UNIVERSIDAD DE CONCEPCIÓN
Facultad de Ingeniería
Departamento de Ingeniería Civil Informática
y Ciencias de la Computación

Profesora Patrocinante:
María Angélica Pinninghoff J.

Comisión:
Ricardo Contreras A.
John Atkinson A.

Metaheurística inspirada en el desarrollo de resistencia a antibióticos por parte de las bacterias, a partir de sus mecanismos de variación genética.

José Matias Orellana Maturana

Informe de Memoria de Título
para optar al título de
Ingeniero Civil Informático

2016

A mis padres, familia, amigos y novia:

Por todo el cariño, la comprensión y ese incondicional apoyo que siempre me han brindado.

¡Muchas Gracias!

Resumen

Las bacterias son microorganismos que poseen una extraordinaria capacidad de supervivencia y adaptación al medio en que se encuentran. Esto es debido en gran parte a los mecanismos de variación genética que poseen, y al carácter colaborativo de la colonia de bacterias. Por ejemplo ante la presencia de un antibiótico, las bacterias resistentes de la colonia pueden traspasar (mediante conjugación genética) una copia de sus genes de resistencia a las bacterias que no lo son. También, las bacterias tienen una alta probabilidad de mutación, por lo que pueden llegar a desarrollar de manera natural resistencia. Además al morir una bacteria, puede liberar parte de su material genético al ambiente, para que éste sea capturado por otra bacteria si es que le resulta de utilidad (proceso llamado transformación genética).

El párrafo anterior describe la idea principal que recoge la metaheurística presentada en la presente memoria de título, la cual se llama Algoritmo Bacteriano de Resistencia Antibiótica (ABRA). En ABRA, una colonia de bacterias (grupo de soluciones a un problema) son iterativamente sometidas a la presencia de antibióticos (un factor de presión que divide a las buenas de las malas soluciones), separando a la población en las bacterias que son resistentes y las no resistentes a ese antibiótico. Es entonces, que las bacterias no resistentes, mediante los mecanismos de variación genética (conjugación, transformación y mutación) tendrán la posibilidad de volverse resistentes (mejorar su calidad como soluciones), y de esta manera sobrevivir a la presencia del antibiótico.

ABRA fue probado con el problema de Ruteo Vehicular con Capacidad Uniforme (CVRP), y logró mejorar los resultados obtenidos por trabajos previos inspirados principalmente en la conjugación bacteriana. Además, demostró que puede obtener tan buenas soluciones e incluso mejores que utilizando un Algoritmo Genético, sin embargo esto a un mayor coste en tiempo de ejecución.

Palabras claves

Bacterias, Resistencia Antibiótica, Optimización Combinatoria, Problema de Ruteo Vehicular con Capacidad Uniforme, Algoritmo, Heurística, Metaheurística.

Índice

Índice de figuras

Índice de tablas

1. Introducción General

Este capítulo tiene por objetivo introducir sobre qué trata el tema de la presente memoria de título, dar a conocer el objetivo general y objetivos específicos, y mostrar cómo está organizado el trabajo que se va a realizar.

1.1. Introducción

Existen ciertos problemas dentro de la informática que se caracterizan por su gran complejidad y para los que no existen algoritmos tradicionales que los resuelvan en tiempo polinomial. La complejidad de algunos de estos problemas radica en que el espacio de las soluciones crece exponencialmente con el tamaño del mismo, por lo tanto, realizar una búsqueda de fuerza bruta en todo el espacio de soluciones se vuelve intratable. Para lidiar con estos problemas, se buscan métodos heurísticos que aunque no necesariamente entreguen la solución óptima, sí encuentren una lo suficientemente satisfactoria, y que lo hagan de manera eficiente.

Algunas heurísticas probadamente buenas para la resolución de estos problemas, son por ejemplo:

- Algoritmos de Colonias de Hormigas, inspirados en cómo las hormigas encuentran el camino más corto desde su colonia hasta la fuente de alimento.
- Algoritmo de Enfriamiento Simulado, inspirado en el proceso mediante el cual se calienta el acero, y luego se enfría lentamente para variar sus cualidades físicas.
- Algoritmos Genéticos, inspirados en la evolución de una población mediante el cruzamiento de sus individuos y la selección natural de los genes más aptos [28].

Como se puede observar en los ejemplos anteriores, un proceso o fenómeno natural sirve como inspiración para desarrollar un método que solucione problemas difíciles de tratar por medio de algoritmos tradicionales.

Las bacterias son microorganismos procariotas (no presentan un núcleo celular definido, por lo que su ADN se encuentra libre en el citoplasma) usualmente de unos pocos micrómetros de tamaño. Las bacterias tienen una capacidad extraordinaria de adaptación a diferentes condiciones ambientales, pueden habitar en lugares terrestres y acuáticos, en condiciones ácidas e incluso en lugares radiactivos [4]. Para comprender la esencia de esta capacidad, es necesario entender sus bases genéticas, es decir, cómo está organizada la información genética en las bacterias, cómo se regula su expresión, y los mecanismos de variación genética que poseen [5].

En los trabajos predecesores a la presente memoria ([1] y [2]]), se propone un algoritmo basado en la conjugación bacteriana, el cual es un mecanismo colaborativo de transferencia genética presente en las bacterias. Como contraparte a los Algoritmos Genéticos, los cuales se basan en un paradigma competitivo para mejorar las soluciones a un problema, este nuevo método adopta un enfoque colaborativo, planteando así que con este paradigma se pueden obtener resultados al menos tan buenos como los que se obtienen a partir del enfoque genético competitivo.

En la presente memoria de título, utilizando parte del trabajo ya realizado en [1] y [2], se propone un nuevo método de enfoque colaborativo, el cual se basa en una colonia de bacterias, la cual se va fortaleciendo al adquirir resistencia a antibióticos cada vez más fuertes. Este fortalecimiento de las bacterias significa para el algoritmo ir obteniendo cada vez mejores soluciones

para el problema que se esté resolviendo.

1.2. Objetivo General

Proponer una nueva metaheurística de carácter colaborativo, basada en el fortalecimiento de una colonia de bacterias, la cual va adquiriendo resistencia hacia los antibióticos que le son aplicados, esto por medio de sus mecanismos de variación genética. La analogía de esto con la informática, es que el fortalecimiento de las bacterias significa para el algoritmo el encontrar mejores soluciones.

1.3. Objetivos Específicos

- Comprender los mecanismos de transferencia genética de las bacterias, y cómo estos favorecen la supervivencia de sus especies.
- Comprender lo básico sobre cómo funcionan las metaheurísticas.
- Determinar qué se puede mantener del actual algoritmo de conjugación bacteriana y aplicarlo a la nueva metaheurística.
- Implementar el nuevo algoritmo, realizar pruebas utilizando el Problema del Ruteo Vehicular con Capacidad Uniforme, y obtener conclusiones.

1.4. Organización del Informe

Este informe está organizado de la siguiente forma:

- El capítulo 1 corresponde a la introducción donde se muestra de qué trata esta memoria de título.
- El capítulo 2 trata sobre la teoría que está detrás de la memoria de título; primero aborda aspectos biológicos sobre las bacterias, sus mecanismos de variación genética y la resistencia antibiótica; se muestra resumidamente qué son los problemas de optimización combinatoria, y se introduce el Problema del Ruteo Vehicular con Capacidad Uniforme (CVRP), con el cual se probará la metaheurística que se va a desarrollar; se presentan los conceptos de algoritmos aproximados, heurísticas y metaheurísticas, junto con ejemplos de estas últimas; se muestran los trabajos previos, correspondientes al Algoritmo de Conjugación Bacteriana (ACB) y Algoritmo de Conjugación Bacteriana con Control de Población (ACBCP).

- El capítulo 3 presenta la metaheurística propuesta, la cual es el Algoritmo Bacteriano de Resistencia Antibiótica (ABRA); se explican sus etapas, parámetros, y cómo será implementado para resolver el CVRP.
- El capítulo 4 corresponde a los experimentos que se realizaron para ajustar los parámetros de ABRA, y validar la calidad de las soluciones entregadas por éste.
- El capítulo 5 son las conclusiones obtenidas de todo el trabajo realizado.

2. Marco Teórico

En este capítulo se trata la teoría que está detrás del modelo que se propondrá. Se mostrarán aspectos biológicos sobre las bacterias, sus mecanismos de variación genética y la resistencia antibiótica. Se introducirá brevemente en la Optimización Combinatoria y el Problema de Ruteo Vehicular con Capacidad Uniforme (CVRP), con el cual se probará la metaheurística que se desarrollará. Se presentarán los conceptos de algoritmos aproximados, heurísticas y metaheurísticas, junto con ejemplos. Y finalmente, se revisarán los trabajos previos a esta memoria de título, correspondientes al Algoritmo de Conjugación Bacteriana (ACB) y Algoritmo de Conjugación Bacteriana con Control de Población (ACBCP).

2.1. Bacterias

Las bacterias son microorganismos procariotas (no presentan un núcleo celular definido, por lo que su ADN se encuentra libre en el citoplasma) usualmente de unos pocos micrómetros de tamaño. Una de sus principales características es que no se reproducen de manera sexual como los seres eucariotas, sino que lo hacen mediante fisión binaria. Una bacteria, se divide en dos nuevas células, las cuales comparten un material genético idéntico al de su predecesora (a menos que ocurra mutación en el proceso).

Las bacterias tienen una capacidad extraordinaria de adaptación a diferentes condiciones ambientales. Pueden habitar en lugares terrestres y acuáticos, en condiciones ácidas e incluso en lugares radiactivos [4]. Para comprender la esencia de esta capacidad, es necesario entender sus bases genéticas, es decir, cómo está organizada la información genética en las bacterias, cómo se regula su expresión, y los mecanismos de variación genética que poseen [5]. Es por esta capacidad de adaptación que las bacterias se utilizan como inspiración para [1], [2] y para el trabajo desarrollado en ésta memoria.

2.1.1. ADN Bacteriano

Toda la información genética esencial para la vida de la bacteria está contenida en una única molécula de ADN de doble cadena circular. Muchas bacterias, además, poseen ADN extracromosómico, también circular y cerrado, denominado plásmido. Los plásmidos portan información para muchas funciones no esenciales de la bacteria bajo condiciones normales. Algunos ejemplos de características concedidas por plásmidos son: adquirir factores de virulencia (lo que por ejemplo, les puede permitir sobrevivir a la respuesta inmunológica de un huésped [7]); poder realizar nuevas actividades metabólicas (por ejemplo, poder degradar alguna molécula presente en el ambiente para obtener energía [8]); y ser capaces de sobrevivir a la acción de antibióticos específicos (la resistencia antibiótica se profundiza en el punto 2.1.4, ya que es la base del método que se presenta en este trabajo).[6]

2.1.2. Variación Genética

Existen dos conceptos relacionados con la información genética de todos los seres vivos, y como ésta se expresa según el ambiente donde se encuentra: el Genotipo y el Fenotipo. El genotipo es toda la información genética que posee un individuo. El fenotipo es cómo se expresa mediante características observables la información genética contenida en el genotipo, siendo el fenotipo influenciado por el ambiente donde se encuentre el individuo. Un mecanismo de variación del fenotipo es el Quorum Sensing, en el cual las bacterias regulan funciones fisiológicas de acuerdo a la densidad celular de la población ([2] utiliza un método de control de población inspirado en el Quorum Sensing, el cual se muestra en el punto 2.4.2 de este capítulo).

También existen mecanismos para modificar el genotipo de la bacteria, es decir, modifican la información genética de la célula: una forma puede ser mediante mutaciones, y la otra, mediante

intercambio de material genético con otras bacterias.

Mutación

Una mutación es un cambio heredable en la información genética de un organismo. Las mutaciones ocurren con baja frecuencia, y generalmente se deben a un error en los procesos de replicación del ADN. Algunas mutaciones pueden conferir al mutante una ventaja frente a la cepa bacteriana que le dio origen, por lo que esta nueva bacteria es capaz de superar el crecimiento de su predecesor y sustituirla [5]. La mutación es uno de los mecanismos de variación genética que son utilizados en [1], [2], y en el actual trabajo, con el propósito de promover la variedad genética de la población.

Transferencia Genética

La recombinación genética es el proceso mediante el cual, los elementos genéticos provenientes de diferentes individuos se combinan. Gracias a esto, el individuo puede adquirir una nueva característica que resulte en una adaptación a cambios en el medio ambiente. A diferencia de los organismos eucariotas, donde la recombinación genética ocurre mediante la reproducción sexual, en las bacterias esta recombinación ocurre de manera horizontal. Esto quiere decir que, los genes no son transferidos desde una generación a su descendencia de manera vertical, sino que pueden ser transferidos entre los miembros de una población indistintamente del proceso de reproducción celular. Por lo tanto, en vez de hablar de células padres-hijos, resulta más adecuado manejar el concepto de células donadoras-receptoras.

Existen tres métodos en las bacterias para realizar intercambio genético. De manera natural, una bacteria puede poseer más de uno de estos mecanismo, así como también puede que no cuente con ninguno de ellos. Los mecanismos son:

- **Conjugación**: Involucra la transmisión directa de ADN de una bacteria a otra. Esta transformación ocurre cuando una célula posee un plásmido conjugativo que contiene la información necesaria para realizar el proceso de conjugación. Un ejemplo es el plásmido F de la E. coli. Una celula F+ (que contiene el plásmido F) puede transferírselo a una F- (que no contiene el plásmido), y al terminar la conjugación se tendrán dos células F+. También existen células denominadas Hfr que poseen el plásmido integrado con el cromosoma. Sin embargo, las células Hfr al realizar el proceso conjugativo con una F-, no transforman a la célula receptora en donadora. Los trabajos realizados en [1] y [2] utilizan como base este método de transferencia. En particular para este trabajo, la conjugación se utilizará como un método de transmitir plásmidos con factores de resistencia antibiótica.
- **Transducción**: Se transfiere ADN mediante algún bacteriófago, usualmente un virus.
- **Transformación**: La bacteria captura ADN que se encuentra libre en el ambiente. Sobre este mecanismo se profundizará en el punto siguiente, ya que será utilizado en el método propuesto.

2.1.3. Transformación Bacteriana

La Transformación Bacteriana es uno de los tres mecanismos mediante los cuales una bacteria puede incorporar ADN a sí misma. En la Transformación, la bacteria captura ADN que se encuentra libre desde el ambiente. Este ADN proviene de una Bacteria Donante (generalmente una bacteria que murió, y de la cual se liberó ADN o trozos de él, el cual puede ser cromosomal o plasmídico), y es tomado por una Bacteria Receptora, la cual pasa a ser una Transformante. El fragmento de ADN capturado se recombina con el ADN cromosomal de la bacteria, y luego este puede ser heredado en las siguientes divisiones celulares.

La mayoría de las bacterias no son capaces de incorporar ADN libre de manera eficiente, a menos que sean expuestas a tratamientos especiales que pueden ser químicos o eléctricos, los cuales hagan sus barreras más permeables. Sin embargo, existen bacterias que sí pueden capturar ADN libre desde el ambiente sin ningún tratamiento, siendo estas bacterias Naturalmente Transformables. No obstante, incluso este tipo de bacterias sólo son capaces de incorporar ADN libre en ciertas etapas de su ciclo de vida, estados en las cuales se denominan Competentes.

El que las bacterias entren en estado de competencia, varía de especie en especie. Algunos factores que pueden influir para que la bacteria sea competente, son: la densidad celular de la población, el pH, la temperatura y la disponibilidad de nutrientes en el ambiente. Por ejemplo, la Haemophilus influenzae entra en estado de competencia cuando el medio no le permite desarrollarse o su división celular se encuentra bloqueada [10]. Hasta el momento, solo se conoce la Neisseria gonorrhoeae como una bacteria que se encuentra en estado permanente de competencia [11].

Hay tres posibles causas, por las cuales existe la transformación de manera natural en algunas especies de bacterias: un componente nutricional, en el cual las bacterias utilizan las moléculas de ADN como una fuente de Carbono, Nitrógeno y energía; una función reparadora, en la cual las bacterias utilizan el ADN capturado para reparar su propio ADN, el cual se encuentra dañado; una función de recombinación, mediante la cual las bacterias intercambian material genético dentro de los miembros de su especie, incrementando la diversidad y promoviendo la evolución [3].

Para el algoritmo que se propone en el capítulo tres, se utilizará la transformación genética como un método de incrementar la diversidad genética y promover la evolución hacia individuos más fuertes en la población. Se asumirá que todas las bacterias son naturalmente competentes, y que al morir pueden liberar parte de su ADN cromosomal.

2.1.4. Resistencia Antibiótica

Los antibióticos, desde que la Penicilina fue descubierta en 1929 por Sir Alexender Fleming, han jugado un rol crítico en la pelea contra enfermedades causadas por bacterias y otros microorganismos. No obstante, en la actualidad esta pelea se ha visto dificultada debido al desarrollo de resistencia a los antibióticos más comúnmente utilizados. Como resultado de lo anterior, enfermedades que antes eran tratables, han pasado a ser cada vez más difíciles de enfrentar.

La adquisición de resistencia antibiótica puede ocurrir por diferentes medios: por mutación espontánea, en el cual una bacteria de manera natural sufre un proceso de mutación el cual le confiere resistencia a algún antibiótico, siendo las probabilidades de esto usualmente muy bajas; y por transferencia horizontal de genes, mediante los mecanismos descritos anteriormente. Cuando una colonia de bacterias se encuentra en presencia de un determinado antibiótico, al aparecer una cepa resistente, ésta rápidamente pasa a ser la predominante al generar descendencia que también será resistente, y por lo tanto tendrán mayor probabilidad de sobrevivir en este ambiente hostil que la cepa no resistente [12].

Si bien el desarrollo de resistencia antibiótica representa un problema para las personas, para las bacterias que adquieren resistencias constituye una mejora evolutiva de su especie, que les confiere una mayor capacidad de supervivencia. El cómo las bacterias de una colonia pueden evolucionar y adaptarse para sobrevivir a la presencia de antibióticos que buscan eliminarlas, puede ser visto como un proceso de mejora de ellas mismas, hacia bacterias cada vez más resistentes y aptas para sobrevivir. Dicho de otra manera y citando el dicho popular, "*lo que no te mata te hace más fuerte*".

El método que se propone en este trabajo se basa en una colonia de bacterias, la cual se va haciendo mas fuerte al adquirir resistencia a antibióticos cada vez más potentes. Este fortalecimiento de las bacterias significa para el algoritmo ir obteniendo cada vez mejores soluciones, lo cual será explicado en el capítulo tres.

2.2. Optimización Combinatoria

Un problema de optimización es aquel que consiste en encontrar los valores de las variables que maximicen o minimicen (según el problema en particular) una función objetivo. Estas variables reciben el nombre de Variables de Decisión y se encuentran sujetas a restricciones. Algunos problemas de optimización pueden ser resueltos de manera relativamente sencilla, como por ejemplo los problemas lineales, en los cuales tanto la función objetivo como las restricciones de las variables de decisión pueden ser representadas de manera lineal.

Los problemas de optimización se pueden dividir dependiendo de si las variables de decisión pueden tomar valores continuos o discretos. Cuando las variables pueden tomar valores discretos, se dice que el problema es de Optimización Combinatoria. En estos problemas, generalmente se busca un objeto dentro de un conjunto finito (típicamente de enteros, conjuntos, permutaciones o grafos) que optimice la solución [13].

En general, los problemas de Optimización Combinatoria se asocian con problemas computacionales difíciles de resolver. Se dice que un problema es fácil de resolver cuando es posible

encontrar un algoritmo que solucione el problema, cuya complejidad en tiempo sea de tipo Polinomial. Por el contrario, un problema se dice difícil cuando no se conoce un algoritmo de tiempo Polinomial que lo resuelva. Más precisamente, hablando en términos de teoría de complejidad, problemas de la clase NP (que pueden ser resueltos en tiempo polinomial por una Máquina de Turing No Determinista) [14]. En la práctica, existe una gran cantidad de problemas que caen dentro de la categoría NP y necesitan ser resueltos de manera eficiente.

A continuación se presentará el Problema del Vendedor Viajero, que es uno de los problemas más emblemáticos y estudiados de Optimización Combinatoria. Luego, se presentará el Problema de Ruteo Vehicular, el cual es una generalización del vendedor viajero, y es el problema que se usara para probar la metaheurística propuesta.

2.2.1. El Problema del Vendedor Viajero

El problema del vendedor viajero es un clásico problema de optimización combinatoria. Conocido como TSP (por sus siglas en inglés: Traveling Salesman Problem), el cual de manera informal se describe como: *"Trata de un vendedor que debe visitar cierto número de ciudades. Partiendo de su ciudad de origen, debe visitar una vez cada ciudad y regresar al punto de origen. El objetivo es encontrar la ruta que reduzca la distancia recorrida"* [15].

Expresado de manera más formal el TSP consiste en: *"Dado un grafo conexo, no dirigido $G = (V, E)$, con costos enteros no negativos $c(u, v)$, para cada arista $(u, v) \in E$, se debe encontrar un Ciclo Hamiltoniano (ciclo que contiene todos los vértices en V) de mínimo costo en G."* [16].

Si N es el número de vértices en el grafo, desde el primer vértice se tiene $N - 1$ posibles caminos para tomar, en el siguiente vértice se tendrán $N - 2$ opciones, en el siguiente $N - 3$ y así sucesivamente. Por lo tanto, la cantidad de rutas posibles es de $(N - 1)!$. Por lo tanto, el problema crece factorialmente conforme se incremente el número de ciudades, por lo que una búsqueda exhaustiva en el total de rutas posibles queda fuera de toda posibilidad.

2.2.2. El Problema de Ruteo Vehicular

El problema de ruteo vehicular (Vehicle Routing Problem en inglés, conocido como VRP), es una generalización del problema del vendedor viajero. A grandes rasgos, el VRP trata de determinar el conjunto óptimo de rutas que deben ser llevadas a cabo por una flota de vehículos, para satisfacer las demandas de un número dado de clientes, de tal manera que:

- Todos los clientes son visitados una sola vez por solo un vehículo.
- Todas las rutas de los vehículos parten y terminan en el depósito.
- Otras condiciones extras [17].

Según cuales sean estas condiciones extras, el VRP deriva en distintas ramas, las cuales se representan en la figura 1 [18]. A continuación se resumirá en qué consisten estas variantes.

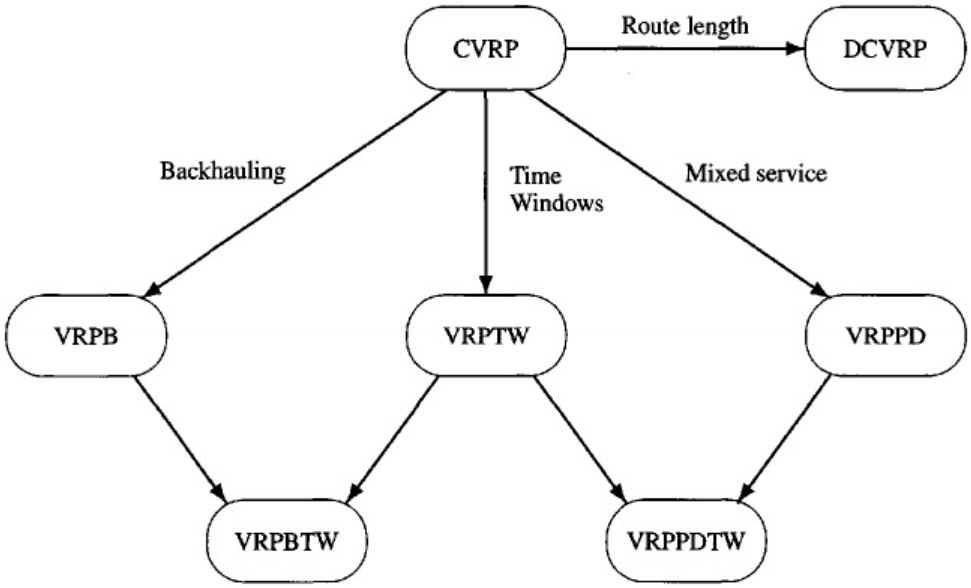

Figura 1: Variantes del Problema de Ruteo Vehicular.

VRP con Capacidad Uniforme (CVRP)

Es la versión más básica del VRP. En el CVRP todos los clientes tienen una demanda conocida de antemano, la cual no puede ser dividida. Todos los vehículos tienen la misma capacidad, y sólo tienen la restricción sobre cuál sea ésta. El objetivo es minimizar la función total de costos (por ejemplo, el largo o tiempo total de los viajes) y satisfacer la demanda de todos los clientes. Es ésta la variante del VRP que se usará para probar la metaheurística que se presentará mas adelante.

Utilizando grafos, el CVRP puede ser descrito de manera más formal. Sea $G = (V, A)$ un grafo completo, donde $V = 0, \ldots, n$ es un conjunto de vértices, y A un conjunto de arcos. Los vértices $i = 1, ..., n$ corresponden a clientes, y el vértice 0 al depósito. Un costo no negativo c_{ij} está asociado con cada arco $(i, j) \in A$, y representa el costo del viaje de ir desde i hasta j. Si G es un grafo dirigido, la matriz de costos C es asimétrica, y corresponde a la instancia del problema llamado CVRP Asimétrico (ACVRP). Si G es no dirigido, tenemos que $c_{ij} = c_{ji}, \forall (i, j) \in A$, y el problema es llamado CVRP simétrico (SCVRP). En algunas instancias, los vértices vienen asociados a coordenadas correspondientes a puntos en el plano, y el costo c_{ij} para cada arco $(i, j) \in A$ es definido como la distancia Euclidiana.

