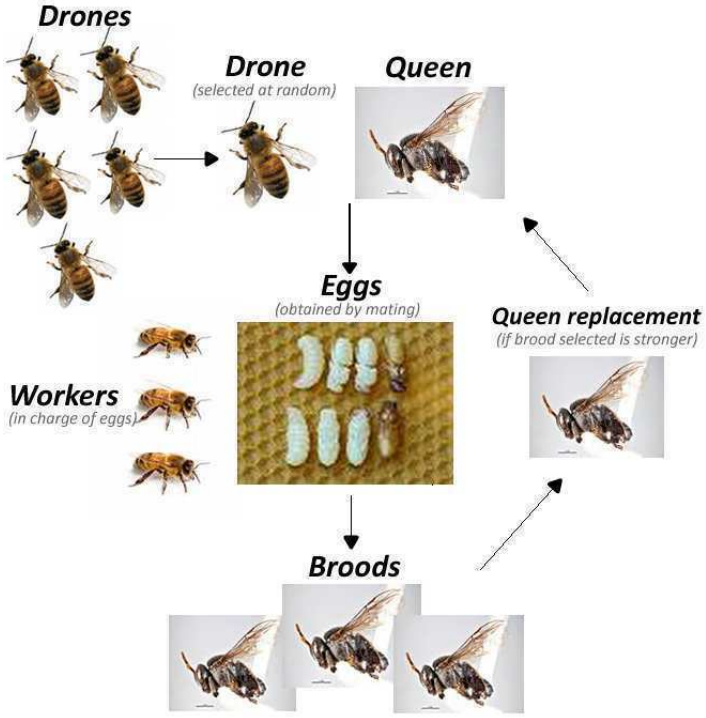

Figura 3.1: HBMO: fases del algoritmo.

3.3.1. Vuelo reproductivo de la abeja reina

Durante un vuelo de la reina, un zángano copula con la reina si se cumple la siguiente condición probabilística:

$$prob(Q, D) = e^{\frac{-\Delta(f)}{S(t)}} \tag{3.1}$$

Donde:

- prob(Q, D) es la probabilidad que un zángano D copule con la reina Q
- $\Delta(f)$ es el error absoluto entre los valores de función objetivo (también conocido como *fitness*) del zángano y de la reina
- S(t) es la velocidad de la reina en el tiempo t. La velocidad de la reina decrece durante su vuelo debido a la copulación con los zánganos; en el tiempo (t+1), se obtiene mediante la siguiente fórmula:

$$s(t+1) = \alpha * s(t) \tag{3.2}$$

siendo α un factor $\in$ (0,1).

A través de esta condición, se desprende que un zángano copula con la reina cuando es similar a ella en términos de función objetivo (es atractivo para la reina) ó cuando la abeja reina recién comienza su vuelo reproductivo.
Luego que un zángano copula con la reina, debe morir. La supervivencia del zángano no es una característica mencionada en la literatura sobre HBMO.

3.3.2. Cruzamiento entre zángano y abeja reina

Tal como se indica en [1], [24], [36], [75], si el zángano elegido cumple con la condición probabilística (3.1), éste se cruza con la abeja reina y el esperma del zángano se guarda en la espermateca de la reina. Luego, usando dicho esperma y el código genético de la reina, las abejas obreras generarán las nuevas crías por cruzamiento y mutación.
La fase de cruzamiento entre zánganos y la abeja reina podría involucrar la aplicación de diferentes operadores de cruzamiento, lo cual podría generar una o varias crías. Para más detalles sobre operadores, se recomiendan [32], [52], entre otros.

3.3.3. Uso de abejas obreras

Como se mencionó anteriormente, en una colonia de abejas, la tarea de las abejas obreras es la de alimentar y cuidar a las nuevas crías.
Las heurísticas cumplen aquí el rol de las abejas obreras. Aunque en la literatura sobre HBMO, se han aplicado operadores de cruzamiento para simular el comportamiento de las abejas obreras, es posible y recomendable siguiendo los lineamientos de los Algoritmos Memeticos ([76], [90]), la incorporación de heurísticas que mejoren las soluciones luego de un cruzamiento.
Finalmente, es importante destacar que la selección de la heurística que actuará sobre la nueva solución, depende del nivel de mejora acumulada que ésta ha logrado sobre soluciones anteriores. Es decir, la aplicación de las heurísticas depende del éxito de las mismas en la obtención de mejores soluciones.

3.3.4. Reemplazo de la abeja reina

La reina actual de la colonia es reemplazada cuando alguna cría es más fuerte.
Luego que las nuevas soluciones (crías) han sido mejoradas por las heurísticas (abejas obreras), éstas son ordenadas según su valor de función objetivo (o fitness). La nueva mejor solución (mejor cría) reemplaza a la mejor solución hasta el momento (abeja reina) si el valor de función objetivo de la primera es mejor que la segunda. Luego, las nuevas soluciones restantes (crías) son eliminadas y el algoritmo nuevamente itera (comienza un nuevo vuelo de la reina) hasta que se cumpla la condición de parada del algoritmo.
Es importante destacar que la colonia de abejas puede tener más de una abeja reina. Por lo tanto y según la situación anterior, el reemplazo de la mejor cría puede ser por alguna reina de la colonia.

3.3.5. Algoritmo HBMO

El siguiente algoritmo [1], muestra el funcionamiento general de la metaheurística HBMO a cualquier problema de optimización usando en este caso, varias abejas reinas. Los parámetros que necesita son:

- Número de reinas: población de reinas de la colonia. Disponer de varias buenas soluciones (óptimos locales) ayuda a mejorar el proceso de diversificación.
- Número de vuelos reproductivos: indica la cantidad de veces que una reina vuela para cruzarse con los zánganos. Es decir, es la cantidad de veces que el algoritmo itera.
- Número de zánganos: población de zánganos que puede cruzarse con la abeja reina. Este parámetro puede descartarse, si se genera un nuevo zángano siempre que la reina siga volando (es decir, mientras tenga energía).
- Número de abejas obreras: si bien no puede considerarse directamente un parámetro ya que de antemano, se conocen las heurísticas a aplicar, éste se considera como el número máximo de heurísticas a aplicar sobre las nuevas soluciones.
- α: es el factor de reducción de velocidad/energía. Siguiendo los mismos lineamientos de Simulated Annealing, el valor usado debe ser cercano a 1 para que la velocidad de

la reina no se reduzca bruscamente. Si eso ocurriese, la reina tendría pocas chances de cruzarse con varios zánganos.

- Energía: valor inicial de energía de la reina. La bibliografía sobre HBMO recomienda un valor aleatorio. No obstante, es importante destacar que este valor debería condicionar al parámetro anterior a los efectos de permitir varios cruzamientos entre reina y zánganos.

Cuadro 3.3: Algoritmo HBMO.

```
procedure HBMO(#queens, #fligths, #drones, #workers,α, energy)

 initialize workers
 randomly generate the queens
 apply local search to get a good queen
 for a pre-defined maximum number of mating-fligths
   for each queen in the queen list
     initialize energy, speed and position
     the queen moves between states and probabilistically chooses drones
     if a drone is selected then
       add its sperm to the queen's spermatheca
     end if
     update the queen's internal energy and speed
   end for each
   generate broods by crossover and mutation
   use workers to improve the broods
   update worker's fitness
   while the best brood is better than the worst queen
     replace the least-fittest queen with the best brood
     remove the best brood from the brood list
   end while
 end for
 return best queen

end procedure
```

3.4. Greedy Randomized Adaptive Search Procedure

La metaheurística GRASP ([37], [38]) propuesta por Resende y Feo, encuentra soluciones aproximadas (de buena calidad, no precisamente óptimas) a problemas de optimización combinatoria. Tiene como premisa la generación de soluciones iniciales diversas y de calidad, que le permitan a los algoritmos de búsqueda local alcanzar mejores óptimos locales [106].

GRASP es un proceso iterativo. Cada iteración consiste de dos fases. Una *fase constructiva*, en la cual una solución es construída y una *fase de búsqueda local*, en la cual un óptimo local en el vecindario de la solución construída previamente, es obtenido. Una vez cumplida la condición de parada, el algoritmo retorna la mejor solución alcanzada.

El principio a cumplir en los algoritmos basados en GRASP es la generación de muchas soluciones iniciales de buena calidad a través de una fase constructiva de soluciones, para luego mediante una fase de mejora, aumentar la probabilidad de encontrar óptimos locales de calidad [120]. GRASP se ha aplicado a diferentes problemas de optimización tales como problemas de agrupamiento ([107], [108]), planificación de máquinas [4], asignación cuadrática [81], árbol mínimo capacitado [25], de corte [6], ruteo de vehículos [102], entre otros.

La simplicidad, rapidez y calidad de las soluciones son características comunmente encontradas en los algoritmos basados en GRASP. El Cuadro 3.4 muestra un pseudocódigo GRASP [105] de alto nivel.

3.4.1. Fase Constructiva

En la fase de construcción de soluciones, cada solución es iterativamente construída, agregando un elemento candidato a la vez. Un elemento candidato es aquel que puede ser elegido para formar parte de la solución parcial (en construcción). Para determinar cuál elemento candidato debe seleccionarse, se usa una función golosa (greedy o miope) la cual mide la contribución que hace el elemento a la solución parcial.

Para evitar que la selección sea miope y determinística, el elemento a incorporar a la solución se elige en forma aleatoria. Sin embargo, esta selección aleatoria no se realiza con todos los elementos candidatos posibles, sino con los mejores. La Lista Restringida de

Cuadro 3.4: Algoritmo GRASP.

```
procedure GRASP(#iterations)

f* ← ∞
for i=1 ... #iterations do
   x ← GreedyRandomizedConstruction()
   x ← LocalSearch(x)
   if f(x) < f* then
      f* ← f(x)
      x* ← x
   end if
end for
return x*
end procedure
```

Candidatos (Restricted Candidate List o RCL), es una lista con buenos elementos candidatos según sus valores obtenidos con la función miope usada. Entonces, el elemento a agregar a la solución es elegido al azar desde esta lista, siendo la misma de tamaño fijo o de tamaño variable dependiendo de los elementos cuyos valores de función miope se encuentren dentro de un intervalo (umbral) previamente determinado. Esta técnica permite que se obtengan diferentes soluciones en cada iteración de GRASP, la cual trabajando con funciones golosas apropiadas, permiten que las soluciones también sean de calidad.
En general, este proceso se repite hasta obtener una solución al problema (en caso que esto no ocurra, debe aplicarse algún operador de reparación sobre la solución parcial). En el Cuadro 3.5, se muestra un pseudo-código de la fase constructiva usando una lista RCL basada en valor. Los parámetros de entrada son α y E. El número real α es usado para determinar el umbral que selecciona elementos de E a incluir en la lista RCL.

3.4.2. Fase de Mejora

La segunda fase de GRASP corresponde al proceso de búsqueda local. Los algoritmos propuestos en esta fase varían desde algoritmos sencillos de búsqueda local a técnicas más avanzadas y eficientes.

Cuadro 3.5: Algoritmo GRASP: fase constructiva.

```
procedure greedyRandomizedConstruction(α, E)
 x ← {}
 C ← E
 Calculate greedy values c(e), ∀ e ∈ C
 while C ≠ {} do
   c* ← min{c(e) | e ∈ C}
   c^* ← max{c(e) | e ∈ C}
   RCL ← {e ∈ C | c(e) ≤ c* + α(c^*-c*) }
   Select candidate element s at random from RCL
   x ← x + {s}
   Update C
   Calculate greedy values c(e), ∀ e ∈ C
 end while
 return x
end procedure
```

En la literatura y para diferentes problemas de optimización, pueden encontrarse algoritmos basados en GRASP que incorporan algoritmos de búsqueda local tales como J-Means [96], ADD and DROP, K-Means, 1-OPT, 2-OPT, entre otros [39]. A su vez, se ha propuesto GRASP para la construcción de soluciones y otras metaheurísticas para la mejora de las mismas, tales como Variable Neighborhood Search [64], Tabu Search/Path Relinking y Scatter Search [40].

En el Cuadro 3.6 se muestra un pseudo-código de la fase de mejora de GRASP.

3.5. Tabu Search

Propuesto por Glover [50], Tabu Search (TS) se basa en la premisa que para resolver un problema de optimización de manera inteligente y lograr soluciones de calidad, es necesario disponer de memoria adaptativa y realizar una exploración de soluciones de manera sensible.

Cuadro 3.6: Algoritmo GRASP: fase de mejora.

```
procedure LocalSearch(x^0)

x ← x^0
while x is not local optima w.r.t. N(x) do
  y ← N(x)
  if f(y) < f(x) then
    x ← y
  end if
end while
return x
end procedure
```

Las memorias adaptativas propuestas por Tabu Search, permiten la exploración del espacio de soluciones de manera eficiente. Esto se debe a que durante el proceso de búsqueda, se recomienda usar los diferentes tipos de memorias disponibles para recolectar información relevante, lo cual contrasta con otras metaheurísticas que no lo hacen (llamadas *sin memoria*). Por otra parte, la exploración sensible tiene que ver con que más allá de propuestas determinísticas o probabilísticas, una mala decisión (en el proceso de búsqueda de soluciones), puede proveer mejor información que una buena decisión de tipo aleatoria [48].

Al igual que un algoritmo de búsqueda local, iterativamente se desplaza de una solución x a otra solución x' $\in$ N(x). Sin embargo, se diferencia de otras metaheurísticas ya que durante el proceso de búsqueda reemplaza el vecindario N(x) por otro $N^*(x)$. Para ello, hace uso de diferentes estructuras de memorias [121].

Tabu Search ha sido aplicado a diferentes problemas de optimización, tanto de aplicación práctica como teórica. Entre ellos, podemos mencionar problemas de planificación [19], agrupamiento [91], computación paralela [48], ruteo de vehículos ([8], [99]), grafos [119], entre otros.

En el Cuadro 3.7, se muestra un pseudo-código de alto nivel [31].

Cuadro 3.7: Algoritmo Tabu Search.

```
procedure simpleTabuSearch()

 x ← GenerateInitialSolution()
 InitializeMemoryStructures()
 x_best ← x
 while termination condition not met do
   A ← GenerateAdmissibleSolutions(x)
   x ← SelectBestSolution(A)
   UpdateMemoryStructures()
   if f(x) < f(x_best) then
     x_best ← x
   end if
 end while
 return x_best
end procedure
```

3.5.1. Memorias de Tabu Search

En vez de almacenar soluciones completas, Tabu Search almacena información sobre propiedades de la solución (*atributos*) en sus estructuras de memoria, información que va cambiando a medida que el algoritmo mediante *movimientos*, se mueve de una solución a otra. Las memorias más conocidas de Tabu Search son las memorias *basadas en recencia* y las memorias *basadas en frecuencia*.

La memoria basada en recencia, también conocida como *memoria de corto plazo*, es implementada mediante una *lista tabú*, la cual registra atributos de las soluciones recientemente visitadas, prohibiendo movimientos que permitan regresar sobre las mismas durante un determinado tiempo (conocido como tenure o tenencia). Por lo tanto, el vecindario de la solución actual está restringido a las soluciones que no se compongan de los atributos pertenecientes a la lista tabú. El uso de la lista tabú previene el ciclaje, forzando a su vez a aceptar soluciones de baja calidad, durante el proceso de búsqueda. Es importante destacar que para determinados problemas de optimización, es suficiente el uso de memorias de corto plazo para obtener soluciones de calidad ([50], [67]).

La memoria basada en frecuencia complementa la información aportada por la memo-

ria de corto plazo. En particular, las memorias basadas en frecuencia se usan para registrar dos eventos particulares: la cantidad de veces que un atributo (de solución) cambia las soluciones visitadas (*medida de transición*) y las veces que un atributo pertenece a las soluciones visitadas sobre una determinada trayectoria (*medida de residencia*). Para incrementar la eficiencia de los *simple Tabu Search* (algoritmos que únicamente implementan memorias de corto plazo), las *memorias de largo plazo* permiten intensificar o diversificar la búsqueda de soluciones. A través del uso de soluciones *elite* completas (óptimos locales) o atributos de las mismas, es posible explorar más agresivamente (*estrategia de intensificación*) sobre determinadas regiones prometedoras del espacio de soluciones. Por otra parte, la introducción de nuevos atributos y la combinación de estos con la información recolectada a lo largo del proceso de búsqueda, permite la exploración de nuevas regiones (*estrategia de diversificación*)[50].

3.5.2. Ejection Chains

Las estrategias de intensificación y diversificación no están únicamente relacionadas a la exploración del espacio de soluciones mediante movimientos *simples*. Uno de los problemas que surgen en los algoritmos de búsqueda local se debe a la ejecución de movimientos acotados, a través de la modificación de la solución actual en dos o tres componentes, generando así soluciones fuertemente dependientes de la solución inicial [103]. Es por ello que es necesario considerar acciones que involucren *movimientos compuestos*.

Una forma de obtener estos movimientos, es a través de la creación de cadenas de movimientos simples que sean activadas o reubicadas (expulsadas de su estado actual). Por lo tanto, un movimiento compuesto puede ejecutarse cuando la cadena no puede crecer más en términos de longitud o bien cuando la calidad de la misma decrece al agregar más elementos a la misma.

Ejection Chains [124] es una técnica avanzada de Tabu Search que consiste en la construcción de vecindarios compuestos por movimientos (elementos) simples para así crear movimientos más complejos y potentes.

Esta técnica ha sido aplicada con excelentes resultados en problemas de planificación [19], asignación ([86], [124], [125]), ruteo de vehículos ([99], [103]) y grafos [119].

3.5.3. Path Relinking

Propuesta como otra técnica avanzada de Tabu Search, Path Relinking [50] es una estrategia de intensificación y diversificación la cual genera nuevas soluciones (al problema de optimización) explorando trayectorias mediante la conexión entre soluciones atractivas (o *elite*).

Para ello, usando una solución llamada *inicial* y otra llamada *guía* desde el conjunto de soluciones elite, se genera un *camino* desde la primera hacia la segunda, mediante movimientos que introducen atributos encontrados en la segunda en la solución cuyo punto de partida fue la primera [105]. Además de ser considerado como un método de combinación de soluciones [107], Path Relinking es visto como una estrategia que busca introducir atributos de soluciones elite mediante los movimientos seleccionados.

En vez de generar directamente una nueva solución combinando dos o más soluciones (tal como lo hace por ejemplo, Algoritmos Genéticos), Path Relinking genera caminos entre y más allá de las soluciones inicial y guía. Para ello, generará nuevas soluciones (factibles y/o infactibles) que compartirán atributos de las dos soluciones usadas, más atributos propios los cuales dependerán de la forma en que se realice el camino. Los caminos se obtienen mediante los atributos de soluciones que son agregados, eliminados o modificados (sobre la solución actual) y ejecutados en el espacio de soluciones.

Dado un conjunto de soluciones elite, los roles que cumplen las soluciones *iniciales* y *guía* pueden intercambiarse. Es por esto, que las soluciones guía e iniciales son conocidas en forma conjunta como *soluciones de referencia*. Las mismas pueden generarse mediante el uso de heurísticas que provean un conjunto de soluciones que sean de calidad ó diversas (originalmente, obtenidas durante la ejecución de Tabu Search).

La estrategia de intensificación mediante Path Relinking puede lograrse generando caminos entre soluciones similares (que provengan de una misma región de soluciones), mientras que la diversificación a través de caminos entre soluciones que provengan de regiones diferentes o que posean atributos diferentes.

Además de ser un mecanismo incorporado en algoritmos basados en Tabu Search, Path Relinking ha sido aplicado conjuntamente con otras metaheurísticas tales como Algoritmos Genéticos y GRASP ([79], [90]) en problemas de agrupación ([29], [107]), asignación [125], ruteo de vehículos ([70], [102]) y planificación [4].

3.6. Scatter Search

Propuesto originalmente por Glover ([80], [88]), Scatter Search (SS) es un procedimiento metaheurístico que opera sobre un conjunto de soluciones llamado *conjunto de referencia*, las cuales se combinan para obtener nuevas y mejores soluciones que las originales. Para ofrecer soluciones competitivas a problemas de optimización, Scatter Search combina la información atractiva y de calidad proveniente de reglas, restricciones o soluciones. A diferencia de los Algoritmos Genéticos, Scatter Search no se basa en operadores de cruzamiento aleatorios como tampoco trabaja con grandes poblaciones de soluciones. Este método evolutivo se basa en el trabajo sistemático y estrátegico sobre un número reducido de soluciones.

Para implementar un algoritmo basado en Scatter Search, deben considerarse los siguientes 5 métodos [80]:

- Diversificación (Diversification Generation Method): este método tiene como función, la generación de una gran colección de soluciones diversas. El tamaño *Psize* recomendable de la colección, es 10 veces el tamaño b del conjunto de referencia. El uso de diferentes tipos de memoria (como las propuestas por Tabu Search) puede ayudar a obtener soluciones diversas de manera eficiente.

- Mejora (Improvement Method): dada una solución inicial (que puede o no ser factible) proveniente de los métodos de diversificación y combinación, la misma es reparada (completa) o mejorada mediante algoritmos heurísticos. Es recomendable analizar si la mejora debe aplicarse a todas las soluciones, dado el tiempo computacional asociado y la rápida convergencia que pudiera ocurrir.

- Actualización (Reference Set Update Method): se encarga de mantener el conjunto de referencia que contiene las b mejores soluciones (según calidad y diversidad), encontradas hasta el momento. Las primeras $b/2$ corresponden a soluciones de calidad, siendo las restantes las más diversas. Inicialmente, el conjunto de referencia se construye a partir de la colección P. Luego, de las soluciones combinadas y luego mejoradas.

- Generación de subconjuntos (Subset Generation Method): a partir del conjunto de referencia, genera subconjuntos de soluciones que luego serán combinados. Una

alternativa simple para la obtención de subconjuntos, es a través de la selección de parejas de soluciones. No obstante, existen propuestas de tres o más soluciones.

- Combinación (Solution Combination Method): crea nuevas soluciones mediante la combinación del subconjunto de soluciones obtenidas por el método anterior. Se recomienda el uso de métodos de combinación aleatorios y determinísticos, condicionando la aplicación (de cada método) a la efectividad en las soluciones obtenidas en el pasado. La inclusión de diversos operadores permite una mejor búsqueda dentro del espacio de soluciones.

En el Cuadro 3.8, se muestra un pseudo-código de la versión básica de Scatter Search [51] incluyendo los parámetros de tamaño ($Psize$) de la colección inicial de soluciones y del conjunto de referencia (b).

En base a la plantilla y a la versión básica de Scatter Search, es importante mencionar la posible interacción con otras metaheurísticas. Respecto al método de generación de soluciones, el uso de Tabu Search o GRASP puede contribuir a la obtención de soluciones iniciales diversas y de calidad. En cuanto al método de mejora, la aplicación de VNS puede colaborar con la obtención de mejores soluciones a bajo costo computacional. En cuanto al método de combinación, la interacción con Algoritmos Genéticos puede facilitar a la combinación de varias soluciones.

Desde su aparición, Scatter Search ha sido aplicado en diferentes problemas de optimización. Entre ellos, problemas de tales como agrupamiento ([29], [95], [113]), ruteo de vehículos [21], planificación [89], ordenamiento [49] y grafos [20].

3.7. Variable Neighborhood Search

Propuesto por Hansen y Mladenovic [120], Variable Neighborhood Search (VNS) es una metaheurística cuya premisa para resolver problemas de optimización, es la búsqueda de soluciones competitivas mediante el cambio sistemático de vecindarios (también conocidos como *entornos*). VNS explora vecindarios distantes de la solución actual de manera incremental, y salta desde el mismo a uno nuevo si se ha obtenido una mejora. La idea detrás de VNS es el cambio dinámico de vecindarios dentro de un algoritmo de búsqueda local [44].

Cuadro 3.8: Algoritmo Scatter Search básico.

```
procedure ScatterSearch(Psize, b)

P = ∅
RefSet = ∅
repeat
  x = diversificationMethod(P)
  x' = improvementMethod(x)
  if x ∉ P then
    P = P ⋃ x'
  end if
until |P| = Psize

RefSet = updateMethod(P,b)
newSolutions = True
while newSolutions do

  newSubsets = generationMethod(RefSet)
  newSolutions = False
  while newSubsets ≠ ∅ do
    s = pop(newSubsets)
    x = combinationMethod(s)
    x' = improvementMethod(x)
    RefSet = updateMethod(RefSet,x')
    if RefSet has changed then
      newSolutions = True
    end if
  end while

end while
return pop(RefSet)
end procedure
```

VNS se basa en tres principios [44]:

- Un óptimo local con una determinada estructura de vecindario, no necesariamente lo sea con otra.
- Un óptimo global es un óptimo local con todas las estructuras de vecindarios.

- Para muchos problemas de optimización, los óptimos locales están relativamente cerca (entre ellos).

En base a los dos primeros hechos, se desprende que es necesario incorporar varias estructuras (parametrizadas) de vecindarios en los algoritmos basados en VNS, para obtener soluciones de calidad. Respecto al último, en base a la información provista por los óptimos locales, es posible alcanzar el óptimo global. Dado que entre un óptimo local y el óptimo global, los valores que toman ciertas variables de decisión (del problema) son los mismos, mediante un estudio adecuado, es posible dirigir el proceso de búsqueda hacia la solución óptima.

En el Cuadro 3.9, se muestra un pseudo-código de VNS básico [90].

Cuadro 3.9: Algoritmo VNS básico.

```
procedure basicVNS()

select the set of neighborhood structures N_k  ∀k = 1, 2, ... k_max
x ← FindInitialSolution()
while termination condition not met do
  k ← 1
  Repeat
    x' ← GenerateAtRandom(N_k(x))     (shaking phase)
    x'' ← LocalSearch(x')
    if f(x'') < f(x) then     (move)
      x ← x''
      k ← 1     (smallest neighborhood)
    else
      k ← k+1     (larger neighborhood)
    end if
  until k=k_max
end while
return x

end procedure
```

El esquema básico de VNS tiene tres fases:

- Agitación (Shaking): en esta fase se obtiene la solución $x' \in N_k(x)$, la cual es generada en forma aleatoria. Dada una determinada estructura de vecindario, generar una

solución sobre un vecindario más grande conduce a una búsqueda (desde la solución actual) de soluciones de manera estocástica [26].

- Búsqueda local (Local Search): se aplican algoritmos de búsqueda local sobre la solución inicial x'. Los mismos no necesariamente están restringidos a las estructuras de vecindarios propios de VNS (por ejemplo, se han usado 1-OPT y OR-OPT en problemas de planificación [64]).

- Movimiento (Move): Al comparar (el valor de función objetivo de) la solución obtenida por búsqueda local con la mejor solución hasta el momento, existen dos caminos. Si existe mejora, se buscará una nueva solución desde un vecindario más pequeño. Por lo tanto, es probable que se obtenga una nueva solución que mantenga buenos atributos del óptimo local (Principio #3). Si no mejora, se buscará una nueva solución desde un vecindario más grande, donde es probable que se obtenga una mejor solución (Principio #1). La posibilidad de moverse entre vecindarios es posible si se define una estructura de vecindario parametrizada.

Los algoritmos basados en VNS deben incluir varias estructuras de vecindarios, para de esta manera, alcanzar soluciones (al menos) cercanas al óptimo global (Principio #2). Para problemas de optimización como el Problema de Cubrimiento de Conjuntos (Set Covering Problem o SCP), una forma de realizar movimientos en el espacio de soluciones es cambiando algunos de los elementos que conforman la solución (recordemos que SCP consiste cubrir todas las filas con la menor cantidad de columnas posible). Estos movimientos permiten obtener diferentes vecindarios si se parametrizan determinados aspectos relacionados con los cambios, como por ejemplo, fijando o acotando el número de elementos (columnas, en este caso) de la solución que se pueden cambiar. Es así que una estructura de vecindario parametrizada para SCP es el *K-Intercambio* (k-Exchange), el cual consiste en intercambiar k columnas de la solución por otras k columnas que no lo están, obteniéndose entonces los vecindarios k-ésimos de la solución x con k=1,2, ..., k_{max}.

VNS ha sido aplicado a diferentes problemas de optimización tales como agrupamiento ([42], [91]), ruteo de vehículos ([66], [98]), planificación [46], ordenamiento [44] y grafos [26].

3.7.1. Variable Neighborhood Descent

La versión básica de VNS (o Reduced VNS) ha resultado útil en problemas de optimización con grandes instancias de datos, donde la búsqueda local resulta computacionalmente costosa [64].

Cuando esto no ocurre, es recomendable considerar otras alternativas. En general, las extensiones realizadas a Variable Neighborhood Search, están mayormente enfocadas a la fase de búsqueda local, siendo VND un claro ejemplo. VND es un procedimiento de mejora iterativo, donde el cambio sistemático de vecindarios y la obtención del óptimo local del vecindario *k-ésimo*, se realiza de manera determinística ([26], [64]). En particular, antes de diversificar o intensificar la búsqueda de soluciones, se debe generar una solución en base a una estructura de vecindario definida y luego proceder a la mejora de la misma. Es en esta fase, donde algoritmos heurísticos específicos del problema u otras metaheurísticas principalmente orientadas a la construcción de soluciones, juegan un rol importante.

En el Cuadro 3.10, se muestra un pseudo-código de VND.

Cuadro 3.10: Algoritmo Variable Neighborhood Descent.

```
procedure VND()

select the set of neighborhood structures N_k  ∀k = 1, 2, ..., k_max
x ← FindInitialSolution()
while termination condition not met do
  k ← 1
  Repeat
    x' ← BestImprovement(N_k(x))
    if f(x') < f(x) then
      x ← x'
      k ← 1
    else
      k ← k+1
    end if
  until k=k_max
end while
return x
end procedure
```

3.7.2. Otras Extensiones de VNS

En la literatura se han propuesto diferentes extensiones de VNS, en base a las tres fases principales:

- Agitación (Shaking): en lugar de generar una solución al azar que sea solución inicial de la fase de mejora, se podría hacer la selección entre b soluciones aleatorias.
- Búsqueda local (Local Search): además de la inclusión de algoritmos constructivos y de búsqueda local a nivel determinístico (VND), una extensión de VNS propone la descomposición del problema en dos niveles. Variable Neighborhood Decomposition Search (VNDS) propone que la creación y mejora de soluciones se realice sobre un subespacio de soluciones. De esta manera, es posible mantener buenos atributos de la solución actual y modificar otros (gracias a la exploración reducida) que permitan alcanzar mejores soluciones. Burke et al. [18] han propuesto un algoritmo VNS para un problema de planificación de personal, el cual incorpora diferentes estructuras de vecindarios que incorporan estas ideas.
- Movimiento (Move): en lugar de aceptar (siempre) una mejor solución, de manera controlada se podría aceptar una de menor calidad siempre que cumpla determinadas propiedades tal que desde la misma se pueda alcanzar un mejor óptimo local. De Souza y Martins [26] han propuesto un algoritmo Skewed VNS para un problema de árbol mínimo, alcanzando resultados competitivos.

  Otra variante está relacionada con el avance incremental de vecindarios ($\mathrm{k} \leftarrow \mathrm{k}+\mathrm{k}_{step}$) y el rango de los mismos ($[k_{min}, k_{max}]$). Haciendo uso de memoria adicional y ciertas reglas heurísticas, es posible proponer algoritmos basados en VNS más robustos.

Para más detalles de VNS, se recomienda [64].

3.8. Biased Random Key Genetic Algorithm

3.8.1. Algoritmos Genéticos

Los Algoritmos Genéticos (GA) son algoritmos adaptativos basados en el proceso genético de los organismos biológicos. Combinan la noción de supervivencia de los individuos más fuertes de una especie (en el tiempo) con el intercambio de información en forma aleatoria entre individuos, para buscar la mejor solución a un problema de optimización. En cada generación, una nueva población de individuos artificiales son creados mediante la reproducción de los individuos más fuertes de la generación anterior, siendo posible que algunos de los nuevos individuos sufran alguna modificación en su composición.

A diferencia de otras metaheurísticas, los Algoritmos Genéticos [52]:

- Trabajan sobre soluciones codificadas en un alfabeto finito.
- Buscan la mejor solución a un problema de optimización, manteniendo una población de soluciones.
- Usan la función objetivo del problema como medida de selección y comparación de soluciones.
- No usan otra información auxiliar.
- Usan reglas de transición probabilísticas en vez de determinísticas.

Cada solución a un problema de optimización también llamada *individuo* o *cromosoma*, debe ser codificada aplicando un determinado alfabeto en una cadena de longitud finita. La forma de codificar una solución en una cadena de bits o números enteros, varía de un problema a otro (por ejemplo, la codificacion de soluciones para el Problema del Viajante podría realizarse usando genes con valores enteros; en cambio, para el Problema de Cubrimiento de Conjuntos, basta con genes binarios). Básicamente, la codificación tiene por objetivo traducir una solución en una cadena de *genes*, permitiendo de esta manera la reproducción entre individuos.

En vez moverse desde una solución (punto) a otra dentro en el espacio de soluciones, los Algoritmos Genéticos trabajan con una colección de soluciones también llamada *población*,

reduciendo de esta manera la probabilidad de explorar en un área reducida del espacio de soluciones y quedar atrapado en algún óptimo local.

En cada iteración del algoritmo, se aplican varios operadores los cuales construyen una nueva generación de individuos (soluciones) basada en la inmediata anterior conservando los genes de los mejores individuos (mejores soluciones). Los operadores genéticos más usados son *reproducción*, *cruzamiento* y *mutación*.

La reproducción es el proceso de preservar (copiar) los mejores individuos (soluciones) de la generación actual (en la nueva). La selección de los mejores individuos se realiza analizando los *fitness* correspondientes (valores devueltos por la función objetivo del problema en cuestión). Luego de la reproducción, se aplica el cruzamiento, el cual consiste en elegir dos individuos en forma aleatoria y combinar sus genes para producir uno o más hijos (*offspring*). El cruzamiento puede realizarse en forma determinística indicando explícitamente cuales genes de cada padre formarán parte del hijo (nueva solución) o bien en forma aleatoria, construyendo al nuevo individuo mediante la selección aleatoria de genes de ambos padres. A su vez, la mutación consiste en la modificación de algunos genes de ciertos individuos. Este operador se considera como un mecanismo de adaptación al ambiente por parte de la población de individuos a lo largo de su evolución.

En el Cuadro 3.11, se muestra un Algoritmo Genético de alto nivel.

3.8.2. Random Key Genetic Algorithm

Bean [10] propuso el uso de claves aleatorias para la codificación de soluciones en algoritmos basados en Algoritmos Genéticos. Random Key Genetic Algorithm (RKGA) representa cada solución a un problema de optimización, como una cadena de números aleatorios reales en el intervalo [0,1) (también conocido en la literatura como alfabeto de claves aleatorias U(0,1)). Para obtener la solución real al problema y/o conocer su valor de función objetivo, la cadena de valores aleatorios respectiva debe ser decodificada.

RKGA resulta conveniente en aquellos problemas que requieren permutaciones de números enteros y donde al aplicarse los operadores de cruzamiento sobre dos soluciones padre, pueden obtenerse (soluciones) hijos infactibles [114]. A continuación, se explicará esta situación mediante un ejemplo.

Cuadro 3.11: Algoritmo Genético.

```
procedure GA(#Pop, #Gens)

Population={}
GeneratePopulation(Population,#Pop)
For i = 1 ... #Gens
  While stopping crossover criteria is not satisfied do
    ch,p1,p2=SelectParents(Population,#Pop)
    Population[ch]=Crossover(Population[p1],Population[p1])
  end while
  while stopping mutation is not satisfied do
    mut=SelectChromosome(Population,#Pop)
    Population[mut]=Mutate(Population[mut])
  end while
end for
Best=SelectBestChromosome(Population,#Pop)
return Population[Best]
end procedure
```

Consideremos el problema del Viajante y una instancia de datos con 6 nodos. Sean 2 soluciones factibles

Parent 1: 1→2→3→4→5→6

Parent 2: 6→1→5→4→3→2

y un operador de cruzamiento con un punto de corte (One-Point Crossover). Si aplicamos dicho operador sobre las dos soluciones para obtener una nueva (child), ésta puede resultar infactible tal como se observa en la Fig. 3.2.

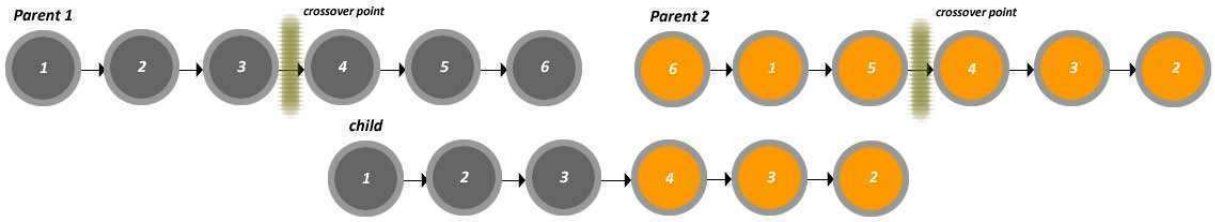

Figura 3.2: Algoritmo Genético: solución infactible al cruzar dos soluciones factibles.

Ahora, obtengamos dos cadenas de 6 números aleatorios reales en el intervalo [0,1), uno por cada nodo a visitar. Recordemos que las cadenas de claves aleatorias codifican una solución (en este caso para TSP) con números aleatorios [10].

Random Keys 1: (0.15, 0.89, 0.42, 0.27, 0.63, 0.11)

Random Keys 2: (0.48, 0.71, 0.83, 0.74, 0.35, 0.56)

Luego, ordenemos en forma ascendente cada una de las dos cadenas. Si realizamos el mapeo entre posición i-ésima del número aleatorio (en la cadena original) y la posición que ocupa en la cadena ordenada en forma ascendente, obtendremos las siguientes soluciones para TSP.

Random Keys 1 (ordenada): (0.11, 0.15, 0.27, 0.42, 0.63, 0.89)
Parent 1 (solución TSP): 6→1→4→3→5→2

Random Keys 2 (ordenada): (0.35, 0.48, 0.56, 0.71, 0.74, 0.83)
Parent 2 (solución TSP): 5→1→6→2→4→3

Si aplicamos nuevamente el operador One-Point Crossover sobre las dos cadenas aleatorias y la nueva cadena es ordenada (y se realiza el mapeo correspondiente), se obtiene una solución factible para TSP tal como se muestra en la Fig. 3.3.

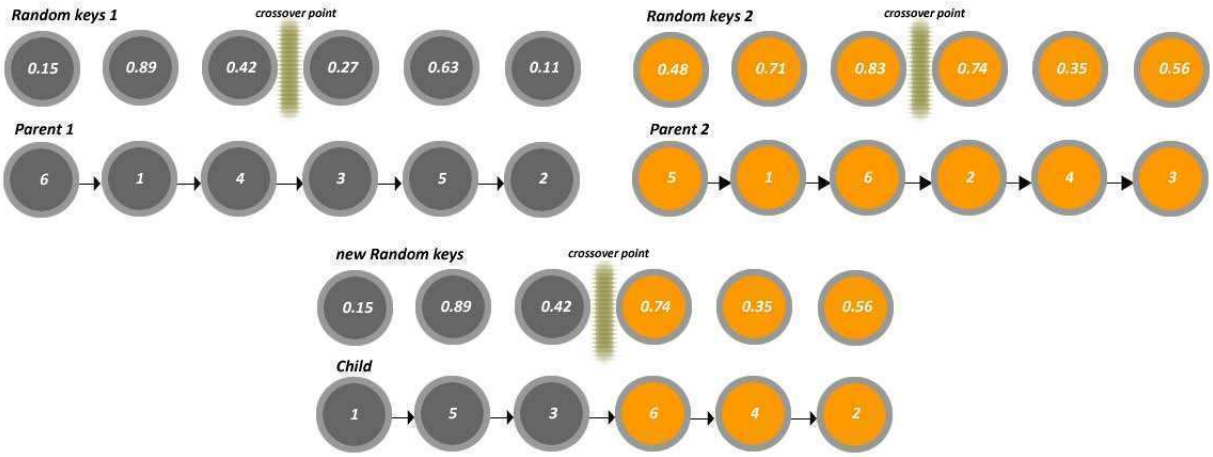

Figura 3.3: RKGA: solución factible entre dos cadenas de claves aleatorias.

La característica más relevante de las cadenas de claves aleatorias es que los nuevos individuos (obtenidos por la aplicación del operador cruzamiento) proporcionen soluciones factibles al problema en cuestión [10].

Para aplicar RKGA a un problema de optimización, inicialmente se debe construir una población de cadenas de claves aleatorias. Luego, y durante las sucesivas generaciones, los operadores genéticos clásicos deben usarse de la siguiente manera:

- Reproducción: mediante la decodificación de las cadenas de claves aleatorias en soluciones reales y obteniéndose sus fitness asociados, se clasifican las soluciones en dos grupos: *Elite* y *No-Elite*. Las cadenas de claves aleatorias de tipo Elite permanecerán en la nueva generación.

- Cruzamiento: se eligen (en forma aleatoria) dos cadenas de claves aleatorias de toda la población. Luego, en lugar de utilizar algún operador clásico de cruzamiento como One-Point o Two-Point, se propone el uso del operador Cruzamiento Uniforme Parametrizado (Parameterized Uniform Crossover)[115]. Este operador propone que cada valor de gen del hijo obtenido por cruzamiento, se obtenga mediante el lanzamiento de una moneda sesgada la cual determinará cual de los dos padres contribuirá con el *allele* correspondiente.

  Siguiendo con el ejemplo del Problema del Viajante con 6 nodos, explicaremos el funcionamiento del operador. Supongamos que si al lanzar la moneda sale Cara(C), el allele elegido para el gen del hijo proviene de la primer cadena; si sale Sello(S), el allele proviene de la segunda cadena. Finalmente, consideremos que la probabilidad que salga Cara es 0.7. En la Fig. 3.4 se observa un potencial resultado obtenido por este operador de cruzamiento.

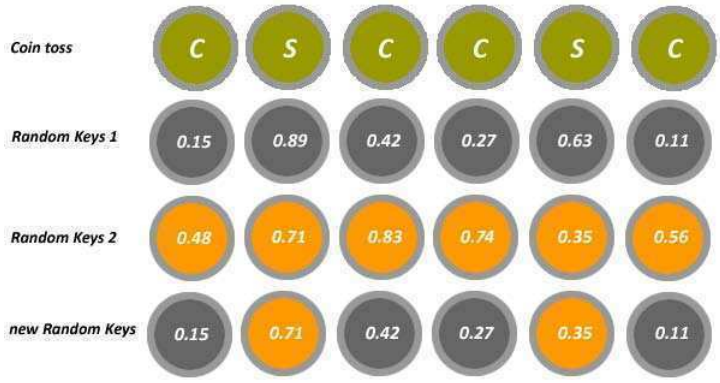

Figura 3.4: RKGA: cruzamiento entre dos cadenas con Prob(C)=0.7.

De la Fig. 3.4, se puede apreciar que la nueva cadena de claves aleatorias

new Random Keys: (0.15, 0.71, 0.42, 0.27, 0.35, 0.11)

produce la siguiente solución

Child (solución TSP): 6→1→4→5→3→2

- Inmigración: en vez de aplicar el operador mutación, el cual modifica genes de ciertos individuos (de la población) con baja probabilidad de ocurrencia, se propone la *inmigración* de nuevos individuos a los efectos de prevenir una prematura convergencia. Esto significa generar (de la misma forma que al inicio del algoritmo) nuevos individuos en introducirlos en la nueva generación.

En la Fig. 3.5 se observa cómo se obtienen los individuos (cadenas de claves aleatorias) de la nueva generación, como así también los porcentajes recomendados de individuos (de la nueva generación) obtenidos por los operadores de reproducción, cruzamiento e inmigración.

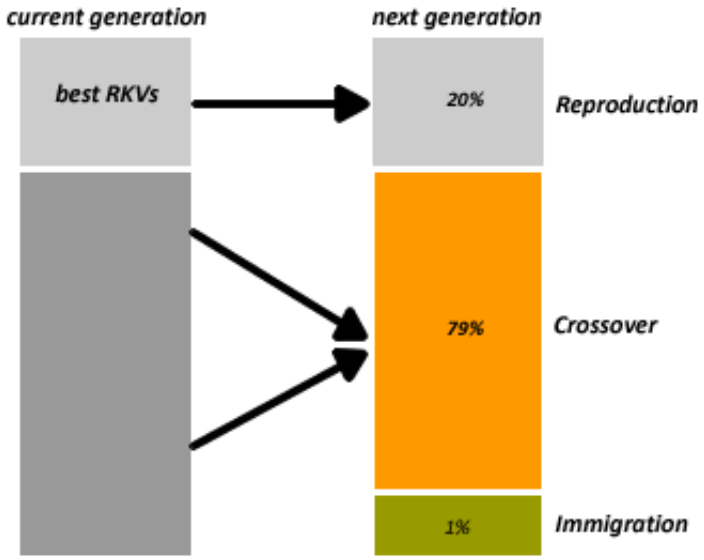

Figura 3.5: RKGA: transición entre generaciones.

3.8.3. Biased Random Key Genetic Algorithm

Biased Random Key Genetic Algorithm (BRKGA) propuesto en forma simultánea por Gonçalves y Almeida [57] y Ericsson et al. [35], difiere de RKGA en la manera en

que se seleccionan los dos padres para aplicar el operador de cruzamiento [59]. BRKGA propone que todo nuevo individuo obtenido por cruzamiento se realice con una cadena de claves aleatorias de tipo Elite y otra de tipo No-Elite (aunque es posible que la segunda cadena pueda seleccionarse de la población entera). De esta manera y mediante el uso del operador Cruzamiento Uniforme Parametrizado (usando $ProbE > 0{,}5$), los nuevos individuos tienen mayor probabilidad de heredar buenos genes de sus padres de tipo Elite.

En la Fig. 3.6 se observa cómo en BRKGA se obtienen los individuos de la nueva generación.

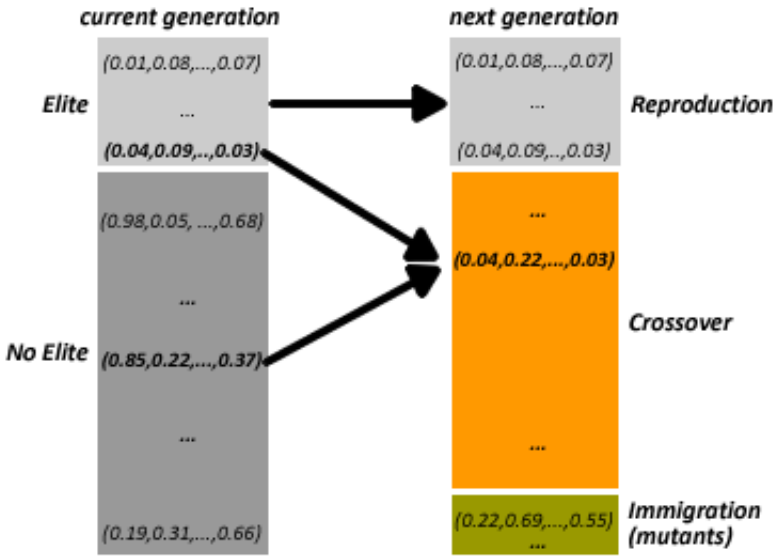

Figura 3.6: BRKGA: transición entre generaciones.

Recientemente, BRKGA ha sido aplicado a difentes problemas de optimización tales como planificación ([58], [60]), ruteo de vehículos [114], de corte [56], balanceo de líneas de ensamblaje [57], entre otros.

Para más detalles sobre BRKGA, se recomienda [59].

Capítulo 4

El Problema de Ruteo de Arcos Capacitados

Este capítulo está dedicado al estado del arte del problema de Ruteo de Arcos Capacitados (CARP). El mismo contiene una introducción al problema, revisión histórica de los problemas de ruteo de arcos, modelización matemática y técnicas aplicadas para su resolución. Finalmente, se describen algunas extensiones del problema y aplicaciones a casos reales.

El problema de Ruteo de Arcos Capacitados (CARP) consiste en atender las demandas sobre determinadas calles de una red vial a través de una flota homogénea de vehículos, los cuales inician y finalizan sus recorridos desde un único depósito. El objetivo del problema es minimizar el costo total de recorrido de manera tal de atender todas las demandas sin exceder la capacidad de carga de los vehículos involucrados.

Uno de los beneficios del estudio de este problema radica en su aplicación real en organismos públicos y empresas. En ambos casos, se busca ofrecer un servicio de mejor calidad y al menor costo posible. No obstante, es probable que sus problemas de ruteo posean características adicionales a CARP; es decir, características relacionadas a su contexto que deben ser consideradas y por tanto, agregadas al modelo original de manera tal de reflejar mejor su realidad; por ejemplo, minimizar el número de vehículos usados, cumplir con las demandas dentro de un cierto intervalo de tiempo especificado, considerar una red vial con calles en un único sentido, en dos o ambos, utilizar varios depósitos desde donde los vehículos puedan comenzar (y finalizar) sus recorridos, considerar vehículos con diferentes capacidad de carga, entre otras.

Entre las diferentes aplicaciones de CARP o sus extensiones, relacionados con los servicios públicos podemos mencionar la recolección de residuos ([28], [34], [47], entre otros) y las tareas de mantenimiento de calles ([93], [126]).

Es por ello, que el estudio del Problema de Ruteo de Arcos Capacitados como así también de sus extensiones, está motivado por la necesidad de ofrecer mejores y variadas alternativas de soluciones a aquellas organizaciones que deben tratar cotidianamente con problemas de ruteo y logística.

4.1. Revisión histórica

Con el objetivo de presentar el problema de estudio, previamente revisaremos algunas definiciones.

Los problemas de ruteo de arcos son definidos mediante un grafo G=(V, E ∪ A) donde V es el conjunto de vértices o nodos, E el conjunto de pares no ordenados (i,j) de vértices de V o aristas y A el conjunto de pares ordenados (i,j) de vértices de V o arcos. Un grafo G es *dirigido* cuando el conjunto E es vacío, es *no dirigido* cuando A es vacío y *mixto*, si ambos conjuntos son no vacíos. El costo c_{ij} de recorrer una arista de E es positivo. Un recorrido T (o ciclo) puede ser representado mediante un vector $(v_1, v_2, \ldots v_n)$ donde (v_i, v_{i-1}) es una arista de E (i=1,. . . , n-1) y $v_1 = v_n$.

Históricamente, el estudio sobre el problema de ruteo de arcos comienza con la presentación de la solución al problema de Königsberg en 1735 por parte de Euler. El problema es el siguiente: *sea un grafo conexo G=(V,E), encontrar un recorrido que visite cada arista de E exactamente una vez o bien informar que no es posible obtenerlo.* Euler probó que es posible obtener tal recorrido siempre que el grado de cada nodo de V sea par.

El siguiente problema es el Problema del Cartero Chino (Chinese Postman Problem o CPP) el cual fue inicialmente propuesto por Kwan Mei-Ko en 1962. Se define así: *sea un grafo conexo G=(V,E,C) donde C es una matriz de costos, encontrar un recorrido tal que visite cada arista al menos una vez al menor costo posible.* Edmonds y Johnson [33] probaron que si el grafo asociado al problema es dirigido o no dirigido, CPP puede ser resuelto en tiempo polinomial. Sino, es un problema NP-Hard.

En 1974, Orloff [65] propuso el Problema del Cartero Rural (Rural Postman Problem o RPP) el cual se define de la siguiente manera: *sea un grafo no dirigido G=(V,E,C) donde*

C es una matriz de costos, encontrar un recorrido de costo mínimo tal que éste visite cada arista de $R \subseteq E$ al menos una vez.

La diferencia entre CPP y RPP es que en el problema del Cartero Rural sólo un conjunto de aristas deben ser visitadas. En 1976, Lenstra y Rinnoy Kan demostraron que RPP es NP-Hard [65].

A continuación, definiremos el problema de Ruteo de Arcos Capacitados (CARP) el cual es una extensión del problema del Cartero Rural, con restricciones de capacidad.

4.2. Modelo matemático

Presentaremos un modelo matemático como problema de programación lineal para CARP [87]. Sea:

- G=(V,E) un grafo,
- V el conjunto de vértices,
- E el conjunto de aristas,
- R $\subseteq$ E el conjunto de aristas requeridas,
- $V_r \subseteq V$ el conjunto de vértices conteniendo los extremos de las aristas de R, más el vértice asociado al depósito,
- K={1,..., M} la flota de vehículos con capacidad de carga Q,
- c_{ij} el costo de recorrido de la arista (i,j) perteneciente a E,
- q_{ij} la demanda asociada a la arista (i,j) perteneciente a E

$$q_{ij} = \begin{cases} 0 & \text{si (i,j)} \notin \text{R} \\ > 0 & \text{si (i,j)} \in \text{R} \end{cases}$$

- Las variables de decisión se definen

$$x_{ijk} = \begin{cases} 1 & \text{si (i,j) es recorrida por el vehículo k} \\ 0 & \text{c.o.c.} \end{cases}$$

El modelo matemático para CARP es el siguiente:

$$\text{Minimizar} \sum_{(i,j)\in E} c_{ij} \sum_{k\in K} x_{ijk} \tag{4.1}$$

$$\sum_{k\in K} x_{ijk} = 1 \qquad (i,j) \in R \tag{4.2}$$

$$\sum_{j\in V_r - \{0\}} \sum_{k\in K} x_{0jk} = \mid K \mid \tag{4.3}$$

$$\sum_{(i,j)\in R} q_{ij} x_{ijk} \leq Q \qquad k \in K \tag{4.4}$$

$$\sum_{j\in V_r} x_{ijk} = \sum_{j\in V_r} x_{jik} \qquad i \in V_r, k \in K \tag{4.5}$$

$$\sum_{i\in S} \sum_{j\notin S} x_{ijk} \geq \sum_{j\in V} x_{hjk} \qquad S \subseteq V_r - \{0\}, k \in K, h \in S \tag{4.6}$$

$$x_{ijk} \in \{0,1\} \qquad (i,j) \in R, k \in K \tag{4.7}$$

Donde la función objetivo (4.1) busca minimizar el costo total de recorrido, sujeta a las restricciones de

- atender las demandas de las aristas requeridas por un único vehículo (4.2),
- el número de vehículos usados es el número de vehículos disponibles (4.3),
- sin exceder la capacidad de carga de ninguno de ellos (4.4)

Además de estas restricciones, se agregan restricciones:

- de igualdad (4.5) y
- de eliminación de sub-recorridos (4.6).

CARP consiste entonces, en determinar un conjunto de rutas para los vehículos a menor costo total de recorrido, tal que cada ruta comienza y finaliza en el depósito, cada una de las demandas asociadas a las aristas de R son atendidas por un único vehículo y ningún vehículo involucrado excede su capacidad de carga Q.

De acuerdo a la formulación antes descripta, puede observarse cierta relación con el Problema de Ruteo Vehicular (Vehicle Routing Problem o VRP). La diferencia reside

en que VRP es un problema de ruteo sobre nodos (o vértices). En la literatura se han propuesto transformaciones de VRP a CARP (por ejemplo, [55]) y de CARP a VRP [87].

Golden y Wong [55] demostraron que CARP es NP-Hard. Incluso si se considera un único vehículo para atender las demandas requeridas, el problema es NP-Hard. Las propuestas existentes en la literatura para ofrecer una solución a CARP están clasificadas en algoritmos heurísticos, metaheurísticas y algoritmos exactos.

4.3. Propuestas para resolver CARP

4.3.1. Algoritmos heurísticos

Para el problema CARP, se han desarrollado algoritmos heurísticos específicos. Asimismo, fueron adaptados otros que originalmente se aplicaron para otros problemas de optimización. A continuación, mencionaremos algunos de ellos:

- Aumentar-Unir (Augment-Merge): propuesto por Golden y Wong [55], construye los recorridos en forma pseudo-paralela mediante dos fases (Aumentar y luego Unir). Primero, para cada arista requerida se construye un recorrido y se ordenan los mismos en orden decreciente según el costo asociado. Luego, para todo recorrido T_i se intenta que absorba a otro recorrido T_j siempre que la arista requerida u no haya sido servida en T_i y no se exceda la capacidad vehicular. A continuación, se intenta concatenar dos recorridos que generen la mayor disminución de costo. El algoritmo finaliza cuando esto no es posible.

- Exploración de recorridos (Path Scanning): propuesto en el trabajo de Golden y Baker [53], este algoritmo construye un recorrido por vez agregando una nueva arista requerida al recorrido actual siempre que no exceda la capacidad vehicular y cumpla que la distancia al depósito es máxima o mínima, o bien que ocupe la mayor capacidad del vehículo o la mínima posible, o considera la máxima o mínima distancia de regreso al depósito según la capacidad remanente del vehículo. Esto se repite hasta que todas las aristas requeridas son atendidas por algún recorrido.

- Algoritmo de Ulusoy (Ulusoy's Algorithm)[117]: construye soluciones en dos fases (metodología conocida en la literatura como Route-First, Cluster-Second). En la

primera, se obtiene una única ruta que visita todos los arcos requeridos (sin considerar restricción de capacidad). En la segunda, se agrupan los arcos requeridos en diferentes rutas, satisfaciendo la restricción de capacidad en cada una de ellas. Para ello, se divide la ruta (obtenida en la primer fase) en varias rutas factibles y se mejora la solución mediante el intercambio de arcos entre rutas.

- Acortar, cambiar, agregar, eliminar y cortar (SHORTEN, SWITCH, ADD, DROP y CUT): Hertz y Mittaz [66] han propuesto varios algoritmos para problemas de ruteo de arcos. Estos trabajan en forma conjunta para obtener soluciones al problema de ruteo. El algoritmo *Cortar* devuelve un conjunto de recorridos factibles (en términos de capacidad vehicular y de atención de demandas de las aristas requeridas) dado un único recorrido que visita todas las aristas requeridas, pero con restricción de capacidad vehicular relajada (situación similar al algoritmo de Ulusoy). A su vez, *Eliminar* quita una arista requerida de un recorrido dado. En tanto, *Agregar* inserta una arista en un recorrido y *Cambiar* invierte el sentido de un recorrido. Finalmente, *Acortar* intenta reducir el costo de un recorrido dado, eliminando cadenas de aristas requeridas previamente servidas.

- Algoritmos adaptados: tal como se indicó anteriormente, se han adaptado algoritmos heurísticos aplicados en problemas similares de optimización, para ser usados en propuestas para CARP. Tal es el caso del algoritmo de búsqueda local 2-OPT propuesto por Croes para el Problema del Viajante (TSP) y adaptado en los trabajos de Lacomme et al. [78] y Beullens et al. [14], el algoritmo 3-OPT para TSP propuesto por Lin y otros algoritmos de mejora descriptos por Toth y Vigo [116], y adaptados en trabajos como el de Maniezzo [87].

Finalmente, es importante mencionar el rendimiento en general de estos algoritmos. La brecha (o GAP) entre la solución obtenida por un algoritmo heurístico y la solución óptima varía entre un 10 % a 40 % [54], lo que remarca la baja calidad de las soluciones devueltas.

En la próxima sub-sección mencionaremos algunos trabajos que hacen uso de estos algoritmos heurísticos como generadores de soluciones iniciales o bien como parte de la fase de mejora.

4.3.2. Metaheurísticas

Se han aplicado diferentes metaheurísticas para resolver CARP desde hace más de 15 años. Mencionaremos algunas de ellas.

Hertz, Laporte y Mittaz [65] diseñaron un algoritmo Tabu Search para CARP. Mediante los algoritmos heurísticos Agregar (ADD) y Eliminar (DROP) (mencionados en la sección anterior), una lista tabú y una función objetivo con penalidades, *CARPET* ha logrado muy buenos resultados sobre instancias de la literatura. A su vez, Brandão y Eglese [16] han propuesto un algoritmo Tabu Search determinístico, el cual construye soluciones aplicando algoritmos heurísticos (entre ellos, Exploración de Recorridos o Path Scanning) las cuales son mejoradas a través de movimientos de aristas y mediante el uso de una lista tabú. De acuerdo a las pruebas realizadas sobre instancias de DeArmon, Belenguer y Eglese, se alcanzaron muy buenos resultados obteniéndose mejoras sobre algunas instancias de Eglese.

Ant Colony Optimization (ACO)[69] está inspirado en el comportamiento que poseen ciertas especies de hormigas, las cuales recorren un determinado camino entre una fuente de alimento y el hormiguero, basado en los altos niveles de feromonas dejados en el suelo por otras. En ACO, los rastros de feromonas funcionan como memoria distribuida que las hormigas usan para construir soluciones para un problema dado. Existen diferentes algoritmos ACO propuestos hasta el momento. Entre ellos, podemos mencionar Ant System, Ant Colony System y Max-Min Ant System. Un algoritmo AS fue propuesto por Doerner et al. [30] para CARP. Usando como información heurística el costo de conexión entre aristas y el conocimiento aprendido por la colonia de hormigas en el pasado, cada una de ellas construye paso a paso recorridos que visitan las aristas requeridas. Luego, se realiza la evaporación de feromona a nivel general y la posterior actualización de los niveles de feromona sobre los distintos caminos elegidos por la colonia. Los resultados obtenidos sobre instancias de DeArmon, muestran que el algoritmo es competitivo. Otro algoritmo AS es el propuesto por Lacomme et al. [78]. En este caso, cada hormiga construye un único recorrido el cual es posteriormente dividido en recorridos factibles en términos de capacidad vehicular. La colonia está formada por hormigas elitistas y no elitistas que permiten intensificar y diversificar el proceso de búsqueda de soluciones. Sobre soluciones factibles y bajo una cierta probabilidad, se aplican algoritmos de búsqueda actual. Las pruebas realizadas sobre instancias de DeArmon, Belenguer y Eglese muestran que el algoritmo es competitivo en términos de resultados pero no en tiempo computacional. En vez de

invertir un alto tiempo computacional en el proceso de búsqueda de soluciones usando un único vecindario, es posible definir y alternar entre vecindades de tamaño reducido hasta encontrar una solución que sea competitiva.

Hertz y Mittaz [66] propusieron un algoritmo Variable Neighborhood Descent (VND) para CARP. El mismo genera un único recorrido que visita todas las aristas requeridas y luego este es dividido y mejorado (en términos de costos) mediante los algoritmos Acortar (SHORTEN), Cortar (CUT) y Cambiar (SWITCH). Las estructuras de vecindarios definidas en este trabajo se relacionan con la unión de k rutas para luego aplicar los algoritmos heurísticos antes indicados. Los resultados sobre instancias de DeArmon, Belenguer e instancias generadas para un trabajo anterior, muestran que el algoritmo obtiene mejores resultados y en menos tiempo que el algoritmo *CARPET*. En tanto, Del Pia y Filippi [28] presentaron un algoritmo VND para CARP con depósitos móviles. Respecto al algoritmo VND propuesto, éste posee características similares al propuesto por Hertz y Mittaz [66], aunque en este caso se propone una estructura de vecindario basada en el movimiento de aristas requeridas de una ruta a otra y otra basada en la modificación de rutas.

Propuesto por Christos Voudoris [121], Guided Local Search (GLS) tiene como premisa usar una nueva función objetivo (respecto al problema de optimización) tal que guíe a los operadores de búsqueda local y evite que el algoritmo quede atrapado en un óptimo local. La nueva función contiene como sumandos, la función objetivo original del problema a resolver y una expresión que penaliza a la solución actual si ésta posee atributos poco deseables en una solución competitiva. Una vez que se alcanzó un óptimo local, se calculan las utilidades de los atributos. A continuación, se procede a incrementar las penalidades de los atributos cuya utilidad es máxima. De esta manera, cuando los operadores de mejora busquen una mejor solución, evitarán soluciones con atributos altamente penalizados. Beullens et al. [14] han propuesto un algoritmo GLS para CARP. Dicha propuesta se basa en movimientos que son aplicados sobre una solución inicial. Estos consisten en el recorrido inverso de secuencia de arcos en una ruta, adaptación del operador 2-OPT, como también movimientos entre dos rutas que ubican una secuencia de arcos de una ruta en otra y el intercambio de arcos entre dos rutas. Para aplicar estos operadores, se hace uso de una lista de vecinos de arcos y el marcado de arcos que producen buenas soluciones. Luego que son aplicados estos operadores sobre una solución dada, se procede a la actualización de las penalidades sobre la solución. Dicha penalidad se aplica entre arcos de una misma ruta y cuya expresión está relacionada con el número de vehículos usados, el costo total de

recorrido y el cardinal del conjunto de aristas requeridas. De acuerdo a pruebas realizadas sobre instancias de DeArmon, Belenguer y otras propuestas por los autores, el algoritmo alcanza resultados competitivos a un bajo costo computacional.