Cada cliente i ($i = 1, \ldots, n$) tiene asociada una demanda no negativa d_i la cual debe ser satisfecha, mientras que el depósito tiene una demanda ficticia $d_0 = 0$. Dado un conjunto de vértices $S \subseteq V$, sea $d(S) = \sum_{i \in S} d_i$, denota la demanda total de ese conjunto. Se tiene un número de M vehículos idénticos, cada uno con capacidad K, que se encuentran disponibles en el depósito. Se asume que $d_i \leqslant K, \forall i = 1, \ldots, n$. Cada vehículo puede llevar a cabo a lo más una ruta, y se asume que M no es menor que M_{min}, donde M_{min} es el mínimo número de vehículos necesarios para satisfacer la demanda de los clientes. Determinar M_{min} corresponde a otro problema en sí mismo, el Bin Packing Problem (BPP), el cual corresponde a la clase NP-Hard [19]. También se puede asumir una cantidad ilimitada de vehículos, y asignar un valor al uso de cada uno en la función de costos del problema. Con todo lo anterior, el CVRP consiste en encontrar M (siendo M un número conocido de antemano u obtenido durante la resolución del problema) circuitos S (cada uno correspondiente a la ruta de un vehículo) con el mínimo valor según la función de costos establecida, tal que:

- $\forall v \in V - \{0\} \exists! S : v \in S$.
- $\forall S, S$ *comienza y termina en* 0.
- $\forall S, d(S) \leqslant K$.

El TSP puede ser visto como la versión del CVRP, donde el número de vehículos $M = 1$, y la capacidad de ese vehículo $K = \infty$.

VRP con Restricción de Distancias (DCVRP)

Es una variante del CVRP, en la cual las aristas (i, j) representan distancias, y se tiene como restricción adicional que las rutas no pueden superar un largo máximo dado.

VRP con Ventanas de Tiempo (VRPTW)

Es una extensión del CVRP, en la cual cada cliente i está asociado con un intervalo $[a_i, b_i]$ llamado ventana de tiempo, y con un tiempo de atención s_i (valor que por ejemplo puede ser interpretado como el tiempo que se demora en descargar una entrega). En el VRPTW, la atención a un cliente i solo puede ser realizada dentro su ventana correspondiente, donde se debe detener s_i tiempo antes de poder partir a atender a otro cliente.

VRP con Backhaul[1] (VRPB)

En esta extensión del CVRP, los clientes son separados en dos subconjuntos. El primer subconjunto L contiene n clientes que requieren una cantidad dada de productos a ser entregados. El segundo subconjunto B contiene m clientes de los cuales, se debe recoger una cantidad dada de productos. La nueva restricción que impone el VRPB, es que en cualquier ruta que tenga clientes con entregas y otros clientes con recogidas, se debe satisfacer primero a todos los clientes con entregas antes que los que tienen recogidas.

VRP con Recogidas y Entregas (VRPPD)

En esta versión del CVRP, cada cliente i tiene asociadas dos cantidades: una demanda d_i y una recogida p_i. Por cada cliente i existe un O_i, el cual, denota el origen de la demanda que se debe entregar a ese cliente; y un D_i, el cual, corresponde al destino de la carga recogida desde ese cliente. Con esto, el VRPPD añade las restricciones de que la carga del vehículo en una ruta debe permanecer siempre no negativa, y por cada cliente i, el cliente O_i debe ser servido en la misma ruta y antes que i, a su vez que el cliente D_i también debe ser servido en la misma ruta pero después que i.[18]

2.3. Métodos Aproximados

Como se mencionó anteriormente, los problemas de Optimización Combinatoria suelen asociarse a problemas de la clase NP. Los Algoritmos Exactos (por ejemplo, Branch and Bound o el Branch and Cut) pueden servir para instancias específicas o acotadas de los problemas, pero su comportamiento sigue siendo no polinomial [23].

Los métodos aproximados nacen como respuesta a la problemática planteada y como una opción a los algoritmos exactos. Tal como su nombre lo indica, estos métodos buscan una solución aproximada, que es lo suficientemente buena para un determinado problema. Esta solución puede no ser la óptima, sin embargo es obtenida en un tiempo razonable. Dentro de los métodos aproximados aparecen las Heurísticas y las Metaheurísticas.

2.3.1. Heurísticas

Una definición de heurística dada en [21] es: "Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva,

[1]Backhaul en el contexto de transporte, es el movimiento de una carga desde un punto B de vuelta al origen A [20]

en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución". Otra definición aportada en [22] es: "Se califica de heurístico a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad o su factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación. Se usa el calificativo heurístico en contraposición a exacto".

Las heurísticas son una buena opción cuando no existe un método exacto de resolución, o cuando sí existe algún método exacto, pero éstos consumen mucho tiempo en entregar una solución. Existen otras razones por las cuales también es buena opción utilizar métodos heurísticos [24].

- El método heurístico es más flexible que el exacto, permitiendo la incorporación de condiciones difíciles de modelar.
- El método heurístico se utiliza como parte de un procedimiento global que garantiza la solución óptima de un problema. La heurística puede aportar ya sea una buena solución inicial como partida para el procedimiento global, o puede participar en un paso intermedio.

Una característica de las heurísticas (en contraposición a las metaheurísticas como se mostrará más adelante) es que son específicas al problema para el que fueron diseñadas. Las heurísticas usan conocimiento acerca del problema y de las técnicas aplicables, y con ello tratan de aportar soluciones (o acercarse a ellas) usando una cantidad razonable de recursos (generalmente tiempo). Los métodos heurísticos son diseñados acorde a cada problema, utilizando toda la información disponible y el análisis teórico del modelo del problema [22].

Algunos ejemplos de heurísticas utilizadas para la resolución del VRP son:

- Algoritmo de Ahorros de Clarke y Wright [25].
- Algoritmo de Inserción Secuencial de Jameson y Mole [26].
- Heuristica de Asignación Generalizada de Fisher y Jaikumar [27].

2.3.2. Metaheurísticas

Las metaheurísticas se sitúan conceptualmente por encima de las heurísticas tradicionales (por ello antepone el prefijo "meta", que quiere decir "más allá" o "por sobre de"). Las metaheurísticas pueden ser definidas como metodologías generales para guiar procedimientos heurísticos, en la resolución de problemas de optimización. Para esta subsección se ha utilizado como principal guía el libro *Metaheuristics. From design to implementation* de El-Ghazali Talbi [28].

Todas las metaheurísticas tienen dos elementos en común, que son esenciales a la hora de ser aplicadas a un problema en particular: el cómo se representan las soluciones de ese problema y la función objetivo a ser optimizada.

Representación de las Soluciones

La representación de las soluciones de un problema debe cumplir con las siguientes características:

- **Completitud:** todas las soluciones asociadas a ese problema deben poder ser representadas.
- **Conexidad:** entre todas las soluciones debe existir un camino para llegar de una hasta cualquier otra, mediante los operadores de la metaheurística.
- **Eficiencia:** la representación debe ser fácil de manipular para los operadores de la metaheurística.

Las representaciones más clásicas son: codificación binaria, vector de valores discretos, vector de valores reales y permutaciones. Esta última resulta especialmente interesante para este trabajo, puesto que es la más cómoda para representar el espacio de soluciones para el problema del VRP, al ser las soluciones de éste, permutaciones de las ciudades que se deben visitar.

Función Objetivo

La función objetivo (también conocida como función de evaluación, función de costo o función de utilidad), es una función f que asocia un valor real a cada solución s del espacio de búsqueda S. Se define como:

- $f : S \to \mathbb{R}, \forall s \in S$

Así, f representa un valor mediante el cual se pueden ordenar todas las soluciones posibles. Sin embargo, puede que sea necesario un paso intermedio de decodificación d de la solución s para que ésta pueda ser evaluada por f:

- $d : s \to s', \forall s \in S$
- $f : s' \to \mathbb{R}, \forall s' \in d(S)$

La función objetivo es de suma importancia, ya que guía al método hacia las buenas soluciones. Una función objetivo mal definida, puede llevar a soluciones no aceptables o de baja calidad.

2.3.3. Ejemplos de Metaheurísticas

A continuación se presentará con más detalle cómo funcionan algunas de las metaheurísticas más conocidas. Cabe destacar que los algoritmos evolutivos se revisarán en una subsección aparte, ya que pertenece al mismo paradigma de la metaheurística que se desarrollará en esta memoria, por lo que comparten propiedades similares y merecen un mayor análisis.

Búsqueda Local

La Búsqueda Local es una de las metaheurísticas más antiguas. En este método, dada una solución inicial dentro del espacio de búsqueda, iterativamente ésta se irá reemplazando por alguna solución vecina que mejore la actual, hasta que llegado un punto todas las vecinas sean peores soluciones. Se dicen vecinas las soluciones cercanas entre sí, siendo el criterio de cercanía dependiente a la representación de las soluciones del problema [29]. Por ejemplo, si la solución se representa como un arreglo binario, las soluciones vecinas pueden ser aquellas que sólo difieran en un bit de la solución actual.

Algorithm 1 Búsqueda Local

Require: Solución inicial s_0
Ensure: Mejor solución encontrada

```
1: s = s_0
2: while No se cumpla criterio de stop do
3:    Generar N(s) /* N(s) es el conjunto de soluciones vecinas de s */
4:    if No hay solución mejor que s en N(s) then
5:       stop
6:    else
7:       s = s' /* s' es algún vecino en N(s) mejor que s */
8:    end if
9: end while
```

Un problema de la búsqueda local, es que resulta muy sencillo quedar atrapado en óptimos locales, y es muy sensible a la solución inicial generada, como se muestra en la figura 2 [28]. En general, funciona bien cuando existen pocos óptimos locales o cuando la diferencia de éstos con el óptimo global es poca.

Enfriamiento Simulado

El Enfriamiento Simulado es una metaheurísticas derivada de la Búsqueda Local, pero que intenta escapar de los óptimos locales mediante la posible aceptación de pasos que a priori

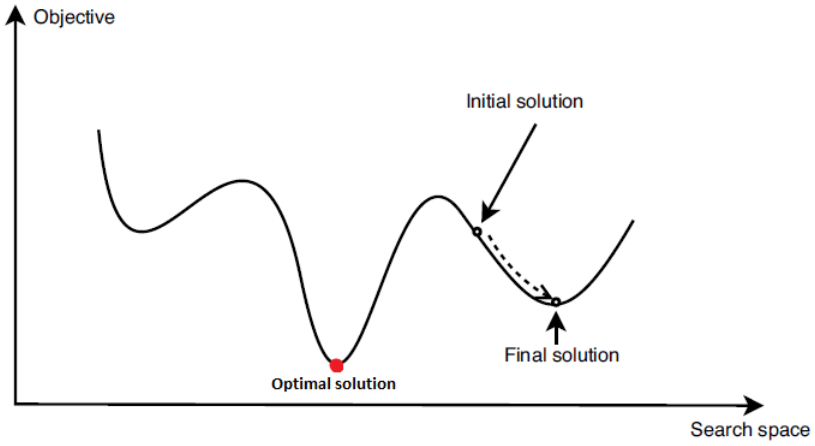

Figura 2: Óptimos locales en búsqueda local.

empeoren la solución actual. Este algoritmo está inspirado en el proceso de recocido al que se someten algunos sólidos, en el cual éstos se calientan hasta altas temperaturas, y luego se enfrían lentamente con tal de variar sus propiedades físicas [30].

El algoritmo parte con una temperatura T dada y una solución inicial generada. Esta solución s se compara con un vecino s'; Si s' es mejor, éste se acepta y pasa a ser la solución actual; Si s' no es mejor, éste puede ser aceptado con una probabilidad de:

- $p(s') = \mathrm{e}^{-\frac{\Delta(E)}{T}}$, *si el problema es de minimización*
- $p(s') = \mathrm{e}^{\frac{\Delta(E)}{T}}$, *si el problema es de maximización*
- *Donde:* $\Delta(E) = f(s')-f(s)$, *f es la función de evaluación*

Este ciclo se itera tantas veces hasta que se alcance el estado de equilibrio para esa temperatura T (por ejemplo, un número fijo de iteraciones). Alcanzado el equilibrio para T, se procede a bajar la temperatura a T' según algún esquema predeterminado de descenso y se repite el ciclo. El algoritmo se repite hasta que $T < T_{min}$.

Comparado con el algoritmo de Búsqueda Local, el de Enfriamiento Simulado sigue siendo sencillo, intuitivo y fácil de implementar. Además se comporta bien en problemas como por ejemplo, el TSP.

Algorithm 2 Enfriamiento Simulado

Require: Solución inicial s_0, Temperatura inicial T_{MAX}, Temperatura mínima T_{min}
Ensure: Mejor solución encontrada

```
1: s = s_0
2: T = T_MAX
3: while T ⩾ T_min do
4:    while No se alcance condición de equilibrio para T do
5:       s' = vecino aleatorio de s
6:       if Δ(E) ⩽ 0 then
7:          s = s'
8:       else
9:          Se acepta s = s' con una probabilidad de p(s')
10:      end if
11:   end while
12:   T = g(T) /* Se actualiza T según algún esquema de descenso de temperatura */
13: end while
```

El Algoritmo 2 corresponde a un problema de minimización. En caso de maximización, la condición de la línea 6 es $\Delta(E) \geqslant 0$.

Algoritmos de Colonia de Hormigas

El método de optimización basado en colonias de hormigas (en inglés, Ant Colony Optimization, conocidos como ACO) es una metaheurística bio-inspirada basada en el comportamiento de las hormigas, y cómo éstas mediante la utilización de feromonas, son capaces de encontrar el camino más corto hasta su fuente de alimento sin usar pistas visuales. Las hormigas al ir caminando van depositando feromonas, y a su vez, el camino que ellas irán tomando conforme avancen, estará probabilísticamente influenciado por la cantidad de feromona depositada anteriormente por otras hormigas. A mayor concentración de feromonas en un camino, más probabilidad que las hormigas sigan esa ruta [39].

Las metaheurísticas presentadas anteriormente, se basan en una solución inicial desde la cual realizan búsquedas de mejores soluciones. A diferencia de ellas, la metaheurística ACO construye una solución basándose en las rutas construidas por hormigas artificiales al recorrer un grafo que representa el problema. Por ello, para poder utilizar este método es necesario reducir el problema a resolver a uno que consista en buscar la mejor ruta en un grafo.

En el esquema básico de un algoritmo ACO, primero, se inicializa una cantidad de feromona en todas las aristas del grafo. Luego, se procede con los dos pasos principales: la construcción de soluciones y la actualización de los montos de feromona.

Durante la construcción de soluciones, cada hormiga va armando su camino eligiendo de manera probabilística por cual arista pasará, hasta completar una solución. La probabilidad de

que una hormiga visite una arista (i, j) queda dada por:

- $p_{ij} = \dfrac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum\limits_{k \in S} \tau_{ik}^{\alpha} \times \eta_{ik}^{\beta}}, \forall j \in S,$ *donde:*
- τ_{ij} *es la feromona en la arista (i,j).*
- $\eta_{ij} = \frac{1}{d_{ij}}, d_{ij}$ *es el costo de pasar entre (i,j). Este valor representa el conocimiento heurístico del problema.*
- S *el conjunto de vértices que faltan por visitar en el tour de la hormiga.*
- α *y* β *representan la influencia que se le asigna a la feromona y al conocimiento heurístico del problema respectivamente.*

Luego, en la fase de actualización de la feromona, ocurre la evaporación de la feromona antigua, y la suma de la dejada por las últimas hormigas. Dependiendo del método ACO en particular, esta fase puede ser realizada mientras las hormigas están construyendo sus caminos. No obstante, el método más utilizado es actualizar las feromonas una vez todas las hormigas encontraron una solución.

La evaporación queda definida por:

- $\tau_{ij} = (1 - \rho)\tau_{ij}, \forall i, j \in [1, n],$ *donde:*
- $\rho \in]0, 1]$ *es el ratio de disminución de la feromona.*

Y la adición de feromona para una solución π está dada por:

- $\tau_{i\pi(i)} = \tau_{i\pi(i)} + \Delta, \forall i \in [1, n],$*donde:*
- $\Delta = \frac{1}{f(\pi)}, f(\pi)$ *es la función de evaluación aplicada sobre la solución* π.

Por lo tanto, a menor costo la solución, mayor será Δ y más feromona será agregada a ese tour.

La construcción de soluciones, seguida por la actualización de las feromonas, se repite hasta que se cumpla un criterio de detención, el cual puede ser por ejemplo un número de iteraciones fijas.

Algorithm 3 Optimización de Colonia de Hormigas

Require: Conjunto de hormigas
Ensure: Mejor solución encontrada
1: **repeat**
2: **for** Cada hormiga **do**
3: Construir una solución usando los rastros de feromona
4: **end for**
5: *Actualizar feromona:*
6: Evaporar Feromona
7: Agregar Feromona
8: **until** Criterio de detención

2.3.4. Algoritmos Evolutivos

La teoría de la selección natural, propuesta por Charles Darwin en su libro *"El Origen de las Especies"*, sostiene que las especies existentes hoy en día, son el resultado de un proceso de millones de años de adaptación a los requerimientos que impone el ambiente. En un momento dado, un número diferente de organismos pueden coexistir y competir por los mismos recursos presentes en el ecosistema, sin embargo, serán sólo los que tengan mayor capacidad para conseguir estos recursos, los que mediante la reproducción, tenderán a ser más numerosos en el futuro. Estos organismos se denominan más aptos para sobrevivir en el ambiente, que los que necesitan de más esfuerzo para la obtención de recursos. Con el tiempo, la población del ecosistema evolucionará conteniendo en promedio, organismos más aptos que las generaciones previas.

Los algoritmos evolutivos, son los métodos que se inspiran en la evolución de las especies y la selección natural, para resolver problemas difíciles de optimización. En su esquema básico, estos algoritmos comienzan con una población inicial generada de manera aleatoria, donde cada individuo de la población es una versión codificada de una posible solución a un problema. La función objetivo del problema, asigna a cada individuo un valor que indica qué tan apto es éste. En cada iteración, los individuos con mejor aptitud serán seleccionados con mayor probabilidad para ser padres. Luego, los individuos seleccionados generan descendencia usando operadores de variación genética (se explicarán más adelante). Finalmente, se aplica algún mecanismo de selección para determinar cuáles individuos de entre los padres y los descendientes conformarán la siguiente generación. Este proceso se itera hasta que se cumpla algún criterio de detención, obteniendo como solución al individuo más apto encontrado (o conjunto de individuos mas aptos) [31]. La figura 3 representa el esquema básico de un algoritmo evolutivo.

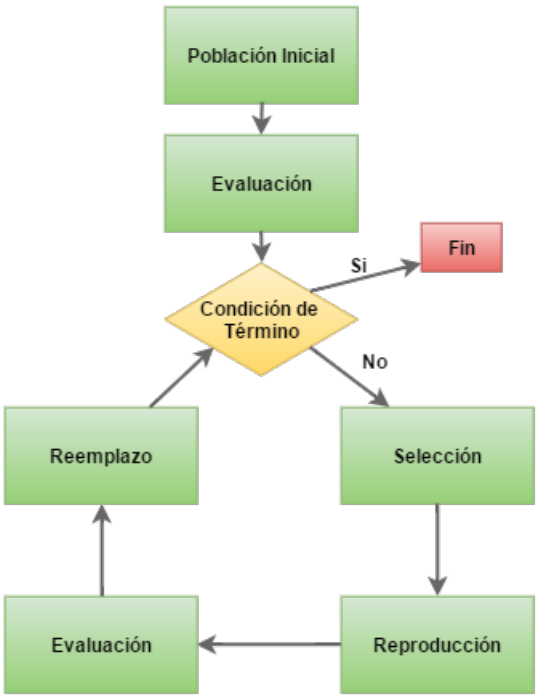

Figura 3: Esquema básico de un algoritmo evolutivo.

Los principales elementos en común para los algoritmos evolutivos son los siguientes:

- Una representación de las soluciones, característica que es compartida por todas las metaheurísticas, tal como se mencionó en el punto 1.3.2.. En los algoritmos evolutivos, siguiendo la metáfora que representan, la representación de las soluciones toma la siguiente nomenclatura:
  - ◇ Una solución codificada corresponde a un *cromosoma*.
  - ◇ Cada variable de decisión dentro de una posible solución (cromosoma) es un *gen*.
  - ◇ Los posibles valores de una variable de decisión (gen) son los *alelos*.
  - ◇ La posición de un gen dentro de un cromosoma se llama *locus*.

- Una población inicial de individuos (representados por un cromosoma).
- Una función objetivo, característica también común a todas las metaheurísticas, pero que por lo general se denomina *Función de Fitness* cuando se habla de algoritmos evolutivos.

- Una estrategia de selección con la cual se determina qué individuos serán los padres de la nueva generación, favoreciendo las probabilidades de los especímenes más aptos.
- Una estrategia de reproducción, en la cual se establece cómo ocurre la variación genética de la población, por ejemplo, mediante mecanismos de mutación y recombinación.
- Una estrategia de reemplazo, con la cual se determina qué individuos de entre los nuevos y los antiguos formarán la siguiente generación.
- Un criterio de detención, que es otro componente común a todas las metaheurísticas.

A continuación, de los anteriores conceptos presentados se explicarán los más distintivos de los algoritmos evolutivos.

1. **Población Inicial**

   La población inicial para los algoritmos evolutivos es generada aleatoriamente. Es deseable que los individuos de la población, sean lo más diversos posible, ya que de esta manera se tiene mayor cobertura de soluciones sobre el espacio de búsqueda. Esto último, puede llevar a pensar que a mayor población mejor se comportará el algoritmo, lo cual no necesariamente es cierto, ya que una población muy grande comprometería la eficiencia en tiempo del método. En general, se recomienda buscar el equilibrio entre estos dos puntos en conflicto para determinar un buen tamaño de población inicial para un problema dado [32].

2. **Selección**

   En los algoritmos evolutivos, la etapa de selección es la que se encarga de velar que los mejores individuos tengan mayor probabilidad de reproducirse, lo que conduce a la población a evolucionar hacia mejores soluciones. Las estrategias de selección, se pueden ver como el modo que tienen los algoritmos evolutivos para intensificar la búsqueda en el espacio de soluciones sobre las que son más prometedoras. Sin embargo, el mecanismo de selección no puede descartar totalmente individuos de mal fitness, ya que el utilizar sólo a los mejores puede dirigir al procedimiento hacia convergencias prematuras y óptimos locales. Algunos mecanismos de selección son: selección por ruleta, selección basada en ranking y selección por torneo [33] [34].

3. **Reproducción**

   El rol de la fase de reproducción es, una vez escogidos los progenitores de la siguiente generación, establecer los mecanismos mediante los cuales ocurrirá la variación genética (es decir, los operadores de búsqueda de la metaheurística). Los métodos más clásicos son la *Mutación* (operador unario) y el *Crossover* (operador binario), los cuales se presentan a continuación:

- **Mutación**: La mutación es un operador unario, que actúa produciendo cambios sobre un cromosoma. Algunas características que idealmente debería cumplir este operador son: todas las soluciones deberían poder ser alcanzadas mediante mutación; los cromosomas mutantes deben ser soluciones válidas según las restricciones que imponga el problema; la mutación debería generar soluciones vecinas al cromosoma mutado, es decir, ocasionar pequeños cambios. La probabilidad de mutación de un cromosoma, queda establecida por un parámetro p_m llamado *probabilidad de mutación*, el cual se recomienda sea un valor pequeño, ya que una alta probabilidad de mutación ocasionaría que la búsqueda se vuelva altamente aleatoria. Existen distintos mecanismos de mutación dependiendo de cuál sea la representación de las soluciones. En particular, para este trabajo interesan los mecanismos de mutación para permutaciones, ya que será la representación que se utilizará para resolver el problema del CVRP. Algunos de éstos son: mutación por inserción, mutación por intercambio y mutación por inversión [35] [36].

- **Crossover**: El crossover es un operador de variación genética binario. El objetivo del crossover, es heredar material genético de los padres a sus descendientes, con el fin de explorar sectores prometedores dentro del espacio de búsqueda. Un operador de crossover debería cumplir con: generar descendientes válidos con respecto a las restricciones del problema; heredar material genético de ambos padres al hijo. Algunos mecanismos de crossover para permutaciones son: crossover ordenado OX, crossover mapeado parcialmente PMX y crossover de dos puntos [28] [35] [36].

4. **Reemplazo**

   La fase de reemplazo, es en la cual se determina qué individuos de entre la generación actual, y los hijos generados, formarán parte de la siguiente generación. Las dos estrategias más frecuentes de reemplazo son el reemplazo generacional y el reemplazo de estado estable [37].

2.4. Algoritmos Inspirados en la Conjugación Bacteriana

A continuación, se presentarán [1] y [2], dos trabajos previos a esta memoria, y que son metaheurísticas que caben en la categoría de algoritmos evolutivos, los cuales están inspirados en las bacterias y se centran principalmente en el mecanismo de conjugación bacteriana.

2.4.1. Algoritmo de Conjugación Bacteriana

El algoritmo de Conjugación Bacteriana (ACB) propuesto por H. Fernández [1] es una metaheurística evolutiva, que toma como principal inspiración la transferencia genética que realizan las bacterias mediante la conjugación bacteriana (ver punto 2.1.3). Este método tiene como característica realizar una transferencia de genes de manera horizontal y no vertical como en la mayoría de los algoritmos evolutivos. Esto quiere decir que no se basa en una reproducción sexual, donde dos individuos padres transfieren sus genes a su descendencia. En cambio, lo que se tiene es que entre dos bacterias, una realiza el papel de donadora y transfiere una copia de algún segmento de su material genético a una bacteria receptora, siendo este proceso absolutamente aparte de la reproducción de las bacterias. Cabe recordar, que la reproducción bacteriana se lleva a cabo por fisión binaria, es decir una bacteria crea una copia suya con exactamente el mismo material genético.

El ACB tiene un principio más colaborativo que competitivo, a diferencia de la mayoría de los algoritmos evolutivos. En un algoritmo evolutivo tradicional, los individuos compiten para generar descendencia. En cambio, en ACB no existe tal concepto, sino que por el contrario, bacterias con buenos genes, cuya característica es la presencia del plásmido F, buscan de manera altruista mejorar el fitness de bacterias de menor calidad, por medio de la transferencia de una copia de su plásmido.

La figura 4 representa el esquema del algoritmo de conjugación bacteriana.

Fases de ACB

1. **Inicializción**: Se genera una población inicial aleatoria de bacterias, donde cada bacteria posee un cromosoma, el cual representa una solución codificada del problema.

2. **Evaluación**: Se utiliza la función de evaluación del problema para evaluar a las bacterias de la población.

3. **Clasificación**: De acuerdo a la evaluación realizada anteriormente, las bacterias se clasifican en células donadoras (las que poseen un plásmido F, las cuales pueden ser F+ o Hfr) y células receptoras (las que no poseen plásmido F, y se denominan F-). Esta clasificación se realiza probabilísticamente, buscando que mejores células sean donadoras.

4. **Selección**: Aleatoriamente se realizan emparejamientos entre células donadoras con células receptoras, para luego determinar si se realizará intercambio genético entre ellas.

5. **Conjugación**: Una vez realizada la selección de parejas, se realiza la conjugación, en la cual ambas células elegidas intercambian parte de su material genético según una probabilidad P_z (*probabilidad de conjugación*). Para determinar el sector inicial del cromosoma que se intercambiará, se utiliza un punto de corte aleatorio i. Luego, se determina el largo L del segmento, el cual queda determinado por $L = \frac{1}{\alpha} \times ln(U)$, donde U es un parámetro llamado *probabilidad de intercambio*, el cual toma valores entre 0 y 1 y varía si la célula es Hfr o F+ (las células Hfr tienen probabilidad de intercambiar un mayor segmento del

Figura 4: Algoritmo de Conjugación Bacteriana.

cromosoma). α representa la influencia de la temperatura en la conjugación, donde a mayor temperatura menor es el largo del segmento a intercambiar y viceversa.
Una conjugación entre una célula F+ y una F- transforma a la receptora en una F+. En cambio, la conjugación entre una Hfr con una F- no transforma a la célula receptora en F+.

6. **Mutación**: En esta etapa las células que realizaron conjugación pueden sufrir mutación en su cromosoma, lo que queda determinado por el parámetro P_m llamado *probabilidad de mutación*.

7. **Curación**: En esta etapa algunas células donadoras pueden perder espontáneamente su plásmido F, pasando de ser donadoras a receptoras. La probabilidad de curación queda determinada por P_c llamado *porcentaje de curación*.

Parámetros de ACB

1. Cp_i: Cepa inicial, la cual determina la cantidad de células en la población inicial, cantidad que permanece constante durante la ejecución del método. En ACB el tamaño que se utiliza de cepa inicial corresponde al número de genes del cromosoma.
2. N_0: Número de iteraciones que correrá el algoritmo.
3. C_i: Corresponde a la cantidad de células que serán donadoras en la población inicial. C_i a su vez se divide en c_f que es la cantidad de células donadoras que serán F+, y c_h el número de células donadoras Hfr.
4. P_z: Probabilidad de que una pareja seleccionada realice conjugación.
5. P_s: Criterio de supervivencia, el cual determina el porcentaje de mejores células que pasarán directamente a la siguiente iteración.
6. P_c: Porcentaje de curación, que es la probabilidad de que una célula donadora pase a ser receptora de una iteración a otra.
7. P_m: Probabilidad de mutación, la cual establece qué tan frecuentemente las células que realizaron conjugación muten alguna parte de su cromosoma.
8. U: Probabilidad de intercambio, el cual determina el largo del sector del cromosoma que se intercambia en una conjugación.

ACB aplicado al CVRP

A continuación se presentan las características de ACB cuando se aplica al CVRP.

- **Representación de las soluciones**: Una solución del CVRP se representa como un arreglo, el cual corresponde a una permutación con el orden en que se visitarán los clientes en esa solución. Por lo tanto, cada arreglo corresponde también a un cromosoma bacteriano.

  Cabe destacar que en esta representación no se incluye el depósito como un punto dentro de las soluciones, ya que cada ruta comienza y termina en él. En una solución, para determinar cuando comienza y termina la ruta de un vehículo, se suma acumulativamente las demandas de los clientes, hasta sobrepasar la capacidad máxima del vehículo.

- **Función de Fitness**: La función que se utilizó para evaluar la calidad de los cromosomas en el ACB aplicado al CVRP, es la suma total de la distancia recorrida en cada solución:

  - $$F = \sum_{(i,j)\in E} c_{ij} \times x_{ij}$$

Donde E es el conjunto de aristas de la instancia del problema, c_{ij} es la distancia entre el cliente i y j, y x_{ij} una variable binaria que toma valor 1 si la arista (i, j) se toma en la solución que se evalúa, o 0 en caso contrario.

- **Implementación de la conjugación y mutación**: La conjugación se realizó mediante crossover OX, donde la célula donadora y receptora intercambiaban material genético *"bi-direccionalmente"*, es decir, ambas actuaban como donadoras y receptoras. En el modelo que se propone más adelante, se realiza un cambio a esta estrategia de conjugación, y para diferenciar se llamará a este método *"Conjugación Bidireccional"*.

En cuanto a la mutación, ésta fue implementada mediante mutación por intercambio.

2.4.2. Algoritmo de Conjugación Bacteriana con Control de Población

El trabajo realizado en [1], fue continuado por V. Ortiz quién realizo algunas correcciones necesarias y modificaciones buscando mejorar el método, trabajo que dio origen al Algoritmo de Conjugación Bacteriana con Control de Población (ACBCP) y quedó plasmado en [2].

Al ejecutar ACB, se observó que a pesar de ejecutar muchos ciclos del algoritmo, el mayor cambio ocurría sólo durante un determinado número de iteraciones (del orden de las 200.000 primeras). Luego, el algoritmo entraba en un estancamiento donde la mejora de las soluciones no era significativa. El diagnóstico fue que esto ocurría porque en la población del algoritmo, las células F- y las Hfr desaparecían después de una cantidad de iteraciones. Esto debido a que las bacterias F- al conjugar con F+, se transforman también en F+. Y las Hfr desaparecían por el proceso de curación, pero a diferencia de las F+ éstas no poseen ningún mecanismo que les permita regenerarse. Por lo anterior, se concluyó que el proceso de curación no se estaba desempeñando correctamente.

Para solucionar el problema planteado, inspirándose en el Quorum Sensing (ver punto 2.1.2.), las bacterias pasaron a ser conscientes de las cantidad de células F+, Hfr, y F- con las que cuenta su población, pudiendo tomar medidas de acuerdo a esos números. Con ello, se consideró como equilibrio que de la población total, la mitad fueran células receptoras, y la otra mitad donadoras. Y dentro de las células donadoras, la mitad fueran F+ y la otra mitad Hfr. Para lograr el equilibrio, se modificó el proceso de curación, el cual se dividió en dos fases que se aplicaban en cada iteración:

- Si la cantidad de bacterias receptoras era menor a la de equilibrio, se realizaría curación hasta lograr el equilibrio.
- Análogamente, se implementó una nueva "curación", la cual transformaba bacterias F+ en Hfr, hasta lograr el valor de equilibrio.

Además, otro cambio que se realizó con respecto al trabajo anterior, fue que la cepa del algoritmo se podía probar con diferentes tamaños en la población inicial.

Con los cambios anteriores, para diferenciar los algoritmos, se bautizó al nuevo método como Algoritmo de Conjugación Bacteriana con Control de Población (ACBCP), cuyo esquema se ve en la figura 5.

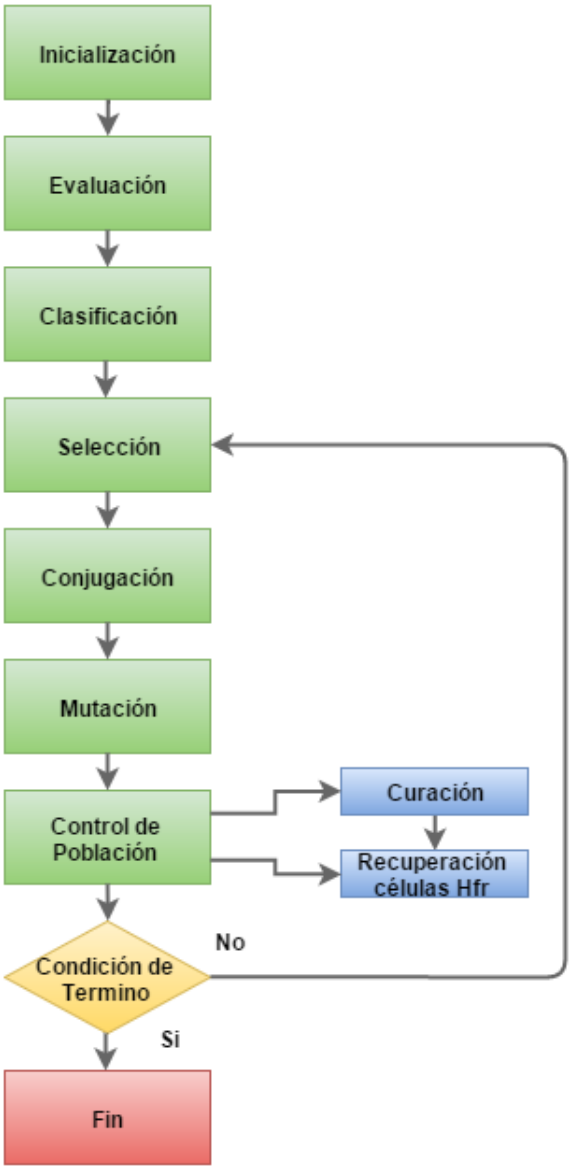

Figura 5: Algoritmo de Conjugación Bacteriana con Control de Población.

ACBCP aplicado al CVRP

En [2] se realizan algunas correcciones importantes a errores sobre cómo estaba aplicándose ACB al CVRP, y que llevaban al método a encontrar soluciones que eran falsamente óptimas.

Corrección 1: Este problema surgía al evaluar un cromosoma para obtener su valor de fitness. Supongamos que se tiene el siguiente cromosoma representando a una solución (cada cambio de color indica una ruta tomada por un vehículo distinto):

En este cromosoma el depósito (punto 0) no se incluye, pero debería ser tomado en cuenta a la hora de la evaluación del fitness. El problema es que al evaluar, se consideraba al depósito sólo al comienzo y final, siendo el siguiente el cromosoma evaluado:

Lo anterior no consideraba que realmente cada vehículo comienza y termina en el depósito, por lo que el cromosoma que realmente debería haber sido evaluado para la solución es el siguiente:

Este cromosoma evalúa 4 aristas adicionales que fueron pasadas por alto en la anterior representación, por lo que el valor real de una solución puede ser drásticamente más alto a como se estaba evaluando anteriormente.

Para solucionar este problema, se planteó realizar una fase de decodificación anterior a la evaluación del fitness de las soluciones. En la fase de decodificación se realizaba una copia de los cromosomas en la cual se insertaban las salidas y vueltas al depósito, copia la cual era evaluada por la función de fitness y posteriormente desechada.

Corrección 2: : Más que la corrección de un error, puede decirse que el siguiente cambio fue una mejora en la evaluación de las soluciones. La función de evaluación tal como estaba planteada, sólo tomaba en cuenta el largo total de los recorridos realizados, y no consideraba la cantidad de vehículos que fueron utilizados para aquello.

Por lo anterior, se asumió que una distancia total recorrida con una cantidad determinada de vehículos, era mejor que una distancia total menor siempre y cuando utilizara menos vehículos (por ejemplo, una distancia recorrida de 745 km con 7 vehículos se considera peor solución que una distancia total de 1000 km con 6 vehículos).

Para incluir el peso de los vehículos en la función de evaluación, se agregó un factor cuadrático que penalizaba la utilización de más vehículos en una solución (c_v es la cantidad de vehículos

utilizados en la solución):

- $F = \sum_{(i,j) \in E} (c_{ij} \times x_{ij}) \times c_v^2$

No obstante, esta función de fitness sólo es utilizada para determinar al mejor individuo de una generación, y no modifica el fitness real de cada solución, el cual sólo toma en cuenta la distancia recorrida.

3. Modelo propuesto

En este capítulo se dará a conocer el modelo de la metaheurística propuesta, la cual recibe el nombre de Algoritmo Bacteriano de Resistencia Antibiótica (ABRA), y que será probada utilizando el CVRP. Se mostrará su inspiración, las novedades respecto a sus predecesoras, se explicará su funcionamiento, etapas y parámetros.

3.1. Algoritmo Bacteriano de Resistencia Antibiótica

El Algoritmo Bacteriano de Resistencia Antibiótica (ABRA) es una metaheurística inspirada en cómo una población de bacterias va evolucionando y volviéndose resistente a la aplicación controlada de distintos antibióticos, siempre y cuando éstos no sean lo suficientemente fuertes como para acabar con la colonia entera de una sola vez (*"Lo que no te mata te hace más fuerte"*).

Este algoritmo, como principal característica distintiva a los trabajos inspirados en la conjugación bacteriana (revisados en el capítulo anterior), utiliza una variable llamada *Antibiótico*, la cual ejerce una presión selectiva sobre la población y la estimula a mejorar hacia individuos más aptos, debido a la necesidad de adaptarse para sobrevivir. A su vez, ABRA también posee el carácter colaborativo de sus predecesores, ya que ante la presencia de un antibiótico, serán las bacterias que sean resistentes las que, mediante el mecanismo de conjugación, buscarán fortalecer a las no resistentes para que puedan mejorar y sobrevivir.

Cabe destacar que en este método no se utilizan los conceptos de células F+, F- y Hfr, que utilizaban los trabajos previos como base para la conjugación (ver Conjugación en el punto 3.1.4.).

Este algoritmo además agrega el mecanismo de transformación bacteriana como otro método de variación genética. En la transformación, una bacteria que murió al no poder sobrevivir a la acción de un antibiótico, puede dejar parte de su material genético libre en el ambiente para que éste luego sea capturado por otra bacteria. El aporte de la transformación al algoritmo, es que añade un nuevo mecanismo que aumenta la variabilidad genética de la población, lo cual aumenta la exploración dentro del espacio de soluciones.

Finalmente, otra nueva característica del algoritmo, es que al detectar que se ha llegado a un óptimo local (es decir se perdió la variabilidad genética, lo cual se produce cuando todas las bacterias poseen el mismo fitness), le da un "remezón" a la población, el cual consiste en permitir que todas las bacterias realicen conjugación, mutación y transformación libremente, de manera indistinta de si esto empeora su fitness, hasta que se recupere la variabilidad genética de la población. Esta fase no posee una contraparte biológica real, pero es de suma importancia para que el algoritmo no se estanque y pueda continuar la búsqueda de soluciones.

El esquema de ABRA queda resumido en la figura 6, y sus etapas serán detalladas a continuación.

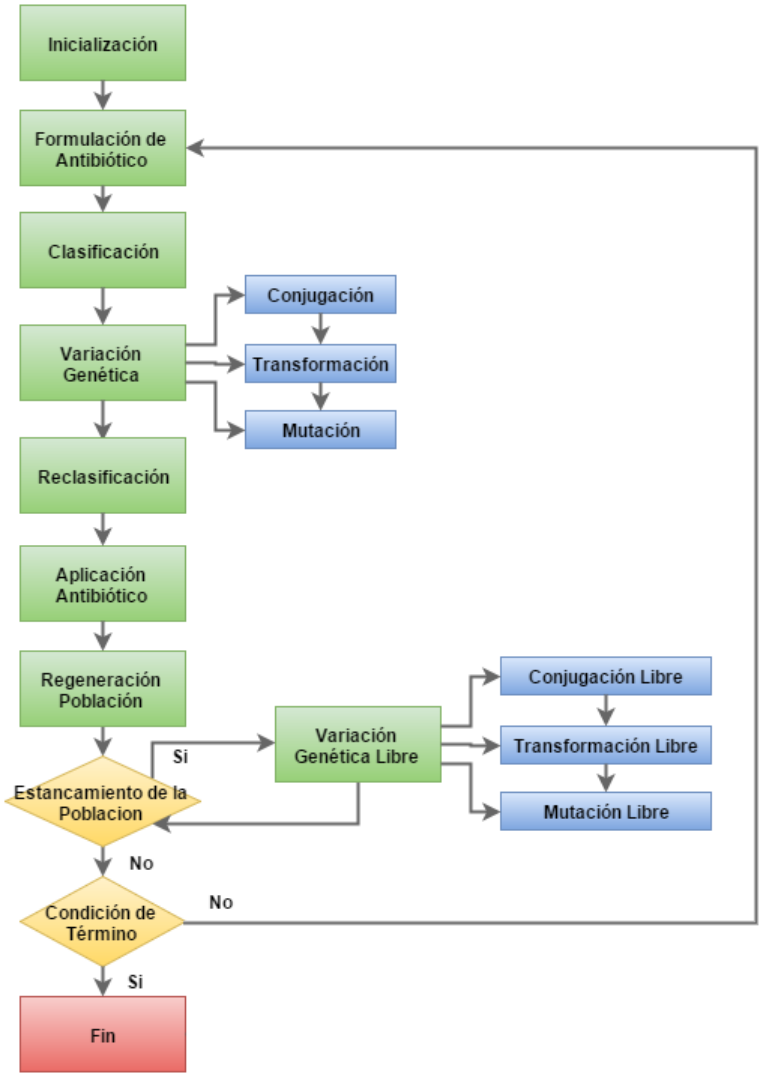

Figura 6: Algoritmo ABRA.

3.1.1. Inicialización

En la primera etapa se genera una población inicial de bacterias, donde cada una posee un cromosoma que es la representación codificada de una solución del problema. El tamaño de la población queda determinado por un parámetro s_p (*tamaño de la población*), el cual permanece constante durante cada iteración del algoritmo, y su valor óptimo debe ser determinado para cada problema en particular. Se debe considerar que una población pequeña puede ser poco representativa y llevar al algoritmo a convergencias prematuras, y una población demasiado grande puede volver al método excesivamente lento.

3.1.2. Formulación del Antibiótico

En la segunda etapa, se evalúa la población de bacterias mediante la función de fitness establecida, y de acuerdo a esta evaluación se formula un antibiótico. En el algoritmo, el antibiótico es alguna estructura que pueda ser evaluada por la función de fitness, y cuyo valor determina su "fuerza" como antibiótico. El antibiótico actúa como una medida de presión y determina qué bacterias dentro de la población deberían formar parte de la siguiente generación, e intenta promover la mejora de los individuos de menor calidad. Sin embargo, el antibiótico no es aplicado inmediatamente luego de formulado, para que bacterias que en un principio debiesen morir por la acción de éste, tengan posibilidad de mejorar mediante los mecanismos de variación genética de las mismas.

El antibiótico que se utilizará, será una solución que se encuentre dentro de la población de bacterias. Para cada iteración de ABRA, luego de evaluar a la población, se elegirán aleatoriamente 2 pares de bacterias. De cada par de bacterias, se seleccionará la que posea el mejor fitness. Luego, de las dos bacterias seleccionadas, se elegirá la que tenga el peor fitness, y su solución pasará a ser el antibiótico para esa iteración. El *Valor del Antibiotico* será el fitness del antibiótico. La figura 7 muestra cómo se lleva a cabo la selección (el número representa el fitness de la solución de cada bacteria), para un caso donde el mejor fitness es el menor (minimización de fitness).

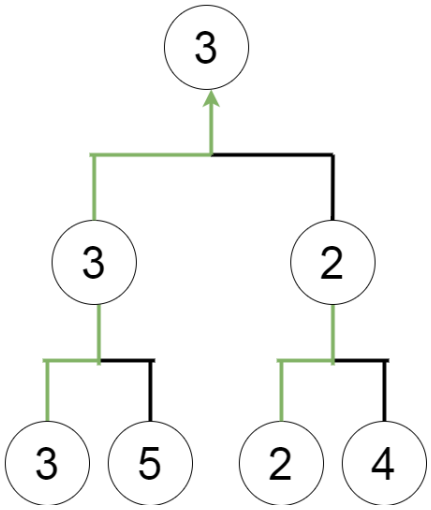

Figura 7: Formulación del antibiótico.

Este método garantiza que el antibiótico tenga un valor que esté dentro del rango de fitness de la población, y que éste probablemente no sea ni el valor más alto ni el más bajo.

La cantidad de bacterias utilizadas para determinar el antibiótico puede ser variable (por ejemplo, se puede correlacionar con el tamaño de la población) y queda a criterio de quien implemente la metaheurística. En este trabajo por motivos de simpleza se optó por 4 bacterias,

como fue mostrado anteriormente.

Otra posible forma de seleccionar un antibiótico, es variando genéticamente soluciones existentes en la población. No obstante, este método puede llevar a que los valores del antibiótico queden fuera del rango del fitness de la población, al poder generarse una solución peor o mejor que todas las existentes. Si lo anterior sucediera, el antibiótico sería inútil o eliminaría a todas las bacterias.

3.1.3. Clasificación

En la tercera etapa se clasifica la población en dos tipos de bacterias: *bacterias resistentes* y *no resistentes* de acuerdo al antibiótico formulado en el paso anterior. El rol de esta clasificación, es determinar qué bacterias se considerarán las mejores dentro de la población y promoverán la mejora de las bacterias no tan buenas.

En la clasificación que se realizará, todas las bacterias con un fitness mejor o igual que el valor del antibiótico pasan a conformar el grupo de bacterias resistentes, mientras que las de fitness peor conforman el grupo de las no resistentes.

- $\forall i \in P, F(i) \leqslant F(A) \Rightarrow i \in R$

- $\forall i \in P, F(i) > F(A) \Rightarrow i \in NR$

Donde P es la población, i una bacteria de la población, F la función de fitness, A el antibiótico, R el conjunto de bacterias resistentes y NR el conjunto de bacterias no resistentes.

3.1.4. Variación Genética

En la cuarta etapa se produce la variación genética de los individuos de la población, la cual se espera que tenga como resultado obtener una mayor cantidad de bacterias resistentes y que posteriormente sobrevivan a la aplicación del antibiótico. Se utilizan tres mecanismos de variación genética, los cuales se realizan en el siguiente orden:

- **Conjugación**: La conjugación en ABRA, se realiza buscando que las bacterias clasificadas como resistentes, transfieran su "factor de resistencia" (el cual en la naturaleza generalmente corresponde a un plásmido) a las bacterias no resistentes. Para ello, todas las bacterias no resistentes tienen una probabilidad p_c (*probabilidad de conjugación*) de realizar el proceso de conjugación con una bacteria resistente aleatoria. En esta conjugación a diferencia de los trabajos anteriores (donde se realizaba una conjugación "bidireccional"), solo la bacteria resistente traspasa material genético a la no resistente, sin producirse el proceso recíproco (a esta conjugación se le llamará *"unidireccional"*). Con esto se busca

promover la mejora de bacterias no tan aptas, sin arriesgarse a perder la calidad de las buenas soluciones ya encontradas.

La conjugación entre bacterias donadora y receptora, se llevará a cabo de manera similar al Crossover OX utilizado en los algoritmos genéticos [38]. Se seleccionan aleatoriamente un punto A y B dentro del cromosoma de la bacteria donadora. La donadora, entrega una copia del trozo de su solución que se encuentra entre los puntos A y B, el cual conserva su posición absoluta al ser insertado en el cromosoma de la bacteria receptora. La receptora, a partir del punto B determina en orden, qué genes suyos no están en el material genético que le ha sido donado, y los re ordena insertándolos desde B hasta completar su cromosoma. La figura 8 ilustra el proceso (en rojo el material donado, el cual conserva su posición absoluta en la receptora; en naranjo, el material que debe ser reordenado en la receptora; a la derecha, el resultado una vez realizada la conjugación).

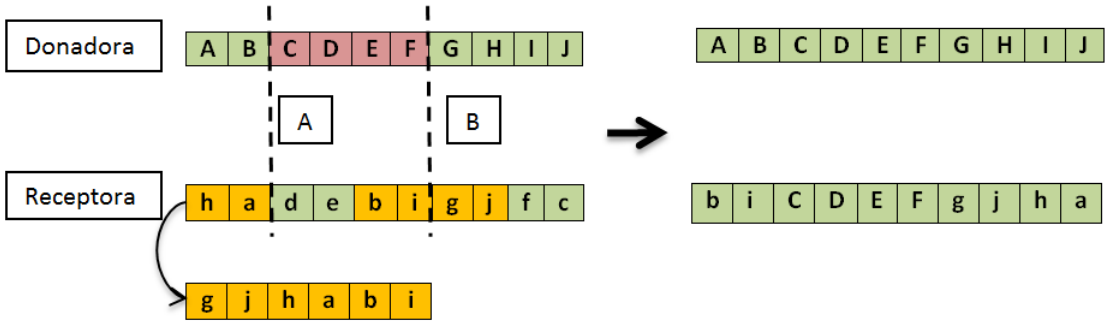

Figura 8: Conjugación entre una bacteria receptora y una donadora.

- **Transformación**: La transformación es un mecanismo nuevo que se implementó en ABRA inspirado en el proceso natural de transformación genética (ver punto 2.1.3). En la transformación, todas las bacterias no resistentes tienen una probabilidad p_{tr} (*probabilidad de transformación*) de recombinar su material genético con material disponible libre en el ambiente (en el punto 3.1.6., sobre la aplicación del antibiótico, se muestra cómo se liberan genes al ambiente), siempre y cuando haya disponibilidad de éste. Al ser utilizado algún trozo de material genético libre para realizar transformación, este material ya no se encontrará más disponible en el ambiente.

  La transformación brinda la posibilidad de rescatar potenciales trozos buenos de una solución que globalmente no sea buena, a la vez que permite aumentar la exploración en el espacio de soluciones, al brindar más variabilidad genética a la población.