Los Algoritmos Meméticos (Memetic Algorithms o MA) [92] son una técnica de optimización en la que convergen ideas de metaheurísticas basadas en población y otras basadas en búsqueda local. Surgen como alternativa para lograr que los algoritmos basados en Algoritmos Genéticos sean más competitivos comparados con otros de la literatura. Uno de los aspectos importantes de estos algoritmos es el uso iterativo de un operador de mutación sobre una solución dada (un agente de la población), el cual también puede aplicarse sobre otras fases de un algoritmo basado en poblaciones. Un algoritmo basado en MA puede ser visto como una población de agentes que realiza una búsqueda de soluciones en forma autónoma basados principalmente en conceptos determinísticos, cooperando ocasionalmente entre ellos mediante el cruzamiento de agentes y compitiendo por la supervivencia a través de la selección y el reemplazo de agentes. Lacomme et al. [77] propusieron un algoritmo MA para un problema CARP extendido. El problema consiste en un multigrafo mixto, con costos de recorrido y servicios asociados a los arcos/aristas, longitud máxima de recorrido entre otras características. El objetivo de CARP extendido es la minimización del costo total del conjunto de recorrido que visitan a todos los arcos/aristas requeridos. La propuesta se basa en la construcción de soluciones a través de los algoritmos heurísticos (modificados) Exploración de Recorridos (Path Scanning), Aumentar-Unir (Augment-Merge) y el algoritmo de Ulusoy, el uso de los operadores de cruzamiento OX y LOX y la mutación de soluciones mediante el movimiento de aristas/arcos requeridas dentro de un recorrido y entre recorridos, entre otros aspectos salientes. Las pruebas se realizaron usando instancias de DeArmon, Belenguer y Eglese. Los resultados obtenidos mostraron que el algoritmo memético propuesto ofrece mejores resultados que el algoritmo *CARPET* (también propuesto por los autores) y en menor tiempo computacional.

4.3.3. Algoritmos exactos

A pesar que CARP es NP-Hard [55], se han propuesto desde la aparición del problema, varios algoritmos exactos basados en Programación Entera (PLE) que lo resuelven en forma óptima. No obstante, es importante destacar que el tamaño de las instancias resueltas

es bastante inferior comparado con las pruebas realizadas por algoritmos heurísticos y metaheurísticas. La mayoría de los métodos exactos propuestos están basados en Branch and Bound, Cutting Plane y Branch and Cut.

Branch and Bound es un método exacto para resolver problemas de programación lineal entera. Dicho método poda (Bound) del árbol que contiene la enumeración completa de las soluciones de un problema lineal, aquellas ramas que no poseen soluciones prometedoras (infactibles o de baja calidad) y ramifica (Branch) aquellos nodos de los cuales pueden surgir ramas (soluciones) que contengan a la solución óptima. Hirabayashi et al. [68] y Kiuchi et al. [74], propusieron un algoritmo Branch and Bound para CARP. El algoritmo usa el procedimiento de acotación conocido como cotas inferiores por duplicación de nodos (Node Duplication Lower Bound), para la poda del árbol. Las pruebas computacionales se realizaron sobre instancias de prueba propuestas por los autores, cuyas aristas requeridas varían entre 15 y 50, alcanzándose las soluciones óptimas.

Por otra parte, la resolución de problemas lineales enteros mediante Cutting Plane es diferente respecto de Branch and Bound. En vez, de resolver sub-problemas para cada uno de los nodos del árbol de enumeración completa para luego podar y repetir hasta alcanzar la solución óptima, Cutting Plane propone trabajar con un único Problema Lineal Entero agregando restricciones (cortes o Cuts) al mismo, los cuales cumplen la función de *corte* de la región convexa de soluciones, procedimiento que se repite hasta obtener la solución óptima (siempre que esto sea posible). Belenguer y Benavent [11] han propuesto un algoritmo basado en Cutting Plane para CARP, incluyendo tres nuevas desigualdades válidas (son desigualdades $ax \leq a_0$ tal que $\forall x \in X$ se verifica que $ax \leq a_0$ [5]). En cada iteración, se resuelve el actual PLE buscándose un conjunto de desigualdades válidas, las cuales son agregadas al actual modelo y se repite el procedimiento. El algoritmo es detenido cuando no es posible agregar una nueva desigualdad o cuando la solución del PLE es igual a una cota inferior conocida provista por algún algoritmo heurístico para CARP. En dicho caso, se alcanza la solución óptima. Las pruebas computacionales fueron realizadas sobre instancias de Benavent, Golden, Kiuchi y Eglese. Los resultados obtenidos sobre las primeras tres colecciones de instancias, las cuales poseen hasta 50 nodos y 97 aristas requeridas, alcanzaron una brecha (GAP) entre mejor solución alcanzada por el algoritmo y mejor solución conocida, inferior al 1 %. Sobre instancias de Eglese, de hasta 140 nodos y 190 aristas, el GAP obtenido es menor al 3 %. Además, se alcanzaron 47 soluciones óptimas sobre las 87 instancias testeadas.

4.4. Variaciones del Problema

Los problemas surgidos como variantes de CARP han sido propuestos como una manera de modelar mejor diferentes situaciones de la realidad. A continuación, serán comentados algunos de ellos.

4.4.1. El Problema de Ruteo de Arcos Capacitados con instalaciones intermedias (CARP-IF)

Introducido por Ghiani et al. [47], para este problema existe un subconjunto I de V de instalaciones intermedias (IF), el cual puede incluir al depósito. Dichas instalaciones son usadas para descargar las demandas de las aristas requeridas del subconjunto R. Cada vehículo de la flota atiende las demandas de las aristas de R y visita alguna de las instalaciones intermedias para proceder a la descarga. La carga acumulada en cada vehículo entre dos instalaciones intermedias consecutivas, no puede en ningún caso exceder Q. El objetivo del problema es determinar un conjunto de recorridos tal que se atiendan todas las demandas de las aristas de R, sin exceder la capacidad de carga de cada vehículo de la flota, a menor costo total de recorrido posible.

Un problema de ruteo de vehículos con características que lo enmarcan dentro de CARP-IF es el trabajo de Amaya et al. [7]. En este caso, el problema propuesto posee dos tipos de vehículos. El primero, es el encargado de servir a las aristas requeridas. Para ello, inicia su recorrido en el depósito y cumple con las demandas de las aristas requeridas hasta que debe descargar (o cargar) su carga sobre un segundo tipo de vehículo llamado de carga (o recarga) el cual permite al primero, proseguir con el servicio para luego retornar al depósito. Como se puede observar, el segundo tipo de vehículo cumple la función de instalación intermedia. En este caso, el problema consiste en la determinación de los recorridos para los dos tipos de vehículos y el objetivo es la minimización del costo total de recorrido. Los autores proponen un algoritmo exacto basado en Cutting Plane para resolver el problema. Las pruebas computacionales se realizaron con instancias generadas en forma aleatoria con grafos con nodos que variaron entre 20 y 70 y con arcos entre 50 y 595, los cuales fueron resueltos en forma óptima.

Un algoritmo sencillo y eficiente es el presentado por Polacek et al. [98]. Basado en Variable Neighborhood Search, la propuesta consiste de la construcción soluciones median-

te el algoritmo de Ulusoy, el uso del operador Cruzar-Intercambiar (o Cross-Exchange, indicado en Toth y Vigo [116]) que realiza intercambios de cadenas entre dos rutas elegidas aleatoriamente (pudiendo aplicarse sobre una única ruta) usado como mecanismo de perturbación (o Shaking) y como fase de mejora de la solución perturbada, se aplica inversión de las rutas modificadas anteriormente más la reconexión de aristas mediante el algoritmo de *Camino más corto*. Las pruebas computacionales fueron realizadas sobre instancias usadas en Golden et al. [53], Ghiani et al. [47] y Eglese [34]. Los resultados alcanzados por el algoritmo, muestran considerables mejoras sobre los de Ghiani y algunas mejoras sobre los de Eglese.

4.4.2. El Problema de Ruteo de Arcos Capacitados con múltiples depósitos (MD-CARP)

MD-CARP (o Multiple Depot CARP) se define como un CARP con varios depósitos (a diferencia de CARP que sólo posee uno, también conocido como 1-CARP) donde los vehículos son estacionados y desde donde los mismos inician y finalizan sus recorridos. Dado que la diferencia distintiva de este problema respecto a CARP es la existencia de varios depósitos, las nuevas restricciones que aparecen con éste, se relacionan con el inicio y fin de los recorridos. En general, puede pensarse que todos los recorridos comienzan y finalizan en el mismo depósito. No obstante, podrían obtenerse recorridos donde el depósito desde donde se inicia sea distinto al depósito donde finaliza el recorrido. Es posible que esta restricción también se relacione con la capacidad operativa de locación que posea cada depósito.

Amberg et al. [8] propusieron un algoritmo para MD-CARP. El algoritmo se basa en la obtención de soluciones al problema de ruteo mediante una versión modificada del problema de *Arbol Mínimo Capacitado* (en este caso, con múltiples centros) y una fase de mejora aplicando lista tabú y Simulated Annealing, los cuales permiten alcanzar mejores soluciones en forma determinística y probabilística respectivamente, a través del movimiento de nodos entre sub-árboles del árbol mínimo capacitado (solución del problema). Las pruebas computacionales fueron realizadas sobre dos instancias reales correspondientes a tareas en época invernal en las localidades germanas de Königstein y Wennigsen (con 1 y 2 depósitos, respectivamente). Los mejores resultados para la primera instancia mejoraron en un 5 % respecto del mejor resultado publicado. En cuanto al análisis

de resultados devueltos por los algoritmos Tabu Search y Simulated Annealing propuestos como estrategias de mejora, la primera alcanzó mejores resultados a menor tiempo computacional.

Como se indicó anteriormente, Del Pia y Filippi [28] aplican un algoritmo VND para un caso real de recolección de distintos tipos de residuos domiciliarios en la ciudad italiana de Due Carrare. El problema analizado consiste en la recolección de residuos domiciliarios a cargo de camiones satélites los cuales vuelcan sus contenidos sobre camiones compactadores (de mayor capacidad) los que a su vez vuelcan sus contenidos sobre los sitios de descarga (en este caso, el depósito). De acuerdo a lo indicado, los camiones compactadores cumplen la función de instalaciones intermedias (CARP-IF) con la salvedad que la ubicación de los mismos es variable, debido a que el objetivo del problema consiste en la minimización de la duración total de los recorridos de los camiones satélites y compactadores, los cuales se inician y finalizan en un único depósito. Los resultados obtenidos de las pruebas sobre instancias reales, muestran que se lograron reducciones en términos de distancia recorrida cercanas al 30 %.

4.4.3. El Problema de Ruteo de Arcos Capacitados periódico (P-CARP)

P-CARP (o Periodic CARP) es una generalización de CARP en términos de horizonte de tiempo. Para este problema, además de las características conocidas de CARP, se considera un horizonte H de np períodos (por ejemplo, días), cada arista requerida e posee una frecuencia *f(e)* que representa el número de servicios requeridos en H, un conjunto de días *comb(e)* que deben ser atendidos y una demanda Q(e, k, p) correspondiente para cada día dentro del horizonte de tiempo.

El objetivo de P-CARP consiste en obtener una combinación temporal para cada arista requerida y la obtención de recorridos para cada combinación temporal, minimizando el tamaño de la flota o el costo total de recorrido de manera tal de satisfacer las demandas de las aristas requeridas sin exceder las capacidades de carga de los vehículos.

Chu et al. [21] desarrollaron un algoritmo basado en Scatter Search (SS). El problema tratado minimiza el tamaño de la flota vehicular y el costo total de recorrido de manera tal que son considerados los costos de inversión y operacionales en forma conjunta. El

algoritmo propuesto consta de una fase de construcción de soluciones realizado a través de un algoritmo heurístico que realiza la mejor inserción posible de aristas requeridas en recorridos dentro del horizonte temporal como también soluciones factibles generadas al azar, siendo ambas incluídas en la primera población de soluciones de Scatter Search. A continuación, y con cierta probabilidad, algunas soluciones de la población son mejoradas aplicando diferentes operadores de búsqueda local. Las nuevas poblaciones se forman con soluciones de calidad más soluciones diversas para lo cual se define una función de distancia entre dos soluciones. A su vez, se cruzan pares de soluciones de la población aplicando el operador PLOX (Periodic LOX) para de esta manera obtener nuevos pares de soluciones que son mejorados con operadores de búsqueda local y que pueden formar parte de la nueva población. Debido a la falta de instancias para este tipo de problema, los autores generaron instancias de prueba con las cuales pudieron mostrar los beneficios de usar Scatter Search comparado con los resultados devueltos por el algoritmo heurístico desarrollado.

Lacomme et al. [77] realizaron un estudio general sobre las distintas características a considerar en la modelización de CARP periódicos para que éstos se asemejen a problemas reales. El problema presentado considera un multigrafo mixto, costos de recorrido y de atención de demanda para cada arco/arista, restricciones en la construcción de recorridos (prohibición de giros en U y penalidades de giros), entre otras características. El objetivo del mismo es la minimización de una función que suma el costo total de los recorridos y el número de vehículos usados. Propusieron un algoritmo basado en Algoritmos Meméticos, el cual trabaja con una población de cromosomas de longitud variable donde cada uno de ellos es una lista de secuencias de aristas/arcos requeridos sin delimitador de recorridos (una por cada día). Como fase de mejora, el algoritmo usa LOX adaptado como operador de cruzamiento y mejora de soluciones a través del movimiento de enlaces y 2-OPT entre recorridos. Las pruebas se realizaron sobre adaptaciones de instancias de Golden y Eglese, para asemejarlas a casos de recolección urbana de residuos. El trabajo no incluyó comparación de resultados, debido al escaso material existente sobre el tema tratado.

4.4.4. El Problema de Ruteo de Arcos Capacitados con ventanas de tiempo (CARP-TW)

Otra extensión a CARP tiene que ver con el momento en que se atienden las demandas de las aristas requeridas. CARP-TW (CARP-Time Windows) se define como CARP más la condición que la demanda de cada arista requerida sea atendida dentro de un intervalo de tiempo [tiempo más $temprano_e$, tiempo más $tardio_e$] específico, siendo posible que el vehículo encargado de cumplir con la demanda pueda llegar antes al lugar y esperar allí para luego cumplir con la misma.

El objetivo de CARP-TW consiste en obtener recorridos que sirvan las demandas de las aristas requeridas dentro de sus ventanas temporales, sin exceder las capacidades vehiculares y minimizando la distancia total recorrida.

Reghioui et al. [102] presentaron un algoritmo para CARP-TW basado en GRASP y Path Relinking. El problema tratado por los autores, minimiza la duración total de los recorridos encargados de atender las aristas requeridas. Para la fase constructiva de GRASP, proponen dos algoritmos constructivos: Recorrido de caminos (Path Scanning) con uso de lista RCL y el algoritmo Ruteo-Agrupación (Route-First, Cluster-Second) basado en el Algoritmo de Ulusoy. En la fase de mejora, combinaron Path Relinking con los algoritmos 2-OPT, OR-OPT y SWAP (basado en Ejection Chains [50]), debido a la necesidad de mejorar la calidad de las soluciones obtenidas por Path Relinking. Las pruebas se realizaron sobre instancias del trabajo de tesis de Wohlk (2005), obteniéndose mejoras sobre 4 de 24 instancias.

Johnson y Wohlk [71] han desarrollado dos métodos basados en Generación de Columnas para CARP-TW. En ambos casos, proponen resolver el Problema de Particionamiento de Conjunto (Set Partitioning Problem) para alcanzar soluciones óptimas. El primer método consta de dos fases. La primera consiste en alcanzar una solución factible. Luego, usando dicha solución como criterio de poda del árbol, la segunda fase intenta alcanzar la solución óptima. El segundo método consiste en alcanzar la solución óptima de manera iterativa. A partir de una solución infactible, durante un cierto tiempo se intenta reparar dicha solución y alcanzar la optimalidad. Para evaluar la calidad de ambos métodos, los autores modificaron 20 instancias de las propuestas por Eglese [34]. Se alcanzaron 19 soluciones óptimas, destancándose el método iterativo por su rapidez.

4.5. Aplicación a casos reales

A continuación, se mencionarán algunos trabajos que describen soluciones a casos reales.

4.5.1. Recolección de Residuos

Ghiani et al. [47] realizaron un estudio sobre recolección de residuos domiciliarios para la ciudad de Castrovillari, ubicada al sur de Italia. El problema es una variación de CARP que incluye instalaciones intermedias (sitios de descarga de residuos), ventanas de tiempos para cumplir con el servicio sobre arcos/aristas requeridas y flota de vehículos heterogénea. Los objetivos del estudio se enfocaron en la reducción de rutas, el número de vehículos utilizados y las horas extras del personal a cargo del servicio de recolección.

Se propuso un algoritmo Agrupación-Ruteo (Cluster-First Route-Second), el cual en su primera fase asigna arcos y aristas requeridas a vehículos siempre que se cumplan las condiciones de ventanas temporales. En la siguiente, se determinan los recorridos que realiza cada vehículo considerando restricciones de capacidad vehicular y longitud de recorrido. Según pruebas realizadas sobre instancias reales, se obtuvo una reducción de longitud de recorridos del 10.6 %, una reducción de tiempo de trabajo del 13.5 %, la eliminación de las horas extras y una mejor distribución de cargas entre los vehículos. En términos generales, se obtuvo una reducción del 8 % en términos de costos totales.

4.5.2. Tareas en época invernal

Muyldermans et al. [93] presentaron un trabajo sobre tareas de mantenimiento de rutas en época invernal (Salt Spreading) y muestran resultados sobre un caso de estudio real sobre la provincia belga de Antwerp. El trabajo realizado por los autores consiste en la partición de regiones geográficas en sub-regiones, para de esta manera facilitar la ejecución de tareas como el mantenimiento de rutas o recolección de residuos, dentro de cada región. El problema de ruteo de vehículos tratado es MD-CARP con flota homogénea, demanda conocida y grafo no dirigido. En particular, los objetivos del problema propuesto son la determinación de distritos con bajo solapamiento de rutas que cruzan entre ellos,

la minimización del número de camiones encargados de esparcir sal sobre las rutas y el balanceo de tareas (demanda) entre distritos.

Se propuso a tal efecto, un algoritmo heurístico de cuatro fases consistente en el cálculo de medidas heurísticas para la asignación de ciclos a distritos, asignación inicial, fase de mejora de asignación e interacción del usuario lo cual puede afectar en la determinación de la mejor solución. Las pruebas se realizaron sobre una instancia real de mantenimiento preventivo correctivo de rutas en la provincia de Antwerp, la cual contiene cuatro distritos y un depósito en cada una de ellos. Los resultados obtenidos muestran una reducción de recorrido total de aproximadamente 15 %, lo cual remarca la importancia de un estudio pormenorizado sobre la determinación de distritos para una mejor realización de tareas en vez de considerar a los distritos (o regiones) como información fija e inamovible.

4.5.3. Limpieza de calles

Zhu et al. [126] han presentado un estudio de caso real para la ciudad china de Chongqing. El mismo fue aplicado para la determinación de recorridos a realizar por aspersores, los cuales son utilizados para la limpieza de calles en ambos lados. El problema de ruteo de vehículos asociado es MD-CARP con flota homogénea, demanda conocida y grafo no orientado, siendo la función objetivo la minimización del costo de recorrido total por parte de los aspersores.

Se propuso un algoritmo híbrido genético, el cual a diferencia de la mayoría de los trabajos de ruteo de vehículos basados en Algoritmos Genéticos, construye una población de individuos donde cada uno de ellos es un recorrido y la población, una solución factible al problema. Además, el algoritmo incluye como operadores de cruzamiento a los operadores 1-Punto de Corte y 2-OPT, y el uso de diferentes movimientos de arcos dentro de un mismo recorrido como mecanismo de mutación, resultando una buena alternativa en términos de mejora de soluciones que simplemente aplicar movimientos de arcos en forma aleatoria. Las pruebas computacionales se realizaron sobre información real provista por el Departamento de Sanidad de dicha ciudad, el cual posee dos depósitos desde donde los aspersores (de capacidad homogénea) inician y finalizan sus recorridos, sin restricción acerca del inicio y fin de cada uno de ellos. De acuerdo a los resultados obtenidos, se obtuvieron recorridos con una reducción de costos en términos de distancias del 48 % comparado con resultados reales.

4.5.4. Planificación de servicios en redes ferroviarias

Grover et al. [61] presentaron un trabajo consistente en la planificación de servicios de mantenimiento y mediciones a realizar sobre redes ferroviarias y la determinación del recorrido a realizar por los vehículos a cargo del mismo, aplicado a la red ferroviaria de Sudáfrica. La planificación está relacionada con el momento en que se realizan tareas de servicio sobre diferentes enlaces de la red ferroviaria cuyas frecuencias de servicio anuales varían según el tipo de vía ferroviaria (minera, principal, secundaria y subterránea). El problema de ruteo asociado es RPP con un único vehículo a cargo de cumplir con las demandas (frecuencias de servicio), frecuencia de servicio conocida entre aristas y grafo no orientado. Los objetivos del problema consisten en la minimización de la distancia total recorrida a cargo del vehículo y el cumplimiento de la frecuencia de servicios a realizar sobre determinados enlaces de la red ferroviaria.

Los autores proponen un algoritmo constructivo que determina las aristas de la red ferroviaria a servir por el vehículo durante las jornadas laborales y aplican diferentes operadores de mejora mediante el uso conjunto de una lista tabú. Las pruebas computacionales fueron realizadas sobre información real provista por la empresa a cargo del servicio ferroviario. El beneficio económico y táctico en términos cuantitativos de la mejor solución alcanzada por el algoritmo desarrollado no fue publicado.

Capítulo 5

Algoritmos para el Problema de Ruteo de Arcos Capacitados

En este capítulo, explicaremos nuestras propuestas para CARP basadas en metaheurísticas.

5.1. Un algoritmo HBMO para CARP

Uno de los algoritmos desarrollados para CARP se basa principalmente en la metaheurística Honey-bee Mating Optimization (HBMO), la cual se inspira en el comportamiento social de las abejas, destacándose el trabajo cooperativo que realizan las abejas obreras para el cuidado de las nuevas crías, la reproducción compartida entre abeja reina y zánganos y la supervivencia de la especie en el tiempo.

Básicamente, se han adoptado los lineamientos generales de HBMO (ver Capítulo 3); sin embargo, se proponen nuevas alternativas para la generación de zánganos, cruzamiento y reemplazo de la abeja reina.

5.1.1. Población inicial de zánganos

La población inicial de soluciones (zánganos) como así también la generación y actualización de soluciones diversas y de calidad aplicadas en el algoritmo HBMO propuesto, son ideas basadas en Scatter Search ([51], [80], [88]).

Para medir la diversidad (o distancia) entre soluciones, es necesario definir una función a tal efecto. Para nuestro problema, la diversidad (o distancia) entre dos soluciones es definida como la menor cantidad de elementos entre rutas que deben ser movidos para que dos soluciones sean iguales, en términos de conjunto.

Al comenzar el algoritmo, se genera una población inicial de soluciones pseudo-aleatorias (de manera similar al algoritmo Cheapest Insertion indicado en la sección 5.2.1) las cuales son posteriormente mejoradas usando diferentes algoritmos de búsqueda local. La mejor solución se transformará en la abeja reina inicial mientras que las soluciones más distantes (respecto de la reina), serán los zánganos iniciales. Es importante destacar que disponer de una población de soluciones diversas es una estrategia sencilla de diversificación. No obstante, dicha diversificación es controlada, dado que si no fuese así, la obtención de soluciones competitivas podría alcanzarse pero a un alto costo computacional.

En la Fig. 5.1 observamos un ejemplo de cómo se obtiene la distancia entre dos soluciones CARP.

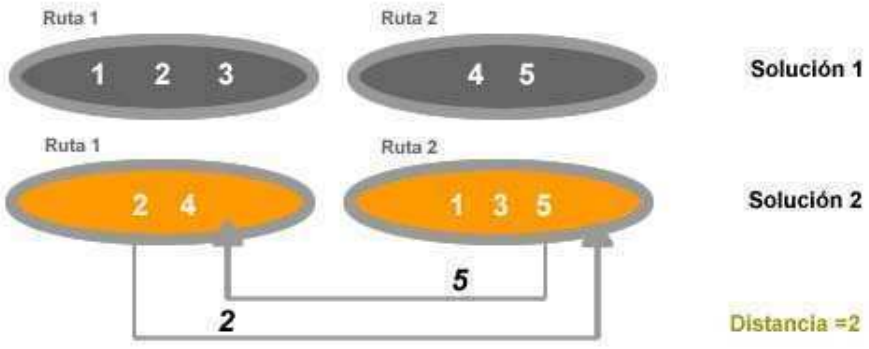

Figura 5.1: HBMO: distancia entre dos soluciones CARP.

5.1.2. Vuelo Reproductivo de la abeja reina

Según pruebas computacionales previas sobre diferentes instancias CARP, para alcanzar buenos resultados a bajo costo computacional (pocos vuelos de la abeja reina, en nuestro caso), es necesario que el número de cruzamientos en cada vuelo entre abeja reina y zánganos, sea alto. Es decir, se necesita una población de mediano o gran tamaño tal que

un número considerable de zánganos satisfagan la ecuación (3.1). Sin embargo, mantener al menos una población mediana redunda en un alto costo computacional.

Como alternativa, se propuso que las nuevas soluciones (crías) sean el resultado de cruzar la abeja reina (mejor solución) con todos los zánganos. Con este cambio y disponiendo de soluciones diversas y calidad, fue posible disminuir el tiempo computacional necesario para alcanzar buenos resultados. Esta propuesta es nueva, dado que no se ha encontrado en la literatura sobre HBMO trabajos con similares características.

5.1.3. Cruce entre abeja reina y zángano

Tal como se indica en Abbas [1], Koudil et al. [75], Curkovic y Jerbic [24], Fathian et al. [36], una vez que el zángano es elegido para copular con la reina, se agrega su esperma a la espermateca de la reina.

Luego, usando dicho esperma y el código genético de la reina, las abejas obreras generarán las nuevas crías por cruzamiento y mutación.

Respecto a operadores de cruzamiento para CARP encontrados en la literatura, podemos mencionar a PLOX (Periodic LOX) usado en el algoritmo Scatter Search para Periodic-CARP de Chu et al. [21] y al operador LOX (Linear Order Crossover) incluido en el algoritmo basado en Algoritmos Meméticos de Lacomme et al. [76].

En nuestra propuesta, consideramos la reproducción entre zángano y abeja reina aplicando el operador de cruzamiento 2-Puntos de Corte (Two-Point Crossover) [122]. Básicamente, dados dos padres *P1* y *P2* (soluciones a un problema dado) y dos puntos de corte aleatorios (posiciones sobre los padres), los hijos *H1* y *H2* (nuevas soluciones) se obtienen tomando los genes anteriores al Corte 1 y los posteriores al Corte 2 del padre *P1* y los genes del padre *P2* ubicados entre los dos puntos de corte. El siguiente hijo *H2*, se obtiene repitiendo las mismas acciones pero cambiando de padre (*P2* por *P1*). Adaptamos el operador de la siguiente manera: dados un zángano y la reina, se eligen dos números aleatorios (puntos de corte) entre 1 y #rutas de la solución. Luego, se toman las rutas de la reina entre los dos puntos de corte y las primeras y últimas rutas del zángano. Una vez realizado el cruzamiento, es posible que la cría obtenida (nueva solución por cruzamiento) posea aristas requeridas repetidas y/o aristas requeridas faltantes. Es por ello, que deben realizarse tareas post-crossover a fin de obtener una solución factible.

En la Fig.5.2 mostramos gráficamente cómo se adaptó el operador para CARP.

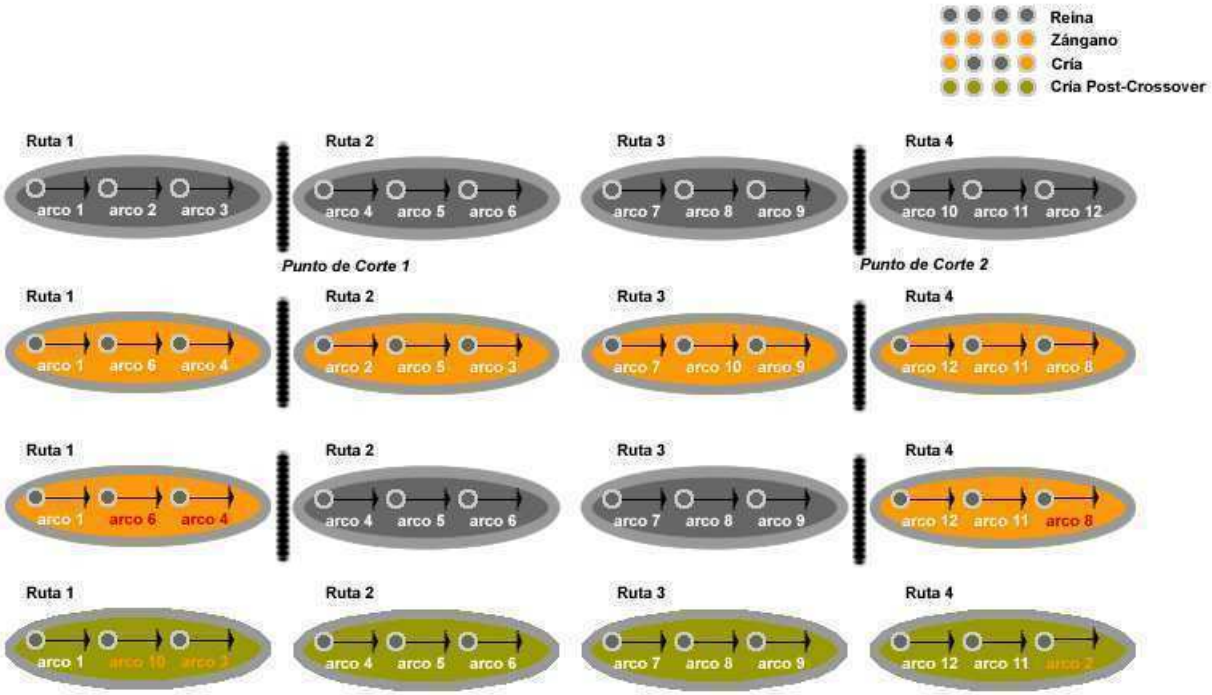

Figura 5.2: HBMO: operador 2-Puntos de Corte adaptado para CARP.

5.1.4. Uso de abejas obreras

El trabajo que realizan las abejas obreras en nuestro algoritmo, es el de mejorar las crías obtenidas luego del cruzamiento, criterio también adoptado en Fathian et al. [36]. En trabajos previos basados en HBMO, la mejora realizada por las abejas obreras se limita en general a aplicar operadores de cruzamiento.

Los algoritmos de mejora aquí aplicados fueron desarrollados originalmente para el Problema del Viajante y variantes. En dichos casos, el problema consiste en visitar (recorrer) nodos, en vez de arcos o aristas [116]. Fueron adaptados algoritmos de mejora inter-ruta e intra-ruta para CARP, de manera de promover la diversificación e intensificación en la búsqueda de mejores soluciones. Los algoritmos que usamos son:

- 2-OPT: este operador de mejora intra-ruta elimina dos conexiones entre aristas requeridas y reconecta la ruta agregando dos nuevas conexiones e invirtiendo el sentido de recorrido de la cadena previamente desconectada. Se ha aplicado este algoritmo para realizar cambios de recorrido en particular sobre rutas con gran

número de aristas/arcos a visitar. En la Fig. 5.3 puede apreciarse el funcionamiento del operador sobre una ruta de una solución CARP.

- Cross-Exchange: este operador intra-ruta intercambia cadenas de aristas requeridas sin invertir el sentido de recorrido. La aplicación de este operador es recomendable en rutas con pocas aristas/arcos a visitar. En la Fig. 5.3 se muestra el funcionamiento de Cross-Exchange.

- String-Cross: consiste en intercambiar cadenas de aristas requeridas entre dos rutas. La diferencia que tiene este operador con String-Exchange consiste en que las cadenas intercambiadas son de mayor longitud. Este operador elimina una conexión por ruta y agrega dos conexiones entre arcos de diferentes rutas. En la Fig. 5.4 se indican las acciones que realiza String-Cross sobre dos rutas de una solución CARP.

- String-Relocation: consiste en mover de una ruta a otra una cadena de aristas que incluyan aristas requeridas, realizando a continuación las reconexiones necesarias. Este operador quita 3 conexiones (en total) y agrega 3 nuevas. La Fig. 5.4 muestra un ejemplo del operador inter-ruta.

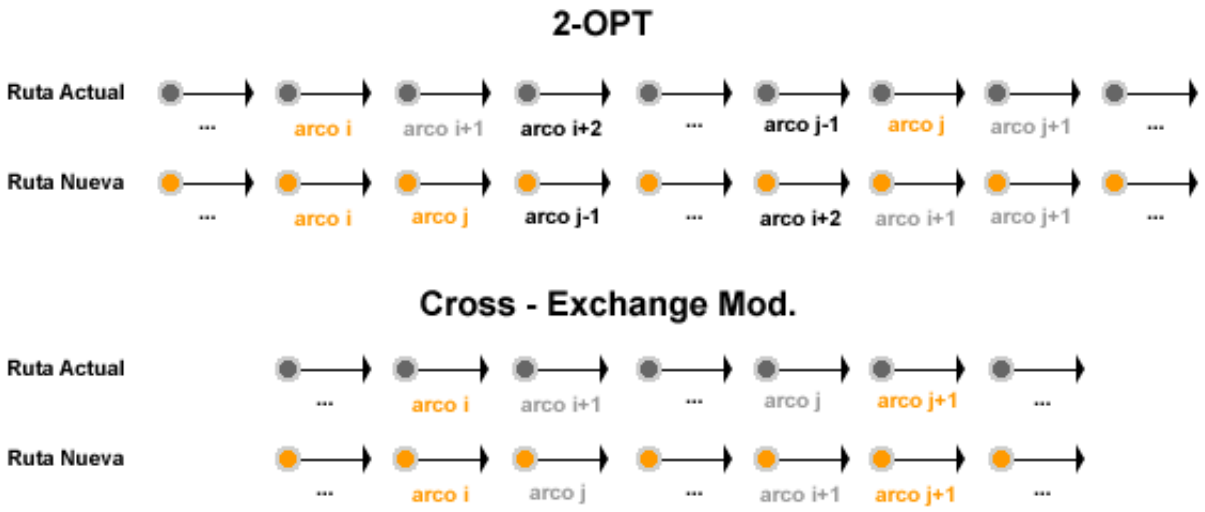

Figura 5.3: HBMO: operadores de mejora intra-ruta.

Es importante destacar que para lograr buenas soluciones para las instancias CARP usadas en las pruebas computacionales, deben perseguirse dos objetivos: reducir el número de rutas y el costo de recorrido total. Durante la etapa de generación de soluciones

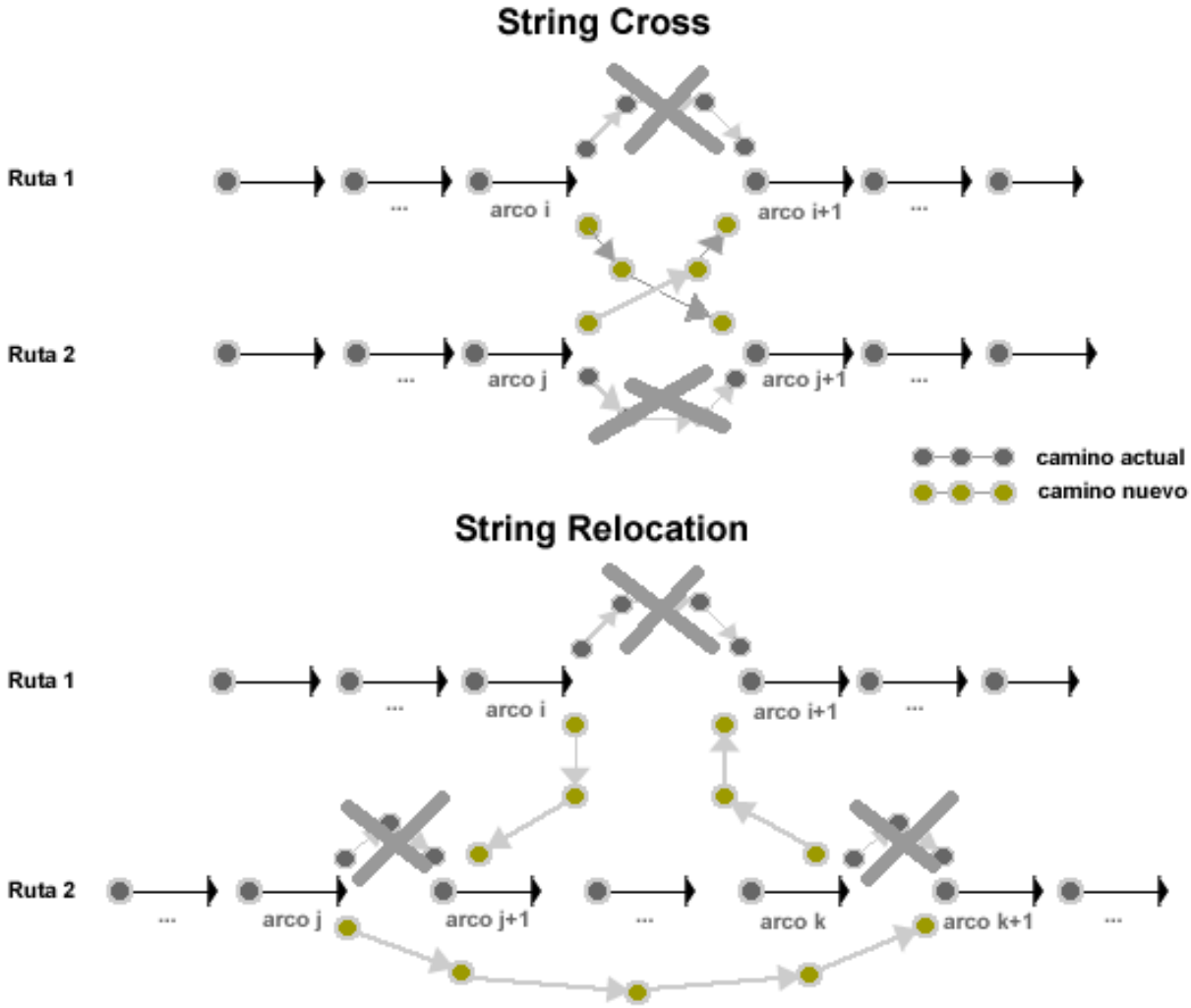

Figura 5.4: HBMO: operadores de mejora inter-ruta.

semi-aleatorias, se busca cumplir el primer objetivo obteniendo soluciones con el menor número de rutas posible. En tanto, mediante el uso de las heurísticas mencionadas, se busca lograr el segundo.

5.1.5. Reemplazo de la abeja reina

La reina actual es reemplazada cuando alguna cría es más fuerte (mejor solución). Es decir, cuando el número de rutas y el costo de recorrido total de una cría (para una instancia CARP) es menor. Luego de la (posible) actualización de la reina, existen otras tareas a realizar antes de repetir los pasos anteriores. Fathian et al. [36] y Curkovic y

Jerbic [24], eliminan todos los zánganos de la generación actual; es decir, en cada vuelo de la abeja reina, se trabaja con una nueva población de zánganos.

En vez de eliminar crías, proponemos analizar la calidad y diversidad de las mismas para reemplazar a los zánganos actuales con menor calidad y/o diversidad. Esta idea se basa en la fase de actualización del *conjunto de referencia* de Scatter Search [88]. Aplicando esta idea en nuestro algoritmo HBMO, consideramos que una cría reemplazará a un zángano cuando ésta tenga calidad de solución cercana a la reina y no supere cierta distancia con la reina (calidad) ó cuando la cría tenga calidad de solución no inferior a cierto límite y la distancia con la reina sea superior a otro límite, también pre-establecido (diversidad).

5.1.6. Algoritmo HBMO para CARP

En el Cuadro 5.1, se muestra un pseudo-código del algoritmo HBMO propuesto. Los parámetros que necesita son el número de vuelos reproductivos, el número de zánganos, el número máximo de soluciones iniciales a generar y el número máximo de vuelos reproductivos sin mejora antes de aplicar el reinicio del algoritmo.

Al comienzo del algoritmo, se genera una cantidad inicial de soluciones semi-aleatorias que luego son mejoradas. La mejor solución será la reina inicial y las más diversas (a la reina) serán los zánganos iniciales. Luego, durante cada vuelo, la reina actual se cruzará con todos los zánganos, obteniéndose la misma cantidad de crías. Dichas crías serán mejoradas con algoritmos de búsqueda local. Si alguna de las crías recién obtenidas es mejor solución que la reina actual, se hará el reemplazo correspondiente. Antes que la reina (actual o nueva) realice un nuevo vuelo, los zánganos actuales pueden ser reemplazados por crías de mejor calidad y/o diversidad. Si el algoritmo no obtuvo una mejor solución durante un determinado número de vuelos reproductivos, se activa el mecanismo de reinicio consistente en reemplazar los zánganos actuales por otros generados de la misma forma que al inicio del algoritmo. Una vez realizados todos los vuelos, se retorna la mejor solución alcanzada.

Cuadro 5.1: Algoritmo HBMO para CARP.

```
procedure HBMO-CARP(#flights, #drones, #MaxTries, #nonImprovement)

Solutions={}
GenerateSolutions(Solutions,#MaxTries,#drones)
queenPos =0, nonImprovement=0
for i = 1 ... #flights
  bestOF= f(Solutions[queenPos]))
  bestPos=-1
  for j = 1 ... #drones
    Solutions[#drones+j]=Crossover(Solutions[queenPos], Solutions[j])
    Solutions[#drones+j]=LocalSearch(Solutions[#drones+j])
    if f(Solutions[#drones+j])<bestOF then
      bestOF=f(Solutions[#drones+j])
      bestPos=#drones+j
    end if
  end for
  if bestPos>-1 then
    updateQueen(Solutions, queenPos, bestPos)
    nonImprovement=0

  else
    nonImprovement++
  end if
  if nonImprovement<#nonImprovement then
    updateDrones(Solutions,#drones, 2*#drones)
  else
    GenerateSolutions(Solutions,#MaxTries,#drones)
  end if

end for
return Solutions[queenPos]
end procedure
```

5.2. Un algoritmo VNS híbrido para CARP

En este caso, nuestra propuesta consiste en la construcción de soluciones mediante la heurística Inserción más Económica (Cheapest Insertion) y otra basada en GRASP [106].

Luego, las soluciones son mejoradas usando diferentes operadores de búsqueda local [116], activados en el marco de Variable Neighborhood Search [64] y Path Relinking [50]. Como mecanismo de escape de óptimos locales, usamos una memoria de largo plazo y una lista RCL basada en valor, para fomentar la exploración de nuevas regiones de soluciones.

5.2.1. Fase constructiva

Algoritmo heurístico basado en GRASP

Teniendo en cuenta que una solución para CARP consiste de un conjunto de rutas que comienzan y terminan en el depósito (es decir, un arco con vértice inicial igual al depósito), las cuales son en síntesis, un recorrido de arcos requeridos (y no requeridos), el algoritmo basado en la fase constructiva de GRASP propuesto es el siguiente: mientras existan arcos requeridos que demanden servicio, siempre que sea necesario se inicia una nueva ruta. Luego, cada arco requerido incluido en la misma, se conecta con un (nuevo) arco requerido no visitado si su demanda no excede la capacidad restante del vehículo. Se agregan arcos requeridos no visitados en la ruta hasta que la misma deba ser cerrada; es decir, hasta que deba construirse un camino de retorno hacia el depósito. Esto da inicio a una nueva ruta, si quedan arcos requeridos sin visitar.

La selección del nuevo candidato a agregar a la solución en construcción, es realizada mediante el uso de una Lista Restringida de Candidatos (RCL) basada en valor. La particularidad de esta lista es que los elementos incluidos son aquellos que se encuentran dentro de un intervalo numérico dado por los valores de la función miope usada. En nuestro caso, se han propuesto 3 funciones que son usadas durante la ejecución del algoritmo VNS propuesto. Las funciones son:

$$\frac{\text{Costo conexión}(vert_j, vert_i) + \text{Costo arco}}{\text{Costo arco}} \tag{5.1}$$

$$\frac{\text{Costo arco}}{\text{Demanda arco}} \tag{5.2}$$

$$(\text{Frecuencia arco}+1) \times \text{Costo conexión}(vert_j, vert_i) + \text{Costo arco} \tag{5.3}$$

Donde:

- Costo conexión($vert_j, vert_i$) es el costo de conexión entre el vértice j del último arco requerido visitado y el vértice i del nuevo arco a insertar en la ruta en construcción.
- Costo arco es el costo de visita (o recorrido) del nuevo arco a insertar.
- Demanda arco es la demanda exigida por el nuevo arco a insertar.
- Frecuencia arco es la frecuencia de uso (en las soluciones construidas hasta el momento) del nuevo arco a insertar.

Respecto a las funciones, en 5.1 se busca elegir arcos cuyo costo de conexión entre el último arco y el nuevo a insertar sea bajo; en 5.2, elegir arcos con bajo costo de recorrido y de gran demanda; en 5.3, arcos con baja de frecuencia de uso y costo de conexión.

Dadas las características de determinadas instancias CARP, es posible que la cantidad de elementos incluidos en las listas RCL sea baja. Esto puede ocasionar que la selección de elementos candidatos se torne miope, dando como resultado soluciones similares a lo largo de la ejecución de la fase constructiva de soluciones. Si esto ocurre, se asigna a cada uno de los elementos candidatos un valor de probabilidad de manera tal que al momento de elegir un nuevo elemento (a formar parte de la solución), se favorezca a aquellos con mejor valuación según la función miope que se esté aplicando. Esta selección de elementos candidatos se basa en la construcción de listas RCL basada en valores con sesgo. Para más detalles, se recomienda [106].

Finalmente y no menos importante, en caso que la conexión entre dos arcos no sea directa, se aplica el Algoritmo de Dikjstra [22] a tal efecto.

Algoritmo Inserción más Económica Modificado

El segundo algoritmo usado para la construcción de soluciones es Inserción más Económica (Cheapest Insertion Heuristic). Como veremos a continuación, es sencillo, rápido y con un enfoque complementario respecto al primero. El algoritmo es el siguiente:

- Paso 1: Creación de rutas vacías. Inicialmente, se crea una solución CARP que consiste de rutas con arcos que indican el inicio (de la ruta) desde el depósito y el retorno al mismo.
- Paso 2:

- Paso 2a: Inserción de arcos. Mientras existan arcos requeridos aún sin ser visitados, todo arco es insertado (contemplando restricción de capacidad vehicular) en alguna ruta si el costo asociado a su inserción en la misma es mínimo. Si es necesario, volver al Paso 1.
- Paso 2b. Mejora de la solución en construcción. Cada K inserciones, la solución es mejorada con algoritmos inter-ruta e intra-ruta para reducir el costo de recorrido total y el número de rutas.

En base a lo anterior, puede asumirse que el algoritmo siempre alcanza la misma solución. Es por ello, que se propuso una variación del algoritmo. Antes de iniciar la inserción de arcos en las diferentes rutas, la lista (de arcos) es *perturbada* aleatoriamente. De esta manera, se pueden obtener diferentes soluciones. A su vez, en lo que respecta a la aplicación de los algoritmos de mejora, durante el proceso de inserción de arcos se usaron algoritmos de mejora intra-ruta y una vez obtenida la solución completa, los de mejora inter-ruta. Es decir, la reducción de costo de recorrido y el número de rutas usadas son objetivos que se buscan en momentos diferentes.

5.2.2. Fase de mejora

La mejora de soluciones se consigue a través de dos acciones específicas: reducción de la cantidad de rutas y reducción del costo total de recorrido. Para lograr esto, se aplicaron diferentes operadores de mejora, los cuales son activados dentro de Variable Neighborhood Search y Path Relinking.

Algoritmos de mejora de rutas

Reducción de rutas

La reducción de rutas en soluciones CARP es realizada considerando dos acciones específicas basadas en la técnica Ejection Chains [50]. La primera consiste en mover todo un bloque de arcos requeridos de una ruta y anexarla a otra, siempre que sea factible en términos de capacidad de carga. Con esta acción, es posible eliminar rutas de la solución actual. La segunda, consiste en mover arcos de una ruta a diferentes rutas. Esta acción,

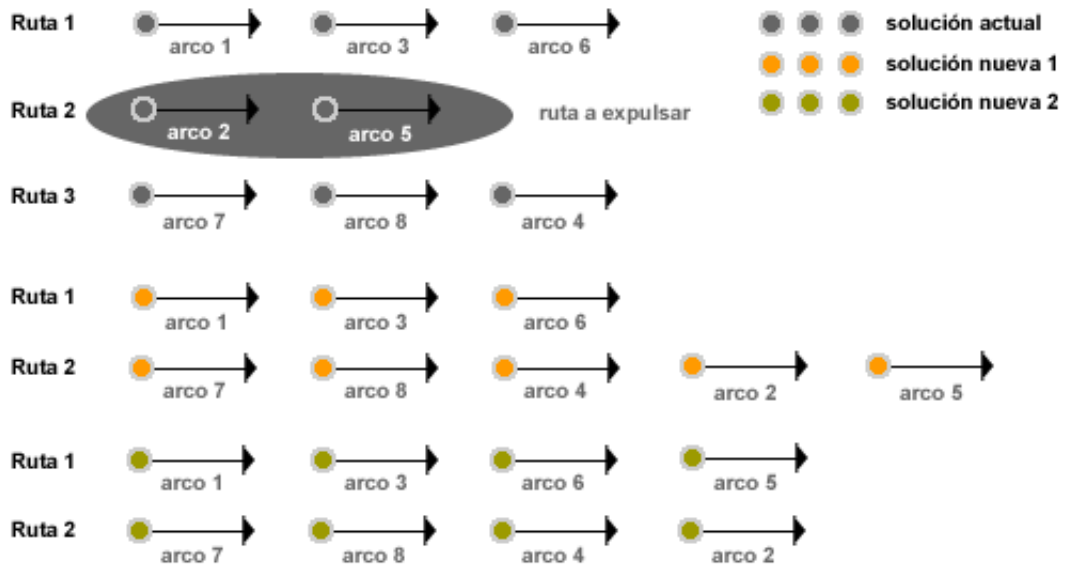

Figura 5.5: VNS-Fase de mejora: operadores basados en Ejection Chains.

cuya complejidad algorítmica es mayor que la anterior, es aplicada siempre que la primera no haya tenido éxito. Ver Fig. 5.5.

Reducción del costo de recorrido total

Se realiza a través de operadores que intentan reducir costos realizando movimientos dentro de una misma ruta como también entre rutas. La aplicación de algoritmos intra-ruta permite intensificar la búsqueda de soluciones dentro de un mismo vecindario. Los algoritmos intra-ruta usados son 2-OPT, 2.5OPT, 3-OPT ([31], [116]) y una variación de Cross-Exchange ([72], [110]).

A través del uso de algoritmos inter-ruta se busca diversificar la búsqueda de mejores soluciones. Los algoritmos de mejora inter-ruta usados son String Cross y String Relocation, descriptos en la sección 5.1.4.

Variable Neighborhood Search

Tal como se ha indicado en el Capítulo 3, Variable Neighborhood Search [64] es una metaheurística cuya premisa para resolver problemas de optimización, es la búsqueda de soluciones competitivas a través de diferentes vecindarios.

Las estructuras de vecindarios incluídas en nuestro algoritmo son:

- Intercambiar J arcos requeridos. Dada una solución CARP consistente de varias rutas, la estructura de vecindario propuesta realiza el intercambio de J arcos requeridos entre dos rutas de la solución. Una vez realizado un intercambio, se aplican operadores de mejora intra-ruta sobre las rutas modificadas para así intensificar la búsqueda en su entorno.

- Eliminar K rutas. Usando diferentes opciones, la estructura de vecindario propuesta consiste en eliminar K rutas de una solución CARP. Una vez realizada la acción anterior, se insertan los arcos requeridos *eliminados* en las rutas existentes o en nuevas, en caso de ser necesario. Finalmente, se aplican operadores de mejora inter-ruta y el operador basado en Ejection Chains anteriormente explicado. Con esto, se intenta principalmente diversificar la búsqueda en nuevos entornos de solución.

En las Fig. 5.6 y 5.7 se muestran las estructuras de vecindarios propuestas y en el Cuadro 5.2 el pseudo-código de la fase de mejora aplicando VNS.

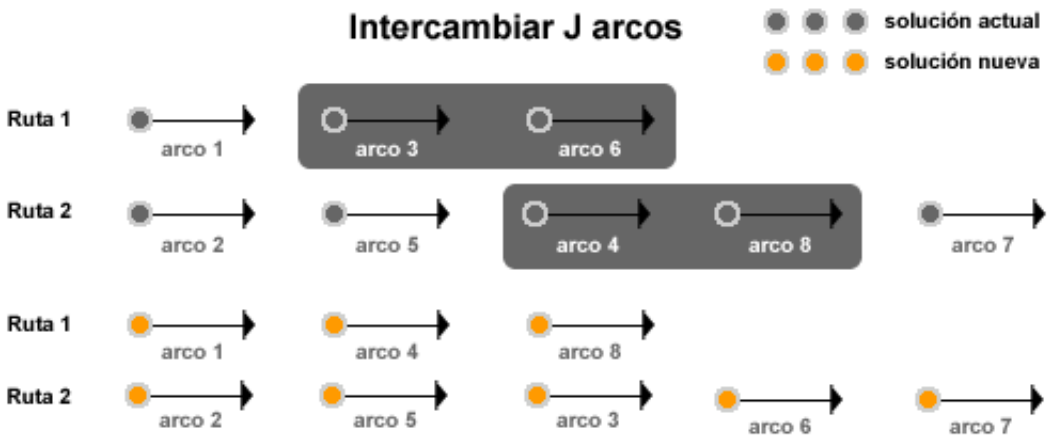

Figura 5.6: VNS-Estructura de vecindario: Intercambiar J arcos.

Cuadro 5.2: Algoritmo VNS-Fase de Mejora.

```
procedure VNS-improvement(prevSolution, arcList, delOption, maxJ, maxK)

bestSolution = {}
bestOF = f(prevSolution)
localOptima = False, k = 1
currSolution = prevSolution
while k ≤ maxK do
  partialSolution = deleteKRoutes(currSolution, delOption)
  currSolution = addArcsToSolution(partialSolution, arcList)
  currSolution = applyInterROperators(currSolution)
  j=1
  while j ≤ maxJ do
    currSolution = addArcExchange(currSolution, j)
    currSolution = applyIntraROperators(currSolution)
    if f(currSolution) < bestOF then
      bestOF = f(currSolution)
      bestSolution = currSolution
      j = 1, localOptima = True
    else
      j++
    end if
    if localOptima == True then
      k=1, localOptima = False
    else
      j++
    end if
  end while
end while
return bestSolution
end procedure
```

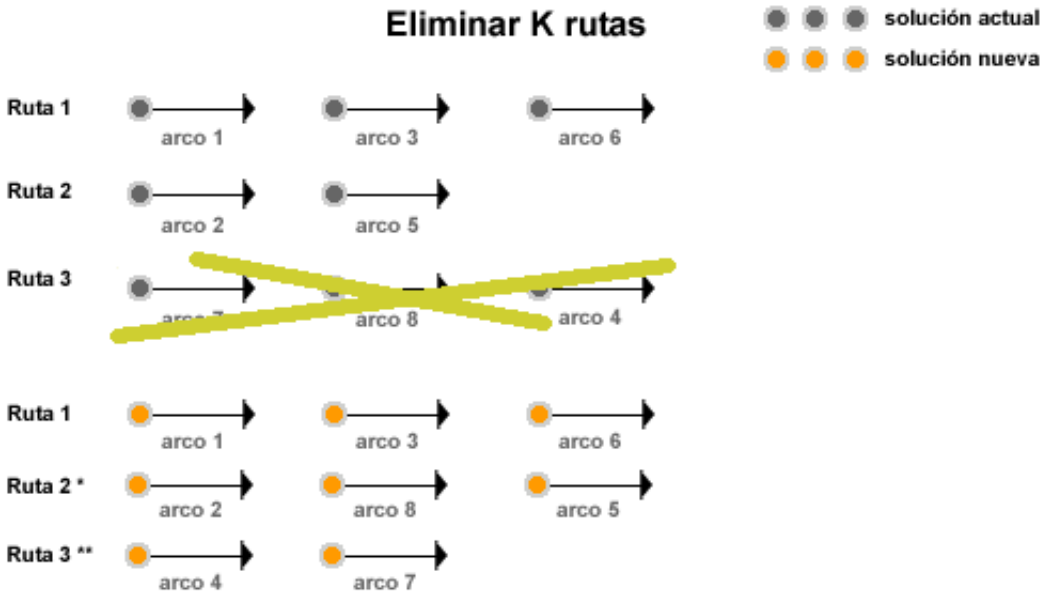

Figura 5.7: VNS-Estructura de vecindario: Eliminar K rutas.

Path Relinking

En nuestra fase de mejora, Path Relinking (Ver Sección 3.5.3) es activado cada vez que se logra un número determinado de soluciones de *referencia* (es decir, en forma periódica). Una vez activado, se intenta que funcione como estrategia de intensificación usando óptimos locales como soluciones iniciales y guía. A su vez, eligiendo soluciones diversas como solución inicial y óptimos locales como solución guía, la estrategia de diversificación es alcanzada. Dada una solución inicial y una solución guía, se aplicó de la siguiente manera:

- Paso 1: Se transforman las dos soluciones, en dos listas de conjuntos (una por solución) de número enteros, donde cada conjunto i-ésimo se corresponde con la ruta i-ésima (de la solución) y cada número k-ésimo (en el conjunto) con un arco requerido atendido en la ruta i-ésima.

- Paso 2: Mapeo de conjuntos de las soluciones. A los efectos de iniciar Path Relinking, se realiza el mapeo entre cada conjunto de la solución inicial con uno de la solución guía tal que los conjuntos relacionados tengan la mayor cantidad de elementos en común.

- Paso 3: Obtención del camino entre solución inicial y guía. Hasta que cada conjunto de la solución inicial tenga los mismos elementos que su conjunto asociado de la solución guía, se realizan las siguientes tareas:

  - Paso 3a: Movimiento de elementos. Si un elemento *i1* del conjunto i de la solución inicial no está en el conjunto asociado de la solución guía, se determina el conjunto j de la solución inicial donde debería estar. Luego, se determina si del conjunto j algún elemento *j1* debe ubicarse en el conjunto i. Si es así, se realiza el intercambio de elementos entre los dos conjuntos; Sino, el elemento *i1* es movido al conjunto j.
  - Paso 3b: Obtención de la solución CARP. Luego de realizado un movimiento de elementos entre conjuntos de la solución inicial, se obtiene una solución CARP la cual puede o no ser factible en términos de capacidad de carga.
  - Paso 3c: Mejora de la solución. Sobre soluciones factibles, se aplican operadores de mejora intra-ruta para de esta manera reducir el costo de recorrido. No se aplicaron operadores inter-ruta ya que el objetivo a alcanzar es lograr que las soluciones construidas desde la solución inicial tiendan a la solución guía.

Finalmente, es importante remarcar que las acciones realizadas por Variable Neighborhood Search y Path Relinking para mejorar las soluciones obtenidas en la fase constructiva, son de tipo complementarias. Esto le permite al algoritmo alcanzar soluciones (al problema) desde diferentes enfoques y reducir de esta manera, el estancamiento en óptimos locales.

5.2.3. Mecanismo de escape de óptimos locales

Nuestra propuesta usa memoria a largo plazo de manera tal de considerar la transición y residencia de componentes de solución. De esta manera, es posible explorar nuevas regiones de soluciones poco visitadas como también analizar aquellas componentes que ingresan o salen de las soluciones. La memoria a largo plazo usada registra la cantidad de veces que el arco i se conecta con el arco j (residencia) como también las veces que el arco i conectado con el arco j, se conecta a posterior con el arco k (transición). Ver Fig. 5.8.

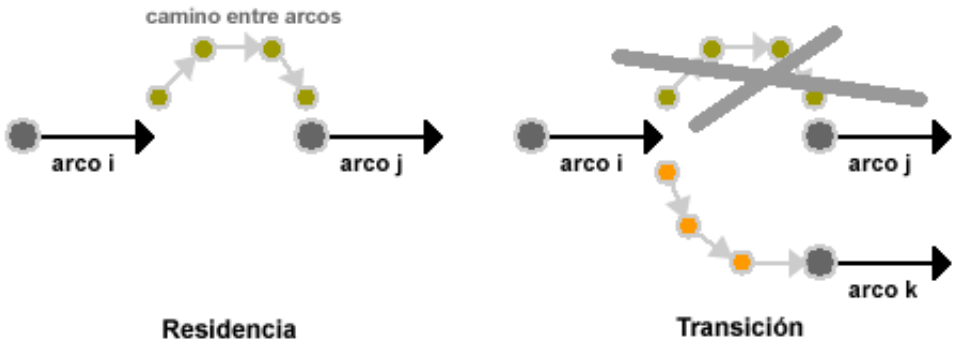

Figura 5.8: VNS-Mecanismo de escape: memorias basadas en frecuencia.

El uso de la información almacenada en la memoria de largo plazo permite explorar regiones (de soluciones) poco visitadas. No obstante, dicha tarea debe ser controlada de manera de reducir la posibilidad de búsqueda en regiones poco atractivas que generen un esfuerzo computacional innecesario. Es decir, se necesita realizar una búsqueda de manera sistemática que explore nuevas regiones pero que a la vez considere el costo computacional asociado. Para ello, proponemos la construcción de nuevas soluciones mediante un algoritmo basado en GRASP con dos funciones miopes que utilizan información de largo plazo. Las funciones son:

$$\frac{\text{Residencia(arco actual, arco)}}{\#\text{iteración}} \tag{5.4}$$

$$\frac{\frac{\text{Costo arco}\times\text{Residencia(arco actual, arco)}}{\#\text{iteración}} + \text{Costo conexión(arco, depósito)}}{\text{Costo arco}} \tag{5.5}$$

Donde:

- Residencia (arco actual, arco) registra el número de veces (hasta el momento) que el arco actual (en la solución actual) se conectó con el nuevo arco a insertar durante las diferentes soluciones construidas o mejoradas.
- #Iteración actual del algoritmo.
- Costo arco es el costo de visita (o recorrido) del nuevo arco a insertar.

- Costo conexión (arco, depósito) es el costo entre el nuevo arco a insertar en la solución y el depósito.

Explicaremos la relación entre memoria de largo plazo y GRASP dentro del mecanismo de escape de óptimos locales. Mientras queden arcos requeridos por visitar, el nuevo arco a agregar a la solución, se elegirá al azar desde una lista restringida de candidatos RCL basada en valor. De acuerdo a la capacidad vehicular restante que posee el vehículo que realiza el recorrido de la ruta en construcción, para incorporar un elemento candidato a la lista RCL, se usará la función 5.4 que mide el porcentaje de uso de conexión del arco candidato (respecto del número de iteraciones) con el último arco insertado en la solución o 5.5 que relaciona el costo de retornar al depósito desde el arco a insertar y su frecuencia de uso hasta el momento. Como puede observarse, la construcción de soluciones (en esta fase) está basada en el algoritmo Exploración de Caminos (Path Scanning).

Todas las soluciones construidas por el algoritmo, son mejoradas con algunos operadores de búsqueda local antes de ingresar a la fase de mejora. Este mecanismo de escape es aplicado sólo cuando el algoritmo VNS híbrido no logra obtener mejores soluciones durante un número determinado de iteraciones.

5.2.4. Algoritmo VNS híbrido para CARP

En el Cuadro 5.3 se muestra el pseudo-código del algoritmo propuesto. Los parámetros son el número de iteraciones, el tamaño del Conjunto de Referencia para Path Relinking, α para la selección de candidatos de la lista RCL, K usado por el algoritmo Inserción más Económica modificado y el número de iteraciones sin mejora para activar el mecanismo de escape.