  La transformación de una bacteria se llevará a cabo de manera similar a la conjugación. Primero, se selecciona al azar qué trozo de material genético dentro de los disponibles en

el ambiente será utilizado para la transformación. Luego, se selecciona aleatoriamente un punto A dentro del cromosoma de la bacteria transformante, y a partir de él, se determina un punto B, tal que B es igual al punto A más el largo del material genético escogido para realizar la transformación. Es entre el punto A y B que se inserta el trozo nuevo de cromosoma, y el resto de material genético de la bacteria se reordena de igual manera que en la conjugación. La figura 9 muestra este proceso (en rojo el materia genético libre que será insertado en la bacteria transformante; en naranjo el material genético que debe ser reordenado en la transformante; a la derecha el resultado una vez realizada la transformación).

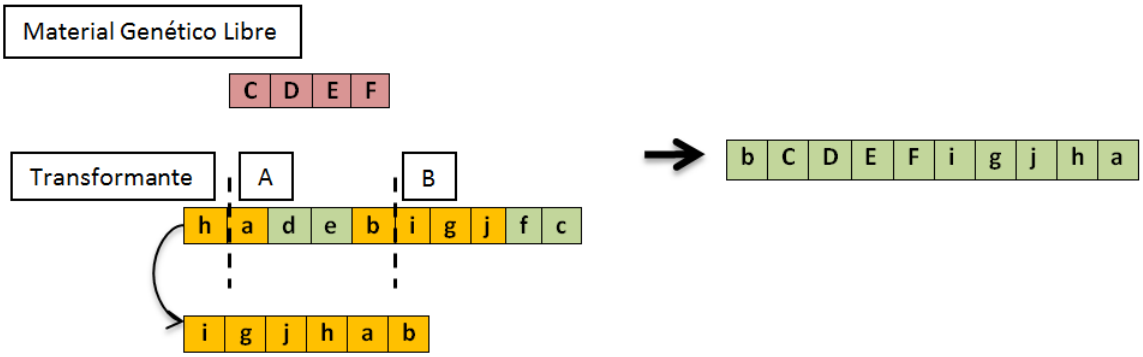

Figura 9: Transformación de una bacteria transformante con material genético libre en el ambiente.

- **Mutación**: La mutación es el último mecanismo de variación genética utilizado, y permite que las bacterias no resistentes tengan la posibilidad de mutar, pudiendo en el mejor de los casos desarrollar resistencia espontáneamente. A su vez, las bacterias resistentes igualmente pueden mutar, pero sólo aceptarán la mutación cuando les confiera mejores características. La probabilidad de que una bacteria mute, queda dada por p_m (*probabilidad de mutación*).

  La mutación de una bacteria se realizará por Intercambio de un Par de Puntos, al igual que como se usa en los algoritmos genéticos. En esta mutación, se seleccionan aleatoriamente 2 puntos dentro del cromosoma bacteriano, y se intercambia el contenido de aquellos genes, como se muestra en la figura 10 (en rojo los genes que intercambiarán su posición dentro del cromosoma, y abajo el resultado de la mutación).

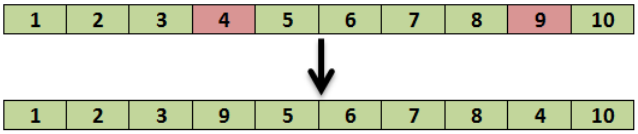

Figura 10: Mutación de una bacteria.

Los tres mecanismos anteriores, fueron descritos según la manera en que se implementaron, pudiendo elegirse otras maneras de realizar las recombinaciones de genes (por ejemplo mediante Crossover PMX o Crossover de Dos Puntos) y la mutación (usando por ejemplo Mutación por Inversión o Mutación por Inserción) [28]
.

3.1.5. Reclasificación

En la quinta etapa, se olvida la clasificación de la etapa tres, y se realiza nuevamente de acuerdo a los fitness de la población actual, la cual fue sometida a procesos de variación genética en la etapa cuatro.

La reclasificación se realizará de la misma forma que en el punto 3.1.3., una vez hayan sido reevaluados los nuevos fitness de las bacterias.

3.1.6. Aplicación del Antibiótico

En la sexta etapa son eliminadas todas las bacterias clasificadas como no resistentes en la fase anterior, es decir, todas aquellas cuyo valor de fitness es peor que el valor del antibiótico. Sin embargo, cada bacteria antes de ser eliminada tiene una probabilidad p_r (*probabilidad de liberación*) de dejar un trozo de su material genético libre en el ambiente.

La liberación de material genético se realizará seleccionando 2 puntos aleatorios A y B dentro del cromosoma, y dejando como genes libres los que se encuentren dentro de ambos puntos, lo cual se ilustra en la figura 11 (en rojo el material genético que será liberado, y a la derecha ya estando libre y disponible para ser utilizado para una transformación en las siguientes iteraciones).

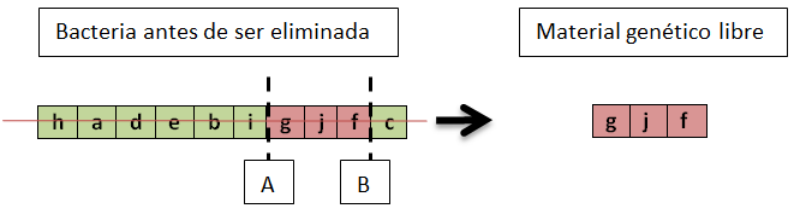

Figura 11: Liberación de parte de su cromosoma por una bacteria antes de morir.

3.1.7. Regeneración de la Población

En la séptima etapa, se debe restablecer el número original de la población de bacterias, reemplazando con nuevos individuos los que fueron eliminados durante la aplicación del antibiótico.

En la regeneración, se determina cuántos individuos le faltan a la población actual (luego de aplicado el antibiótico) para volver a tener su tamaño normal (dado por el parámetro s_p del algoritmo). Para esto, por cada individuo que falte se simulará que alguna bacteria de la población, la cual es elegida aleatoriamente, realizará fisión celular con *100 %* de probabilidad de mutación, hasta restablecer el tamaño original de la colonia. Sin embargo, esta mutación es realizada por intercambio de dos pares de puntos, para que de esta manera, sea mayor la diferencia genética con la bacteria progenitora, buscando así la variabilidad genética de la población. La figura 12 muestra cómo se realiza la fisión y mutación para generar una nueva bacteria (a partir de una bacteria aleatoria, se crea una copia de ésta, sobre la cual se realiza mutación por intercambio de dos pares de puntos; en rojo y azul los dos pares respectivamente y abajo, el resultado de la fisión).

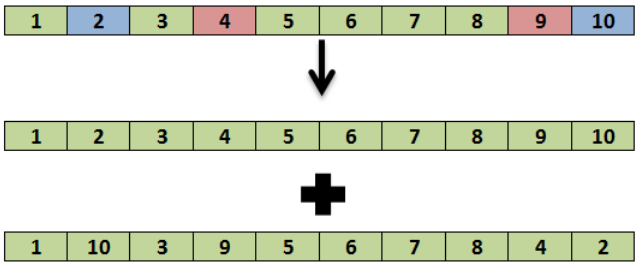

Figura 12: Generación de una nueva bacteria a partir de una existente.

Luego de finalizada esta etapa, se determinan los individuos con mejor y peor fitness de la generación. Además, desde la primera iteración del algoritmo se va guardando al mejor individuo encontrado hasta ese momento en todas las iteraciones, el cual se obtiene por comparar la mejor bacteria encontrada en la generación actual, con la mejor encontrada durante todo el algoritmo.

3.1.8. Estancamiento de la Población

En el algoritmo, se dirá que se produjo un estancamiento de la población cuando se pierde la variabilidad genética de la misma (es decir, se llegó a un óptimo local), por lo que se vuelve necesario promover la variación genética de las bacterias en caso de que esto suceda.

El criterio para decir que se produjo estancamiento de la población, será cuando en una iteración del algoritmo el mejor fitness encontrado sea igual al peor dentro de la población de bacterias. Si se concluye que hay estancamiento de la población, se procede con la variación genética libre.

3.1.9. Variación Genética Libre

Esta etapa sólo ocurre cuando se ha producido un estancamiento en la búsqueda de soluciones. El fin de la variación genética libre, es que la población de bacterias pueda restablecer su variabilidad, al realizar los individuos conjugación, transformación y mutación libres de la presión de un antibiótico.

La variación genética libre se realizará hasta que deje de cumplirse el criterio de estancamiento. El objetivo de esta etapa más que mejorar la solución, es restablecer la variabilidad genética de la misma, por lo cual no se utiliza antibiótico.

- **Conjugación Libre**: Para esta conjugación, cada bacteria tiene una probabilidad p_{cl} de realizar conjugación con cualquier bacteria aleatoria de la población. A diferencia de la conjugación normal, en este caso el intercambio genético es "bidireccional", por lo que ambas bacterias juegan el papel de donadora y receptora de material genético.

  La conjugación libre se llevará a cabo de manera similar a la conjugación normal, como se muestra en la figura 13 (en rojo el material que le donarán a la otra bacteria, y en letras naranjas los genes que deberán re-acomodar en su cromosoma).

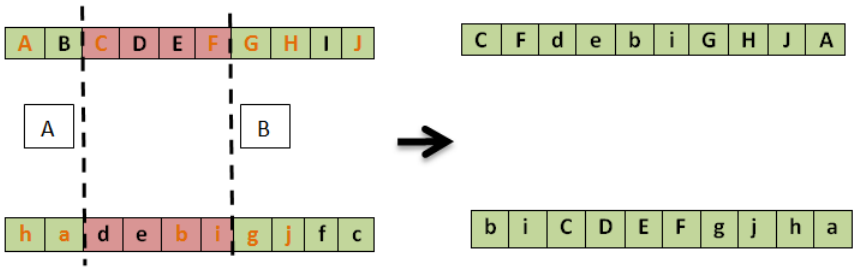

Figura 13: Conjugación libre entre dos bacterias.

- **Transformación Libre**: En la transformación libre cada bacteria de la población tiene una probabilidad p_{trl} de realizar transformación con material genético libre en el ambiente (siempre que haya disponibilidad de éste), independiente de si esta transformación mejora o empeora su fitness.

  La transformación libre se realizará de igual manera que la transformación normal. En general se recomienda que $p_{trl} > p_{tr}$, buscando acelerar la variación genética.

- **Mutación Libre**: De manera similar a la transformación libre, en esta mutación, todas las bacterias pueden mutar con una probabilidad p_{ml}, independiente si el resultado es favorable o no para su fitness.

  La mutación libre se realizará de igual manera que la mutación normal. En general se recomienda que $p_{ml} > p_m$, buscando acelerar la variación genética.

3.1.10. Condición de Término

La condición de término dispara el fin del algoritmo, y puede ser por ejemplo un número máximo fijo n_{it} de iteraciones, que es lo que se utilizará en este trabajo.

3.1.11. Resumen de los Parámetros de ABRA

- n_{it} = número de iteraciones máximas que ejecuta el algoritmo.
- s_p = tamaño de la población de bacterias.
- p_c = probabilidad de conjugación.
- p_{cl} = probabilidad de conjugación libre.
- p_{tr} = probabilidad de transformación.
- p_{trl} = probabilidad de transformación libre.
- p_m = probabilidad de mutación.
- p_{ml} = probabilidad de mutación libre.
- p_r = probabilidad de liberación.

3.2. ABRA orientado al CVRP

Como se mencionó anteriormente, el Problema de Ruteo Vehicular con Capacidad Uniforme (CVRP) es el que será utilizado para probar la metaheurística propuesta en este trabajo. Para ello, se debe determinar la representación de las soluciones y la función de evaluación que utilizará el método.

3.2.1. Representación de las Soluciones

Como se vio en el capítulo 2, el CVRP consiste en encontrar un conjunto óptimo de rutas para satisfacer la demanda de una determinada cantidad de clientes, tal que se cumplen una serie de condiciones (ver CVRP en el punto 2.2.2).

Para este trabajo se utilizará la misma representación usada por el ACB y ACBCP (ver 2.4), la cual consiste en un arreglo que corresponde a una permutación de los clientes que se deben visitar. Una solución representada como un arreglo corresponde al cromosoma de una bacteria. El algoritmo en primera instancia maneja las soluciones ignorando el depósito (no se incluye en el arreglo), puesto que generar soluciones válidas y manejar los operadores genéticos se volvería demasiado complejo. Por ello se utiliza el mismo método que en ACBCP, en el cual a la hora de evaluar una solución se crea una copia del cromosoma, el cual se "extiende" incluyendo las idas y vueltas al depósito, siendo esta versión del cromosoma la que es evaluada. En ABRA el depósito se representa como el punto 0.

Por ejemplo, en la figura 14 se representa gráficamente un problema acompañado de una posible solución. La figura 15 muestra el cromosoma que representa aquella solución, acompañado de su copia "extendida" que se utiliza para la evaluación.

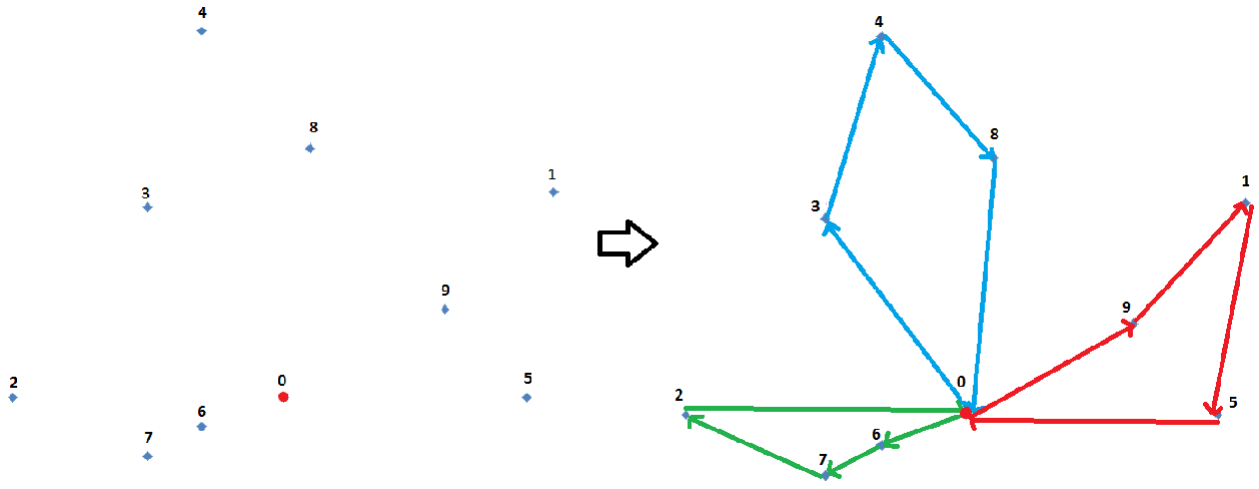

Figura 14: Representación gráfica de un problema acompañada de una solución.

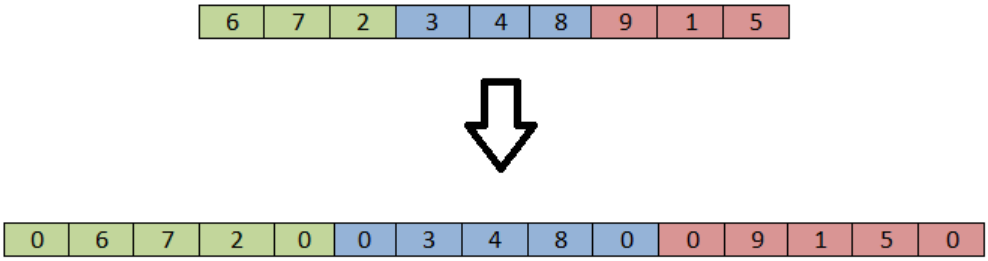

Figura 15: Cromosoma correspondiente a una solución y su versión extendida.

3.2.2. Función de Evaluación

Para obtener el fitness de cada bacteria, se utilizará la misma función de evaluación que en ACB y ACBCP, la cual devuelve la distancia recorrida en una determinada solución:

$$\diamond \; F = \sum_{(i,j) \in E} c_{ij} \times x_{ij}$$

Donde E es el conjunto de aristas de la instancia del problema, c_{ij} es la distancia entre el cliente i y j, y x_{ij} una variable binaria que toma valor 1 si la arista (i, j) se toma en la solución que se evalúa, o 0 en caso contrario.

Y para determinar la mejor solución de una generación, se utilizará la función aplicada por el ACBCP, la cual incluye un factor que penaliza el uso de más vehículos:

$$\diamond \; F = \sum_{(i,j) \in E} \left(c_{ij} \times x_{ij}\right) \times c_v^2$$

Donde c_v es la cantidad de vehículos utilizados en la solución.

4. Experimentos

En el siguiente capítulo se mostrarán las pruebas realizadas a ABRA utilizando el CVRP. Los experimentos primero van orientados a encontrar la mejor combinación de parámetros para el algoritmo, y luego a validar la calidad de las soluciones entregadas por este mismo, mediante la comparación con los resultados obtenidos por ACBCP y un Algoritmo Genético (AG) [40].

4.1. Metodología de las Pruebas

Con el fin de validar el Algoritmo de Resistencia Antibiótica (ABRA) presentado en el capítulo anterior, éste será probado con un problema de optimización combinatoria, el cual corresponde al del Ruteo Vehicular con Capacidad Uniforme (CVRP, ver punto 2.2.1.), y con el cual se medirá la calidad de las soluciones entregadas por el algoritmo.

Al igual que en los trabajos anteriores, se utilizarán las instancias de Augerat[2] para probar el algoritmo, las cuales poseen el valor de la solución óptima para cada uno de los problemas. La solución óptima para una instancia de Augerat, corresponde a la menor distancia con la que se puede resolver el problema, con el menor número de vehículos posible.

Las instancias de Augerat que se utilizarán, son 50 problemas que se dividen en tipos A y tipos B. Las de tipo A (27 de los 50) son instancias que van desde los 32 hasta los 80 clientes (incluyendo al depósito), los que se encuentran aleatoriamente ubicados en el espacio, mientras que en las de tipo B (23 de los 50) van desde los 31 hasta los 78 clientes (igualmente incluyendo al depósito), los cuales están agrupados en cúmulos. En las imágenes podemos observar las diferencias en cómo se encuentran ubicados los clientes para una instancia de tipo A (figura 16) y B (figura 17). Para ambas imágenes, el depósito es el punto rojo y los azules representan a los clientes.

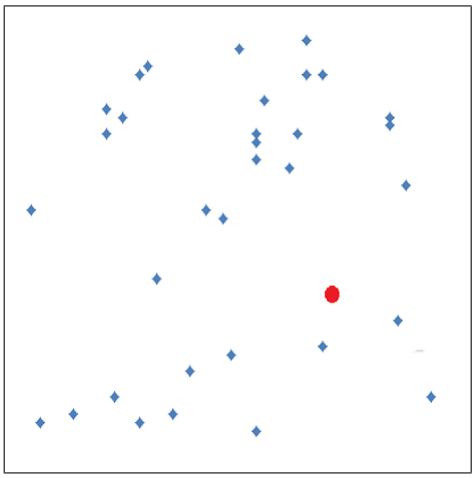

Figura 16: Instancia A-n34-k5.

[2] `http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/`

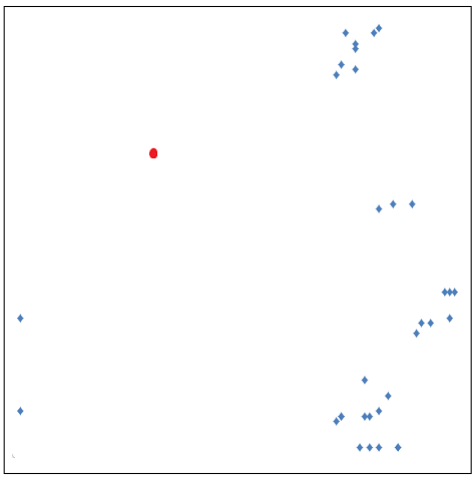

Figura 17: Instancia B-n34-k5.

Cada instancia de Augerat corresponde a un archivo de texto, el cual contiene las coordenadas de cada cliente y su demanda asociada, la capacidad de los vehículos y la solución óptima para ese problema. Cada archivo es nombrado con la forma T-nCC-V, donde T es el tipo de instancia (A o B), CC es el número de clientes (incluyendo al depósito) y V es la cantidad de vehículos que utiliza la solución óptima de ese problema.

La métrica que se utilizará para medir la calidad de las soluciones entregadas por el algoritmo, será el Porcentaje de Desviación Relativa (RPD) de la solución encontrada con respecto al óptimo conocido. Este valor, tal como su nombre lo indica, representa qué tan desviada se encuentra la solución obtenida del óptimo. Por ejemplo, si la solución óptima de un problema fuese 10, y la solución encontrada fuese 11, ésta se encontraría desviada un 1 % del óptimo. El RPD se obtiene con la siguiente fórmula (orientado a un problema de minimización):

- $RPD(\%) = \frac{Sol_{heur} - Sol_{opt}}{Sol_{opt}} \times 100$

Donde Sol_{heur} es la solución encontrada por el algoritmo, y Sol_{opt} es el óptimo conocido para el problema en particular.

Los experimentos fueron realizados en un computador con procesador AMD de 3.0GHz y dos núcleos, con 3 GB de RAM.

4.2. Ajuste de los Parámetros de ABRA

Los experimentos que se realizaron, en primer lugar tuvieron por objetivo encontrar una buena configuración de parámetros para el algoritmo, por lo que se eligieron algunas instancias,

las cuales fueron resueltas varias veces variando sus parámetros, para de esa manera evaluar con qué valores arroja soluciones de mayor calidad.

Los experimentos se realizaron sobre 4 instancias elegidas seudo-aleatoriamente (se buscó que no fueran muy similares en tamaño y tipo), las cuales fueron: A-n32-k5, A-n45-k6, B-n31-k5 y B-n50-k8.

Y los parámetros que se ajustaron fueron: probabilidad de conjugación, probabilidad de mutación, probabilidad de transformación, tamaño de la población y número de iteraciones. Para los parámetros en su versión libre, por simpleza se ajustaron como el doble del valor en su versión normal. Y para la probabilidad de liberación se utilizó el mismo valor de la probabilidad de transformación.

Un supuesto importante es que se consideró que el efecto de un parámetro era independiente del valor de los otros. Gracias a lo anterior, se pudo buscar el valor óptimo de cada parámetro haciéndolo variar mientras se mantenían fijos los otros.

Por cada parámetro se realizaron 15 ejecuciones (con 100.000 iteraciones cada una), y se conservó la mejor solución encontrada. Para las probabilidades y el tamaño de la población, como valor óptimo se escogió aquel con el que se obtuviera la mayor cantidad de mejores soluciones para las 4 instancias.

El orden de las pruebas fue:

1. Probabilidad de Conjugación.
2. Probabilidad de Mutación.
3. Probabilidad de Transformación.
4. Tamaño de la Población.
5. Número de Iteraciones.

Para las probabilidades se probó con 0.1, 0.25, 0.5, 0.75 y 0.9. Además, para el caso de la transformación se probó con 0.0, para ver si el método realmente ofrecía algún aporte al algoritmo. La población se midió con 10, 50, 100 y 200 bacterias. Y el número de iteraciones se probó con un máximo de 3 millones.

La configuración inicial se puede apreciar en la tabla 1. Al ir realizando las pruebas, se iban reemplazando los valores por los óptimos que se iban obteniendo.

Parámetros	
Numero de Iteraciones	100.000
Tamaño de la Población	10
Probabilidad de Conjugación	0.5
Probabilidad de Conjugación Libre	1
Probabilidad de Transformación	0.5
Probabilidad de Transformación Libre	1
Probabilidad de Mutación	0.5
Probabilidad de Mutación Libre	1
Probabilidad de Liberación	0.5

Tabla 1: Configuración inicial para ajuste de parámetros.

Las probabilidades no mostraron conclusiones muy claras en las pruebas. El algoritmo pareció funcionar mejor con una **probabilidad de conjugación de 0.75** (al igual que en los trabajos anteriores), **probabilidad de mutación de 0.75** (más alta que en los trabajos anteriores), y una **probabilidad de transformación de 0.25**, por lo que estos valores fueron elegidos como óptimos. Los resultados se ven en los gráficos 18, 19 y 20.

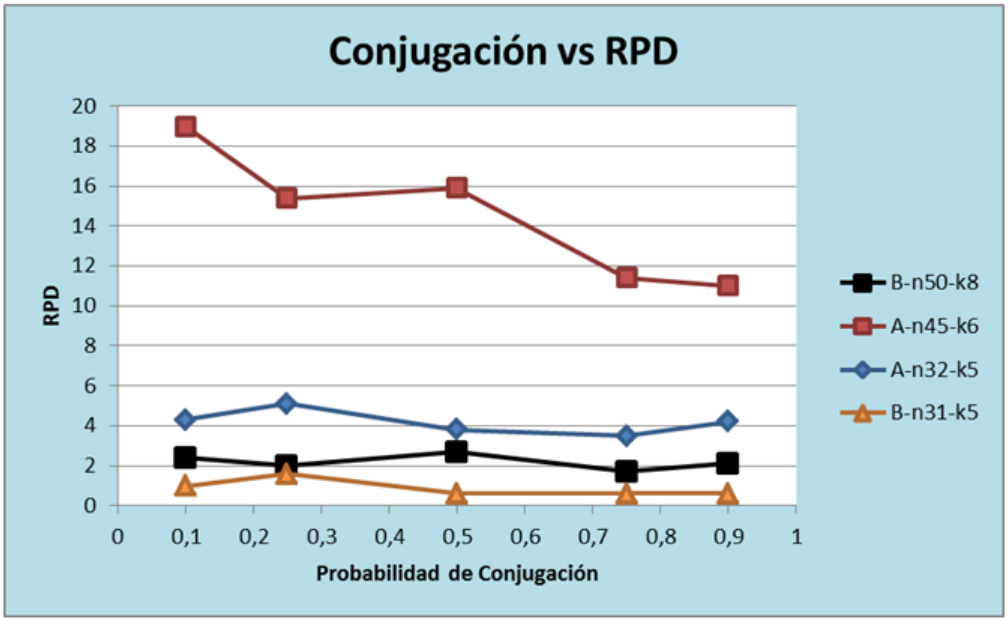

Figura 18: Variación del RPD con respecto a la probabilidad de conjugación.

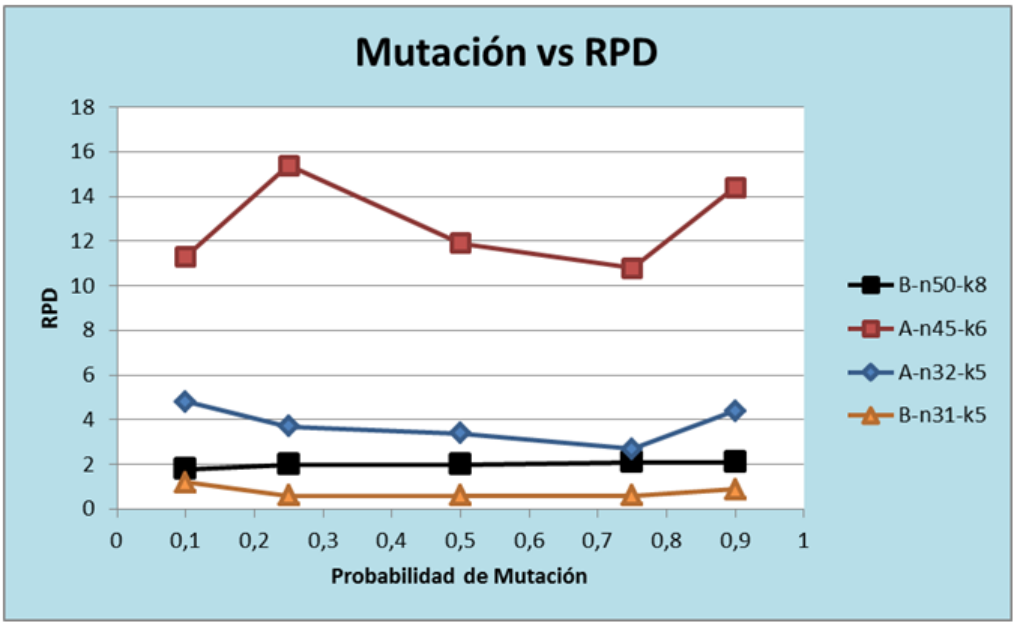

Figura 19: Variación del RPD con respecto a la probabilidad de mutación.

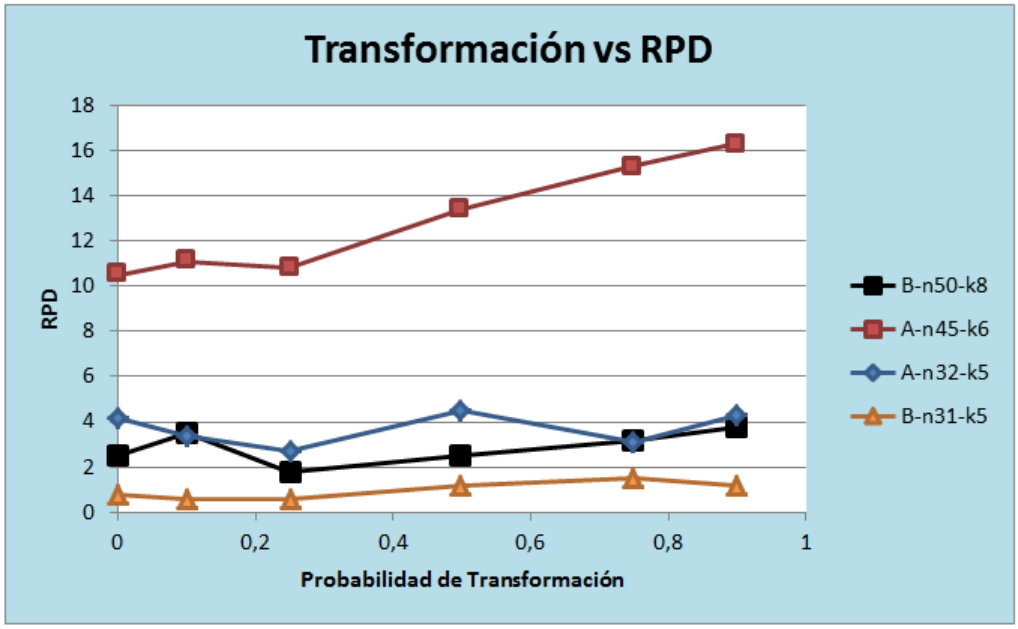

Figura 20: Variación del RPD con respecto a la probabilidad de transformación.

El tamaño de la población demostró mejorar la calidad de las soluciones encontradas al tratar con 50-100 bacterias, y empeorando desde ese punto hacia adelante obteniendo malas soluciones con 200 bacterias, como se observa en el gráfico 21. Sin embargo, el tiempo que demora el algoritmo crece proporcionalmente con el tamaño de la población, por lo que se vuelve necesario intentar utilizar poblaciones pequeñas (**10 bacterias**) siempre que con estas se pueden

alcanzar buenas soluciones (por ejemplo con un RPD alrededor del 2 %). Es por esto, que con el tamaño de la población, se decidió utilizar el más pequeño a pesar de no ser el que entrega las mejores soluciones, y para las instancias donde no se alcancen buenos resultados, se irán realizando pruebas con una mayor población.

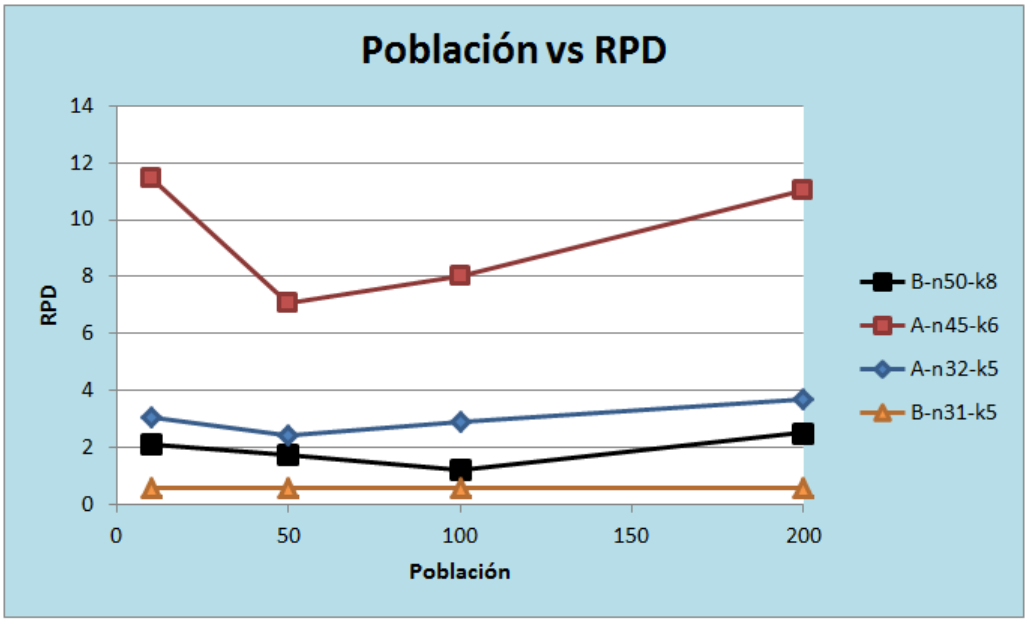

Figura 21: Variación del RPD con respecto al tamaño de la población.

Para el número de iteraciones, como óptimo, se escogió la máxima iteración donde fue encontrada una de las mejores soluciones para las instancias probadas. De lo anterior se escogió como número máximo 1.500.000 iteraciones, que fue donde por última vez disminuyó una de las mejores soluciones encontradas para las instancias, como se ve en el gráfico 22.

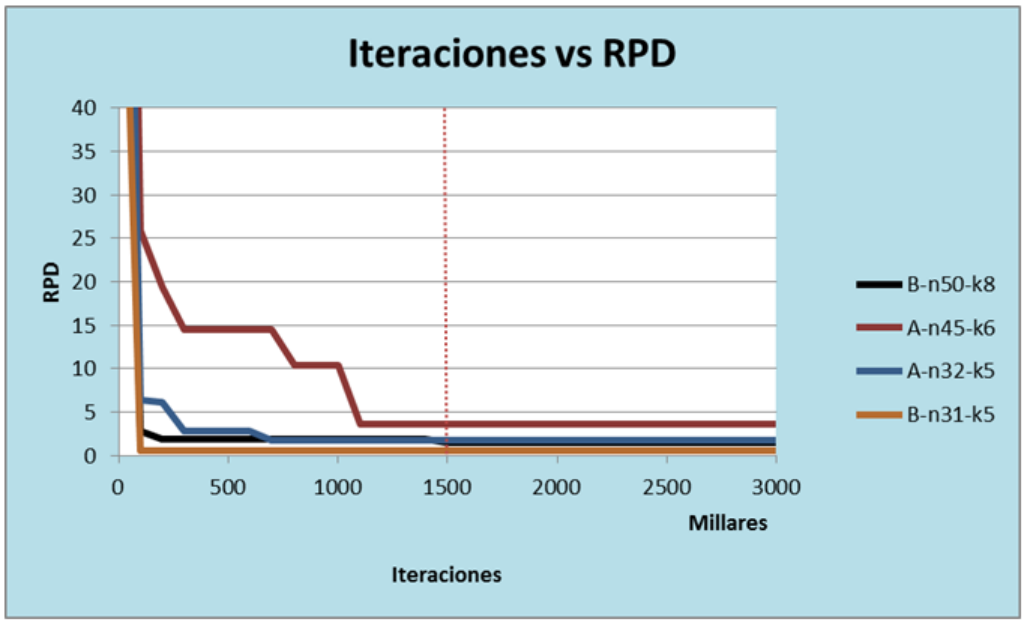

Figura 22: Variación del RPD con respecto al número de iteraciones.

Con los ajustes realizados, finalmente los parámetros quedaron como se muestra en la tabla 2.

Parámetros	
Numero de Iteraciones	1.500.000
Tamaño de la Población	**10 (pudiendo aumentarse donde las soluciones no sean buenas)**
Probabilidad de Conjugación	0.75
Probabilidad de Conjugación Libre	1
Probabilidad de Transformación	0.25
Probabilidad de Transformación Libre	0.5
Probabilidad de Mutación	0.75
Probabilidad de Mutación Libre	1
Probabilidad de Liberación	0.25

Tabla 2: Configuración final de parámetros.

4.3. Análisis y comparaciones de los resultados encontrados por ABRA

Para comparar los resultados obtenidos por ABRA, se utilizaron, por un lado los resultados obtenidos por ACBCP en [2], al ser el trabajo predecesor a la metaheurística acá presentada, y por otro lado, los resultados obtenidos por un Algoritmo Genético (AG) en [40], ya que éste

obtuvo muy buenos resultados y sirve para validar la hipótesis inicial de este trabajo, la cual dice que utilizando bacterias y un enfoque colaborativo se pueden obtener al menos tan buenos resultados como con un AG de enfoque competitivo.

En total, se realizaron **1395 pruebas** (15 para cada instancia con un tamaño de población dado). Desde las pruebas iniciales que utilizaban una población de 10 bacterias, se fue probando con más individuos en las instancias que entregaron malos resultados (lo cual se estimaba principalmente de manera relativa a los resultados obtenidos por AG), aumentando primero a 50 bacterias, y luego a 100 si los resultados seguían siendo insatisfactorios. El total de las pruebas se desglosa en el gráfico de la figura 23 según el tamaño de población utilizado.

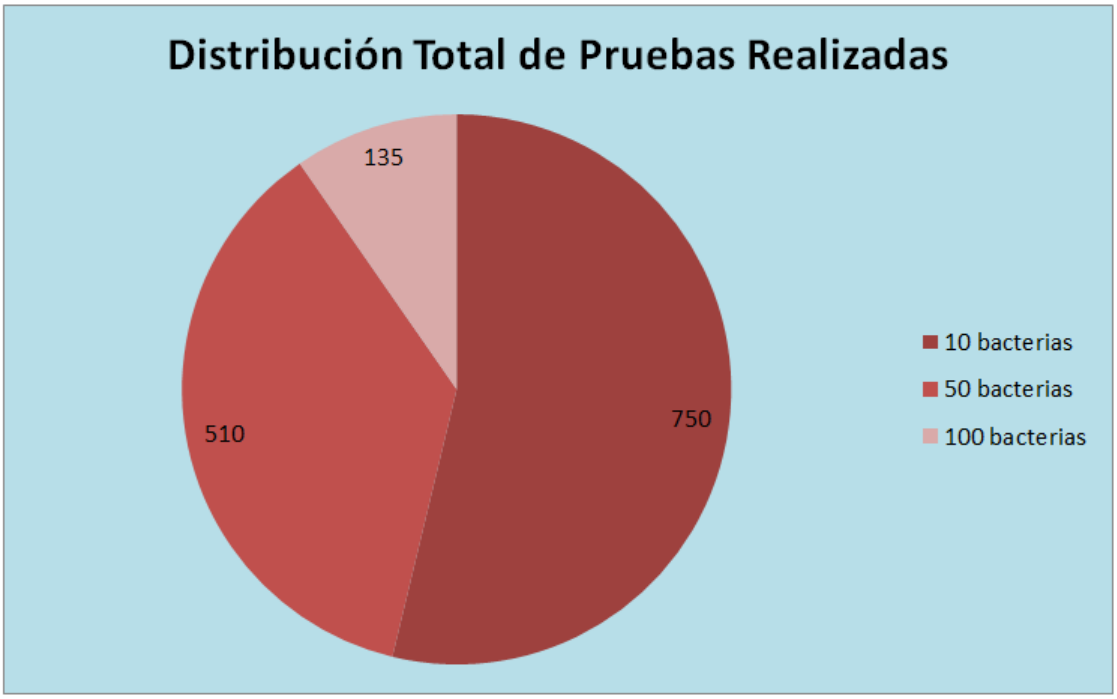

Figura 23: Desglose del total de pruebas según el tamaño de la población.

En la mayoría de los casos se logró el objetivo de mejorar la solución al aumentar la población. Sólo en dos instancias no se logró mejorar el resultado al aumentar la población; una al pasar de 10 a 50 bacterias (para esa misma instancia se obtuvo el mismo resultado entre 10 y 100 bacterias) y otra al pasar de 50 a 100. Del total de las 50 instancias, la distribución de mejores soluciones encontradas según el tamaño de la población utilizado, se ve en el gráfico 24.

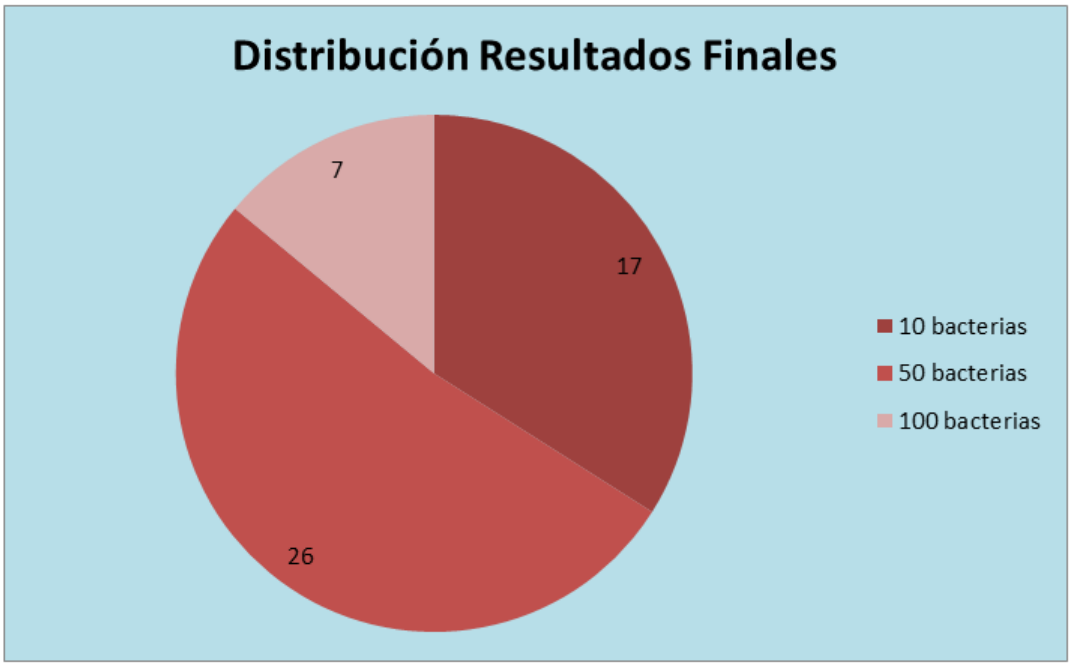

Figura 24: Distribución de soluciones finales según el tamaño de la población.

El RPD más bajo fue de **0,13477089 %** y se logró para la instancia A-n33-k6, y el más alto fue de **5,89765828 %** para la instancia B-n57-k7. Cabe destacar que el RPD promedio del algoritmo para las instancias tipo A y tipo B fue casi idéntico. En las de tipo A se logró un RPD promedio de **1,543013616 %**, y en las de tipo B **1,54795339 %**, dando en conjunto para las 50 instancias un RPD promedio del **1,545285912 %**.

Al observar el desempeño del algoritmo según el número de clientes, se observa que en general (no es estricto) la calidad disminuye al aumentar la cantidad de clientes. Para las instancias tipo A esto se nota a partir de los 48 clientes y para las de tipo B desde los 56. En los gráficos 25 y 26 se observa el RPD por instancias A y B respectivamente, obtenido por ABRA (se cambia la nomenclatura de las instancias por: TCCxV, ver punto 4.1.).

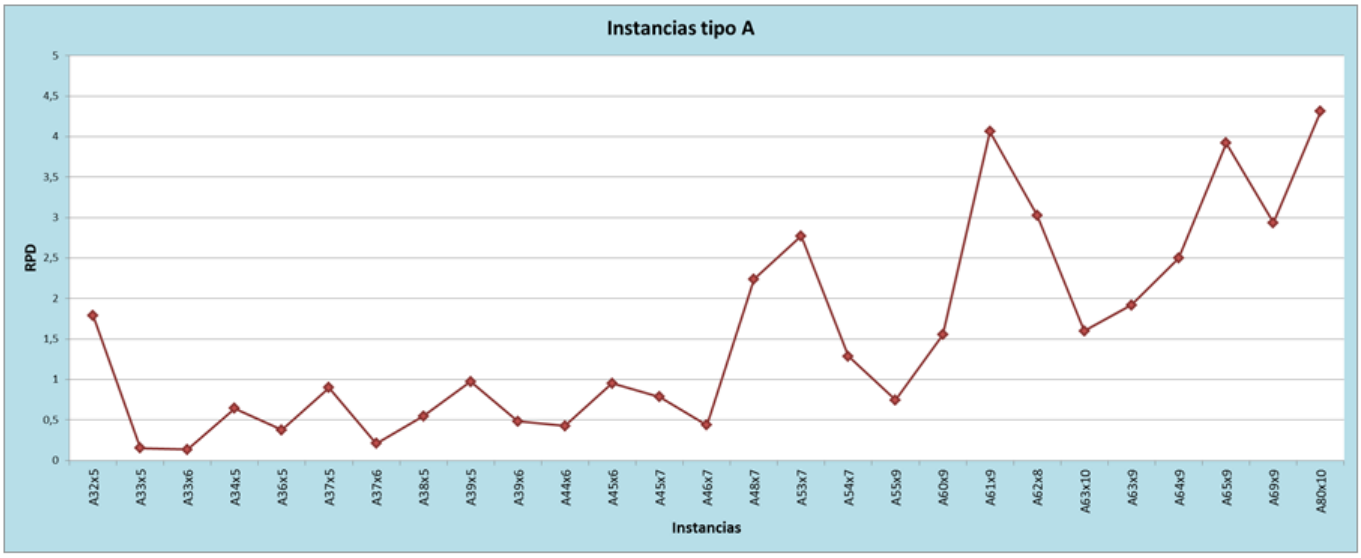

Figura 25: RPD de ABRA para las instancias A.

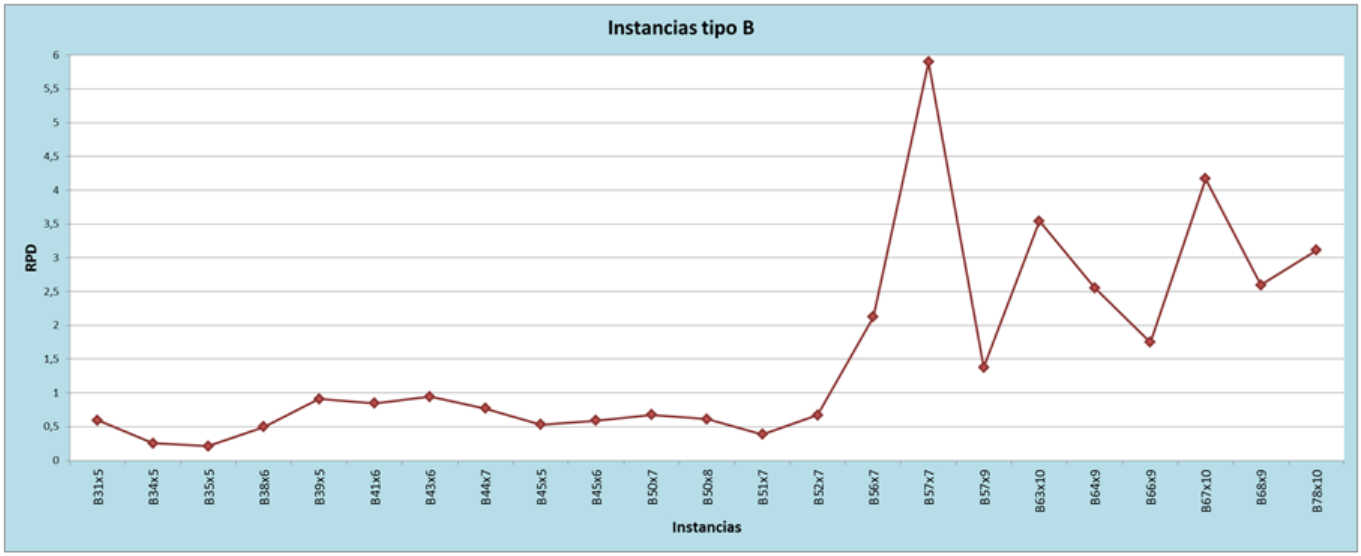

Figura 26: RPD de ABRA para las instancias B.

De las 50 instancias en total (conjuntamente las tipo A y tipo B), se obtuvo que en **28 de ellas se logró un RPD bajo el 1 %**, y sólo en 4 un RPD sobre el 4 %, lo que indica el buen nivel de las soluciones obtenidas (ver gráfico de la figura 27).

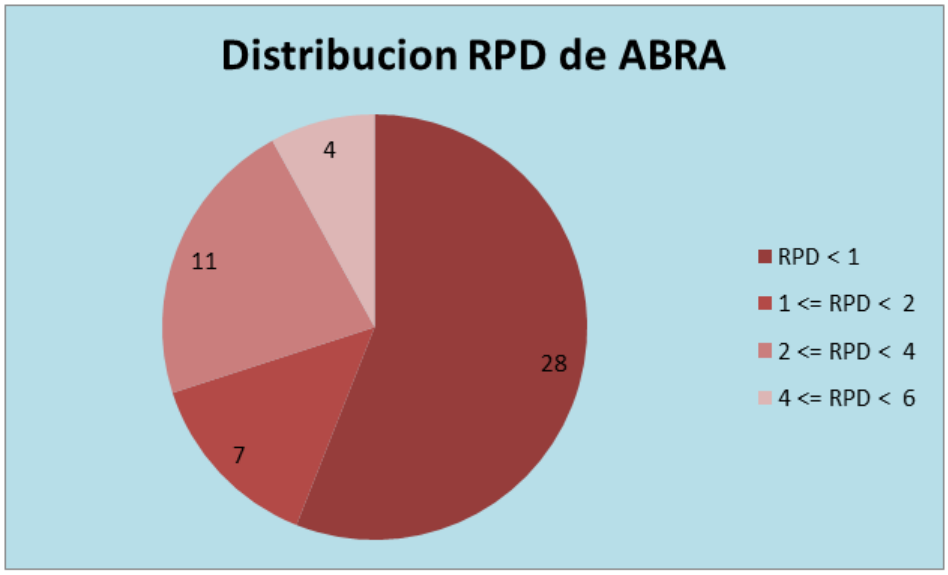

Figura 27: Distribución por segmentos de los RPD obtenidos por ABRA.

En cuanto al tiempo que demoró el algoritmo, la prueba más rápida fue para la instancia A-n32-k5 con 10 bacterias, con un tiempo de **236 segundos**, y la más lenta, la instancia A-n80-k10 con 100 bacterias, con **10821 segundos**.

Al analizar cómo aumenta el tiempo de ejecución del algoritmo, con respecto al número de clientes, se observa que al comparar puntos donde la cantidad de clientes aumenta por 2 veces, o cercano a ello (por ejemplo, 32 con 64, 34 con 68 y 39 con 80 clientes), **el tiempo utilizado aumentó aproximadamente 3.5 veces**. Este comportamiento fue peor de lo que se suponía, ya que se esperaba un crecimiento proporcional del tiempo con respecto al número de clientes. Los gráficos 28 y 29 (para instancias tipo A y B respectivamente) muestran cómo aumenta el tiempo de ejecución del algoritmo con respecto al número de clientes, para un tamaño de población de 10 bacterias (se observa una anomalía en el tiempo de la instancia B-n57-k7, el cual probablemente se deba a alguna alteración en las condiciones de benchmark a la hora de realizar la prueba).