En cada iteración, se construyen soluciones las cuales son mejoradas mediante la fase de mejora de VNS o Path Relinking. Si durante un número de iteraciones no se obtuvo mejora, se construirán soluciones usando información de largo plazo. Finalmente, se devuelve la mejor solución alcanzada.

Cuadro 5.3: Algoritmo VNS híbrido para CARP.

```
procedure VNS-CARP(#iterations,#refSet,α,K,#nonImprovement)
 bestOF=∞, bestSolution = {}, refSolutions={}
 nonImprovement=0, flag=false, counter=0
 for i = 1 ... #iterations
   if flag==true then
     currSolution=greedyBasedOnGrasp&Memory(α, i)
     counter++
     if counter==#nonImprovement then
       counter=0, flag=false, nonImprovement=0
     end if
   else
     if i mod 2==0 then
       currSolution=greedyBasedOnGrasp(α, greedyFunction)
     else
       currSolution=cheapestIHM(K)
     end if
   end if
   AnalyzeSolution(currSolution, refSolutions)
   if |refSolutions| <#refSet  then
     currSolution=VNSImprovementPhase(currSolution)
   else
     currSolution=PathRelinking(refSolutions)
   end if
   if f(currSolution)<bestOF  then
     bestOF = f(currSolution), bestSolution = currSolution
     nonImprovement = 0
   else
     if flag==false then
       nonImprovement++
     end if
   end if
   if nonImprovement==#nonImprovement then
     flag=true
   end if
 end for
 return bestSolution
end procedure
```

5.3. Un algoritmo BRKGA para CARP

A continuación, explicaremos otro de los algoritmos desarrollados para CARP, el cual adopta ciertos lineamientos propuestos por la literatura (Gonçalves [56], Gonçalves y Al-

meida [57], Gonçalves y Resende [59], entre otros) pero además incluye algunas variaciones respecto a la generación de soluciones, la mutación y se propone una codificación de claves aleatorias para el problema de estudio.

5.3.1. Codificación y decodificación

La codificación de cadenas de claves aleatorias utilizada para CARP se basa en lo propuesto por la literatura (Samanlioglu et al., [112]). Cada cadena de claves aleatorias se compone de números enteros entre $MinU$ y $MaxU$, siendo ambos los límites numéricos para la generación de las claves. La longitud de las cadenas es igual al número de arcos requeridos de la instancia CARP correspondiente. Es decir, cada cadena representa a una solución CARP.

En cuanto a la codificación específica de cadenas de claves aleatorias para CARP, no es la misma que la codificación (de claves) aplicada para el Problema del Viajante (ver sección 3.8.2). Para ello, se propone incluir marcas de *fin de ruta*(FR) en las cadenas de claves aleatorias. De esta manera, es posible detectar rápidamente el conjunto de rutas que forman parte de la solución.

Consideremos el siguiente ejemplo. Sea una instancia CARP con 6 arcos requeridos cada uno con demanda=1, capacidad vehicular=4 y $MinU$=50 y $MaxU$=400 para la generación de claves aleatorias. Dada la siguiente cadena:

Random keys: 120 85 31 FR 367 55 107

si la ordenamos (en forma ascendente) y mantenemos fija la posición de la marca de fin de ruta, obtenemos la siguiente solución:

Solución CARP (2 rutas): 3→5→2, 6→1→4

Tal como puede observarse, la codificación propuesta resulta sencilla para el cruzamiento entre cadenas de claves aleatorias, como así también para efectuar una mejora de la solución CARP (decodificada) mediante operadores de búsqueda local y realizar luego, la actualización de la cadena inicial.

5.3.2. Población inicial

Cada cadena de claves aleatorias de la población inicial es obtenida a través de los siguientes pasos:

1. Generación de la cadena inicial: las claves de la cadena son obtenidas aleatoriamente dentro de un rango numérico $[MinU, MaxU]$, siendo el tamaño de la cadena igual a la cantidad de arcos requeridos de la instancia CARP correspondiente.

2. Decodificación de la cadena inicial: ordenando las claves de la cadena en forma ascendente, se obtiene una solución aleatoria CARP sin considerar la restricción de capacidad vehicular.

3. Obtención de solución CARP: la solución es obtenida aplicando el algoritmo Particionamiento Iterativo de Tour (Iterated Tour Partitioning), propuesto por Haimovich y Kan [63]. Comenzando desde una posición inicial de la solución ($\forall i = 1 \ldots |\text{solución}|$), ésta se divide en varias rutas cumpliendo con las restricciones vehiculares obteniéndose así una solución CARP factible. Este proceso se repite varias veces. Luego, la mejor solución en términos de función objetivo, es devuelta.

4. Mejora de la solución CARP: aplicando una fase de búsqueda local (sección 5.3.6), la solución es mejorada.

5. Codificación de la solución CARP mejorada: dada la cadena inicial la cual generó una secuencia de arcos requeridos y la solución CARP mejorada, se realiza el mapeo inverso para así obtener una cadena de claves inicial, la cual ahora incluye marcas de fin de ruta.

Es importante mencionar que en el Paso 3, el valor de capacidad vehicular es multiplicado por un factor (entre 0 y 1), que varía a lo largo de la ejecución del algoritmo BRKGA. Esto genera soluciones iniciales para los algoritmos de mejora (en el Paso 4) con tamaños variables, permitiendo un mejor balanceo de trabajo entre operadores de mejora intra-ruta e inter-ruta.

5.3.3. Operadores genéticos

En cada generación del algoritmo propuesto, la nueva población se obtiene aplicando los siguientes operadores genéticos:

- Reproducción: una determinada cantidad de cadenas de claves aleatorias proviene directamente de la generación anterior aplicando este operador. Ordenando las cadenas en base a los valores de función objetivo (*fitness*), las cadenas clasificadas de tipo Elite son directamente copiadas a la generación actual.
- Cruzamiento: usando cadenas de tipo Elite y No-Elite de la generación anterior, se obtienen nuevas aplicando cruzamiento.
- Mutación: a través de la mutación de cadenas de la generación anterior, se obtiene una determinada cantidad de nuevas cadenas.

5.3.4. Cruzamiento

Como se indicó anteriormente, el operador de cruzamiento usado es el propuesto en la literatura [112].

Antes de aplicar el operador para la obtención de nuevas cadenas, todas las cadenas de la generación anterior son ordenadas según sus respectivos valores de fitness. Luego, éstas son clasificadas en Elite, No-Elite y cadenas a mutar. Tomando una cadena de tipo Elite y otra de tipo No-Elite, se procede a aplicar el operador *Parameterized Uniform Crossover*. La nueva cadena se formará entonces con claves de la primera con probabilidad *ProbE* y con claves de la segunda con probabilidad (*1-ProbE*). En la sección 3.8.2, se muestra un ejemplo de uso del operador.

La selección de cadenas para el cruzamiento es determinística. Es decir, dado que las cadenas están ordenadas por *fitness*, se elige de manera secuencial las cadenas de ambos tipos. En base a los porcentajes de cadenas de diferentes tipos que debe tener cada población (especificado en cada corrida del algoritmo), es posible que una cadena de tipo Elite se cruce con varias de tipo No-Elite.

Una vez aplicado el operador de cruzamiento, debe controlarse que la solución CARP obtenida mediante la decodificación de la nueva cadena, sea factible. Para ello, se controlan

que todas las demandas de las rutas de la solución no excedan la capacidad vehicular. Si esto ocurre, cada ruta excedida se divide hasta que alcance la factibilidad.

Luego del control anterior, la solución CARP es mejorada con algoritmos de búsqueda local. Finalmente, es codificada como una cadena de claves aleatorias.

5.3.5. Mutación

A diferencia de lo propuesto en la literatura sobre la creación de nuevas cadenas de claves aleatorias (mutantes) en cada generación ([56], [112]), nuestro algoritmo aplica el operador clásico de mutación sobre una cierta cantidad de cadenas previamente clasificadas de la generación anterior. Se aplicó el operador de esta manera debido a que el reemplazo de algunas claves pertenecientes a la cadena a mutar, genera una solución CARP diferente respecto a la solución proveniente de la decodificación de la cadena original. Esta situación puede observarse en la Fig. 5.9, sobre una instancia para el Problema del Viajante.

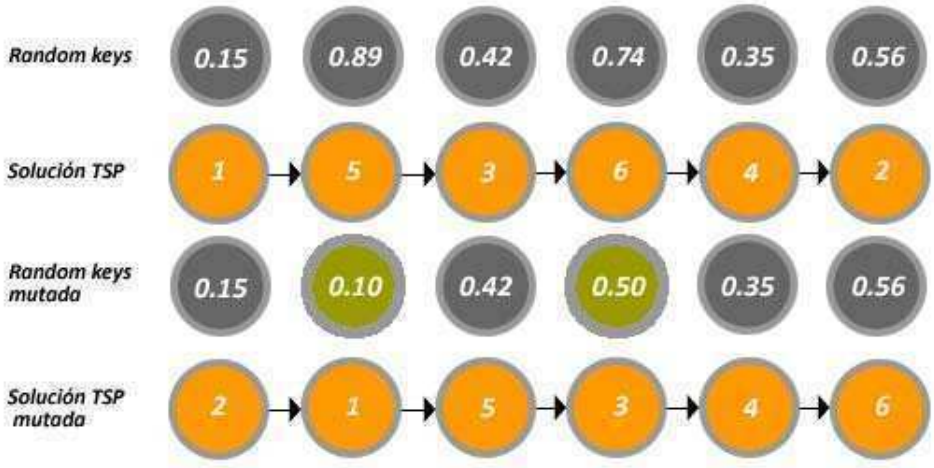

Figura 5.9: BRKGA: mutación sobre una cadena de claves aleatorias.

Una vez modificada una determinada cantidad de claves (la cual varía durante la ejecución del algoritmo) sobre una cadena a mutar, la solución CARP asociada es mejorada con operadores de búsqueda local. Mapeando esta última, se obtiene la cadena de claves final.

5.3.6. Búsqueda local

Los operadores de búsqueda local (Snyder y Daskin [114]) han provisto a los algoritmos basados en Algoritmos Genéticos, la posibilidad de obtener resultados atractivos en menor tiempo computacional. Incorporando una fase de mejora es posible una estrategia de intensificación eficiente.

La fase de mejora incluida en nuestro algoritmo que se aplica sobre soluciones CARP obtenidas luego del cruzamiento y la mutación de cadenas, está basada en las ideas propuestas en el trabajo de Beullens et al. [14]. Consiste en aplicar operadores de mejora inter-ruta e intra-ruta mediante el uso de lista de vecinos de arcos y una estrategia de inspección de determinados arcos tal que el proceso de búsqueda de soluciones se realice de manera eficiente.

Una vez obtenida la lista de vecinos, la cual consiste de una lista de arcos requeridos vecinos para cada arco requerido, se procede a explorar el entorno de una solución inicial de la siguiente manera:

- Inicio: si la lista está vacía, entonces Retornar mejor solución; sino elegir el primer arco e de la lista y luego ir a Siguiente. Todos los arcos de la lista de vecinos son etiquetados como *no examinados*.
- Siguiente: elegir el primer arco *vecino(e)* de la lista de vecinos del arco e. Si todos los arcos de la lista del arco e fueron examinados, ir a Desmarcar; sino determinar si ambos arcos están en la misma ruta de la solución CARP. Si es así, ir a Mejora-1R, sino ir a Mejora-2R.
- Mejora-1R: determinar si es posible mejorar la solución actual aplicando el operador de mejora intra-ruta (ver Cuadro 5.4). Si es posible, aplicar el movimiento, marcar los arcos involucrados, guardar la nueva solución y luego ir a Inicio. Sino, el arco *vecino(e)* es ahora examinado. Ir a Siguiente.
- Mejora-2R: determinar si es posible mejorar la solución actual aplicando los operadores de mejora inter-ruta (ver Cuadro 5.4). Si es posible, aplicar el movimiento, marcar los arcos involucrados, guardar la nueva solución y luego ir a Inicio. Sino, ir a Siguiente para examinar el arco *vecino(e)*.
- Desmarcar: eliminar el arco e de la lista de vecinos y luego ir a Inicio.

La estrategia de marcar arcos tiene como finalidad intensificar la búsqueda sobre arcos cuyas conexiones permitieron obtener una mejor solución en términos de la función objetivo. Si el arco marcado tras una mejora no se encuentra actualmente en la lista, éste es agregado (con sus vecinos etiquetados como *no examinados*).

Cuadro 5.4: BRKGA-Fase de mejora: estrategia de marcado de arcos.

Posición del arco e	Posición del arco *vecino(e)*	Movimiento	Marcar arcos
i	j	2-Opt(i, i+1, j, j+1)	arco(i+1), arco(j), arco(j+1)
j+1	i+1	2-Opt(i, i+1, j, j+1)	arco(i), arco(i+1), arco(j)
i	j	Relocate(i-1, i, i+1, j, j+1)	arco(i-1), arco(i+1), arco(j), arco(j+1)
i	j	Relocate(i-1, i, i+1, j-1, j)	arco(i-1), arco(i+1), arco(j-1), arco(j)
i	j	Relocate(i, i+1, j-1, j, j+1)	arco(i+1), arco(j-1), arco(j), arco(j+1)
i	j	Relocate(i-1, i, j-1, j, j+1)	arco(i-1), arco(j-1), arco(j), arco(j+1)
i	j	Cross(i, i+1, j-1, j)	arco(i+1), arco(j-1), arco(j)
i	j	Cross(i-1, i, j, j+1)	arco(i-1), arco(j), arco(j+1)

En el Cuadro 5.4, se detallan los movimientos a analizar procurando obtener una mejor solución evitando realizar comparaciones sobre determinados pares de arcos por resultar poco atractivos. En nuestra propuesta, se han aplicado los siguientes operadores:

- Operador Intra-ruta: se usó 2-OPT como algoritmo de mejora dentro de una ruta de la solución CARP. Básicamente, consiste en agregar dos nuevas conexiones, eliminar dos existentes y cambiar el sentido de recorrido de un bloque de arcos.
- Operadores Inter-ruta: se usaron dos operadores de mejora entre dos rutas de la solución CARP. El primero de ellos, Relocate, elimina un arco de una ruta y lo inserta en la otra realizando a continuación las reconexiones necesarias. El segundo, Cross, intercambia bloques de arcos entre las dos rutas.

5.3.7. Reinicio

De acuerdo a la literatura sobre RKGA/BRKGA la obtención de cadenas mutantes en cada generación, tiene como objetivo evitar el estancamiento en óptimos locales. Una alternativa propuesta en este trabajo fue la aplicación del operador mutación sobre determinadas cadenas en cada generación. Aún así, debe incorporarse otro mecanismo que reduzca la posibilidad del estancamiento.

Si durante un número determinado de generaciones no se pudo obtener un mejor óptimo local, entonces se genera una nueva población de cadenas de claves aleatorias pero manteniendo las cadenas de tipo Elite de la generación actual. La supervivencia de estas cadenas es una propuesta que busca acelerar la convergencia a nuevos óptimos locales.

5.3.8. Algoritmo BRKGA para CARP

En el Cuadro 5.5 se muestra el pseudo-código del algoritmo propuesto. Los parámetros son el número de generaciones, el tamaño de la población, el porcentaje (de la población) de cadenas de tipo Elite, el porcentaje de cadenas a mutar y la probabilidad para aplicar cruzamiento entre dos cadenas.

Inicialmente, se generan cadenas de claves aleatorias. Luego, durante cada generación, se ordenan las cadenas de claves según su fitness para luego aplicar los operadores genéticos clásicos. Si el algoritmo no obtuvo mejora durante un determinado número de

generaciones, se generan nuevas cadenas de claves aleatorias, manteniendo aquellas que son de tipo Elite. Finalmente, se retorna la mejor solución.

Cuadro 5.5: Algoritmo BRKGA para CARP.

```
procedure BRKGA-CARP(#generations,#population,%ElitePop,%MutPop,%ProbE)

rkeyVector={}
bestOF=∞, bestSolution = {}
GenerateRKVs(rKeyVector,#population)
for i = 1 ... #generations
  sortByFitness(rKeyVector,%ElitePop,%MutPop)
  k=0
  repeat
    j=getNonEliteRKV(rKeyVector)
    k=getEliteRKV(rKeyVector,k)
    applyUCrossover(rKeyVector[k],rKeyVector[j],%ProbE)
  until getNonEliteRKV(rKeyVector)==-1
  repeat
    j=getMutantRKV(rKeyVector)
    applyMutation(rKeyVector[j])
  until getMutantRKV(rKeyVector)==-1
  repeat
    j=getModifiedRKV(rKeyVector)
    prevCarpSolution = decodeRKV(rKeyVector[j])
    improvedSolution = applyLSearch(prevCarpSolution)
    updateRKV(rKeyVector[j],improvedSolution)
    if f(rkeyVector[j])<bestOF  then
      bestOF = f(rkeyVector[j])
      bestSolution = improvedSolution
    end if
  until getModifiedRKV(rKeyVector)==-1
  if stagnation==True then
    GenerateRKVs(rKeyVector,#population-#ElitePop)
  end if
end for
return bestSolution
end procedure
```

Capítulo 6

Resultados Computacionales

En este capítulo, se evaluará la calidad de los resultados obtenidos por los algoritmos propuestos para CARP usando instancias de la literatura.

6.1. Introducción

En el capítulo anterior se propusieron nuevos algoritmos HBMO, VNS y BRKGA para CARP. De cada uno de ellos, se describieron sus principales características y su aplicación al Problema de Ruteo de Arcos Capacitado.

Este capítulo está dedicado a evaluar la calidad de los resultados obtenidos por los algoritmos propuestos. Para ello, el experimento empírico consta de dos partes. La primera consiste en obtener la mejor configuración posible de los parámetros de los algoritmos, de manera tal que asegure la obtención de buenos resultados en pruebas computacionales posteriores. Para ello, se analizará la calidad de los resultados devueltos por cada algoritmo usando diferentes configuraciones. La segunda parte está abocada al estudio comparativo de los resultados alcanzados por los algoritmos sobre diferentes instancias CARP de la literatura, usando en cada uno de ellos la mejor configuración de parámetros.

Respecto a las instancias utilizadas durante las pruebas, se han tomado instancias Kshs propuestas originalmente por Kiuchi et al. [74], instancias Gdb del trabajo de Golden et al. [53] e instancias Val de Benavent et al. [13]. Estos conjuntos de instancias fueron generados aleatoriamente siguiendo diferentes patrones de construcción, de capacidad de

vehículos y de demandas, y son de uso frecuente para evaluar la calidad de algoritmos exactos y heurísticos.[1]

Para la obtención de la mejor configuración de parámetros para cada uno de los algoritmos propuestos, se han usado 3 instancias CARP. En el Cuadro 6.1 se muestra para cada una de ellas (además del nombre), el número de vértices, el número de aristas requeridas, la demanda requerida, la capacidad de carga vehicular y el resultado óptimo extraído de la literatura ([16], [84]) consistente de la distancia total recorrida por los vehículos y el número de vehículos usados para atender las demandas requeridas.

Cuadro 6.1: Pruebas preliminares: información de instancias usadas.

Instancia	$\|V\|$	$\|R\|$	Demanda	Q	Optimo	
					TD	Veh.
Kshs6	9	15	389	150	**10197**	**3**
Gdb1	12	22	22	5	**316**	**5**
Gdb16	8	28	116	24	**127**	**5**

A los efectos de obtener configuraciones de parámetros robustas, se consideró conveniente para la primera fase de pruebas, la utilización de instancias que posean cantidades de aristas requeridas y capacidades vehiculares diferentes, como así también soluciones que empleen varios vehículos para atender las demandas requeridas. La descripción completa de algunas instancias de prueba usadas como así también los mejores resultados alcanzados por los algoritmos, se encuentran en el Apéndice.

Para las pruebas computacionales, se usó el ambiente Eclipse dada la potencialidad y facilidad que brinda para el desarrollo y prueba de software. A su vez y con el propósito de realizar un objetivo análisis comparativo de los algoritmos, todas las pruebas se realizaron en una PC con procesador CORE 2 4300 1.80Ghz, con 2GB de memoria RAM y Windows Vista como sistema operativo.

En las secciones 6.2, 6.3 y 6.4, se describirán los parámetros de los algoritmos propuestos, se mostrarán los resultados alcanzados usando diferentes configuraciones y serán

[1]Las instancias se encuentran disponibles en www.uv.es/belengue/carp.html

analizados los mismos para así determinar las mejores configuraciones. Luego, en la sección 6.5, se realizarán nuevas pruebas computacionales para analizar en conjunto, la calidad de los resultados.

6.2. Algoritmo HBMO para CARP

Tal como se mencionó anteriormente (sección 3.3.5), los parámetros requeridos en los algoritmos HBMO son:

- Número de reinas: corresponde al número de reinas (mejores soluciones) que se usará para cruzarse con la población de zánganos. Por ejemplo, Abbas [1] realizó pruebas para instancias SAT (Propositional Satisfability Problem) con diferentes números de reinas (de 1 a 5), obteniéndose los mejores resultados con 2 abejas reinas. Afshar et al. [2] en cambio, para un problema real en el campo de la administración y planificación de recursos hídricos, usaron tan sólo 1 abeja reina.

- Número de vuelos reproductivos: indica el número de vuelos que realiza la abeja reina, pudiendo en cada uno de ellos cruzarse con varios zánganos. En el trabajo de Afshar et al. [2], se utilizaron 3000 vuelos reproductivos (iteraciones del algoritmo) para las pruebas computacionales. Por otra parte, Koudil et al. [75] realizaron pruebas con diferentes combinaciones de abejas reinas y números de vuelos para un Problema de Partición y Asignación, siendo la más apropiada la formada por un número bajo de reinas y un número alto de vuelos reproductivos la que produjo los mejores resultados a bajo costo computacional.

- Número de zánganos: es el número de zánganos que potencialmente pueden cruzarse con la reina. Puede ser un parámetro fijo o bien depender de la energía de la abeja reina para que se genere uno nuevo. Abbas [1] adoptó esto último. En cambio Afshar et al. [2], utilizaron 130 zánganos en cada vuelo reproductivo. Fathian et al. [36] propusieron que para cada vuelo y mientras la abeja reina posea energía, un zángano sea elegido al azar para determinar si cumple con la condición de apareo.

- Número de abejas obreras: es el número de heurísticas encargadas de mejorar las nuevas soluciones obtenidas por cruzamiento entre abeja reina y zángano. Como se

mencionó previamente, no es exactamente un parámetro dado que depende de las heurísticas incluidas en el algoritmo propuesto.

- Velocidad: también conocido como Energía, es la velocidad que posee la abeja reina durante su vuelo reproductivo. No es un valor constante sino que decrece durante el tiempo que transcurre el vuelo. El valor inicial debe permitir que la abeja reina (en cada vuelo) tenga chances de aparearse con varios zánganos. Para sus pruebas, tanto Fathian et al. [36] como Curkovic y Jerbic [24], generaron al azar el valor de velocidad inicial (entre 0.5 y 1).

- α: factor de reducción de velocidad (o energía). Al igual que la velocidad, el valor de este parámetro debe contribuir a que la reina se aparee con varios zánganos. Fathian et al. [36] usaron 0.98 para las pruebas. En cambio, Curkovic y Jerbic [24] adoptaron como factor de reducción un valor dependiente de la energía inicial de reina y el tamaño de espermateca.

El algoritmo HBMO propuesto (sección 5.1.6) posee los siguientes parámetros requeridos:

- Número de vuelos reproductivos: este parámetro fue reemplazado por el tiempo máximo de ejecución del algoritmo, a los efectos de realizar una mejor comparación de resultados entre los algoritmos propuestos para CARP.

- Número de zánganos: indica el número de zánganos que se cruzarán con la reina en cada vuelo reproductivo. Al final de cada vuelo, los zánganos actuales pueden o no ser eliminados (o sea, reemplazados por crías obtenidas recientemente). Esto dependerá de su nivel de diversidad y/o calidad de solución.

- Número de soluciones iniciales: indica el número de soluciones (tambień aplicable luego de cada reinicio del algoritmo) generadas en forma semi-aleatoria, de las cuales se elegirán aquellas que formen parte de la población inicial de zánganos.

- Número máximo de vuelos sin mejora: indica el número de vuelos que deben transcurrir (sin que el algoritmo obtenga una mejor solución) antes de activar el mecanismo de reinicio. Durante la fase de pruebas, se fijó un máximo de 3 reinicios.

En base a resultados publicados de pruebas sobre algoritmos HBMO, y a los resultados obtenidos en pruebas preliminares, se han propuesto 3 valores posibles para el número de zánganos y 3 para el número de soluciones iniciales generadas, obteniéndose un total de 9 configuraciones de parámetros para el algoritmo.

El Cuadro 6.2 muestra para cada configuración de parámetros propuesta, los resultados alcanzados por el algoritmo HBMO. Por cada configuración, se realizaron 50 corridas independientes y se fijó el tiempo máximo de CPU en 20 segundos por corrida. La información de resultados listada consiste de la distancia total promedio $\bar{TD}$, el desvío estándar DE, la moda MO, la menor distancia total (alcanzada en las 50 corridas) $Min.TD$, el porcentaje de ocurrencia (durante las 50 corridas) del valor anterior $\%Min.TD$ y el tiempo promedio $\bar{Tiempo}$ (en segundos) que necesita el algoritmo para alcanzar el mejor resultado en cada corrida.

Cuadro 6.2: Pruebas preliminares de HBMO con instancia Kshs6 (TD=10197 y Vh=3).

Configuración			Resultados					
Id	Zánganos	Sols.In.	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	20	150	10216.50	137.88	10197	10197	98.0	0.7
C2	20	500	10205.64	29.59	10197	10197	92.0	1.0
C3	20	1000	10199.16	15.27	10197	10197	98.0	1.3
C4	30	150	10199.16	15.27	10197	10197	98.0	0.6
C5	30	500	10201.32	21.37	10197	10197	96.0	1.4
C6	30	1000	10197.00	0.0	10197	10197	100.0	2.3
C7	50	150	10201.32	21.37	10197	10197	96.0	0.6
C8	50	500	10203.48	25.90	10197	10197	94.0	1.0
C9	50	1000	10199.16	15.27	10197	10197	98.0	1.9

De los resultados mostrados en el Cuadro 6.2, se observa que en general todas las combinaciones alcanzan el valor óptimo (del total de corridas realizadas para cada configuración, cada una de ellas lo alcanza en al menos un 92 %) a bajo costo computacional

(ninguna configuración supera los 2.3 segundos en alcanzarlo en cada corrida). Al menos para esta instancia, los resultados alcanzados son similares en todas las configuraciones.

En el Cuadro 6.3 se listan los resultados alcanzados por cada configuración de parámetros, usando Gdb1 como instancia de prueba. Para cada configuración, se realizaron 50 corridas independientes y se fijó el tiempo máximo de CPU en 30 segundos .

Cuadro 6.3: Pruebas preliminares de HBMO con instancia Gdb1 (TD=316 y Vh=5).

Configuración			Resultados					
Id	Zánganos	Sols.In.	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	20	150	318.12	3.15	316	316	68.0	5.1
C2	20	500	318.20	3.25	316	316	68.0	6.6
C3	20	1000	318.18	3.19	316	316	66.7	7.2
C4	30	150	317.94	3.06	316	316	70.0	7.3
C5	30	500	318.04	3.07	316	316	68.0	8.3
C6	30	1000	318.08	3.12	316	316	68.0	8.1
C7	50	150	318.62	3.39	316	316	62.0	6.1
C8	50	500	318.36	3.32	316	316	66.0	6.3
C9	50	1000	318.80	3.27	316	316	56.0	9.0

En el Cuadro 6.3, puede apreciarse una mayor variabilidad de resultados alcanzados por las configuraciones. Si bien todas poseen como moda y menor distancia total recorrida al valor óptimo TD=316, los porcentajes (de ocurrencia) de la menor distancia total y el tiempo promedio (en alcanzar dicho valor por corrida) varían entre ellas. Por ejemplo, se observa que la configuración C1 (20 zánganos y 150 soluciones iniciales generadas) alcanza un mejor porcentaje de menor distancia (68 %) y un menor tiempo computacional (5.1 segundos) comparado con la configuración C9 (50 zánganos y 1000 soluciones iniciales), dado que ésta alcanza la menor distancia total en un 56 % de las veces con un tiempo promedio de 9 segundos. De todas las configuraciones, se destaca C4 (30 zánganos y 150

soluciones iniciales), dado que obtiene la menor distancia total promedio (317.94), un mejor porcentaje de ocurrencia de la distancia recorrida óptima (70 %) y un bajo tiempo computacional promedio (7.3 segundos).

En el Cuadro 6.4, se listan los resultados alcanzados por las configuraciones propuestas usando la instancia Gdb16 como instancia de prueba. Se realizaron 50 corridas para cada configuración y por cada corrida se fijó un tiempo máximo de CPU de 45 segundos.

Cuadro 6.4: Pruebas preliminares de HBMO con instancia Gdb16 (TD=127 y Vh=5).

Configuración			Resultados					
Id	Zánganos	Sols.In.	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	20	150	127.84	1.28	127	127	64.0	10.2
C2	20	500	127.60	1.08	127	127	74.0	11.8
C3	20	1000	127.60	1.01	127	127	72.0	12.0
C4	30	150	127.76	1.13	127	127	66.0	10.9
C5	30	500	127.28	0.80	127	127	88.0	8.7
C6	30	1000	127.60	1.01	127	127	72.0	11.3
C7	50	150	127.76	1.33	127	127	72.0	7.4
C8	50	500	127.56	0.99	127	127	74.0	13.9
C9	50	1000	127.68	1.03	127	127	68.0	12.14

Los resultados mostrados en el Cuadro 6.4 indican que todas las configuraciones tienen como moda y menor distancia total (durante las 50 corridas) a la distancia óptima TD=127. Además, en todos los casos, la distancia total promedio es cercana a dicho valor; todas poseen un bajo desvío estándar (en el mejor de los casos es 0.8 y en el peor es 1.33) y el porcentaje de ocurrencia de la menor distancia total es bueno (en el mejor de los casos es 88 % y en el peor 64 %). Respecto a las configuraciones C1 y C9, no se aprecian grandes diferencias si se comparan sus resultados. De todas las configuraciones, se destaca C5 (30 zánganos y 500 soluciones iniciales), dado que obtiene la menor distancia promedio

(127.28), el menor desvío (0.80), el mejor porcentaje de menor distancia total (88 %) y un bajo tiempo computacional promedio (8.7 segundos).

En base a los resultados de las pruebas obtenidas por las configuraciones de parámetros propuestas con las instancias Kshs6, Gdb1 y Gdb16, la configuración C4 (30 zánganos y 150 soluciones iniciales) es la que se usará en las pruebas finales. Dicha selección se debe a que la misma alcanzó buenos resultados en las pruebas preliminares y que por sus características intrínsecas permitirá al algoritmo (en las pruebas finales) intensificar y diversificar el proceso de búsqueda de soluciones.

En las Fig. 6.1, 6.2 y 6.3 se observan los histogramas de las corridas realizadas para la configuración C4 con las instancias Kshs6, Gdb1 y Gdb16, respectivamente. En dichos histogramas, pueden apreciarse los buenos resultados que obtiene la configuración. En tanto, las Fig. 6.4, 6.5 y 6.6 muestran un diagrama de dispersión correspondiente a una corrida por cada una de las instancias de prueba usando la configuración elegida. En los diagramas, se observa cómo el algoritmo para alcanzar buenos resultados intensifica y diversifica la búsqueda de soluciones, no limitándose a realizar una búsqueda aleatoria o miope.

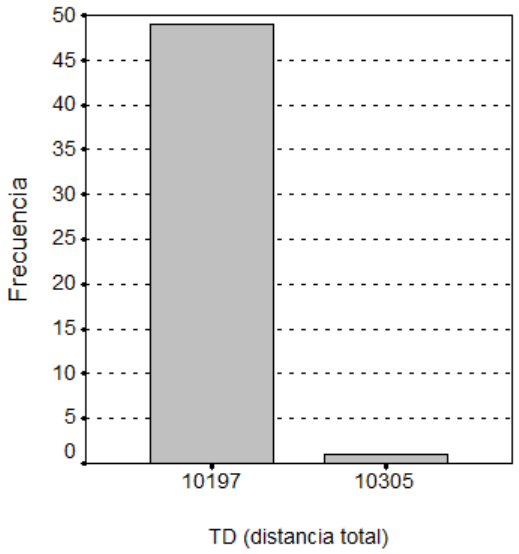

Figura 6.1: Pruebas HBMO: histograma de Kshs6.