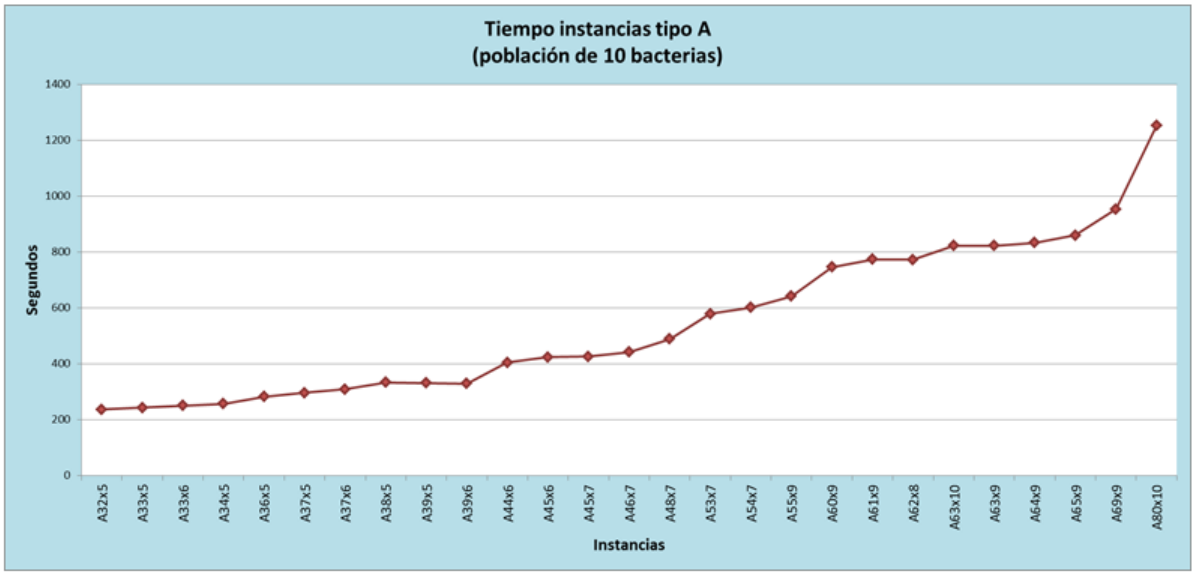

Figura 28: Tiempos de ABRA para las instancias A con población de 10 bacterias.

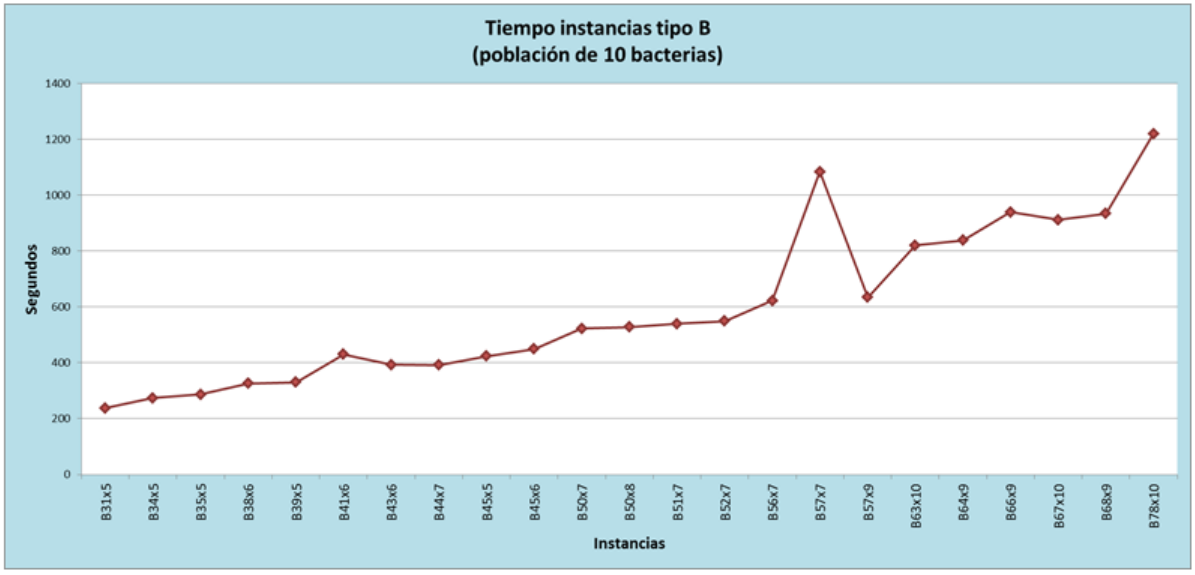

Figura 29: Tiempos de ABRA para las instancias B con población de 10 bacterias.

En cuanto al tiempo con respecto al tamaño de la población, analizando las instancias donde se probó con los 3 tamaños de población, al aumentar 5 veces la población (de 10 a 50) el tiempo aumento aproximadamente **4.4 veces**. Al aumentar 10 veces (de 10 a 100) el aumento fue cercano a **8.3 veces**. Y finalmente, al aumentar al doble (desde 50 a 100), el aumentó en tiempo fue de **1.8 veces** aproximadamente. Este comportamiento es más cercano, e incluso levemente mejor a lo que se esperaba, que era un aumento proporcional del tiempo con respecto al tamaño de la población. El gráfico 30 muestra la comparación de los tiempos para los 3 tamaños de

población, en las instancias en los que fueron probados.

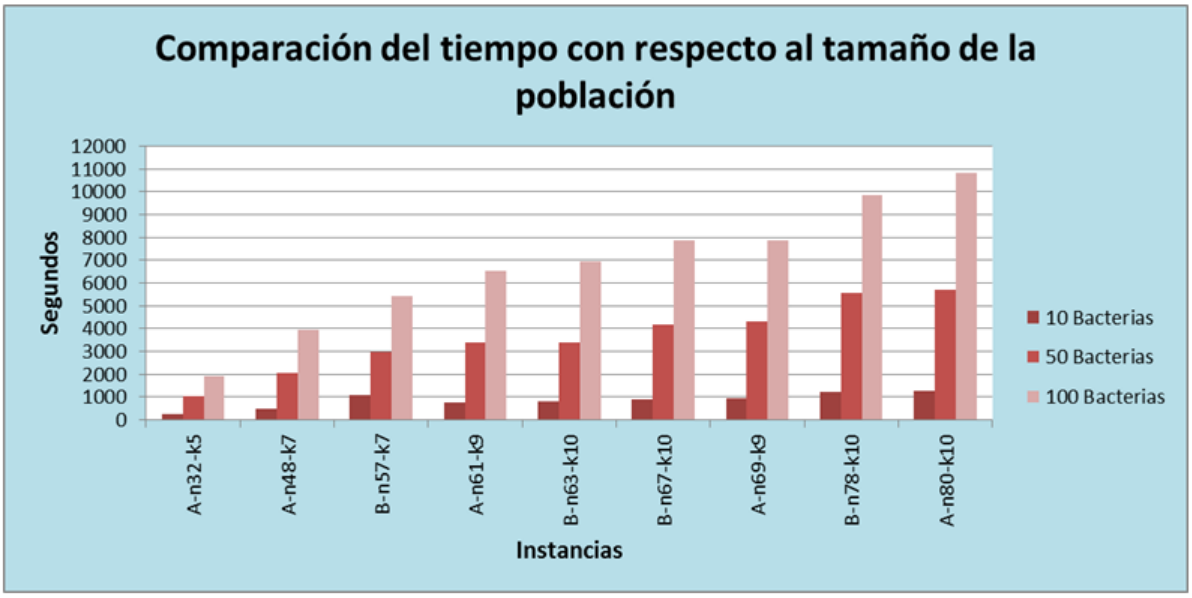

Figura 30: Comparación de los tiempos de ABRA para los distintos tamaños de población.

4.3.1. ABRA comparado con ACBCP

La comparación entre los resultados obtenidos por ABRA y los de ACBCP no resiste análisis. La nueva metaheurística superó categóricamente a su predecesora, al mejorar todos los resultados obtenidos por ésta, logrando mejorar el RPD promedio de todas las instancias en un **7,769934994 %** (para la instancia B-n34-k5 no se disponía del resultado obtenido por ACBCP). Los gráficos 31 y 32 muestran la comparación del RPD por instancias tipo A y B respectivamente.

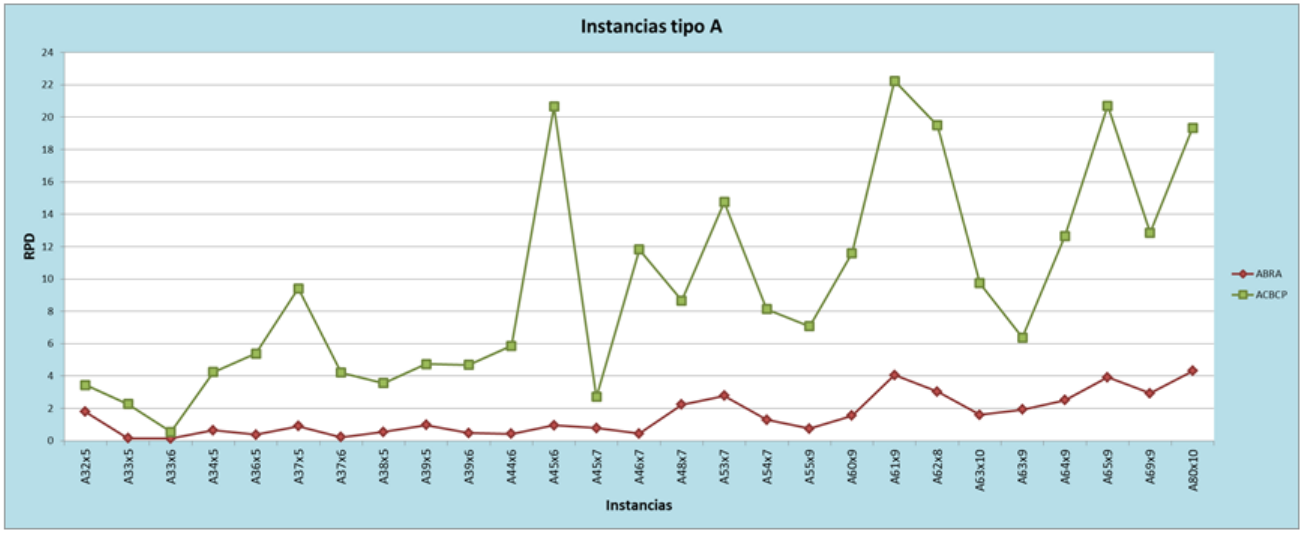

Figura 31: RPD de ABRA vs ACBCP para las instancias A.

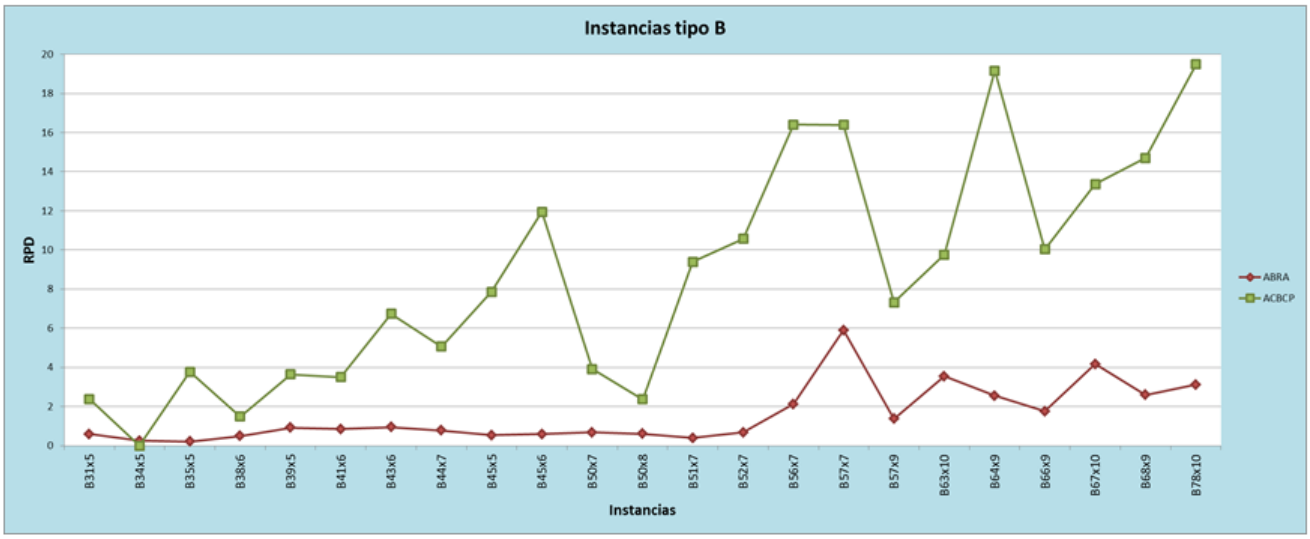

Figura 32: RPD de ABRA vs ACPBCP para las instancias B.

No se disponía de los tiempos de ejecución obtenidos por ACBCP, por lo que comparar ambos algoritmos en ese ítem fue imposible.

4.3.2. ABRA comparado con AG

Entre ABRA y AG las diferencias entre las soluciones obtenidas no fueron tan grandes. Analizando el RPD promedio por tipo de instancias, para las A, ABRA mejoró en un **0,5538068 %** los

resultados obtenidos por AG, para las B, en un **0,22771656 %**, y en el global, un **0,403805287 %**. Un detalle importante al comparar estos resultados, es considerar que la implementación del AG utilizada en [40] no penalizaba el uso adicional de vehículos. Por lo anterior, en 3 instancias tipos A y en 2 tipo B, la mejor solución encontrada por AG utiliza más vehículos que el óptimo de Augerat. Esto nunca ocurre en la implementación de ABRA, ya que en ella sí se penaliza el uso adicional de vehículos.

Debido a la diferencia anteriormente mencionada, AG en 2 instancias B encuentra soluciones las cuales tienen RPD negativo, ya que recorren menos distancias que el óptimo de Augerat, a costo de utilizar vehículos adicionales. En particular, una de las instancias donde ocurre este fenómeno es en B-n57-k7, la cual es la instancia con peor comportamiento para ABRA. Por lo tanto, esa instancia contribuye significativamente a disminuir la diferencia en promedio de RPD entre AG y ABRA para las tipo B, y por lo tanto en el total.

Los gráficos 33 y 34 muestran la comparación entre ABRA y AG para las instancias tipo A y B respectivamente.

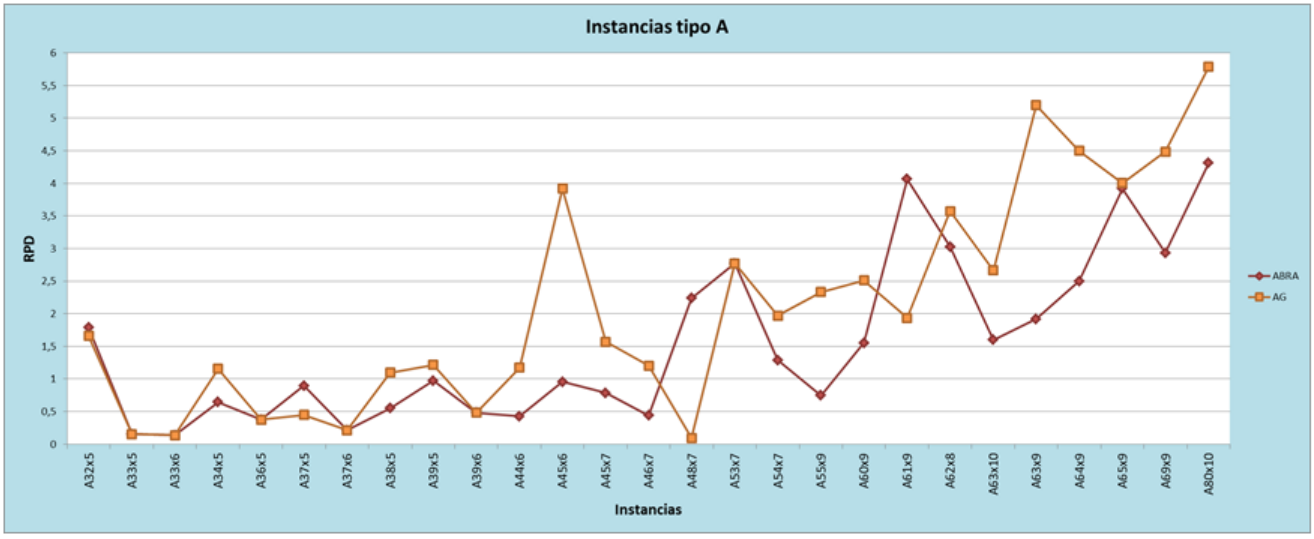

Figura 33: RPD de ABRA vs AG para las instancias A.

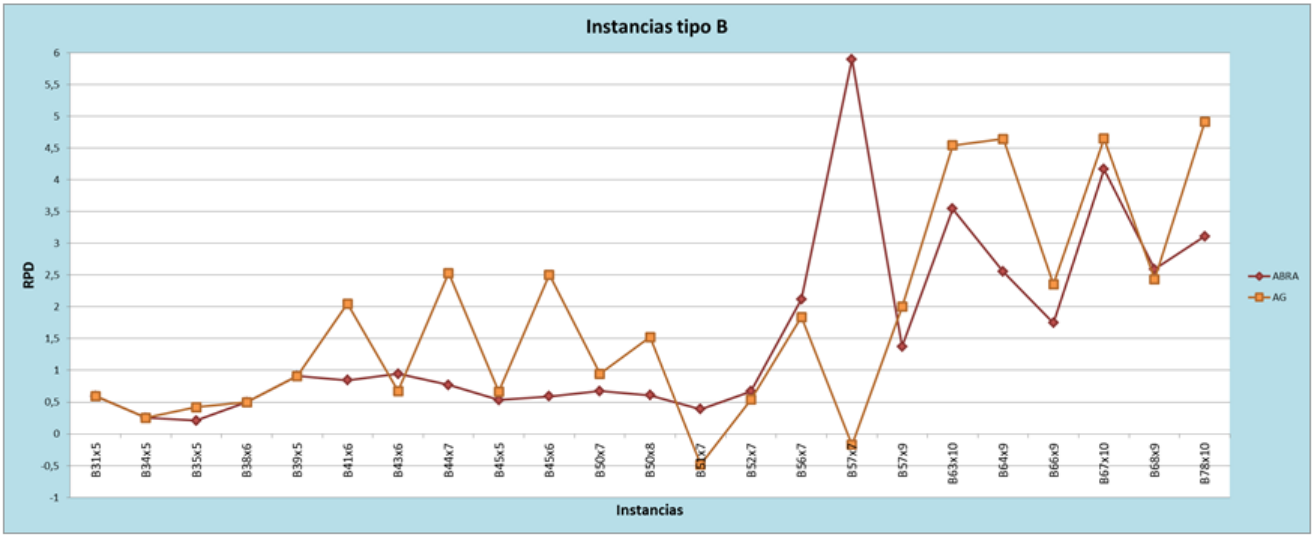

Figura 34: RPD de ABRA vs AG para las instancias B.

Analizando el total de las instancias, se tiene que en **30 de las 50 (17 tipo A y 13 tipo B) ABRA llegó a una mejor solución que AG**, en 10 (6 tipo A y 4 tipo B) ambos lograron la misma solución, y finalmente en 10 (4 tipo A y 6 tipo B) ABRA obtuvo una peor solución que AG, como se aprecia en el gráfico 35.

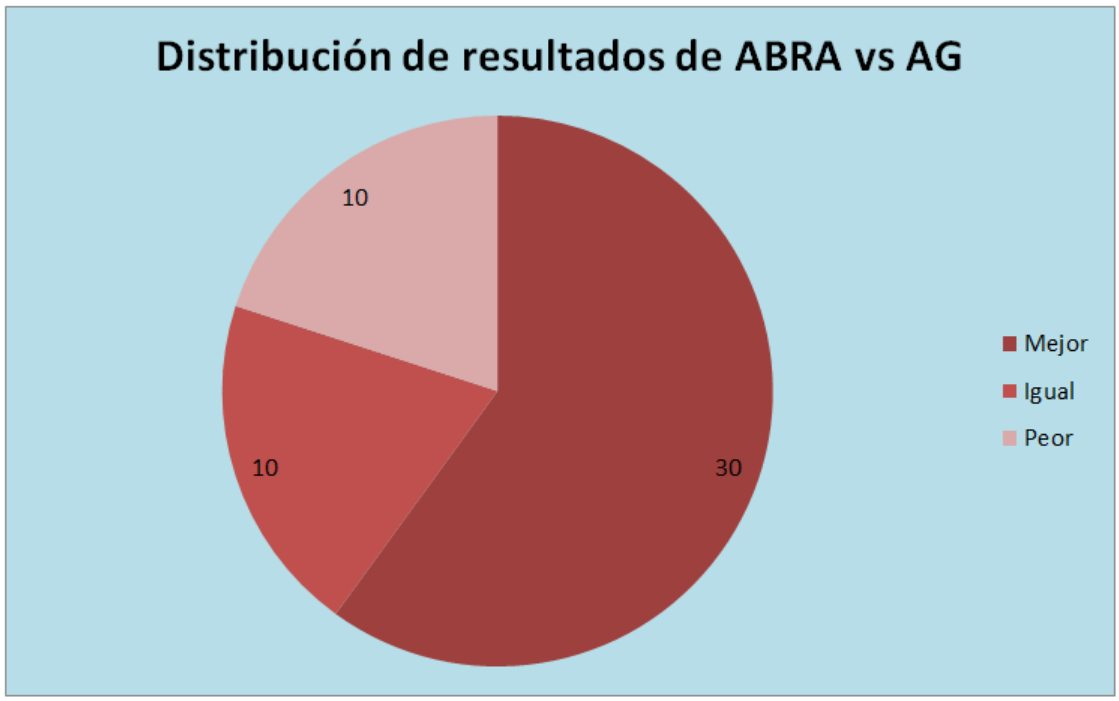

Figura 35: Distribución de resultados mejores, iguales y peores entre ABRA y AG.

Y en cuanto a los segmentos en que caen los RPD de las soluciones obtenidas, **ABRA obtuvo 10 soluciones más que AG con un RPD inferior al 1 % (mejor segmento)**, y 5 soluciones menos con un RPD entre el 4 % y 6 % (peor segmento), como se observa en el gráfico 36.

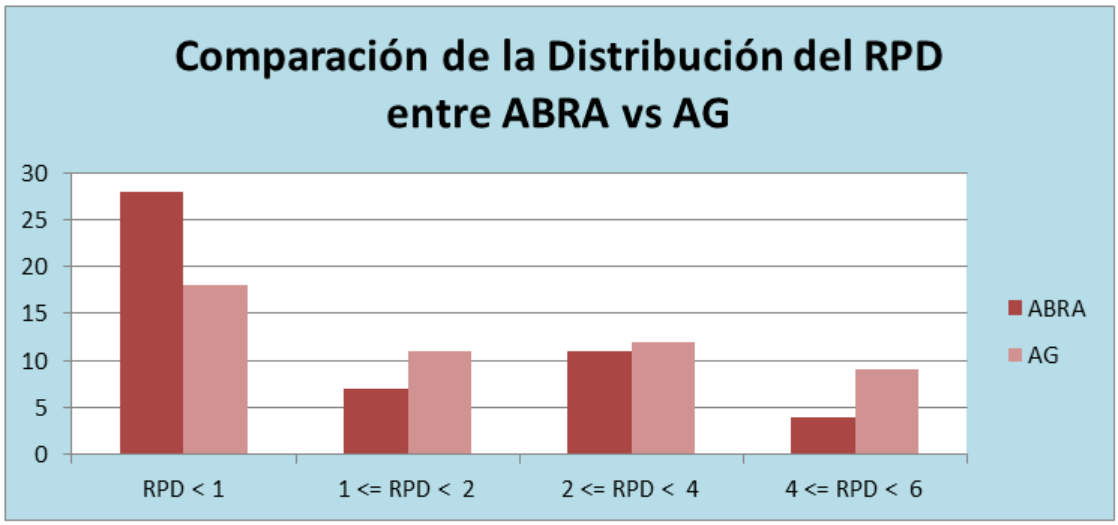

Figura 36: Comparación de la distribución por segmentos de los RPD obtenidos por ABRA y AG.

Al comparar los tiempos con que obtuvieron sus resultados, AG es completamente superior en ese aspecto, ya que para todas sus instancias tardó 60 segundos (debido a que su parámetro de detención era ese tiempo), mientras que ABRA en su instancia mas lenta tardó 180 veces más. No obstante, las condiciones de benchmark no fueron idénticas, ya que en [40] se utilizó un computador con procesador AMD de 2.1 GHz con 4 núcleos y 8GB de RAM, lo cual puede distorsionar las reales diferencias entre los tiempos utilizados por ambos algoritmos. Obviando este último punto, el gráfico 37 muestra la dimensión de las diferencias en el tiempo utilizado por ambos algoritmos, para obtener las mejores soluciones que encontraron para todas las instancias.

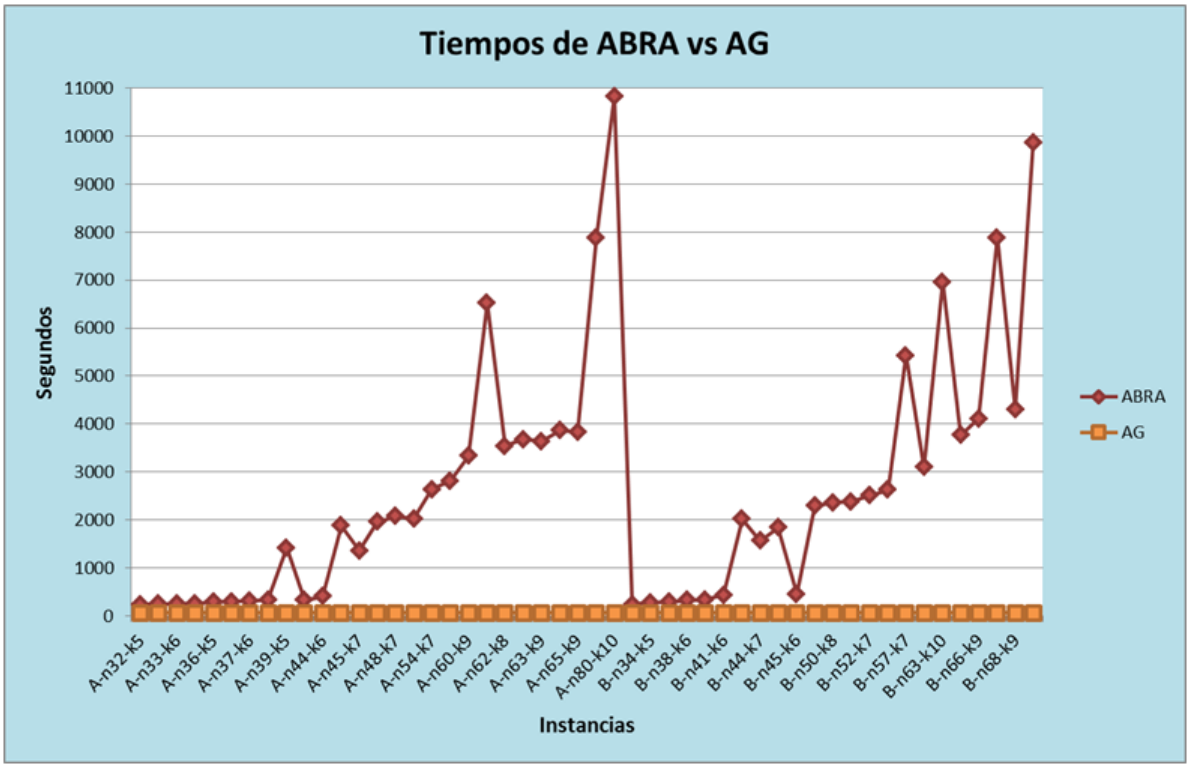

Figura 37: Comparación de los tiempos de ABRA con respecto a los de AG, con los que fueron encontradas las mejores soluciones para todas las instancias.

5. Conclusiones

En este capítulo se presentan las conclusiones finales sobre el trabajo realizado, además de las ideas sobre lo que se puede mejorar en trabajos futuros.

5.1. Conclusiones

En esta memoria de título se propuso una nueva metaheurística llamada Algoritmo Bacteriano de Resistencia Antibiotica (abreviado ABRA), la cual está inspirada en cómo las bacterias pueden sobrevivir a la presencia de antibióticos, mediante el desarrollo de resistencias a partir de sus mecanismos de variación genética. El desarrollo de resistencias y la mayor capacidad de supervivencia de las bacterias, puede verse como un proceso de mejora en el cual se van volviendo cada vez más fuertes. Además, el fortalecimiento de la colonia de bacterias se puede ver como un proceso colaborativo entre las mismas, puesto que son las bacterias resistentes a un determinado antibiótico, las que promueven el desarrollo de resistencia por parte de las no resistentes. Esa es la analogía que toma como base ABRA, en el cual un grupo de soluciones a un problema (representados por una colonia de bacterias), va mejorando iterativamente (desarrollando resistencias), hasta algún criterio de detención. ABRA continúa una línea de metaheurísticas inspiradas en el comportamiento colaborativo de las bacterias, cuyo predecesor era el Algoritmo de Conjugación Bacteriana con Control de Población (ACBCP).

Con respecto a su predecesor, el trasfondo de ABRA se diferencia del de ACBCP en que se quitan conceptos biológicos complejos (por ejemplo, ya no se habla de bacterias F+, F- y Hfr, porcentaje de curación, entre otros términos), y se agregan ideas mas universales, como antibióticos, bacterias resistentes y no resistentes. Esto dota de más simplicidad semántica al modelo, volviendo más sencilla la implementación y la potencial introducción de cambios.

Para probar el funcionamiento de ABRA, se utilizó el problema de Optimización Combinatoria de Ruteo Vehicular con Capacidad Uniforme (CVRP). Los resultados obtenidos por ABRA se compararon con los logrados anteriormente por ACBCP y un Algoritmo Genético (AG), para el mismo conjunto de instancias del CVRP.

Las comparaciones demostraron que ABRA mejoró por mucho la calidad de todas las soluciones obtenidas por ACBCP, sin ser posible realizar una comparación de los tiempos de ejecución, ya que no se contaban con éstos para ACBCP.

Al comparar con los resultados obtenidos por AG, ABRA demostró lograr soluciones tan buenas, e incluso mejores que las logradas por éste. Además, se agrega el extra de que en todas se utilizaban el número óptimo de vehículos posibles, a diferencia de AG. No obstante, para lograr sus soluciones, ABRA fue considerablemente más lento y requirió de mucho más tiempo que el utilizado por AG.

Con lo anterior, se valida la hipótesis inicial de este trabajo, la cual dice que con un enfoque colaborativo se pueden obtener tan buenos resultados como con Algoritmos Genéticos y un enfoque competitivo. Por lo tanto, ABRA puede ser una opción válida, sobre todo si la calidad de la soluciones prima por sobre el tiempo que se tarda en encontrarlas, en el contexto del problema que se desee solucionar.

5.2. Trabajo Futuro

ABRA a pesar de ser la continuación de dos trabajos previos (ACB y ACBCP) y tomar sus bases, optó por un enfoque totalmente nuevo a la hora de utilizar las bacterias para solucionar un problema. Es por ello que ABRA se puede considerar el primer algoritmo en su línea, al utilizar la inspiración de bacterias, antibióticos y resistencia. Por lo tanto, ésta es la propuesta inicial de un modelo que quizás puede llegar a dar mucho más de sí.

Resultaría interesante probar nuevas formas de seleccionar un Antibiótico en cada iteración, ya que en este trabajo se optó por una en la cual se privilegió la simpleza.

También, se puede estudiar la Transformación como medio de variación genética, ya que en este trabajo sólo se utilizó como un nuevo operador para explorar el espacio de soluciones. Si con conocimiento heurístico del problema en particular, se pueden llegar a seleccionar trozos del cromosoma de una bacteria los cuales realmente sean potencialmente buenos, en vez de realizar este proceso de manera azarosa, la transformación podría llegar a cooperar activamente en el mejoramiento de los individuos no resistentes, en vez de solo transformarlos aleatoriamente.

Otro punto de interés, sería ver si mejorando la implementación de ABRA realizada en este trabajo, se pueden mejorar los tiempos de ejecución que acá se obtuvieron, o si estos están directamente ligados al modelo de la metaheurística.

Finalmente, y a mi parecer, el trabajo futuro más importante, sería validar el funcionamiento de ABRA con distintos problemas de optimización combinatoria, comparando sus resultados con los métodos vigentes. De esta manera, se podría llegar a decir con más argumentos que ABRA, es una opción válida a la hora de seleccionar un método para la resolución de problemas de optimización combinatoria.

Bibliografía

[1] Fernández H., *Memoria de Título: Metaheurística basada en la conjugación bacteriana*, Universidad de Concepción, 2014.

[2] Ortiz V., *Memoria de Título: Metaheurística basada en el comportamiento colaborativo de microorganismos celulares, para la solución de problemas de optimización combinatoria*, Universidad de Concepción, 2014.

[3] Larry Snyder, Wendy Champness, *Molecular Genetics of Bacteria, Third Edition*, ASM Press, 2007.

[4] Fredrickson J., Zachara J., Balkwill D., et al, *Geomicrobiology of high-level nuclear waste-contaminated vadose sediments at the hanford site, Washington state*, Applied and Environmental Microbiology Vol.70, No.7, 2004.

[5] Betancor L., Gadea M., Flores K., *Genética Bacteriana*, Documento de la Facultad de Medicina de la Universidad de la República de Uruguay, 2009. Disponible en: `http://www.higiene.edu.uy/cefa/2008/GeneticaBacteriana.pdf`, Accedido el 03 de Agosto del 2015.

[6] Jeremy W. Dale, Simon F. Park, *Molecular Genetics of Bacteria, Fourth Edition*, Wiley, 2005.

[7] Cornelis GR, Boland A, Boyd AP, Geuijen C, Iriarte M, Neyt C, Sory MP, Stainier I., *The virulence plasmid of Yersinia, an antihost genome*, Microbiology and Molecular Biology Reviews Vol.62 No.4, 1998.

[8] E. M. Johnson, J. A. Wohlhieter, Bruce P. Placek, R. B. Sleet, L. S. Baron, *Plasmid-Determined Ability of a Salmonella tennessee Strain to Ferment Lactose and Sucrose*, Journal of Bacteriology Vol.125 No.1, 1976.

[9] Bassler BL., *How bacteria talk to each other: regulation of gene expression by quorum sensing*, Current Opinion in Microbiology Vol.2 No.6, 1999.

[10] Smith HO, Danner DB, Deich RA., *Genetic transformation*, Annual Review of Biochemistry Vol.50 41-68, 1981.

[11] Sparling PF., *Genetic transformation of Neisseria gonorrhoeae to streptomycin resistance*, Journal of Bacteriology Vol.92 No.5, 1966.

[12] Kenneth Todar, *Bacterial Resistance to Antibiotics*, Todar's Online Textbook of Bacteriology. Disponible en: `http://textbookofbacteriology.net/resantimicrobial.html`, Accedido el 08 de Octubre del 2015.

[13] Christos H. Papadimitriou, Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, 2000.

[14] Korte Bernhard, Vygen Jens, *Combinatorial Optimization: Theory and Algorithms, Second Edition*, Springer, 2002.

[15] Cruz Chavez Marco, Moreno Bernal Pedro, Peralta Abarca Jesús del Carmen, *Aplicacion de la teoria de la complejidad en optimizacion combinatoria*. Disponible en: `http://dialnet.unirioja.es/descarga/articulo/4733827.pdf`, Accedido el 10 de Agosto del 2015.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms, Third Edition*, The MIT Press, 2009.

[17] Gilbert Laporte, *The vehicle routing problem: An overview of exact and approximate algorithms*, European Journal of Operational Research Vol.59 No.3, 1992.

[18] Paolo Toth, Daniele Vigo, *The Vehicle Routing Problem*, SIAM, 2001.

[19] Weisstein Eric W, *Bin-Packing Problem* From MathWorld–A Wolfram Web Resource. Disponible en: `http://mathworld.wolfram.com/Bin-PackingProblem.html`, Accedido el 14 de Agosto del 2015.

[20] *ShipNorthAmerica Transportation*. Disponible en: `http://www.shipnorthamerica.com/htmfiles/glossary/gloss_shipterms_back.html`, Accedido el 14 de Agosto del 2015.

[21] Belarmino Díaz, *Optimización heurística y redes neuronales en dirección de operaciones e Ingeniería*, Paraninfo, 1996.

[22] Melián B, Moreno Pérez JA, Moreno Vega JM, *Metaheurísticas: una visión global*, Revista Iberoamericana de Inteligencia Artificial Vol.7 No.19, 2003.

[23] Gerhard J. Woeginger, *Exact Algorithms for NP-Hard Problems: A Survey*, Lecture Notes in Computer Science Vol.2570, 2003.

[24] Rafael Martí, *Procedimientos Metaheurísticos en Optimización Combinatoria*, 2003. Disponible en: `http://www.uv.es/rmarti/paper/docs/heur1.pdf`, Accedido el 31 de Agosto del 2015.

[25] Jens Lysgaard, *Clarke & Wright's Savings Algorithm*, 1997. Disponible en: `http://pure.au.dk/portal-asb-student/files/36025757/Bilag_E_SAVINGSNOTE.pdf`, Accedido el 31 de Agosto del 2015.

[26] R. H. Mole, S. R. Jameson, *A Sequential Route-Building Algorithm Employing a Generalised Savings Criterion*, Operational Research Quarterly Vol.27 No.2, 1976.

[27] Marshall L. Fisher, Ramchandran Jaikumar, *A generalized assignment heuristic for vehicle routing*, Networks Vol.11 No.2, 1981.

[28] El-Ghazali Talbi, *Metaheuristics: From Design to Implementation*, Wiley, 2009.

[29] Vélez MC, Montoya JA, *Metaheurísticos: Una alternativa para la solución de problemas combinatorios en administración de operaciones*, Revista EIA Vol.4 No.8, 2007.

[30] Felipe Diaz, Alberto Reyes, *Aceros, estructuras y tratamientos térmicos*, 2012. Disponible en: `http://olimpia.cuautitlan2.unam.mx/pagina_ingenieria/mecanica/mat/mat_mec/m6/`, Accedido el 02 de Septiembre del 2015.

[31] Talib S. Hussain, *An Introduction to Evolutionary Computation*. Disponible en: `http://neo.lcc.uma.es/cEA-web/documents/hussain98.pdf`, Accedido el 11 de Septiembre del 2015.

[32] Siamak Sarmady, *An Investigation on Genetic Algorithm Parameters*. Disponible en: `http://sarmady.com/siamak/papers/genetic-algorithm.pdf`, Accedido el 11 de Septiembre del 2015.

[33] Noraini Mohd Razali, John Geraghty, *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*, Proceedings of the World Congress on Engineering Vol.1, 2011.

[34] Daniel W. Dyer, *Evolutionary Computation in Java. A Practical Guide to the Watchmaker Framework*, 2010. Disponible en: `http://watchmaker.uncommons.org/manual/index.html`, Accedido el 11 de Septiembre del 2015.

[35] Marek Obitko, *Introduction to Genetic Algorithms*. Disponible en: `https://courses.cs.washington.edu/courses/cse473/06sp/GeneticAlgDemo/`, Accedido el 11 de Septiembre del 2015.

[36] Analía Amandi, Virginia Yannibelli, *Introducción a la Computación Evolutiva*. Disponible en: `http://users.exa.unicen.edu.ar/~icompevol/filminasenpdf/terceraclase.pdf`, Accedido el 11 de Septiembre del 2015.

[37] F. Vavak, T. C. Fogarty, *A comparative study of steady state and generational genetic algorithms for use in nonstationary environments*, Lecture Notes in Computer Science Vol.1143, 2005.

[38] Z. Michalewicz, D. B. Fogel, *How to solve it: Modern Heuristics*, Springer, 1998.

[39] Marco Dorigo, Luca Maria Gambardella, *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*, IEEE Transactions on Evolutionary Computation Vol.1 No.1, 1997.

[40] Pantoja C., *Memoria de Título: Resolución del Problema de Enrutamiento Vehicular con Capacidad Homogénea utilizando Algoritmos Genéticos*, Universidad de Concepción, 2013.

A. Apéndice

En este apéndice se adjuntan los resultados mas importantes obtenidos en las pruebas, junto con las instrucciones para ejecutar el programa que fue creado para implementar ABRA.

A.1. Tablas de resultados y comparativas

A continuación se muestran los resultados y tablas comparativas de las pruebas más importantes realizadas en este trabajo.

A.1.1. Mejores resultados según el tamaño de la población de bacterias.

En rojo la que fue utilizada como solución final.

Instancia	Óptimo	Población	Fitness	RPD	Tiempo (s)
A-n32-k5	784	**10**	**798**	**1,78571429**	**236**
		50	803	2,42346939	1037
		100	798	1,78571429	1938
A-n33-k5	661	**10**	**662**	**0,15128593**	**242**
A-n33-k6	742	**10**	**743**	**0,13477089**	**250**
A-n34-k5	778	**10**	**783**	**0,64267352**	**256**
A-n36-k5	799	**10**	**802**	**0,37546934**	**282**
A-n37-k5	669	**10**	**675**	**0,89686099**	**296**
A-n37-k6	949	**10**	**951**	**0,21074816**	**308**
A-n38-k5	730	**10**	**734**	**0,54794521**	**333**
A-n39-k5	822	10	834	1,45985401	331
		50	**830**	**0,97323601**	**1415**
A-n39-k6	831	**10**	**835**	**0,48134777**	**329**
A-n44-k6	937	**10**	**941**	**0,42689434**	**404**
A-n45-k6	944	10	978	3,60169492	423
		50	**953**	**0,95338983**	**1876**
A-n45-k7	1146	10	1162	1,39616056	426
		50	**1155**	**0,78534031**	**1360**
A-n46-k7	914	10	926	1,31291028	441
		50	**918**	**0,43763676**	**1957**
A-n48-k7	1073	10	1122	4,5666356	488
		50	**1097**	**2,23671948**	**1957**
		100	1106	3,07548928	3943
A-n53-k7	1010	10	1050	3,96039604	579
		50	**1038**	**2,77227723**	**2023**
A-n54-k7	1167	10	1214	4,02742074	601
		50	**1182**	**1,28534704**	**2631**
A-n55-k9	1073	10	1092	1,77073625	642
		50	**1081**	**0,74557316**	**2809**
A-n60-k9	1354	10	1400	3,39734121	746
		50	**1375**	**1,55096012**	**3336**
A-n61-k9	1034	10	1162	12,3791103	774
		50	1113	7,64023211	3399
		100	**1076**	**4,06189555**	**6520**
A-n62-k8	1288	10	1359	5,51242236	773
		50	**1327**	**3,02795031**	**3539**
A-n63-k10	1314	10	1348	2,58751903	822
		50	**1335**	**1,59817352**	**3676**
A-n63-k9	1616	10	1696	4,95049505	822
		50	**1647**	**1,91831683**	**3634**
A-n64-k9	1401	10	1475	5,28194147	833
		50	**1436**	**2,49821556**	**3866**
A-n65-k9	1174	10	1257	7,06984668	860
		50	**1220**	**3,91822828**	**3837**

A-n69-k9	1159	10	1225	5,69456428	953
		50	1208	4,22778257	4295
		100	**1193**	**2,93356342**	**7883**
A-n80-k10	1763	10	1900	7,77084515	1252
		50	1841	4,4242768	5682
		100	**1839**	**4,31083381**	**10821**

Tabla 3: Fitness, RPD y tiempo de las mejores soluciones por tamaño de población para las instancias tipo A.

Instancia	Óptimo	Población	Fitness	RPD	Tiempo (s)
B-n31-k5	672	10	**676**	**0,5952381**	**238**
B-n34-k5	788	10	**790**	**0,25380711**	**274**
B-n35-k5	955	10	**957**	**0,20942408**	**287**
B-n38-k6	805	10	**809**	**0,49689441**	**326**
B-n39-k5	549	10	**554**	**0,91074681**	**330**
B-n41-k6	829	10	**836**	**0,84439083**	**430**
B-n43-k6	742	10	751	1,21293801	393
		50	**749**	**0,94339623**	**2018**
B-n44-k7	909	10	921	1,32013201	392
		50	**916**	**0,77007701**	**1562**
B-n45-k5	751	10	759	1,06524634	424
		50	**755**	**0,53262317**	**1835**
B-n45-k6	678	10	**682**	**0,5899705**	**449**
B-n50-k7	741	10	751	1,34952767	523
		50	**746**	**0,67476383**	**2286**
B-n50-k8	1312	10	1332	1,52439024	528
		50	**1320**	**0,6097561**	**2360**
B-n51-k7	1032	10	1049	1,64728682	539
		50	**1036**	**0,3875969**	**2383**
B-n52-k7	747	10	759	1,6064257	549
		50	**752**	**0,66934404**	**2506**
B-n56-k7	707	10	735	3,96039604	622
		50	**722**	**2,12164074**	**2628**
B-n57-k7	1153	10	1290	11,8820468	1084
		50	1236	7,19861232	2972
		100	**1221**	**5,89765828**	**5426**
B-n57-k9	1598	10	1646	3,00375469	634
		50	**1620**	**1,3767209**	**3099**
B-n63-k10	1496	10	1582	5,7486631	820
		50	1563	4,47860963	3394
		100	**1549**	**3,54278075**	**6956**
B-n64-k9	861	10	915	6,271777	838
		50	**883**	**2,55516841**	**3764**
B-n66-k9	1316	10	1348	2,43161094	939
		50	**1339**	**1,74772036**	**4105**
B-n67-k10	1032	10	1097	6,29844961	912
		50	1077	4,36046512	4181
		100	**1075**	**4,16666667**	**7880**
B-n68-k9	1272	10	1326	4,24528302	934
		50	**1305**	**2,59433962**	**4303**
B-n78-k10	1221	10	1317	7,86240786	1220
		50	1298	6,30630631	5569
		100	**1259**	**3,11220311**	**9862**

Tabla 4: Fitness, RPD y tiempo de las mejores soluciones por tamaño de población para las instancias tipo B.

A.1.2. Rutas de las mejores soluciones encontradas.

Cada par de corchetes corresponde a la ruta tomada por un vehículo, donde el depósito está representado con el número 0.

Instancias	Rutas
A-n32-k5	[0, 29, 22, 9, 15, 10, 25, 5, 20, 0] - [0, 6, 17, 19, 31, 21, 13, 26, 0] - [0, 30, 16, 7, 1, 12, 0] – [0, 2, 3, 23, 28, 4, 11, 8, 18, 14, 0] - [0, 24, 27, 0]
A-n33-k5	[0, 29, 16, 3, 9, 17, 15, 0] - [0, 10, 30, 25, 27, 12, 4, 0] - [0, 2, 32, 13, 8, 7, 26, 5, 20, 0] – [0, 24, 6, 19, 14, 21, 1, 31, 11, 0] - [0, 23, 18, 28, 22, 0]
A-n33-k6	[0, 28, 27, 30, 16, 25, 32, 0] - [0, 21, 12, 10, 0] - [0, 17, 11, 29, 19, 7, 0] – [0, 5, 4, 8, 3, 2, 15, 9, 20, 0] - [0, 13, 6, 18, 1, 14, 0] - [0, 22, 26, 24, 23, 31, 0]
A-n34-k5	[0, 7, 6, 15, 8, 29, 14, 0] - [0, 30, 24, 5, 26, 4, 0] - [0, 1, 27, 23, 11, 19, 17, 10, 0] – [0, 18, 2, 9, 12, 3, 22, 16, 33, 0] - [0, 13, 25, 31, 28, 32, 21, 20, 0]
A-n36-k5	[0, 20, 5, 25, 27, 24, 11, 16, 0] - [0, 12, 31, 19, 4, 3, 6, 9, 0] – [0, 21, 22, 32, 18, 33, 29, 30, 17, 13, 1, 0] - [0, 15, 8, 35, 2, 23, 34, 14, 28, 0] - [0, 10, 7, 26, 0]
A-n37-k5	[0, 7, 4, 33, 5, 6, 10, 13, 22, 0] - [0, 1, 12, 2, 19, 20, 23, 14, 17, 0] – [0, 15, 3, 24, 9, 11, 27, 8, 25, 35, 18, 26, 0] - [0, 30, 31, 28, 32, 29, 36, 34, 0] - [0, 16, 21, 0]
A-n37-k6	[0, 4, 26, 21, 9, 19, 31, 0] - [0, 18, 17, 34, 1, 3, 5, 8, 20, 0] – [0, 10, 11, 12, 22, 23, 28, 2, 33, 0] – [0, 24, 29, 36, 6, 14, 0] - [0, 13, 30, 15, 32, 27, 0] - [0, 16, 35, 25, 7, 0]
A-n38-k5	[0, 20, 37, 31, 28, 34, 19, 18, 0] - [0, 9, 17, 36, 13, 15, 2, 14, 24, 0] – [0, 21, 10, 30, 23, 35, 33, 8, 0] – [0, 29, 6, 25, 16, 4, 1, 3, 12, 26, 0] - [0, 32, 22, 27, 11, 5, 7, 0]
A-n39-k5	[0, 14, 19, 25, 33, 12, 18, 4, 0] - [0, 8, 11, 26, 34, 37, 35, 24, 17, 0] – [0, 2, 22, 3, 7, 16, 5, 1, 0] – [0, 31, 23, 20, 29, 32, 10, 15, 38, 9, 0] - [0, 6, 36, 27, 28, 13, 30, 21, 0]
A-n39-k6	[0, 14, 25, 35, 31, 37, 38, 12, 3, 0] - [0, 11, 36, 17, 23, 21, 1, 6, 0] – [0, 18, 22, 34, 27, 16, 10, 32, 20, 0] - [0, 13, 15, 0] - [0, 2, 33, 19, 4, 8, 7, 30, 0] – [0, 24, 29, 28, 9, 5, 26, 0]
A-n44-k6	[0, 29, 43, 40, 23, 30, 17, 0] - [0, 1, 35, 18, 20, 16, 11, 10, 26, 0] – [0, 33, 37, 42, 32, 21, 5, 24, 19, 0] – [0, 31, 15, 27, 28, 7, 8, 0] - [0, 4, 34, 39, 12, 3, 25, 6, 0] - [0, 22, 36, 9, 38, 13, 14, 41, 2, 0]
A-n45-k6	[0, 37, 19, 30, 40, 11, 17, 18, 0] - [0, 15, 25, 2, 38, 31, 35, 14, 0] – [0, 32, 20, 5, 21, 33, 41, 8, 10, 3, 9, 0] - [0, 26, 27, 34, 24, 6, 44, 1, 0] – [0, 22, 16, 4, 42, 36, 39, 12, 0] - [0, 23, 28, 7, 13, 43, 29, 0]
A-n45-k7	[0, 10, 15, 25, 23, 17, 38, 0] - [0, 27, 41, 29, 19, 36, 31, 0] - [0, 32, 13, 18, 7, 16, 20, 40, 0] – [0, 22, 5, 30, 37, 1, 42, 8, 0] - [0, 9, 14, 33, 44, 24, 21, 0] - [0, 39, 35, 34, 26, 2, 12, 0] – [0, 4, 6, 28, 3, 43, 11, 0]
A-n46-k7	[0, 37, 2, 30, 4, 38, 0] - [0, 44, 25, 18, 35, 12, 11, 0] – [0, 21, 29, 31, 34, 40, 1, 27, 20, 41, 28, 0] – [0, 32, 3, 22, 10, 7, 39, 5, 36, 0] - [0, 23, 14, 26, 13, 43, 6, 45, 0] – [0, 19, 15, 24, 16, 42, 33, 8, 0] – [0, 9, 17, 0]
A-n48-k7	[0, 7, 3, 20, 26, 39, 8, 15, 27, 0] - [0, 45, 42, 11, 4, 13, 46, 30, 21, 28, 0] – [0, 14, 17, 47, 16, 40, 36, 0] - [0, 6, 22, 38, 19, 25, 37, 32, 0] - [0, 23, 43, 31, 1, 5, 12, 0] – [0, 41, 2, 10, 33, 29, 24, 9, 34, 0] - [0, 44, 35, 18, 0]
A-n53-k7	[0, 39, 3, 5, 14, 13, 34, 25, 0] - [0, 45, 42, 48, 12, 47, 0] - [0, 21, 52, 24, 11, 41, 17, 9, 0] – [0, 7, 16, 32, 15, 19, 23, 43, 50, 36, 2, 37, 22, 4, 0] - [0, 51, 27, 8, 46, 28, 0] – [0, 38, 18, 40, 26, 10, 49, 29, 44, 30, 1, 0] - [0, 33, 20, 6, 35, 31, 0]
A-n54-k7	[0, 38, 9, 33, 21, 45, 26, 10, 35, 0] - [0, 32, 37, 48, 40, 31, 8, 19, 11, 13, 0] – [0, 20, 49, 39, 50, 7, 5, 18, 28, 4, 0] - [0, 53, 22, 3, 44, 0] – [0, 30, 25, 47, 51, 24, 42, 46, 41, 34, 52, 0] - [0, 16, 6, 27, 12, 2, 14, 0] – [0, 15, 17, 1, 36, 29, 23, 43, 0]