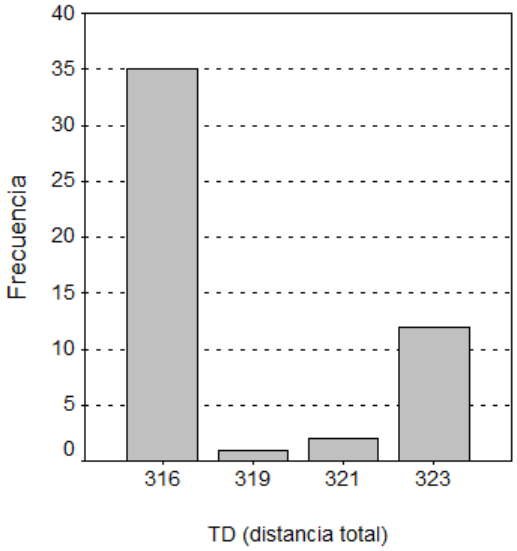

Figura 6.2: Pruebas HBMO: histograma de Gdb1.

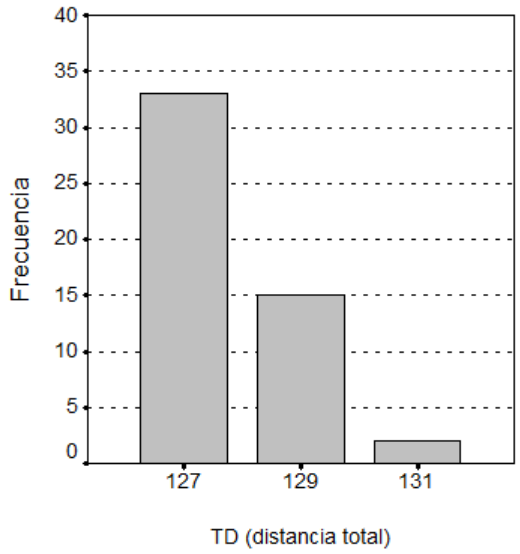

Figura 6.3: Pruebas HBMO: histograma de Gdb16.

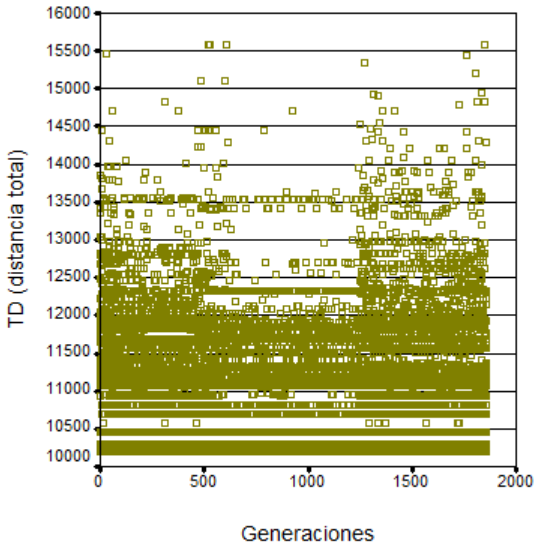

Figura 6.4: Pruebas HBMO: diagrama de dispersión de Kshs6.

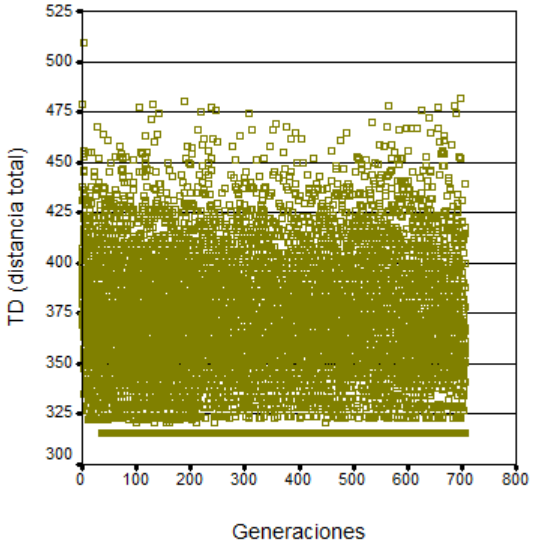

Figura 6.5: Pruebas HBMO: diagrama de dispersión de Gdb1.

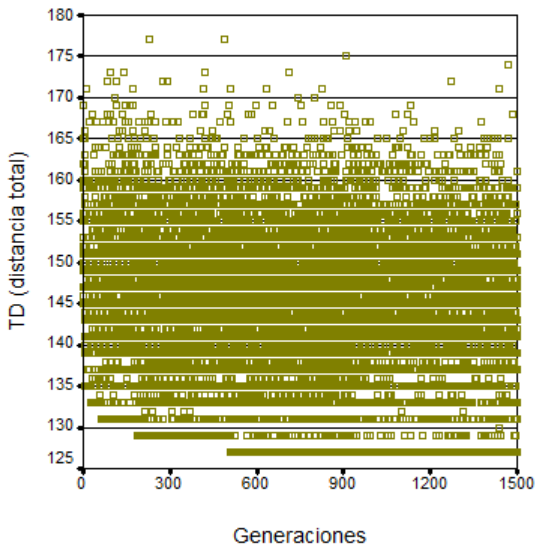

Figura 6.6: Pruebas HBMO: diagrama de dispersión de Gdb16.

6.3. Algoritmo híbrido-VNS para CARP

En base a lo expuesto en sección 3.7 y de la literatura ([64], entre otros), los parámetros requeridos en las diferentes extensiones de VNS son:

- Condición de parada del algoritmo: en la literatura, se han propuesto el tiempo máximo de CPU, el número de iteraciones y el número máximo de iteraciones entre dos sucesivas mejoras (este último, más relacionado con el estancamiento en óptimos locales). Para las pruebas del algoritmo para el Problema de P-Centros (P-Center Problem) usando instancias TSP-Lib de diferentes tamaños, Mladenovic et al. [91] usaron diferentes tiempos máximos de CPU. De manera similar, Fleszar y Hindi [42], han usado el tiempo de CPU como condición de parada para las pruebas sobre el algoritmo propuesto para el Problema de la P-Mediana (P-Median Problem).

- Número de vecindarios: es el número máximo de vecindarios (de una estructura propuesta) a partir de los cuales se obtienen soluciones iniciales que luego son mejoradas. Una mejor exploración del espacio de soluciones es posible mediante el uso de varias estructuras de vecindarios. No obstante, la definición de las mismas depende del problema a tratar. Mladenovic et al. [91] propusieron para el Problema de P-Centros, la estructura de vecindario K-Intercambio (K-Interchange). En sus

pruebas, el número máximo de vecindarios corresponde al valor p, siendo éste el número de centros a obtener en cada una de las instancias. Para un Problema de Asignación de Estudiantes a Tópicos (o temas), Geiger y Wenger [46] propusieron las estructuras de vecindario Mover (Shift), Intercambiar2 (Swap2), Intercambiar3 (Swap3) y Mover+Intercambiar2, las que se aplican luego que una solución es obtenida por un algoritmo constructivo. Dado el tamaño de las instancias reales usadas y el tiempo computacional usado, los valores máximos de vecindarios corresponden al número de estudiantes y tópicos de cada instancia.

El algoritmo VNS híbrido propuesto posee los siguientes parámetros:

- Número de iteraciones: para realizar una mejor comparación de resultados entre los algoritmos propuestos, este parámetro fue reemplazado por el tiempo máximo de CPU como condición de finalización del algoritmo.
- α: es un parámetro ($0 \leq \alpha \leq 1$) usado en forma similar a GRASP [106], que permite seleccionar a partir del costo o beneficio, un subconjunto de elementos a incorporar a la solución en construcción. De esta lista restringida, se elegirá aleatoriamente el nuevo elemento que se agregará a la solución. A pesar que resulta difícil determinar el mejor valor [105], es recomendable que no adopte valores extremos.
- K: parámetro usado por el algoritmo Inserción más Económica, corresponde al número de inserciones (de arcos) que deben realizarse sobre la solución en construcción, antes que la misma sea mejorada por operadores de búsqueda local. En base a pruebas previas usando diferentes valores, K=$\frac{|R|}{4}$ resultó el valor más adecuado para activar a los algoritmos de mejora.
- Tamaño del Conjunto de Referencia: corresponde al número de soluciones diversas y de calidad que se necesitan para activar Path Relinking como una estrategia adicional de intensificación y diversificación. Esto significa que Path Relinking no es continuamente aplicado, sino periódicamente (tal como se indica en [105]).
- Número máximo de iteraciones sin mejora: es el número de iteraciones que debe transcurrir desde el último óptimo local obtenido, para activar el mecanismo de escape de óptimos locales. Una vez alcanzado este valor, se procede a construir soluciones usando el algoritmo constructivo basado en GRASP que utiliza memoria de largo plazo. De acuerdo a pruebas previas, $\frac{TiempoCPU}{10}$ produjo los mejores resultados.

Siguiendo las ideas de diferentes algoritmos basados en GRASP y Path Relinking, y de acuerdo a pruebas previas del algoritmo VNS híbrido, se propusieron 3 valores posibles para α y 3 para el número de soluciones élite, obteniéndose un total de 9 configuraciones de parámetros.

El Cuadro 6.5 muestra los resultados obtenidos por el algoritmo VNS usando la instancia Kshs6, para todas las configuraciones propuestas. Por cada una, se realizaron 50 corridas independientes y se fijó un tiempo máximo de CPU de 20 segundos por corrida. Además de los valores de los parámetros usados en cada configuración, se lista la distancia total promedio $\bar{TD}$, el desvío estándar DE, la moda MO, la menor distancia total (alcanzada en las 50 corridas) $Min.TD$, el porcentaje de ocurrencia de dicho valor $\%Min.TD$ y el tiempo promedio $\bar{Tiempo}$ (necesario para obtener el mejor resultado).

Cuadro 6.5: Pruebas preliminares de VNS con instancia Kshs6 (TD=10197 y Vh=3).

Configuración			Resultados					
Id	α	$\|RefSet\|$	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	0.25	20	10197.00	0.0	10197	10197	100.0	2.8
C2	0.25	30	10197.00	0.0	10197	10197	100.0	2.7
C3	0.25	40	10197.00	0.0	10197	10197	100.0	3.0
C4	0.3	20	10197.00	0.0	10197	10197	100.0	1.9
C5	0.3	30	10197.00	0.0	10197	10197	100.0	2.5
C6	0.3	40	10197.00	0.0	10197	10197	100.0	3.3
C7	0.35	20	10197.00	0.0	10197	10197	100.0	2.4
C8	0.35	30	10197.00	0.0	10197	10197	100.0	2.6
C9	0.35	40	10199.16	15.27	10197	10197	98.0	3.2

De los resultados listados en el Cuadro 6.5, se observa que todas las configuraciones tienen como moda y menor distancia total recorrida al valor óptimo TD=10197. Además, los tiempos promedios (en alcanzarlo) tienen muy poca variación entre sí (entre 1.9 y

3.2 segundos). La configuración C9 (α=0.35 y $|RefSet|$=40 soluciones elite) es la única que posee desvío estándar (15.27) y C4 (α=0.30 y $|RefSet|$=20) es de todas las configuraciones, la que alcanza mejores resultados en menor tiempo promedio de CPU (1.9 segundos).

Continuando con las pruebas preliminares, el Cuadro 6.6 muestra para las mismas configuraciones de parámetros, los resultados obtenidos con la instancia Gdb1. Los mismos corresponden a 50 corridas por configuración y el tiempo máximo de CPU fue fijado en 30 segundos.

Cuadro 6.6: Pruebas preliminares de VNS con instancia Gdb1 (TD=316 y Vh=5).

Configuración			Resultados					
Id	α	$\|RefSet\|$	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	0.25	20	318.00	2.92	316	316	64.0	12.8
C2	0.25	30	319.60	3.23	316	316	40.0	14.3
C3	0.25	40	320.20	3.38	316	316	32.0	14.0
C4	0.3	20	317.80	2.76	316	316	66.0	14.3
C5	0.3	30	319.80	3.51	316	316	40.0	15.1
C6	0.3	40	319.88	3.55	316	316	38.0	14.8
C7	0.35	20	318.42	3.07	316	316	58.0	14.6
C8	0.35	30	319.30	3.35	316	316	46.0	15.9
C9	0.35	40	319.86	3.53	316	316	38.0	16.2

Del cuadro anterior, se aprecia que al igual que con los resultados obtenidos con la instancia Kshs6, todas las configuraciones tienen como moda y menor distancia total, al valor óptimo TD=316. Además, las distancias totales promedio distan como máximo 1 % de dicho valor y los tiempos computacionales promedios son bajos (entre 12.8 y 16.2 segundos). Si se analizan las configuraciones C1 ($\alpha = 0{,}25$ y $|RefSet|$=20) y C9 ($\alpha = 0{,}35$ y $|RefSet|$=40), puede notarse que un mayor número de soluciones élite y un valor alto

de α, produjeron un mayor tiempo promedio (16.2 segundos) y un menor porcentaje de menor distancia total (38 %). Por otra parte, la configuración C4 que posee un valor de α intermedio y un tamaño de Conjunto de Referencia mencionado en la literatura de Scatter Search, obtuvo el menor promedio de distancia total (317.80), el menor desvío (2.76), el mejor porcentaje (de ocurrencia) de menor distancia (66 %) y un bajo costo computacional (14.3 segundos).

Los resultados del Cuadro 6.7 corresponden a las pruebas realizadas con la instancia Gdb16. Por cada configuración de parámetros, se realizaron 50 corridas independientes del algoritmo y se estableció un tiempo máximo de CPU de 45 segundos por corrida.

Cuadro 6.7: Pruebas preliminares de VNS con instancia Gdb16 (TD=127 y Vh=5).

Configuración			Resultados					
Id	α	$\|RefSet\|$	$\bar{TD}$	DE	Mo	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	0.25	20	127.00	0.0	127	127	100.0	10.2
C2	0.25	30	127.32	0.74	127	127	84.0	11.9
C3	0.25	40	127.00	0.0	127	127	100.0	13.4
C4	0.3	20	127.04	0.28	127	127	98.0	10.4
C5	0.3	30	127.00	0.0	127	127	100.0	11.2
C6	0.3	40	127.00	0.0	127	127	100.0	12.5
C7	0.35	20	127.00	0.0	127	127	100.0	9.8
C8	0.35	30	127.00	0.0	127	127	100.0	11.0
C9	0.35	40	127.04	0.28	127	127	98.0	13.2

Los resultados indicados en el Cuadro anterior, tienen características similares a los del Cuadro 6.5. Es decir, todas las configuraciones obtienen como moda y menor distancia total, al valor óptimo TD=127. Además, el desvío estándar en la mayoría de las configuraciones es cero (salvo en las configuraciones C2, C4 y C9 los cuales son menores a 1)

y la distancia total promedio alcanzada coincide con dicho valor (en 6 de las 9 configuraciones). Si analizamos nuevamente las configuraciones C1 y C9, esta última posee un mayor tiempo computacional promedio (13.2 segundos), un menor porcentaje de menor distancia total (98 %) y un leve valor de desvío (0.28). Finalmente, los mejores resultados fueron obtenidos con la configuración C7 (α = 0,35 y $|RefSet|$=20), dado que ésta alcanza en todas las corridas realizadas el valor óptimo con el menor tiempo computacional (9.8 segundos).

De las pruebas preliminares realizadas con las instancias Kshs6, Gdb1 y Gdb16 usando diferentes configuraciones de parámetros, para las pruebas finales se usará la configuración C4 (α = 0,30 y $|RefSet|$=20) para el algoritmo VNS híbrido, dado que ésta ha logrado buenos resultados a un bajo costo computacional. Esto se debe principalmente a los valores de sus parámetros, los cuales son usualmente utilizados en la literatura.

Las Fig. 6.7, 6.8 y 6.9, muestran los histogramas de las corridas efectuadas usando la configuración C4 sobre las instancias Kshs6, Gdb1 y Gdb16. En todos los casos, puede apreciarse que la configuración permite que el algoritmo alcance los mejores resultados en un alto porcentaje de las corridas realizadas. A su vez, las Fig. 6.10, 6.11 y 6.12, corresponden a un diagrama de dispersión de una corrida elegida por cada instancia, usando la configuración antes indicada. En cada diagrama de dispersión se observa cómo a lo largo de la ejecución del algoritmo, éste diversifica e intensifica la búsqueda de mejores soluciones. Los picos existentes en los diagramas marcan momentos de inicio y reinicio del algoritmo. Además, a pesar de ser un algoritmo con un comportamiento más determinístico que el algoritmo HBMO propuesto, la búsqueda de soluciones se realiza sobre un espacio amplio de soluciones.

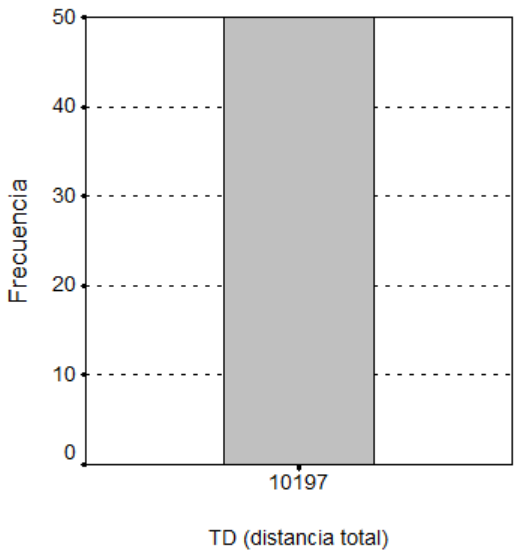

Figura 6.7: Pruebas VNS: histograma de Kshs6.

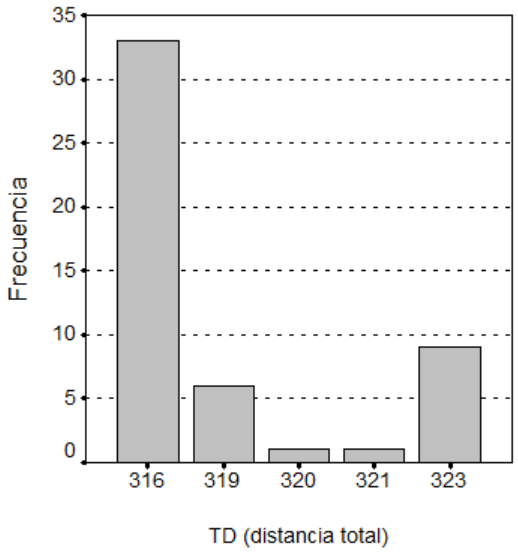

Figura 6.8: Pruebas VNS: histograma de Gdb1.

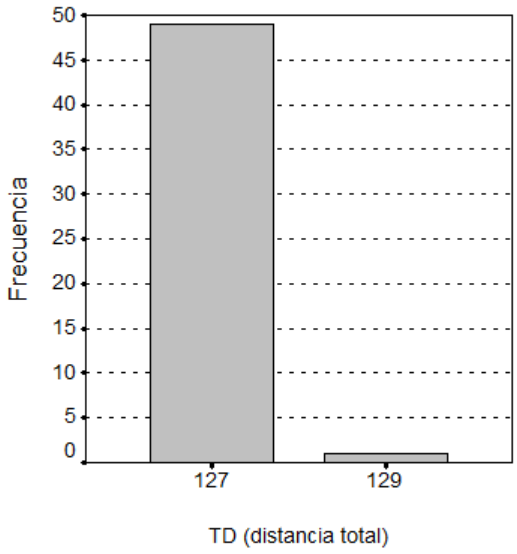

Figura 6.9: Pruebas VNS: histograma de Gdb16.

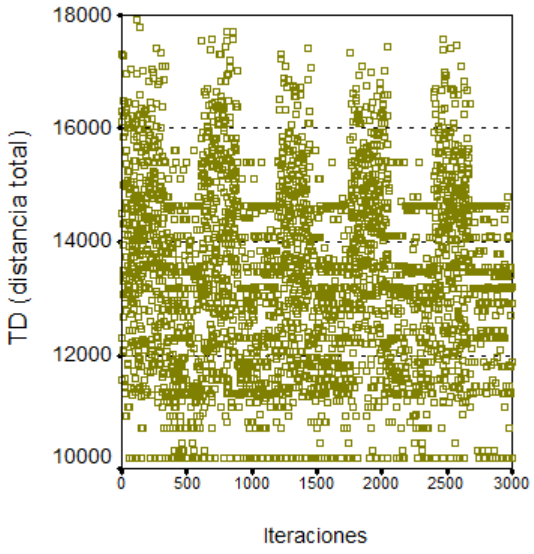

Figura 6.10: Pruebas VNS: diagrama de dispersión de Kshs6.

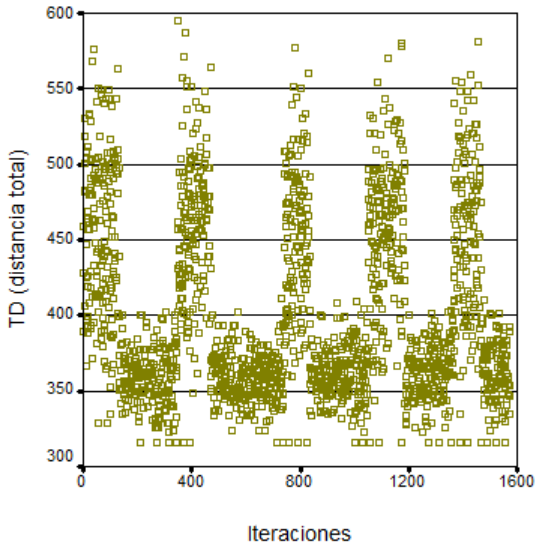

Figura 6.11: Pruebas VNS: diagrama de dispersión de Gdb1.

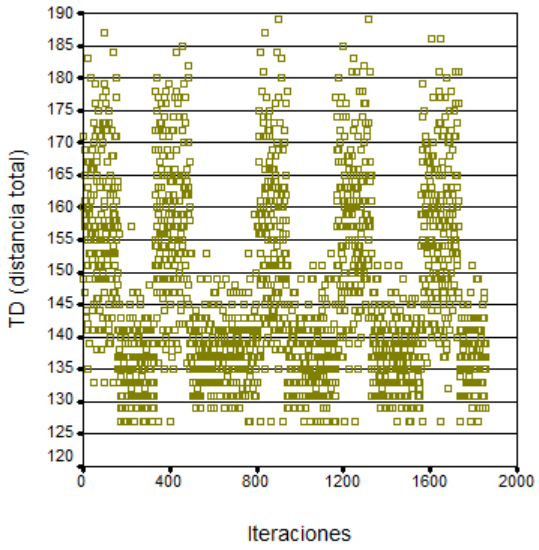

Figura 6.12: Pruebas VNS: diagrama de dispersión de Gdb16.

6.4. Algoritmo BRKGA para CARP

De lo indicado en las secciones 3.8.2, 3.8.3 y 6.3, y de la literatura ([56], [57], [59], entre otros), los parámetros usados en los algoritmos BRKGA son:

- Tamaño de la población: es el número de cadenas de claves aleatorias (que luego son decodificadas para obtener soluciones al problema en cuestión) que el algoritmo mantiene durante un cierto tiempo y sobre las cuales se aplican diferentes operadores genéticos a los efectos de obtener mejores soluciones. Puede ser un valor fijo o resultar de una expresión dependiente de otros parámetros. Por ejemplo, Snyder y Daskin [114], evaluaron su propuesta para GTSP (Generalized Travelling Salesman Problem) con diferentes instancias de la librería TSPLib usando una población fija de 100 cromosomas. En cambio, Gonçalves [56] para las diferentes pruebas efectuadas al algoritmo BRKGA para TDOPP (Two-dimensional orthogonal Packing Problem) utilizó como tamaño de la población 10 veces el número de rectángulos incluidos en cada instancia usada.

- Número de individuos Elite: es el número de individuos de alta calidad que son preservados de una generación a otra; es decir, es la aplicación del operador *reproducción* sobre una población de cadenas de claves aleatorias. El valor de este parámetro corresponde a un porcentual sobre la población. Snyder y Daskin [114] fijaron en 20 % de la población anterior la que se preserva (copia) en la actual. En tanto Gonçalves [56], aplicó el operador reproducción sobre el 15 % de la población.

- Número de individuos Mutantes: corresponde al número de individuos obtenidos mediante *inmigración*. Al igual que el operador *mutación*, la aplicación de este operador contribuye a evitar una prematura convergencia de la población. Mediante inmigración, Snyder y Daskin [114] en cada generación obtienen un 10 % de nuevos individuos de manera aleatoria los cuales son mejorados con algoritmos heurísticos. Por su parte, Gonçalves [56] aplicó el mismo operador sobre el 15 % de cada población.

- Probabilidad de cruzamiento (p_e): es la probabilidad usada por el operador *Parameterized Uniform Crossover* para obtener nuevas cadenas de claves aleatorias mediante el cruzamiento entre cadenas de tipo elite y No-elite (de la población

anterior). Snyder y Daskin [114] obtienen el 70 % de cada población con probabilidad de cruzamiento del 70 %. Gonçalves [56] adoptó el mismo criterio.

- Número de generaciones: o número máximo de iteraciones establecido para que la población pueda evolucionar. Este parámetro es comunmente adoptado como condición de parada del algoritmo. Snyder y Daskin [114] fijaron 100 generaciones como condición de parada ó 10 generaciones consecutivas sin mejora. De manera similar, Gonçalves [56] utilizó 2000 generaciones para sus pruebas.

Los parámetros usados en el algoritmo propuesto son los siguientes:

- Tamaño de la población: considerando las características de las instancias utilizadas en las pruebas preliminares, se han propuesto diferentes tamaños para la población de claves aleatorias los que serán mostrados en los siguientes cuadros.
- Número de individuos Elite: se propusieron varias opciones para este parámetro, los cuales también se indican en cuadros posteriores.
- Número de individuos Mutantes: a diferencia de los algoritmos BRKGA de la literatura, el número de individuos mutantes serán obtenidos a través del operador *mutación*. Este operador se aplica para obtener el 20 % de individuos de cada población.
- Probabilidad de cruzamiento: se adoptó 70 % como el valor de probabilidad para aplicar el cruzamiento entre cadenas de tipo Elite y No-Elite.
- Número de generaciones: para comparar objetivamente los resultados de los algoritmos propuestos, se reemplazó este parámetro por el tiempo máximo de CPU.
- Número máximo de generaciones sin mejora: para reducir la probabilidad que el algoritmo quede atrapado en un óptimo local, se propuso que toda vez que no exista mejora durante un cierto tiempo ($\frac{TiempoCPU}{4}$), los individuos de la nueva generación serán obtenidos mediante *inmigración*.

Para realizar las pruebas preliminares del algoritmo BKRGA, se propusieron 3 valores posibles para el tamaño de la población y 3 valores posibles para el número de individuos de tipo Elite, obteniéndose un total de 9 configuraciones posibles entre ambos parámetros.

En el Cuadro 6.8 aparecen los resultados alcanzados por el algoritmo BRKGA usando la instancia Kshs6, realizando 50 corridas para cada una de las configuraciones propuestas. El cuadro muestra información de la configuración (Identificador, tamaño de la población y % de población Elite), la distancia total promedio $\bar{TD}$, el desvío estándar DE, la moda MO, la menor distancia total $Min.TD$, el porcentaje de ocurrencia de la menor distancia alcanzada en las 50 corridas $\%Min.TD$ y el tiempo computacional promedio $\bar{Tiempo}$ que necesita el algoritmo en alcanzar el mejor resultado. En este caso, el tiempo máximo de CPU fue fijado en 20 segundos.

Cuadro 6.8: Pruebas preliminares de BRKGA con instancia Kshs6 (TD=10197 y Vh=3).

Configuración			Resultados					
Id	Pobl.	% Elite	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	20	15	10199.16	15.27	10197	10197	98.0	0.8
C2	20	20	10197.00	0.0	10197	10197	100.0	0.6
C3	20	25	10197.00	0.0	10197	10197	100.0	0.8
C4	30	15	10197.00	0.0	10197	10197	100.0	0.6
C5	30	20	10197.00	0.0	10197	10197	100.0	0.7
C6	30	25	10199.16	15.27	10197	10197	98.0	0.7
C7	40	15	10197.00	0.0	10197	10197	100.0	0.8
C8	40	20	10197.00	0.0	10197	10197	100.0	0.6
C9	40	25	10197.00	0.0	10197	10197	100.0	0.8

Todas las configuraciones dispuestas en el cuadro anterior, tienen como moda y menor valor de distancia total recorrida, al valor óptimo TD=10197. Por otra parte, 7 de las 9 configuraciones logran en todas sus corridas dicho valor y en los 2 casos restantes lo alcanzan en el 98 % de las veces. A su vez, el tiempo computacional promedio para cada configuración, varía entre 0.6 y 0.8 segundos. En síntesis, la calidad de los resultados aún no permiten diferenciar una configuración por sobre las restantes.

A continuación, en el Cuadro 6.9 se muestran los resultados de las pruebas preliminares del algoritmo usando la instancia Gdb16. Nuevamente, se realizaron 50 corridas independientes por cada configuración y el tiempo máximo de CPU se fijó en 30 segundos por corrida.

Cuadro 6.9: Pruebas preliminares de BRKGA con instancia Gdb1 (TD=316 y Vh=5).

Configuración			Resultados					
Id	Pobl.	% Elite	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	20	15	316.46	1.35	316	316	88.0	12.4
C2	20	20	316.30	1.09	316	316	92.0	12.3
C3	20	25	316.46	1.31	316	316	88.0	11.1
C4	30	15	316.60	1.70	316	316	88.0	9.6
C5	30	20	316.46	1.48	316	316	90.0	12.7
C6	30	25	316.76	1.96	316	316	86.0	10.8
C7	40	15	316.80	1.69	316	316	80.0	11.0
C8	40	20	316.88	1.71	316	316	76.0	11.1
C9	40	25	316.90	2.09	316	316	82.0	10.6

Respecto a los resultados del cuadro anterior, se puede apreciar que todas las configuraciones alcanzan el valor óptimo en sus modas y menor distancia total recorrida. Por otra parte, la distancia total promedio en todos los casos es levemente mayor (varían entre 316.30 y 316.90). A su vez, el porcentaje de ocurrencia de la menor distancia recorrida es alto en todos los casos (entre 76 y 92 %) y son bajos los tiempos computacionales (entre 9.6 y 12.7 segundos). Considerando el porcentaje de ocurrencia de la menor distancia recorrida y el tiempo computacional promedio, las mejores configuraciones son C2 (Población=20, % Elite=20) y C5 (Población=30, % Elite=20). Finalmente, comparando las configuraciones C9 y C1, se observa que una mayor población de cadenas de claves aleatorias al igual que un mayor porcentaje de tipo Elite, no garantizan la obtención de mejores resultados.

Antes de determinar la mejor configuración de parámetros del algoritmo BRKGA para luego proceder a la fase de pruebas finales, se muestran en el Cuadro 6.10 los resultados obtenidos con la instancia Gdb16. Por cada configuración, se efectuaron 50 corridas y el tiempo máximo de CPU asignado por corrida fue establecido en 45 segundos.

Cuadro 6.10: Pruebas preliminares de BRKGA con instancia Gdb16 (TD=127 y Vh=5).

Configuración			Resultados					
Id	Pobl.	% Elite	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$\bar{Tiempo}$
C1	20	15	128.80	0.60	129	127	10.0	6.9
C2	20	20	128.80	0.60	129	127	10.0	6.6
C3	20	25	128.92	0.39	129	127	4.0	4.5
C4	30	15	128.76	0.65	129	127	12.0	6.0
C5	30	20	128.80	0.60	129	127	10.0	6.7
C6	30	25	128.68	0.74	129	127	16.0	9.3
C7	40	15	128.69	0.73	129	127	15.7	8.6
C8	40	20	128.76	0.65	129	127	12.0	4.8
C9	40	25	128.84	0.54	129	127	8.0	5.3

De las últimas pruebas efectuadas, todas las configuraciones obtienen como moda un valor levemente mayor al valor óptimo TD=127, alcanzándose este último, como menor distancia total recorrida. Respecto a los tiempos computacionales promedio, estos son bajos (entre 4.8 y 9.3 segundos). Las configuraciones C6 (Población=30 y % Elite=25) y C7 (Población=40 y % Elite=15) resultaron ser las mejores, considerando el porcentaje de ocurrencia de la menor distancia recorrida y el tiempo computacional promedio. De nuevo, de las configuraciones C1 y C9, se puede apreciar que una mayor población de cadenas de claves aleatorias y de tipo Elite no aseguran mejores resultados.

En base a los resultados de las pruebas preliminares usando las configuraciones de parámetros propuestas con el objetivo de alcanzar la menor distancia recorrida en el

menor tiempo computacional posible, se usará la configuración C5 (Población=30 y % Elite=20) para las pruebas finales. Si bien esta configuración no es la mejor en todas las pruebas, la misma obtiene resultados competitivos en todos los casos.

Las Fig. 6.13, 6.14 y 6.15 corresponden a los histogramas de las corridas del algoritmo BRKGA usando la configuración C5 para las instancias Kshs6, Gdb1 y Gdb16. En las mismas, se observa un alto porcentaje de efectividad alcanzando los valores óptimos, salvo en la instancia Gdb16. Luego, en las Fig. 6.16, 6.17 y 6.18 se muestra el comportamiento del algoritmo sobre 3 corridas seleccionadas (usando la misma configuración), una por cada instancia usada en esta fase de prueba preliminar. De los diagramas de dispersión, se aprecia cómo a lo largo de las generaciones alcanzadas para cada caso (según los tiempos de CPUs adoptados), el algoritmo procede a buscar las mejores soluciones de manera no aleatoria y monótona.

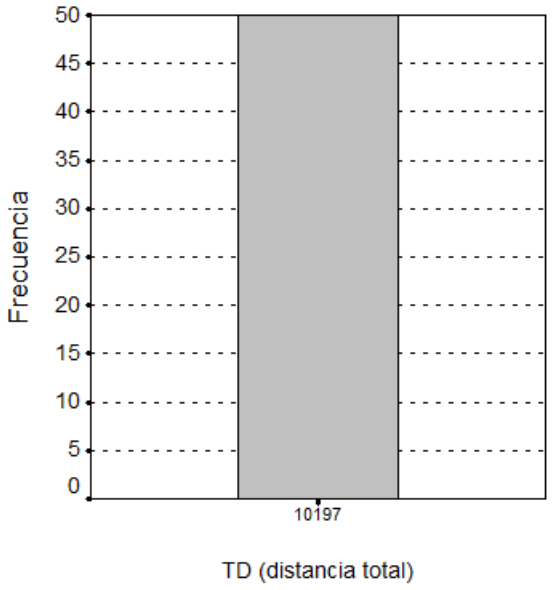

Figura 6.13: Pruebas BRKGA: histograma de Kshs6.

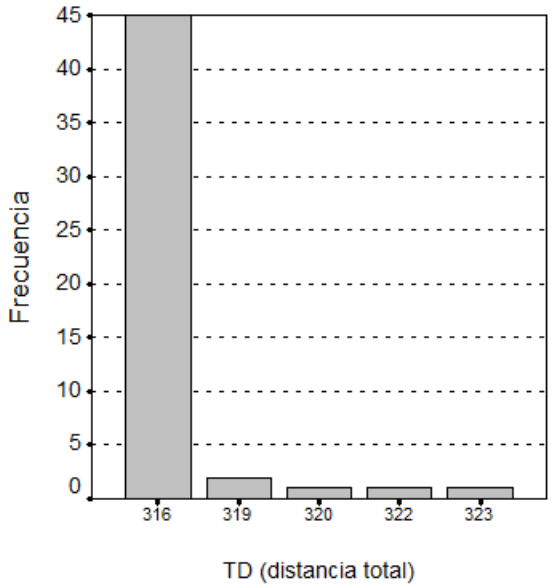

Figura 6.14: Pruebas BRKGA: histograma de Gdb1.

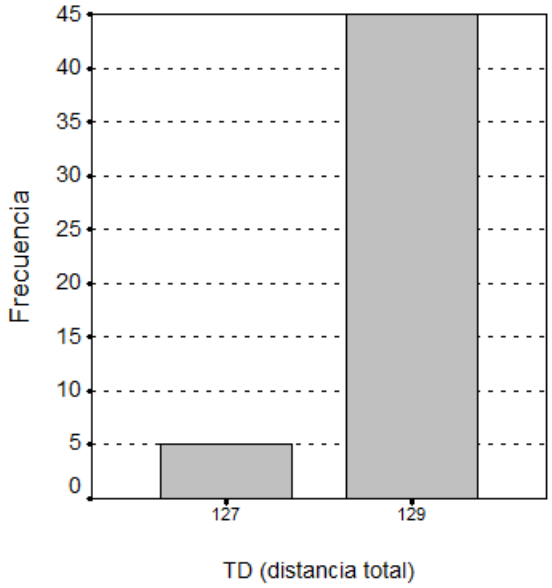

Figura 6.15: Pruebas BRKGA: histograma de Gdb16.

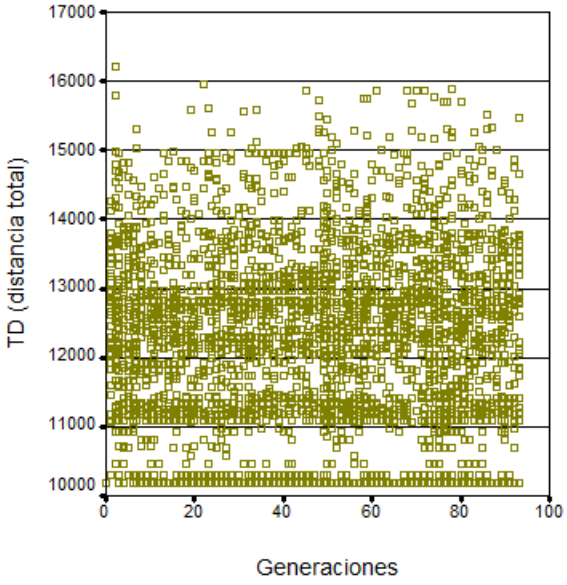

Figura 6.16: Pruebas BRKGA: diagrama de dispersión de Kshs6.

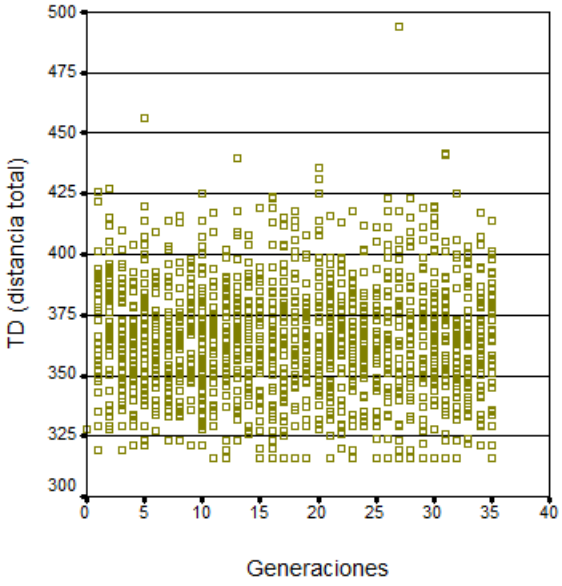

Figura 6.17: Pruebas BRKGA: diagrama de dispersión de Gdb1.

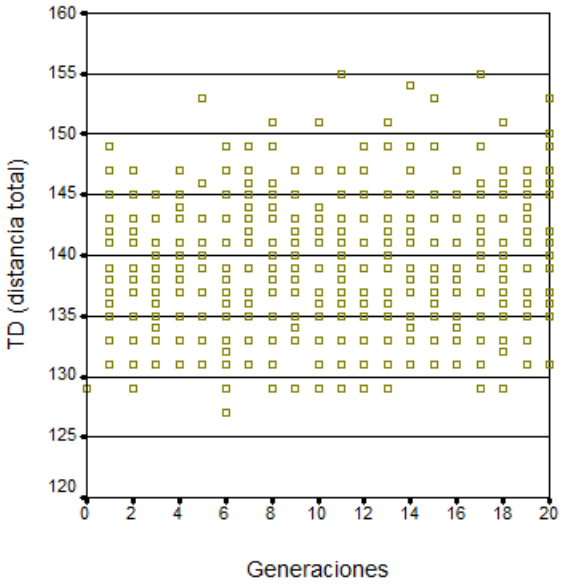

Figura 6.18: Pruebas BRKGA: diagrama de dispersión de Gdb16.

6.5. Comparación entre los algoritmos propuestos

Usando las mejores configuraciones de parámetros obtenidas a través de pruebas preliminares (secciones 6.2, 6.3 y 6.4), en esta sección se procederá a evaluar de manera conjunta, los algoritmos propuestos para CARP usando nuevas instancias de prueba provenientes de la literatura.

En el Cuadro 6.11, se lista información de las instancias utilizadas en las pruebas finales. Para cada una de ellas, se indica además del nombre, el número de vértices, el número de aristas requeridas, la demanda requerida, la capacidad vehicular, el resultado óptimo ([16], [84]) consistente de la menor distancia total recorrida y número de vehículos, y el tiempo máximo de CPU fijado para las corridas correspondientes. Para las pruebas finales de los algoritmos se realizaron 50 corridas independientes por instancia. Las mismas se realizaron con el software y hardware empleado previamente (ver sección 6.1).

En el Cuadro 6.12 se muestran los resultados obtenidos por el algoritmo HBMO adoptando una única configuración de parámetros (consistente de 1 abeja reina, 30 zánganos, 150 soluciones iniciales y 3 reinicios de algoritmo) sobre las instancias finales y las 3 instancias utilizadas durante las pruebas preliminares. Además del nombre de la instancia, se muestra la distancia total promedio $\bar{TD}$, el desvío estándar DE, la moda MO, la menor distancia total alcanzada $Min.TD$, el porcentaje de ocurrencia de dicho valor $\%Min.TD$, la mayor distancia alcanzada $Max.TD$ y el tiempo computacional promedio $\bar{Tiempo}$ (en segundos) para alcanzar el mejor resultado.

Cuadro 6.11: Pruebas finales: información de instancias usadas.

Instancia	$\|V\|$	$\|R\|$	Demanda	Q	Optimo		Tpo. CPU(s.)
					TD	Veh.	
Kshs1	8	15	535	150	**14661**	**4**	15
Kshs2	10	15	497	150	**9863**	**4**	15
Kshs3	6	15	565	150	**9320**	**4**	15
Kshs4	8	15	594	150	**11498**	**4**	15
Kshs5	8	15	443	150	**10957**	**3**	15
Gdb3	12	22	22	5	**275**	**5**	20
Gdb10	12	25	37	10	**275**	**4**	20
Gdb14	7	21	89	21	**100**	**5**	20
Gdb15	7	21	112	37	**58**	**4**	20
Gdb17	8	28	168	41	**91**	**5**	25
Gdb18	9	36	153	37	**164**	**5**	90
Gdb20	11	22	107	27	**121**	**4**	20
Gdb21	11	33	154	27	**156**	**6**	30
Gdb22	11	44	205	27	**200**	**8**	60
Val1A	24	39	358	200	**173**	**2**	90
Val2A	24	34	310	180	**227**	**2**	120
Val3A	24	35	137	80	**81**	**2**	120
Val3B	24	35	137	50	**87**	**3**	90
Val6A	31	50	451	170	**223**	**3**	360
Val8A	30	63	566	200	**386**	**3**	600
Val8B	30	63	566	150	**395**	**4**	600
Val9A	50	92	654	235	**323**	**3**	1800

Cuadro 6.12: Pruebas finales de HBMO.

Instancia	Resultados						
	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$Max.TD$	$\bar{Tiempo}$
Kshs1	14689.56	33.90	14661	14661	56.9	14729	1.7
Kshs2	9863	0.0	9863	9863	100.0	9863	0.8
Kshs3	9339.82	57.56	9320	9320	86.3	9525	2.3
Kshs4	13335.04	1379.27	12095	11909	2.0	17424	6.2
Kshs5	11028.76	169.20	10957	10957	82.0	11729	0.8
Kshs6	10199.16	15.27	10197	10197	98.0	10305	0.6
Gdb1	317.94	3.06	316	316	70.0	323	7.3
Gdb3	277.46	4.81	275	275	76.0	288	5.5
Gdb10	275.36	1.43	275	275	92.0	283	1.6
Gdb14	100.72	0.96	100	100	64.0	102	4.7
Gdb15	58	0.0	58	58	100.0	58	0.6
Gdb16	127.76	1.13	127	127	66.0	131	10.9
Gdb17	91.92	0.27	92	91	8.0	92	2.9
Gdb18	165.32	1.28	164	164	44.0	168	16.9
Gdb20	121.88	1.84	121	121	64.0	131	4.5
Gdb21	160.20	2.83	158	156	5.9	169	15.1
Gdb22	207.80	2.95	209	202	2.0	215	32.4
Val1A	173.64	1.90	173	173	88.0	180	16.1
Val2A	231.46	4.88	227	227	28.0	249	21.4
Val3A	82.68	1.63	81	81	32.0	87	24.3
Val3B	88.64	1.61	88	87	27.5	92	21.6
Val6A	225.26	2.81	223	223	42.0	235	67.6
Val8A	400.38	10.71	392	386	4.0	433	232.1
Val8B	416.10	10.45	423	395	2.0	451	298.2
Val9A	336.37	5.88	332	325	2.0	348	692.6

Respecto de los resultados del cuadro anterior, se observa que:

- En la mayoría de las instancias, se obtuvo como menor distancia total recorrida, al valor óptimo. En 2 de las 3 instancias (Kshs4, Gdb22 y Val9A) en las que no se obtuvo dicho valor, la distancia es del 1 %.
- Los porcentajes de ocurrencia (sobre 50 corridas) de la menor distancia total recorrida son altos (en 18 de las 25 instancias, los porcentajes de ocurrencia son superiores al 25 %).
- Las distancias totales promedio, no superan el 5 % del valor óptimo (a excepción de la instancia Kshs4).
- Las modas alcanzadas en 17 de las 25 instancias, coinciden con los valores óptimos. Sobre las 8 instancias restantes, 4 de ellas (Gdb17, Gdb21, Val3B y Val8A) distan a lo sumo en un 1.5 % y las restantes no superan el 7 %.
- Los rangos (diferencia entre la máxima y la menor distancia total) varían en la mayoría de las instancias entre 0 y 10 % respecto del valor óptimo (a excepción de las instancias Kshs4, Val8A y Val8B).
- Los tiempos computacionales son altamente competitivos, incluso con instancias de complejidad media/alta. Dichos tiempos no superan el 50 % del tiempo máximo de CPU fijado para cada caso.

En síntesis, los resultados obtenidos muestran cómo el algoritmo HBMO propuesto alcanza resultados competitivos a bajo costo computacional. Esto se sustenta en una buena selección y actualización de zánganos (generación a generación) por crías de calidad y diversas, lo que permite un mejor proceso de diversificación e intensificación, dado que la exploración del espacio de soluciones se realiza de manera controlada y no aleatoria. Por otra parte, el uso de operadores clásicos de mejora inter-ruta e intra-ruta luego del cruzamiento entre abeja reina y zángano, contribuye al proceso de intensificación a bajo costo computacional. A su vez, la generación semi-aleatoria (con posterior mejora) de soluciones para una selección de los zánganos iniciales (de la colonia) y los posteriores al reinicio del algoritmo, le permiten al algoritmo alcanzar los mejores resultados de manera temprana, diferenciándose de los algoritmos basados en Algoritmos Genéticos que denotan una lenta convergencia hacia los mejores resultados. Finalmente, en algunos casos en que se

notó cierto estancamiento en óptimos locales, el reinicio propuesto le permitió al algoritmo salir de los mismos.

A continuación, se muestra en el Cuadro 6.13 los resultados alcanzados por el algoritmo VNS híbrido sobre las instancias usadas para las pruebas finales y preliminares, manteniendo una misma configuración de parámetros (α = 0,30, $|RefSet|$=20, K= $\frac{|R|}{4}$ y Número de No-Mejoras= $\frac{Tiempo\ CPU}{10}$). Para cada instancia, se incluye el nombre de la misma, la distancia total promedio, el desvío estándar, la moda, la menor distancia total alcanzada, el porcentaje de ocurrencia de dicho valor, la mayor distancia recorrida y el tiempo computacional promedio que se necesita para alcanzar el mejor resultado.

Cuadro 6.13: Pruebas finales de VNS.

Instancia	Resultados						
	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$Max.TD$	$\bar{Tiempo}$
Kshs1	14696.36	34.31	14729	14661	48.0	14729	6.1
Kshs2	9903.56	34.97	9863	9863	34.0	10022	4.7
Kshs3	9320.00	0.0	9320	9320	100.0	9320	3.0
Kshs4	12453.24	80.12	12487	12160	4.0	12487	3.9
Kshs5	10957.00	0.0	10957	10957	100.0	10957	3.3
Kshs6	10197.00	0.0	10197	10197	100.0	10197	1.9
Gdb1	317.80	2.76	316	316	66.0	323	14.34
Gdb3	280.74	2.32	281	275	2.0	287	10.6
Gdb10	275.00	0.0	275	275	100.0	275	4.7
Gdb14	101.76	0.82	102	100	15.7	103	7.9
Gdb15	58	0.0	58	58	100.0	58	1.9
Gdb16	127.04	0.28	127	127	98.0	129	10.4
Gdb17	91.00	0.0	91	91	100.0	91	7.9
Gdb18	164.08	0.39	164	164	96.0	166	32.1
Gdb20	122.14	1.73	121	121	64.0	126	8.3
Gdb21	158.82	1.74	158	156	7.8	163	16.4
Gdb22	202.16	1.20	203	200	14.0	204	34.7
Val1A	174.26	1.88	173	173	56.6	179	31.9
Val2A	231.04	3.03	231	227	18.0	239	50.7
Val3A	81.70	0.58	82	81	34.0	84	46.6
Val3B	91.74	1.32	92	88	2.0	94	42.6
Val6A	228.60	2.74	229	223	4.0	235	190.9
Val8A	403.38	5.37	401	389	2.0	416	264.8
Val8B	428.70	6.60	433	411	2.0	442	303.6
Val9A	338.25	3.11	339	329	2.0	344	858.4

De los resultados alcanzados por el algoritmo VNS híbrido, se puede apreciar que:

- De las pruebas realizadas, en la mayoría de los casos la menor distancia total alcanzada coincide con el valor óptimo. Respecto a las instancias que no alcanzaron dicho valor (5 de 25 instancias), las brechas en 3 de ellas no superan el 1.8 % (Val8A 1 %, Val8B 1.8 % y Val3B 1.1 %).
- En más de la mitad de las intancias (14 sobre 25 instancias), los porcentajes de ocurrencia de la menor distancia total coincidentes con el valor óptimo, superan el 25 %. En 6 instancias (Kshs3, Kshs5, Kshs6, Gdb10, Gdb15, Gdb17) se obtuvo el 100 % de coincidencia; en tanto en otras 2 instancias (Gdb16 y Gdb18), se alcanzó el 98 y 96 % respectivamente.
- La mayoría de las distancias totales promedio no superan el 5 % de los respectivos valores óptimos. Las 3 instancias que superan dicho umbral (Kshs4, Val3B y Val8B) tienen una brecha de a lo sumo 8.5 %.
- Las modas obtenidas en 12 de las 25 instancias coinciden con el valor óptimo. De las 13 restantes, 6 instancias (Kshs1, Gdb14, Gdb21, Gdb22, Val2A y Val3A) tienen brechas no superiores a 2 %, 4 instancias (Gdb3, Val6A, Val8A y Val9A) alcanzan brechas inferiores a 5 % y las 3 restantes (Kshs4, Val3B y Val8B) no superan 9.6 %.
- Los rangos entre menor y mayor distancia total recorrida son bajos. En 20 instancias, los rangos no superan el 5 % de los respectivos valores óptimos. En las 5 instancias restantes (Val2A, Val3B, Val6A, Val8A y Val8B), sus rangos no superan el 7.8 %.
- Los tiempos computacionales promedio necesarios para alcanzar buenos resultados para cada instancia son competitivos. A excepción de 1 instancia (Gdb1), dichos tiempos no superan el 60 % del tiempo de CPU dispuesto para las corridas.

De los resultados alcanzados, puede apreciarse que la conjunción de GRASP para la construcción de soluciones y la mejora de las mismas mediante VNS (junto a operadores de mejora de tipo inter-ruta e intra-ruta) y Path Relinking resultó exitosa. Respecto a la construcción de soluciones, tanto GRASP como la heurística Inserción más Económica logran soluciones de calidad y diversidad media, a bajo costo computacional. Con valores porcentuales similares (según información estadística de las corridas), ambas alcanzan soluciones que luego de ser mejoradas, se convierten en los mejores resultados de las

corridas. A su vez, la construcción de soluciones usando información de largo plazo (basado en Tabu Search) y GRASP, le ha permitido al algoritmo además de explorar nuevas regiones de soluciones, obtener en varias instancias las mejores soluciones iniciales. Es por ello que, la construcción de soluciones a partir de información parcial obtenida a lo largo de una corrida, ha resultado ser más efectiva que el reinicio del algoritmo desde una buena solución o desde una generada al azar (propuesto por Tabu Search y Iterated Local Search). Respecto a la mejora de soluciones, VNS y Path Relinking brindan al algoritmo propuesto la posibilidad de intensificar el proceso de búsqueda de soluciones desde dos propuestas diferentes y al mismo tiempo, efectivas. A partir de una solución inicial, usando dos estructuras de vecindarios y operadores de búsqueda local, VNS obtiene resultados competitivos. Además, ofrece una participación diferente de los operadores de búsqueda local comparado con lo propuesto en el algoritmo HBMO. Activado de manera periódica y mediante el uso de soluciones de calidad y diversas obtenidas a lo largo de la ejecución del algoritmo, Path Relinking intensifica el proceso de búsqueda a través de trayectorias desde soluciones iniciales hacia soluciones guía, incluyendo a los operadores de mejora, los que se activan luego de realizar movimientos en las soluciones intermedias hacia una solución guía.

Finalmente, en el Cuadro 6.14, se muestran los resultados alcanzados por el algoritmo BRKGA sobre las instancias seleccionadas para las pruebas finales y preliminares, manteniendo una misma configuración de parámetros (Tamaño de la población=30, Individuos elite=20 %, Individuos mutantes=20 %, probabilidad de cruzamiento=70 %, Tiempo para reinicio= $\frac{Tiempo\ CPU}{4}$). Por cada instancia, se indica el nombre, la distancia total promedio, el desvío estándar, la moda, la menor distancia total, el porcentaje de ocurrencia del valor anterior, la mayor distancia recorrida y el tiempo de CPU promedio (en segundos).

Cuadro 6.14: Pruebas finales de BRKGA.

Instancia	Resultados						
	$\bar{TD}$	DE	MO	$Min.TD$	$\%Min.TD$	$Max.TD$	$\bar{Tiempo}$
Kshs1	14661	0.0	14661	14661	100.0	14661	3.6
Kshs2	9863	0.0	9863	9863	100.0	9863	0.6
Kshs3	9323.48	17.22	9320	9320	96.0	9407	3.4
Kshs4	12310.7	818.49	11498	11498	18.0	15767	8.6
Kshs5	10994.04	99.15	10957	10957	86.0	11299	4.5
Kshs6	10197.00	0.0	10197	10197	100.0	10197	0.7
Gdb1	316.46	1.48	316	316	90.0	323	12.7
Gdb3	277.12	2.30	275	275	52.0	281	8.7
Gdb10	278.52	3.86	275	275	42.0	287	9.9
Gdb14	100.36	0.77	100	100	82.0	102	6.7
Gdb15	58	0.0	58	58	100.0	58	0.6
Gdb16	128.80	0.60	129	127	10.0	129	6.7
Gdb17	91.00	0.0	91	91	100.0	91	1.6
Gdb18	165.08	1.00	166	164	46.0	166	32.5
Gdb20	134.02	9.50	129	121	6.0	160	13.3
Gdb21	168.58	4.72	172	156	2.0	178	25.9
Gdb22	211.24	4.12	209	204	2.0	226	54.4
Val1A	173.14	0.60	173	173	92.0	177	35.5
Val2A	227.56	1.21	227	227	76.0	233	51.6
Val3A	81.58	0.67	81	81	52.0	83	43.5
Val3B	91.38	1.56	92	87	2.0	95	41.7
Val6A	229.46	2.91	229	223	2.0	237	190.9
Val8A	402.22	6.10	403	387	2.0	414	317.6
Val8B	428.34	6.06	430	409	2.0	440	259.5
Val9A	340.12	2.22	340	336	4.0	344	884.3

Según los resultados logrados por el algoritmo BRKGA:

- La menor distancia total coincide con el valor óptimo en 21 de las 25 instancias. Respecto a las 4 instancias restantes, las brechas no sobrepasan el 4 % (Gdb22 2 %, Val8A 0.2 %, Val8B 3.5 % y Val9A 4 %).
- El porcentaje de ocurrencia de la menor distancia coincidente con el valor óptimo, es alto. En 15 instancias, el porcentaje es superior al 25 %, obteniéndose en 5 de ellas (Kshs1, Kshs2, Kshs6, Gdb15 y Gdb17) 100 % de efectividad.
- La distancia total promedio no supera 10.7 % del valor óptimo. En 19 instancias, la brecha no sobrepasa el 5 %.
- La moda coincide con el valor óptimo en 15 de 25 instancias. De las 10 restantes, 5 de ellas (Gdb16, Gdb18, Gdb22, Val6A y Val8A) tienen brechas inferiores al 5 %.
- El rango entre mayor y menor distancia total recorrida es inferior al 5 % (del valor óptimo) en 17 de 25 instancias. De las 8 restantes, 4 instancias tienen rangos inferiores al 10 %.
- El tiempo de CPU utilizado para alcanzar buenos resultados es inferior al 60 % (del máximo dispuesto) en 22 de 25 instancias (a excepción de Gdb20, Gdb21 y Gdb22).

La calidad de los resultados alcanzados se debe a la elección e interacción acertada de las partes que forman el algoritmo BRKGA propuesto. Es necesario obtener poblaciones iniciales tal que permitan al algoritmo intensificar el proceso de búsqueda en diferentes regiones del espacio de soluciones y de esta manera, alcanzar los mejores resultados. Es asi que la heurística Iterated Tour Partitioning junto a operadores de búsqueda local, han proporcionado poblaciones iniciales que, según resultados estadísticos de las corridas, permitieron al algoritmo obtener los mejores resultados incluso en la fase inicial del mismo. Aunque en menor proporción, también desde los reinicios del algoritmo (consistente en la generación de una nueva población de inviduos No-Elite y posterior mejora de los mismos), se alcanzaron los mejores resultados. A su vez, el proceso de intensificación debe ser tal que a medida que transcurre el tiempo de CPU dispuesto, la calidad de los resultados en las sucesivas generaciones, debe incrementarse de manera rápida. El cruzamiento dispuesto entre individuos Elite y No-Elite (y viceversa) ha permitido una exploración de dos tipos de entornos (de calidades distintas), desde los cuales se han alcanzado buenos resultados. El

uso de listas de vecinos (de aristas requeridas) con algunos operadores de mejora inter-ruta e intra-ruta, permitieron al algoritmo a partir de una solución obtenida por cruzamiento, profundizar la búsqueda de mejores soluciones únicamente en regiones atractivas. Por otra parte, el uso del operador mutación utilizado en Algoritmos Genéticos, ha permitido alcanzar buenos resultados, aunque en menor proporción que el cruzamiento. Finalmente, es destacable la velocidad del algoritmo para obtener buenos resultados, dada que dicha característica está ausente en la mayoría de los algoritmos evolutivos.

En el Cuadro 6.15, puede apreciarse (de manera parcial) los resultados alcanzados por los tres algoritmos propuestos para CARP. Para cada instancia, se indica además del nombre, la menor distancia total alcanzada, el porcentaje de ocurrencia de dicho valor, el tiempo promedio de CPU (en segundos) que demora cada algoritmo en alcanzar el mejor resultado posible y el costo total de recorrido óptimo.

Cuadro 6.15: Pruebas finales: comparación de resultados.

Instancia	Algoritmo HBMO			Algoritmo VNS			Algoritmo BRKGA			Opt.
	$Min.TD$	$\%Min.TD$	$Tiempo$	$Min.TD$	$\%Min.TD$	$Tiempo$	$Min.TD$	$\%Min.TD$	$Tiempo$	
Kshs1	14661	56.9	1.7	14661	48.0	6.1	14661	100.0	3.6	14661
Kshs2	9863	100.0	0.8	9863	34.0	4.7	9863	100.0	0.6	9863
Kshs3	9320	86.3	2.3	9320	100.0	3.0	9320	96.0	3.4	9320
Kshs4	11909	2.0	6.2	12160	4.0	3.9	11498	18.0	8.6	11498
Kshs5	10957	82.0	0.8	10957	100.0	3.3	10957	86.0	4.5	10957
Kshs6	10197	98.0	0.6	10197	100.0	1.9	10197	100.0	0.7	10197
Gdb1	316	70.0	7.3	316	66.0	14.34	316	90.0	12.7	316
Gdb3	275	76.0	5.5	275	2.0	10.6	275	52.0	8.7	275
Gdb10	275	92.0	1.6	275	100.0	4.7	275	42.0	9.9	275
Gdb14	100	64.0	4.7	100	15.7	7.9	100	82.0	6.7	100
Gdb15	58	100.0	0.6	58	100.0	1.9	58	100.0	0.6	58
Gdb16	127	66.0	10.9	127	98.0	10.4	127	10.0	6.7	127
Gdb17	91	8.0	2.9	91	100.0	7.9	91	100.0	1.6	91
Gdb18	164	44.0	16.9	164	96.0	32.1	164	46.0	32.5	164
Gdb20	121	64.0	4.5	121	64.0	8.3	121	6.0	13.3	121
Gdb21	156	5.9	15.1	156	7.8	16.4	156	2.0	25.9	156
Gdb22	202	2.0	32.4	200	14.0	34.7	204	2.0	54.4	200
Val1A	173	88.0	16.1	173	56.6	31.9	173	92.0	35.5	173
Val2A	227	28.0	21.4	227	18.0	50.7	227	76.0	51.6	227
Val3A	81	32.0	24.3	81	34.0	46.6	81	52.0	43.5	81
Val3B	87	27.5	21.6	88	2.0	42.6	87	2.0	41.7	87
Val6A	223	42.0	67.6	223	4.0	190.9	223	2.0	190.9	223
Val8A	386	4.0	232.1	389	2.0	264.8	387	2.0	317.6	386
Val8B	395	2.0	298.2	411	2.0	303.6	409	2.0	259.5	395
Val9A	325	2.0	692.6	329	2.0	858.4	336	4.0	884.3	323

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

En esta tesis, se ha propuesto el desarrollo de algoritmos basados en la hibridización de metaheurísticas para el Problema de Ruteo de Arcos Capacitados. La posibilidad de dotar a las metaheurísticas de nuevas características basadas en otras, permite no sólo un mejor tratamiento integral de un problema de optimización (como por ejemplo, los algoritmos basados en GRASP para la construcción de soluciones y posterior mejora a cargo de Path Relinking [105]), sino también de nuevas propuestas para abordar un problema conocido de la literatura ([29], [99], [107], entre otros).

Respecto a la primera propuesta, el algoritmo Honey-Bee Mating Optimization (HBMO) para CARP ha alcanzado de manera exitosa los valores óptimos (en 22 de 25 instancias y en 2 de las 3 instancias restantes, la brecha fue del 1 %) y a bajo costo computacional (en todas las pruebas, se necesitó menos del 50 % del máximo tiempo de CPU asignado). Los aportes del algoritmo consisten en la hibrización de HBMO con ciertas ideas basadas en Scatter Search, el bajo número de parámetros que necesita para su ejecución (número de zánganos, tamaño de población de soluciones generadas inicialmente y número de vuelos reproductivos) comparado con otras propuestas también basadas en HBMO y la aplicación de HBMO para CARP. La generación inicial de zánganos con posterior selección (en términos de función objetivo y diversidad) para obtener la población de zánganos inicial y de cada reinicio, el cruzamiento de todos los zánganos con la abeja reina, el reemplazo de zánganos por crías mejores y el reinicio del algoritmo ante

estancamiento en óptimos locales, pueden proveer a HBMO de mayor potencialidad y robustez en su aplicación a diferentes problemas de optimización. Por otra parte, el bajo número de parámetros que posee el algoritmo permite para todo problema, la posibilidad de realizar una fase preliminar de pruebas de manera intensiva a los efectos de obtener una buena configuración de parámetros que permita en las pruebas finales, la posibilidad de alcanzar los mejores resultados de manera robusta. De las pruebas realizadas y del análisis global de resultados, consideramos a esta propuesta como la mejor de las tres dado que ha alcanzado muy buenos resultados con poco uso de CPU sobre instancias de diferentes tamaños y complejidades de resolución.

Respecto a la segunda propuesta, ésta ha demostrado una gran cooperación para alcanzar buenos resultados, entre la metaheurística VNS con GRASP y Path Relinking. En base a las pruebas efectuadas, el algoritmo VNS híbrido ha alcanzado resultados competitivos (se alcanzaron los valores óptimos en 20 de 25 instancias y en 3 de las instancias restantes alcanza una brecha no superior al 1.8 %), con baja variabilidad de resultados sobre diferentes corridas (en las corridas realizadas sobre 8 instancias, el porcentaje de ocurrencia del valor óptimo fue superior al 95 %) usando poco tiempo de CPU (el tiempo de CPU promedio no superó el 60 % del máximo dispuesto), entre otros. Es por ello que la hibridización de VNS a través de la construcción de soluciones (con información de corto plazo) mediante GRASP, la reutilización de soluciones mejoradas por VNS a través de Path Relinking como nuevo mecanismo de mejora y el mecanismo de reinicio del algoritmo ante estancamiento en óptimos locales, por medio del uso de memoria de largo plazo (basado en Tabu Search) y la construcción de soluciones con dicha información mediante GRASP, han sido aportes importantes para la aplicación de VNS al problema de ruteo elegido. El éxito de la hibridización del algoritmo VNS responde básicamente a la reutilización constante de soluciones generadas a lo largo de la ejecución del algoritmo, tanto en la fase de construcción (incluido el reinicio del algoritmo) como en la de mejora. Finalmente, es destacable el bajo número de parámetros que posee el algoritmo (α para GRASP, $|RefSet|$ para Path Relinking, y número de iteraciones), dada la posibilidad de realizar más y mejores pruebas. En base a los resultados obtenidos por el algoritmo sobre diferentes tipos de instancias, consideramos que es una propuesta competitiva y aplicable sobre instancias pequeñas y medianas. Sobre instancias grandes, requiere de un mayor tiempo de CPU (al usado sobre las pruebas) para alcanzar resultados de mejor calidad.

Respecto a la tercera, los resultados de las pruebas han sido muy satisfactorios (se alcanzaron los valores óptimos en 21 de 25 instancias y en 2 de las 4 instancias restantes, las brechas no superaron el 2 %) lo que de alguna manera, valida las características incluidas en el algoritmo BRKGA propuesto. Dos aspectos desfavorables que presentan los algoritmos evolutivos es la gran variabilidad de los resultados y la lenta convergencia hacia resultados de calidad. De las pruebas efectuadas, ninguna de estas características está presente en el algoritmo BRKGA, dado que éste logra una baja variabilidad de resultados (en 19 instancias, la distancia total promedio en cada caso alcanza una brecha inferior al 5 %), y gran rapidez en la obtención de los mejores resultados (en general, el tiempo de CPU utilizado no superó el 60 % del máximo tiempo asignado). Esto fue posible mediante la generación de soluciones con una heurística constructiva con posterior mejora de soluciones mediante operadores de búsqueda local inter-ruta e intra-ruta, el cruzamiento rápido de soluciones (de tipo Elite y No-Elite) mediante la codificación de soluciones CARP en cadenas de claves aleatorias, la mejora de soluciones mediante el uso de lista de vecinos y operadores de búsqueda local aplicados sobre dos tipos de entornos (uno por cada nuevo individuo obtenido por cruzamiento) y el reinicio periódico del algoritmo sobre la población de soluciones No-Elite. Estas características convierten al algoritmo BRKGA desarrollado en una propuesta interesante a ser considerada más alla del problema de ruteo de vehículos elegido. Del análisis de resultados sobre las pruebas efectuadas, consideramos esta propuesta competitiva (al igual que el algoritmo VNS híbrido) sobre instancias medianas y pequeñas. En instancias de mayor tamaño, precisa un mayor tiempo de CPU que las dos propuestas anteriores para lograr resultados de mejor calidad.

Es importante destacar que tanto HBMO como BRKGA no habían sido aplicadas previamente al CARP.

También se compararon los resultados alcanzados por los tres algoritmos propuestos con los del Tabu Search de Hertz et al.[65], Tabu Search de Brandão y Eglese [16], ACO de Lacomme et al.[78] y Algoritmos Meméticos de Lacomme et al.[76]. Estos algoritmos fueron testeados en las mismas instancias que los nuestros y reportan haber obtenido también los valores óptimos en la mayor parte de dichos casos.

7.2. Trabajo futuro

El trabajo futuro está definido en tres líneas de acción, las cuales se esperan concretar a corto plazo.

La primera consiste en la mejora de determinados aspectos de los algoritmos propuestos. Respecto a HBMO, la incorporación de otros operadores de cruzamiento basados en Algoritmos Genéticos y Scatter Search por ejemplo, permitirían una mayor variedad del tipo de soluciones iniciales a ser mejoradas. Respecto a VNS, un uso más intensivo de diferentes memorias de largo plazo podrían permitir la construcción de soluciones siguiendo diferentes propuestas que ayuden a alcanzar mejores resultados en menor tiempo. A su vez, nuevas estructuras de vecindarios pueden proporcionar mayor efectividad al algoritmo. Respecto a BRKGA, la generación de soluciones iniciales con diferentes algoritmos permitiría una exploración (posterior) más variada del espacio de soluciones. Así también, la posibilidad de medir la diversidad en las soluciones, puede contribuir a una mejor selección de soluciones, tanto en la generación inicial de las mismas como en la selección y/o reemplazo de inviduos involucrados en el cruzamiento, lo cual redundaría en una reducción del tiempo computacional.

La segunda está relacionada con la aplicación de los algoritmos propuestos a casos reales. Esto es factible a través de la firma de convenios de reciprocidad entre la Universidad (mediante proyectos de investigación) y organismos o empresas del medio. La concreción de convenios posibilitaría un mayor acercamiento al medio, mediante la transferencia de soluciones tecnológicas, la capacitación del personal y el asesoramiento en temas específicos, entre otras acciones. Como contrapartida, también brinda la posibilidad de captar desde las organizaciones, nuevos problemas de estudio y tratamiento desde el ámbito académico.

La tercera está referida a las extensiones de CARP. A partir del estudio realizado sobre el problema en cuestión y dada la aplicación real y concreta que provee el ruteo de arcos capacitados en diferentes problemas reales (tales como recolección de residuos, entrega de mercadería, limpieza de calles, medición de medidores eléctricos, etc), se prevee adaptar los algoritmos propuestos a diferentes extensiones del problema. La posibilidad de considerar múltiples depósitos, puntos intermedios de carga/descarga, ventanas de tiempo para atender los servicios requeridos por mencionar sólo algunas extensiones, permitirían

resolver de manera eficiente y rápida, los problemas que comunmente tienen las empresas/organizaciones relacionadas con la planificación de rutas.

Bibliografía

[1] Abbass, H. A. MBO: Marriage in honey bees optimization: A haplometrosis polygynous swarming approach. *Congress on Evolutionary Computation* (2001), 207–214.

[2] Afshar, A., Haddad, B., Mariño, M., and Adams, B. Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation. *Journal of the Franklin Institute 344* (2007), 452–462.

[3] Ahuja, R., Orlin, J., and Tiwari, A. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research 27* (2000), 917–934.

[4] Aiex, R., Binato, S., and Resende, M. Parallel GRASP with Path Relinking for job shop scheduling. *Parallel Computing 29* (2003), 393–430.

[5] Alvarez, R., Corberán, A., and Tamarit, J. La combinatoria poliédrica y el problema del viajante. Aplicación al caso de ciento tres ciudades españolas. *Qüestió 9-3* (1985), 199–213.

[6] Alvarez-Valdés, R., Parajón, A., and Tamarit, J. A Tabu Search algorithm for large-scalle guillotine (un)constrained two-dimensional cutting problems. *Computers and Operations Research 29* (2002), 925–947.

[7] Amaya, A., Langevin, A., and Trepanier, M. The capacitated arc routing problem with refill points. *Operations Research Letters 35-1* (2007), 45–53.

[8] Amberg, A., Domschke, W., and Voss, S. Multiple center capacitated arc routing problems: A Tabu Search algorithm using capacitated trees. *European Journal of Operational Research 124* (2000), 360–376.

[9] Applegate, D., Bixby, R., Chvátal, V., and Cook, W. *The Travelling Salesman Problem: a computational study.* Princeton University Press, New Jersey, 2007.

[10] Bean, J. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing 6* (1994), 154–160.

[11] Belenguer, J., and Benavent, E. A Cutting Plane algorithm for the capacitated arc routing problem. *Computers and Operations Research 30* (2003), 705–720.

[12] Belov, G., and Scheithauer, G. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research 171* (2006), 85–106.

[13] Benavent, E., Campos, V., Corberán, A., and Mota, E. The capacitated arc routing problem: lower bounds. *Networks 22* (1992), 669–690.

[14] Beullens, P., Muyldermans, L., Cattrysse, D., and Oudheusden, D. V. A Guided Local Search heuristic for the capacitated arc routing problem. *European Journal of Operational Research-Discrete Optimization 147* (2003), 629–643.

[15] Boyer, V., Elkihel, M., and Baz, D. E. Heuristics for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research 199-3* (2009), 658–664.

[16] Brandão, J., and Eglese, R. A deterministic Tabu Search algorithm for the capacitated arc routing problem (CARP). *Computers and Operations Research 35-4* (2008), 1112–1126.

[17] Burkard, R., Karisch, S., and Rendl, F. Qaplib: A quadratic assignment problem library. *Journal of Global Optimization 10* (1997), 391–403.

[18] Burke, E., Causmaecker, P. D., Petrovic, S., and Bergue, G. V. Variable Neighborhood Search for nurse rostering problems. In *Metaheuristics: Computer decision-making*, M. Resende and P. de Sousa, Eds., vol. 86 of *Applied Optimization.* Kluwer Academic Publishers, 2004, pp. 153–172.

[19] Cavique, L., Rego, C., and Themido, I. Subgraph Ejection Chains and Tabu Search for the scheduling problem. *Journal of the Operational Research Society 50* (1999), 608–616.

[20] Cavique, L., Rego, C., and Themido, I. A Scatter Search for the maximum clique problem. In *Essays and Surveys in Metaheuristics*, C. Ribeiro and P. Hansen, Eds., vol. 15 of *Operations Research/Computer Science Interfaces*. Kluwer Academic Publishers, 2001, pp. 227–244.

[21] Chu, F., Labadi, N., and Prins, C. A Scatter Search for the periodic capacitated arc routing problem. *European Journal of Operational Research 169* (2004), 586–605.

[22] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to algorithms*. MIT Press, Massachusetts, 2001.

[23] Croes, G. A method for solving travelling salesman problems. *Operations Research 6* (1958), 791–812.

[24] Curkovic, P., and Jerbic, B. Honey-bees optimization algorithm applied to path planning problem. *International Journal of simulation modelling 6-3* (2007), 154–164.

[25] de Souza, M., Duhamel, C., and Ribeiro, C. A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In *Metaheuristics: Computer decision-making*, M. Resende and J. de Sousa, Eds. Kluwer Academic Publishers, 2003, pp. 237–261.

[26] de Souza, M., and Martins, P. Skewed VNS enclosing second order algorithm for the degree constrained minimum spanning tree problem. *European Journal of Operational Research 191* (2008), 677–690.

[27] Degraeve, Z., and Jans, R. A new Dantzig-Wolfe reformulation and branch-and-price algorithm for the capacitated lot-sizing problem with setup times. *Operations Research 55-5* (2007), 909–920.

[28] del Pia, A., and Filippi, C. A Variable Neighborhood Descent algorithm for a real waste collection problem with mobile depots. *International transactions in Operational Research 13* (2006), 125–141.

[29] Díaz, J., and Fernández, E. Hybrid Scatter Search and Path Relinking for the capacitated p-median problem. *European Journal of Operational Research 169* (2006), 570–585.

[30] Doerner, K., Hartl, R., and Maniezzo, V. An Ant System metaheuristic for the capacitated arc routing problem. In *Proceedings of the Fifth Metaheuristics International Conference* (2003), pp. 1–6.

[31] Dorigo, M., and Stützle, T. *Ant Colony Optimization*. MIT Press, Massachusetts, 2004.

[32] Dowsland, K., and Díaz, B. A. Heuristic design and fundamentals of the Simulated Annealing. *Inteligencia Artificial 19* (2003), 93–102.

[33] Edmonds, J., and Johnson, E. Matching euler tours and the chinese postman. *Mathematical Programming 5-1* (1973), 88–124.

[34] Eglese, R. Routing winter gritting vehicles. *Discrete applied mathematics 48-3* (1994), 231–244.

[35] Ericsson, M., Resende, M., and Pardalos, P. A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization 6* (2002), 299–333.

[36] Fathian, M., Amiri, B., and Maroosi, A. Aplication of honey-bee mating optimization algorithm on clustering. *Applied Mathematics and Computation 190* (2007), 1502–1513.

[37] Feo, T., and Resende, M. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters 8* (1989), 67–71.

[38] Feo, T., and Resende, M. Greedy randomized adaptive search procedure. *Journal of Global Computing 6* (1995), 109–133.

[39] Festa, P., and Resende, M. An annotated bibliography of GRASP, Part I: algorithms. *International Transactions in Operational Research 16* (2009), 1–24.

[40] Festa, P., and Resende, M. An annotated bibliography of GRASP, Part II: applications. *International Transactions in Operational Research 16* (2009), 131–172.

[41] Fischetti, M., Monaci, M., and Salvagnin, D. Three ideas for the quadratic assignment problem. *Operations Research submitted* (2011).

[42] Fleszar, K., and Hindi, K. S. An effective VNS for the capacitated p-median problem. *European Journal of Operational Research 191* (2008), 612–622.

[43] Fréville, A. The multidimensional 0-1 knapsack: An overview. *European Journal of Operational Research 155* (2004), 1–21.

[44] García, C., Perez-Brito, D., Campos, V., and Martí, R. Variable Neighborhood Search for the linear ordering problem. *Computers and Operations Research 33* (2006), 3549–3565.

[45] Garey, M., and Johnson, D. *Computers and Intractability: a guide to the theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[46] Geiger, M. J., and Wenger, W. On the assignment of students to topics: a Variable Neighborhood Search approach. *Socio-Economic Planning Sciences 44* (2010), 25–34.

[47] Ghiani, G., Guerriero, F., Improta, G., and Musmanno, R. Waste collection in southern italy: solution of a real-life arc routing problem. *International Transactions in Operational Research 12* (2005), 135–144.

[48] Glover, F. Tabu Search and adaptive memory programming - advances, applications and challenges. In *Advances in metaheuristics, optimization and stochastic modeling technologies*, R. Barr, R. Helgason, and J. Kennington, Eds., vol. 7 of *Interfaces.* Kluwer Academic Publishers, 1996, pp. 1–75.

[49] Glover, F. A template for Scatter Search and Path Relinking. *Lecture Notes in Computer Science 1363* (1997), 13–54.

[50] Glover, F., and Laguna, M. *Tabu Search.* Kluwer Academic Publishers, Boston, 1997.

[51] Glover, F., Laguna, M., and Martí, R. New ideas and applications of Scatter Search and Path Relinking. In *New Optimization Techniques in Engineering*, G. Onwubolu and B. Babu, Eds., vol. 141 of *Studies in Fuzziness and Soft Computing.* Springer-Verlag, 2004, pp. 367–384.

[52] Goldberg, D. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, Massachusetts, 1989.

[53] Golden, B., DeArmon, J., and Baker, E. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research 10-1* (1983), 47–59.

[54] Golden, B., Raghavan, S., and Wasil, E. *The vehicle routing problem: latest advances and new challenges*. Springer, New York, 2008.

[55] Golden, B., and Wong, R. Capacitated arc routing problems. *Networks 11* (1981), 305–315.

[56] Gonçalves, J. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research 183* (2007), 1212–1229.

[57] Gonçalves, J., and Almeida, J. A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics 8* (2002), 629–642.

[58] Gonçalves, J., de Magalhães Mendes, J., and Resende, M. A hybrid genetic algorithm-heuristic for the job shop scheduling problem. *European Journal of Operational Research 167* (2005), 77–95.

[59] Gonçalves, J., and Resende, M. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics published Online* (2010).

[60] Gonçalves, J., Resende, M., and Mendes, J. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics published Online* (2010).

[61] Groves, G., Roux, J. L., and Houren, J. V. Network service scheduling and routing. *International Transactions in Operational Research 11* (2004), 613–643.

[62] Haddad, O., Afshar, A., and Mariño, M. HBMO in engineering optimization. *Ninth International Water Technology Conference* (2005), 1053–1063.

[63] Haimovich, M., and Kan, A. Mathematics of operations research. *Science 10-4* (1985), 527–542.

[64] Hansen, P., and Mladenovic, N. Variable Neighborhood Search. In *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds., vol. 57 of *Operations Research and Management Science*. Kluwer Academic Publishers, 2003, pp. 145–184.

[65] Hertz, A., Laporte, G., and Mittaz, M. A Tabu Search heuristic for the capacitated arc routing problem. *Operations Research 48* (2000), 129–135.

[66] Hertz, A., and Mittaz, M. A Variable Neighborhood Descent algorithm for the undirected capacitated arc routing problem. *Transportation Science 35-4* (2001), 425–434.

[67] Hertz, A., Taillard, E., and de Werra, D. A tutorial on Tabu Search. In *Proceedings of Giornate di Lavoro AIRO 95* (1995), pp. 13–24.

[68] Hirabayashi, R., Saruwatari, Y., and Nishida, N. Tour construction algorithm for the capacitated arc routing problem. *Asia-Pacific Journal of Operational Research 9* (1992), 155–175.

[69] Hoos, H., and Stützle, T. *Stochastic local search: foundations and applications.* McGraw-Hill, San Francisco, 2005.

[70] Jaszkiewicz, A., and Zielniewicz, P. Pareto memetic algorithm with Path Relinking for bi-objective travelling salesman problem. *European Journal of the Operational Research 193* (2009), 885–890.

[71] Johnson, E., and Wohlk, S. Solving the capacitated arc routing problem with time windows using column generation. CORAL Working Paper L-2008-09, University of Aarhus, Aarhus School of Business, Department of Business Studies, 2009.

[72] Kim, B., Kim, S., and Sahoo, S. Waste collection vehicle routing problem with time windows. *Computers and Operations Research 33-12* (2006), 3624–3642.

[73] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. Optimization by Simulated Annealing. *Science 220-4598* (1983), 671–680.

[74] Kiuchi, M., Shinano, Y., Hirabayashi, R., and Saruwatari, Y. An exact algorithm for the capacitated arc routing problem using parallel Branch and Bound method. *Abstracts of the 1995 Spring National Conference of the Operational Research Of Japan* (1995), 28–29.

[75] Koudil, M., Benatchba, K., Tarabet, A., and Sahraoui, E. Using artificial bees to solve partitioning and scheduling problems in codesign. *Applied Mathematics and Computation 186* (2007), 1710–1722.

[76] Lacomme, P., Prins, C., and Ramdane-Cherif, W. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research 131-1* (2004), 159–185.

[77] Lacomme, P., Prins, C., and Ramdane-Cherif, W. Evolutionary algorithms for periodic arc routing problems. *European Journal of Operational Research 165* (2005), 535–553.

[78] Lacomme, P., Prins, C., and Tunguy, A. First competitive Ant Colony scheme for the CARP. *Lecture Notes in Computer Science 3172* (2004), 426–427.

[79] Laguna, M., and Martí, R. GRASP and Path Relinking for 2-layer straight line cross minimization. *INFORMS Journal on Computing 11-1* (1999), 44–52.

[80] Laguna, M., and Martí, R. Scatter Search. In *Metaheuristic procedures for training neural networks*, E. Alba and R. Martí, Eds., vol. 35 of *Interfaces*. Springer-Verlag, 2006, pp. 309–368.

[81] Li, Y., Pardalos, P., and Resende, M. A greedy randomized adaptive search procedure for the quadratic assignment problem. In *Quadratic assignment and related problems*, P. Pardalos and H. Wolkowicz, Eds., vol. 16 of *Discrete Mathematics and Theoretical Computer Science (DIMACS)*. American Mathematical Society, 1994, pp. 237–261.

[82] Lin, S. Computer solutions of the travelling salesman problem. *Bell System Technology Journal 44* (1965), 2245–2269.

[83] Lin, S., and Kernighan. An effective heuristic algorithm for the travelling-salesman problem. *Operations Research 21-2* (1973), 498–516.

[84] Longo, H., de Aragão, M. P., and Uchoa, E. Solving capacitated arc routing problems using a transformation to the CVRP. *Computers and Operations Research 33-6* (2006), 1823–1837.

[85] Lourenço, H., Martin, O., and Stützle, T. A gentle introduction to Iterated Local Search. *4th Metaheuristics International Conference* (2001), 1–11.

[86] Lourenço, H., and Serra, D. Adaptive search heuristics for the generalized assignment problem. *Matware and Soft Computing 7* (2000), 1–15.

[87] Maniezzo, V. Algorithms for large directed CARP instances: urban solid waste collection operational support. Technical Report UBLCS-2004-16, University of Bologna, Department of Computer Science, 2004.

[88] Martí, R., and Laguna, M. Scatter Search: basic design and advances strategies. *Inteligencia Artificial 19* (2003), 123–130.

[89] Martí, R., Laguna, M., and Campos, V. Scatter Search vs genetic algorithms: an experimental evaluation with permutation problems. In *Metaheuristic optimization via memory and evolution*, C. Rego and B. Alidaee, Eds., vol. 30 of *Operations Research/Computer Science Interfaces*. Kluwer Academic Publishers, 2005, pp. 263–283.

[90] Mills, P., Tsang, E., Zhang, Q., and Ford, J. A survey of AI-based meta-heuristics for dealing with local optima in local search. Technical Report Series CSM-416, University of Essex, Department of Computer Science, 2004.

[91] Mladenovic, N., Labbé, M., and Hansen, P. Solving the p-center problem with Tabu Search and Variable Neighborhood Search. *Networks 42* (2003), 48–64.

[92] Moscato, P., and Cotta, C. Introducción a los Algoritmos Meméticos. *Inteligencia Artificial 19* (2003), 131–148.

[93] Muydelrmans, L., Cattrysse, D., Oudheusden, D. V., and Lotan, T. Districting for salt spreading operations. *European Journal of Operational Research 139* (2002), 521–532.

[94] Nemhauser, G., and Wolsey, L. *Integer and combinatorial optimization*. Wiley-Interscience, New York, 1988.

[95] Pacheco, J. A Scatter Search approach for the minimum sum-of-squares clustering problem. *Computers and Operations Research 32-5* (2005), 1325–1335.

[96] Pacheco, J., and Valencia, O. Design of hybrids for the minimum sum-of-squares clustering problem. *Computational Statistics and Data Analysis 43-2* (2003), 235–248.

[97] Padberg, M., and Rinaldi, G. A branch-and-cut algorithm for the resolution of large-scale travelling salesman problem. *SIAM Review 33* (1991), 60.

[98] Polacek, M., Doerner, K., Hartl, R., and Maniezzo, V. A Variable Neighborhood Search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics 14-5* (2008), 405–423.

[99] Potvin, J., and Naud, M. Tabu Search with Ejection Chains for the vehicle routing problem with private fleet and common carrier. CIRRELT Technical Report 2009-50, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, 2009.

[100] Puchinger, J., Raidl, G., and Pferschy, U. The multidimensional knapsack problem: structure and algorithms. *Informs Journal on Computing 22-2* (2010), 250–265.

[101] Ralphs, T. Parallel branch and cut for capacitated vehicle routing. *Parallel Computing 29* (2003), 607–629.

[102] Reghioui, M., Prins, C., and Labadi, N. GRASP with Path Relinking for the capacitated arc routing problem with time windows. *Lecture Notes in Computer Science 4448* (2007), 722–731.

[103] Rego, C. A subpath ejection method for the vehicle routing problem. *Management Science 44-10* (1998), 13–54.

[104] Rego, C., and Glover, F. Local search and metaheuristics for the travelling salesman problem. In *The travelling salesman problem and its variations*, G. Gutin and A. Punnen, Eds., vol. 12 of *Combinatorial Optimization*. Kluwer Academic Publishers, 2002, pp. 309–368.

[105] Resende, M., and Ribeiro, C. GRASP with Path Relinking: recent advances and applications. In *Metaheuristics: Progress as real problem solvers*, T. Ibaraki, K. Nonobe, and M. Yagiura, Eds., vol. 32 of *Operations Research/Computer Science Interfaces*. Springer, 2005, pp. 29–63.

[106] Resende, M., and Velarde, J. GRASP: Procedimientos de búsqueda miopes aleatorizados y adaptativos. *Inteligencia Artificial 19-2* (2003), 61–76.

[107] Resende, M., and Werneck, R. A hybrid heuristic for the p-median problem. *Journal of Heuristics 10* (2004), 59–88.

[108] Resende, M., and Werneck, R. A fast swap-based local search procedure for location problems. *Annals of Operations Research 150* (2007), 205–230.

[109] Sahni, S., and Gonzalez, T. NP-Complete approximation problems. *Journal of the Association for Computing Machinery 23* (1976), 555–565.

[110] Sahoo, S., Kim, S., Kim, B., Kraas, B., and Popov, A. Routing optimization for waste management. *Interfaces 35-1* (2005), 24–36.

[111] Salkin, H., and Mathur, K. *Foundations of Integer Programming.* North Holland, New York, 1989.

[112] Samanlioglu, F., Ferrell, W., and Kurz, M. A memetic random-key genetic algorithm for a symmetric multi-objective salesman problem. *Computers and Industrial Engineering 55* (2008), 439–449.

[113] Scheuerer, S., and Wendolsky, R. A Scatter Search heuristic for the capacitated clustering problem. *European Journal of Operational Research 169* (2006), 533–547.

[114] Snyder, L., and Daskin, M. A random key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research 174* (2006), 38–53.

[115] Spears, W., and DeJong, K. On the virtues of Parameterized Uniform Crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (1991), pp. 230–236.

[116] Toth, P., and Vigo, D. *The Vehicle Routing Problem.* SIAM, Pennsylvania, 2001.

[117] Ulusoy, G. The fleet size and mixed problem for capacitated arc routing. *European Journal of Operational Research 22* (1985), 329–337.

[118] Vasquez, M., and Hao, J. A hybrid approach for the 0-1 multidimensional knapsack problem. In *Proceedings of the International Joint Conference on Artificial Intelligence* (2001), pp. 328–333.

[119] Velarde, J. G., and Laguna, M. Tabu Search with simple Ejection Chains for coloring graphs. *Annals of Operations Research 117* (2002), 165–174.

[120] Voss, S. Meta-heuristics: The state of the art. *Local Search for Planning and Scheduling 2148* (2001), 1–23.

[121] Voudouris, C. *Guided Local Search for combinatorial optimisation problems.* PhD thesis, University of Essex, Department of Computer Science, 1997.

[122] Whitley, D. A Genetic algorithm tutorial. *Statistics and Computing 4* (1994), 65–85.

[123] Wolsey, L. *Integer Programming.* Wiley-Interscience, New York, 1998.

[124] Yagiura, M., Ibaraki, T., and Glover, F. An Ejection Chains approach for the generalized assignment problem. *Informs Journal on Computing 16-2* (2004), 133–151.

[125] Yagiura, M., Ibaraki, T., and Glover, F. A Path Relinking approach with Ejection Chains for the generalized assignment problem. *European Journal of Operational Research 169* (2006), 548–569.

[126] Zhu, Z., Li, X., Yang, Y., Deng, X., Xia, M., and Xie, Z. A hybrid genetic algorithm for the multiple depot capacitated arc routing problem. In *Proceedings of the IEEE International Conference on Automation and Logistics* (2007), pp. 2253–2258.

Apéndice A

Instancias y Soluciones CARP

Cuadro A.1: Instancia CARP: Kshs4

Lista de aristas requeridas			
i	j	costo	demanda
1	4	873	42
1	5	640	58
1	7	444	34
2	3	583	20
2	7	740	53
3	4	471	63
3	5	293	16
3	6	441	65
4	6	866	50
4	8	318	37
5	6	334	42
5	7	466	1
6	7	602	51
6	8	563	22
7	8	357	40

Cuadro A.2: Instancia CARP: Gdb10

Lista de aristas requeridas			
i	j	costo	demanda
1	8	9	1
1	9	4	2
1	10	13	1
1	11	12	2
2	1	11	1
4	1	7	1
5	1	9	2
2	3	15	1
2	4	18	2
2	5	8	2
3	4	8	2
3	8	18	2
3	9	6	1
4	6	10	2
4	11	17	1
5	6	15	1
6	11	3	2
7	2	6	1
7	4	11	2
7	5	5	1
8	9	14	2
8	12	5	1
9	10	19	1
10	12	2	2
11	12	7	1

Cuadro A.3: Instancia CARP: Val3A

Lista de aristas requeridas			
i	j	costo	demanda
1	2	2	4
1	3	1	3
2	4	1	3
3	4	2	4
3	9	1	1
4	5	2	6
5	6	3	5
5	10	2	4
5	11	4	4
6	7	3	5
6	12	1	3
7	8	1	6
8	14	3	7
9	10	2	2
9	19	1	3
10	11	3	5
10	20	2	4
11	12	1	3
11	15	1	4
12	13	1	2
12	16	1	1
13	14	2	4
13	17	3	5
14	18	3	6
15	16	1	3
15	21	1	4
16	17	4	4

(continúa en la siguiente página)

i	j	costo	demanda
17	18	1	3
19	20	1	2
19	22	3	7
20	21	2	4
20	23	1	4
21	24	1	3
22	23	3	4
23	24	1	5

(fin de la tabla)

Cuadro A.4: Solución CARP: Kshs4

Ruta	Detalle
#1	1→5, 5→3, **3→6**, **6→7**, **7→1**
#2	**1→5**, **5→3**, **3→2**, **2→7**, 7→1
#3	**1→4**, **4→3**, 3→6, **6→5**, **5→7**, 7→1
#4	1→7, **7→8**, **8→4**, **4→6**, **6→8**, 8→7, 7→1

Cuadro A.5: Solución CARP: Gdb10

Ruta	Detalle
#1	**1→11**, 11→6, **6→4**, **4→2**, **2→7**, **7→5**, **5→1**
#2	**1→2**, **2→5**, **5→6**, **6→11**, **11→12**, **12→8**, **8→1**
#3	**1→4**, **4→7**, 7→2, **2→3**, **3→8**, **8→9**, **9→1**
#4	**1→10**, **10→12**, 12→10, **10→9**, **9→3**, **3→4**, **4→11**, 11→1

Cuadro A.6: Solución CARP: Val3A

Ruta	Detalle
#1	**1→2**, **2→4**, **4→3**, **3→9**, **9→19**, **19→22**, **22→23**, **23→24**, **24→21**, 21→24, 24→23, **23→20**, **20→10**, **10→11**, **11→15**, **15→21**, **21→20**, **20→19**, 19→9, 9→3, 3→1
#2	**1→3**, 3→4, **4→5**, **5→6**, **6→12**, **12→13**, **13→17**, **17→18**, **18→14**, **14→8**, **8→7**, **7→6**, 6→12, 12→13, **13→14**, 14→18, 18→17, **17→16**, **16→15**, 15→16, **16→12**, **12→11**, **11→5**, **5→10**, **10→9**, 9→3, 3→1