A-n55-k9	[0, 11, 15, 51, 2, 33, 0] - [0, 43, 35, 9, 49, 39, 47, 0] - [0, 6, 45, 1, 0] – [0, 32, 38, 16, 40, 53, 5, 10, 12, 0] - [0, 4, 7, 42, 31, 20, 26, 0] – [0, 37, 14, 46, 17, 34, 3, 36, 0] – [0, 21, 8, 29, 41, 25, 0] - [0, 28, 54, 13, 27, 19, 22, 30, 0] - [0, 23, 52, 24, 44, 50, 48, 18, 0]
A-n60-k9	[0, 34, 24, 23, 47, 14, 0] - [0, 25, 40, 21, 4, 11, 3, 20, 16, 0] – [0, 7, 29, 37, 57, 17, 15, 19, 18, 0] – [0, 52, 59, 38, 33, 41, 0] - [0, 48, 22, 10, 36, 1, 2, 0] - [0, 56, 32, 9, 51, 12, 43, 50, 0] – [0, 54, 5, 45, 42, 58, 0] - [0, 6, 28, 44, 49, 30, 53, 31, 46, 0] - [0, 13, 8, 27, 26, 39, 55, 35, 0]
A-n61-k9	[0, 14, 51, 34, 16, 1, 44, 0] - [0, 43, 32, 26, 4, 22, 12, 13, 0] - [0, 24, 53, 41, 6, 20, 58, 0] – [0, 17, 57, 52, 31, 11, 60, 28, 39, 0] - [0, 49, 59, 45, 37, 30, 42, 33, 0] - [0, 15, 38, 3, 0] – [0, 40, 8, 5, 54, 46, 2, 10, 0] - [0, 25, 29, 36, 21, 27, 56, 47, 35, 18, 48, 0] – [0, 19, 7, 23, 55, 50, 9, 0]
A-n62-k8	[0, 20, 40, 57, 32, 6, 61, 7, 0] - [0, 56, 48, 52, 36, 15, 0] - [0, 16, 37, 22, 44, 2, 19, 38, 0] – [0, 12, 13, 30, 1, 9, 11, 49, 25, 60, 5, 14, 0] - [0, 55, 50, 17, 58, 33, 26, 59, 0] – [0, 42, 39, 27, 23, 41, 21, 4, 35, 31, 34, 28, 0] - [0, 46, 45, 47, 24, 3, 54, 18, 53, 0] – [0, 29, 43, 8, 10, 51, 0]
A-n63-k10	[0, 51, 58, 22, 55, 33, 7, 0] - [0, 53, 6, 27, 35, 8, 0] - [0, 24, 46, 23, 21, 0] – [0, 48, 30, 43, 52, 36, 0] - [0, 18, 5, 3, 42, 19, 10, 59, 0] – [0, 57, 32, 34, 11, 47, 44, 31, 62, 50, 0] – [0, 25, 15, 49, 29, 56, 17, 0] - [0, 38, 4, 26, 13, 28, 20, 0] – [0, 40, 14, 2, 12, 9, 41, 1, 39, 45, 0] – [0, 61, 60, 37, 16, 54, 0]
A-n63-k9	[0, 61, 9, 41, 3, 48, 0] - [0, 8, 27, 59, 24, 36, 49, 7, 0] - [0, 42, 38, 31, 43, 46, 1, 18, 4, 47, 0] – [0, 16, 10, 6, 37, 17, 28, 12, 0] - [0, 26, 20, 34, 60, 13, 25, 0] – [0, 21, 55, 32, 58, 40, 14, 50, 53, 0] – [0, 11, 33, 44, 15, 56, 0] - [0, 22, 35, 51, 57, 62, 52, 23, 39, 0] - [0, 54, 29, 30, 2, 45, 5, 19, 0]
A-n64-k9	[0, 46, 47, 63, 2, 58, 50, 31, 59, 0] - [0, 25, 29, 45, 33, 6, 0] - [0, 11, 57, 17, 1, 30, 3, 0] – [0, 56, 52, 32, 8, 16, 60, 24, 61, 13, 41, 0] - [0, 15, 53, 62, 48, 0] – [0, 38, 43, 28, 12, 22, 18, 0] – [0, 55, 49, 44, 39, 36, 40, 7, 21, 26, 35, 14, 27, 0] - [0, 10, 20, 51, 19, 9, 42, 0] – [0, 34, 4, 54, 5, 37, 23, 0]
A-n65-k9	[0, 55, 21, 56, 59, 44, 0] - [0, 41, 2, 38, 42, 61, 58, 20, 32, 0] – [0, 29, 18, 25, 19, 13, 12, 1, 33, 62, 0] – [0, 7, 11, 63, 54, 48, 57, 23, 28, 0] - [0, 43, 27, 3, 50, 16, 60, 0] - [0, 15, 14, 22, 9, 34, 17, 0] – [0, 5, 45, 30, 37, 35, 36, 4, 49, 0] - [0, 53, 40, 10, 8, 52, 24, 0] – [0, 47, 31, 26, 6, 64, 46, 39, 51, 0]
A-n69-k9	[0, 67, 64, 61, 21, 27, 34, 0] - [0, 68, 41, 35, 45, 20, 59, 8, 38, 0] - [0, 43, 52, 12, 22, 24, 0] – [0, 29, 40, 39, 56, 62, 50, 0] - [0, 19, 58, 23, 53, 18, 31, 0] – [0, 28, 57, 42, 11, 63, 46, 5, 54, 26, 0] – [0, 66, 13, 44, 15, 3, 6, 51, 30, 9, 49, 33, 2, 0] - [0, 48, 1, 36, 10, 32, 17, 37, 16, 0] – [0, 14, 25, 47, 4, 60, 55, 65, 7, 0]
A-n80-k10	[0, 29, 59, 17, 74, 3, 77, 51, 0] - [0, 42, 53, 67, 66, 36, 73, 49, 0] - [0, 13, 12, 44, 5, 23, 62, 0] – [0, 38, 58, 32, 4, 22, 45, 72, 70, 0] - [0, 24, 6, 8, 37, 2, 34, 52, 11, 0] – [0, 27, 20, 75, 57, 61, 16, 43, 68, 78, 30, 0] – [0, 64, 33, 15, 56, 69, 65, 35, 26, 19, 47, 25, 31, 60, 39, 0] - [0, 1, 63, 7, 21, 40, 0] – [0, 76, 50, 54, 9, 55, 41, 46, 0] - [0, 28, 79, 18, 48, 14, 71, 10, 0]

Tabla 5: Rutas de las mejores soluciones encontradas para las instancias tipo A.

Instancias	Rutas
B-n31-k5	[0, 21, 16, 18, 5, 25, 4, 29, 0] - [0, 7, 17, 13, 9, 6, 22, 0] - [0, 8, 30, 23, 12, 28, 26, 0] – [0, 3, 1, 19, 24, 11, 15, 14, 0] - [0, 20, 27, 10, 2, 0]
B-n34-k5	[0, 10, 23, 33, 32, 22, 0] - [0, 9, 16, 19, 1, 0] - [0, 11, 30, 2, 7, 21, 4, 20, 0] – [0, 18, 14, 24, 29, 27, 3, 26, 8, 12, 0] - [0, 15, 13, 28, 31, 17, 25, 6, 5, 0]
B-n35-k5	[0, 15, 30, 5, 11, 19, 2, 32, 13, 0] - [0, 22, 14, 4, 28, 24, 27, 34, 0] – [0, 20, 12, 16, 10, 17, 29, 0] – [0, 6, 33, 3, 18, 26, 1, 23, 9, 7, 0] - [0, 8, 31, 25, 21, 0]
B-n38-k6	[0, 23, 33, 18, 4, 5, 27, 0] - [0, 36, 17, 8, 34, 24, 29, 12, 7, 25, 0] – [0, 21, 28, 13, 37, 35, 2, 30, 0] – [0, 26, 16, 9, 19, 6, 31, 0] - [0, 20, 10, 1, 15, 14, 0] - [0, 22, 3, 32, 11, 0]
B-n39-k5	[0, 5, 15, 11, 12, 25, 32, 38, 27, 0] - [0, 2, 33, 37, 21, 24, 23, 30, 16, 22, 26, 1, 0] – [0, 19, 20, 18, 6, 13, 10, 14, 0] - [0, 3, 28, 4, 35, 17, 36, 31, 0] - [0, 8, 7, 34, 9, 29, 0]
B-n41-k6	[0, 8, 13, 22, 31, 15, 5, 0] - [0, 16, 12, 32, 17, 14, 29, 1, 0] - [0, 4, 38, 27, 26, 19, 39, 0] – [0, 25, 6, 33, 34, 11, 28, 0] - [0, 24, 35, 2, 9, 37, 3, 18, 21, 0] - [0, 30, 7, 20, 40, 10, 23, 36, 0]
B-n43-k6	[0, 22, 24, 6, 36, 38, 15, 17, 0] - [0, 1, 23, 2, 8, 12, 31, 10, 0] - [0, 9, 7, 16, 34, 39, 42, 26, 0] – [0, 37, 28, 14, 19, 18, 5, 21, 20, 33, 0] - [0, 3, 11, 13, 30, 29, 32, 25, 27, 0] - [0, 41, 40, 35, 4, 0]
B-n44-k7	[0, 4, 8, 42, 30, 22, 38, 32, 0] - [0, 15, 33, 43, 31, 27, 35, 20, 0] - [0, 25, 28, 3, 12, 1, 40, 0] – [0, 9, 26, 24, 39, 14, 36, 2, 17, 5, 0] - [0, 7, 16, 29, 0] - [0, 23, 10, 11, 13, 0] – [0, 18, 19, 37, 41, 34, 21, 6, 0]
B-n45-k5	[0, 29, 11, 25, 26, 19, 33, 16, 4, 43, 0] - [0, 42, 40, 6, 12, 21, 8, 22, 41, 5, 39, 0] – [0, 1, 37, 32, 15, 2, 13, 28, 0] - [0, 9, 7, 34, 27, 14, 44, 18, 36, 17, 0] – [0, 10, 20, 35, 38, 31, 3, 24, 23, 30, 0]
B-n45-k6	[0, 38, 41, 31, 21, 40, 0] - [0, 30, 1, 42, 13, 43, 0] - [0, 5, 11, 10, 36, 2, 17, 14, 24, 0] – [0, 29, 18, 34, 6, 26, 25, 16, 0] - [0, 7, 20, 15, 3, 9, 33, 44, 35, 12, 0] – [0, 4, 27, 19, 37, 23, 32, 28, 8, 22, 39, 0]
B-n50-k7	[0, 38, 9, 23, 29, 46, 12, 22, 33, 0] - [0, 39, 26, 13, 1, 4, 35, 0] – [0, 34, 49, 44, 19, 17, 48, 2, 6, 0] – [0, 25, 18, 14, 43, 24, 27, 16, 10, 42, 3, 0] - [0, 37, 11, 20, 8, 5, 36, 45, 21, 0] – [0, 40, 28, 32, 15, 0] - [0, 7, 30, 41, 31, 47, 0]
B-n50-k8	[0, 35, 39, 45, 41, 23, 3, 0] - [0, 27, 12, 33, 9, 49, 36, 20, 6, 0] - [0, 10, 21, 42, 17, 8, 15, 0] – [0, 30, 16, 5, 0] - [0, 13, 22, 38, 46, 14, 31, 0] - [0, 32, 19, 29, 28, 1, 0] – [0, 11, 43, 24, 47, 26, 18, 4, 25, 37, 0] - [0, 2, 34, 48, 40, 7, 44, 0]
B-n51-k7	[0, 21, 44, 20, 1, 9, 49, 15, 0] - [0, 18, 39, 29, 13, 0] - [0, 46, 22, 24, 30, 25, 47, 45, 23, 50, 0] – [0, 41, 27, 32, 37, 31, 12, 8, 0] - [0, 33, 48, 19, 3, 35, 5, 36, 40, 0] – [0, 17, 10, 34, 38, 2, 16, 11, 6, 0] - [0, 4, 42, 14, 7, 28, 26, 43, 0]
B-n52-k7	[0, 26, 13, 17, 22, 50, 12, 23, 0] - [0, 3, 31, 24, 28, 11, 21, 39, 14, 45, 4, 0] – [0, 16, 9, 46, 49, 15, 19, 32, 34, 38, 48, 2, 0] - [0, 36, 1, 18, 30, 20, 42, 40, 0] – [0, 51, 5, 8, 27, 29, 10, 44, 0] - [0, 35, 33, 43, 7, 37, 47, 0] - [0, 25, 6, 41, 0]
B-n56-k7	[0, 2, 30, 34, 54, 31, 20, 55, 11, 33, 49, 0] - [0, 21, 26, 4, 39, 47, 42, 24, 6, 53, 0] – [0, 16, 35, 48, 32, 36, 13, 3, 0] - [0, 45, 41, 29, 37, 15, 7, 19, 0] – [0, 18, 46, 8, 28, 44, 51, 14, 17, 10, 0] - [0, 40, 5, 12, 9, 38, 50, 22, 23, 43, 0] – [0, 27, 52, 1, 25, 0]
B-n57-k7	[0, 31, 6, 11, 15, 29, 0] - [0, 45, 26, 41, 17, 34, 19, 56, 0] - [0, 30, 22, 14, 39, 37, 0] – [0, 16, 49, 44, 21, 10, 3, 24, 13, 48, 27, 46, 0] - [0, 7, 43, 20, 47, 32, 42, 50, 52, 2, 55, 1, 0] – [0, 18, 25, 28, 8, 9, 51, 38, 36, 53, 12, 0] - [0, 4, 40, 35, 54, 5, 33, 23, 0]
B-n57-k9	[0, 6, 32, 22, 56, 0] - [0, 11, 44, 26, 39, 19, 0] - [0, 4, 49, 35, 33, 12, 0] – [0, 52, 24, 27, 18, 28, 9, 50, 0] - [0, 14, 34, 13, 16, 23, 2, 0] - [0, 7, 42, 20, 37, 25, 51, 15, 8, 0] – [0, 46, 10, 17, 31, 55, 3, 40, 0] - [0, 30, 38, 48, 54, 43, 45, 1, 0] - [0, 5, 47, 36, 21, 41, 29, 53, 0]

B-n63-k10	[0, 23, 11, 17, 24, 43, 14, 58, 12, 54, 41, 0] - [0, 39, 62, 33, 26, 6, 35, 25, 52, 22, 0] – [0, 49, 56, 59, 46, 8, 30, 0] - [0, 28, 29, 55, 13, 32, 0] - [0, 44, 16, 40, 3, 4, 31, 0] – [0, 18, 50, 1, 27, 36, 20, 7, 0] - [0, 34, 2, 21, 51, 19, 0] - [0, 48, 60, 5, 15, 9, 47, 0] – [0, 42, 61, 57, 38, 53, 10, 37, 0] - [0, 45, 0]
B-n64-k9	[0, 23, 25, 39, 42, 63, 20, 57, 0] - [0, 2, 45, 44, 58, 53, 6, 3, 32, 0] - [0, 47, 61, 50, 10, 12, 0] – [0, 33, 29, 11, 28, 56, 13, 16, 0] - [0, 38, 40, 34, 27, 21, 8, 7, 0] - [0, 1, 52, 43, 54, 31, 17, 0] – [0, 49, 62, 26, 41, 46, 30, 0] - [0, 19, 15, 24, 5, 4, 14, 60, 9, 0] – [0, 37, 36, 48, 35, 59, 51, 55, 22, 18, 0]
B-n66-k9	[0, 18, 23, 31, 57, 11, 55, 0] - [0, 51, 34, 43, 22, 13, 56, 19, 45, 0] – [0, 5, 4, 60, 15, 3, 29, 14, 39, 21, 36, 50, 1, 0] - [0, 26, 46, 63, 64, 44, 59, 0] – [0, 49, 24, 12, 9, 27, 0] - [0, 41, 20, 17, 54, 53, 52, 0] - [0, 6, 2, 10, 48, 25, 16, 28, 0] – [0, 47, 61, 62, 37, 30, 35, 38, 7, 0] - [0, 42, 33, 65, 8, 40, 58, 32, 0]
B-n67-k10	[0, 32, 8, 23, 37, 65, 33, 0] - [0, 47, 41, 27, 10, 38, 17, 0] - [0, 11, 19, 20, 7, 59, 2, 0] – [0, 44, 40, 64, 6, 34, 42, 53, 0] - [0, 39, 36, 1, 49, 35, 50, 14, 4, 46, 0] – [0, 60, 3, 57, 30, 61, 62, 0] – [0, 28, 13, 58, 15, 56, 0] - [0, 31, 18, 26, 66, 51, 24, 29, 43, 16, 0] – [0, 12, 45, 52, 22, 54, 55, 21, 5, 0] - [0, 9, 48, 25, 63, 0]
B-n68-k9	[0, 63, 50, 40, 55, 23, 25, 0] - [0, 28, 18, 24, 10, 58, 6, 32, 67, 36, 42, 0] – [0, 49, 22, 2, 8, 41, 37, 0] – [0, 56, 1, 17, 9, 48, 44, 34, 16, 59, 0] - [0, 65, 43, 13, 46, 7, 47, 0] – [0, 33, 52, 20, 57, 51, 61, 26, 0] – [0, 31, 4, 60, 35, 27, 64, 0] - [0, 66, 5, 14, 62, 54, 21, 30, 45, 12, 15, 0] – [0, 29, 3, 11, 39, 38, 53, 19, 0]
B-n78-k10	[0, 53, 68, 10, 25, 18, 16, 42, 0] - [0, 51, 46, 34, 12, 15, 71, 8, 0] – [0, 3, 20, 54, 35, 59, 55, 33, 0] – [0, 2, 27, 63, 49, 47, 56, 0] - [0, 62, 61, 48, 13, 4, 36, 14, 41, 60, 66, 0] – [0, 1, 52, 24, 21, 43, 69, 67, 38, 58, 0] - [0, 50, 39, 74, 28, 19, 9, 6, 0] – [0, 22, 45, 31, 73, 76, 32, 7, 57, 0] - [0, 65, 75, 37, 29, 70, 30, 5, 0] – [0, 77, 23, 44, 26, 11, 64, 40, 17, 72, 0]

Tabla 6: Rutas de las mejores soluciones encontradas para las instancias tipo B.

A.1.3. Comparación de los mejores resultados obtenidos por ABRA y ACBCP.

En la columna *RPD ACBCP - ABRA*, un resultado positivo indica una mejor solución de ABRA, un cero indica soluciones igualmente buenas, y un negativo una mejor solución para ACBCP.

Instancia	Optimo	Fitness ACBCP	Fitness ABRA	RPD ACBCP	RPD ABRA	RPD ACBCP - ABRA
A-n32-k5	784	811	798	3,44387755	1,78571429	1,658163265
A-n33-k5	661	676	662	2,26928896	0,15128593	2,118003026
A-n33-k6	742	746	743	0,53908356	0,13477089	0,404312668
A-n34-k5	778	811	783	4,24164524	0,64267352	3,598971722
A-n36-k5	799	842	802	5,38172716	0,37546934	5,006257822
A-n37-k5	669	732	675	9,41704036	0,89686099	8,520179372
A-n37-k6	949	989	951	4,21496312	0,21074816	4,004214963
A-n38-k5	730	756	734	3,56164384	0,54794521	3,01369863
A-n39-k5	822	861	830	4,74452555	0,97323601	3,771289538
A-n39-k6	831	870	835	4,69314079	0,48134777	4,21179302
A-n44-k6	937	992	941	5,86979723	0,42689434	5,442902882
A-n45-k6	944	1139	953	20,6567797	0,95338983	19,70338983
A-n45-k7	1146	1177	1155	2,70506108	0,78534031	1,919720768
A-n46-k7	914	1022	918	11,8161926	0,43763676	11,3785558
A-n48-k7	1073	1166	1097	8,66728798	2,23671948	6,4305685
A-n53-k7	1010	1159	1038	14,7524752	2,77227723	11,98019802
A-n54-k7	1167	1262	1182	8,14053128	1,28534704	6,855184233
A-n55-k9	1073	1149	1081	7,08294501	0,74557316	6,337371855
A-n60-k9	1354	1511	1375	11,5952733	1,55096012	10,04431315
A-n61-k9	1034	1264	1076	22,2437137	4,06189555	18,18181818
A-n62-k8	1288	1539	1327	19,4875776	3,02795031	16,45962733
A-n63-k10	1314	1442	1335	9,7412481	1,59817352	8,143074581
A-n63-k9	1616	1719	1647	6,37376238	1,91831683	4,455445545
A-n64-k9	1401	1578	1436	12,633833	2,49821556	10,13561742
A-n65-k9	1174	1417	1220	20,6984668	3,91822828	16,7802385
A-n69-k9	1159	1308	1193	12,8559103	2,93356342	9,922346851
A-n80-k10	1763	2104	1839	19,3420306	4,31083381	15,03119682
PROMEDIOS				9,52480822	1,54301362	7,981794603

Tabla 7: Comparación de las mejores soluciones encontradas por ABRA y ACBCP para las instancias tipo A.

Instancia	Optima	Fitness ACBCP	Fitness ABRA	RPD ACBCP	RPD ABRA	RPD ACBCP - ABRA
B-n31-k5	672	688	676	2,380952381	0,5952381	1,785714286
B-n34-k5	788	No disponible	790	No disponible	0,25380711	No Disponible
B-n35-k5	955	991	957	3,769633508	0,20942408	3,560209424
B-n38-k6	805	817	809	1,49068323	0,49689441	0,99378882
B-n39-k5	549	569	554	3,64298725	0,91074681	2,732240437
B-n41-k6	829	858	836	3,498190591	0,84439083	2,653799759
B-n43-k6	742	792	749	6,738544474	0,94339623	5,795148248
B-n44-k7	909	955	916	5,060506051	0,77007701	4,290429043
B-n45-k5	751	810	755	7,856191744	0,53262317	7,323568575
B-n45-k6	678	759	682	11,94690265	0,5899705	11,35693215
B-n50-k7	741	770	746	3,913630229	0,67476383	3,238866397
B-n50-k8	1312	1343	1320	2,362804878	0,6097561	1,75304878
B-n51-k7	1032	1129	1036	9,399224806	0,3875969	9,011627907
B-n52-k7	747	826	752	10,57563588	0,66934404	9,906291834
B-n56-k7	707	823	722	16,40735502	2,12164074	14,28571429
B-n57-k7	1153	1342	1221	16,39202082	5,89765828	10,49436253
B-n57-k9	1598	1715	1620	7,321652065	1,3767209	5,944931164
B-n63-k10	1496	1642	1549	9,759358289	3,54278075	6,21657754
B-n64-k9	861	1026	883	19,16376307	2,55516841	16,60859466
B-n66-k9	1316	1448	1339	10,03039514	1,74772036	8,282674772
B-n67-k10	1032	1170	1075	13,37209302	4,16666667	9,205426357
B-n68-k9	1272	1459	1305	14,70125786	2,59433962	12,10691824
B-n78-k10	1221	1459	1259	19,49221949	3,11220311	16,38001638
PROMEDIOS				9,058000111	1,54795339	7,45122189

Tabla 8: Comparación de las mejores soluciones encontradas por ABRA y ACBCP para las instancias tipo B.

A.1.4. Comparación de los mejores resultados obtenidos por ABRA y AG.

En la columna *RPD AG - ABRA*, un resultado positivo indica una mejor solución de ABRA, un cero indica soluciones igualmente buenas, y un negativo una mejor solución para AG.

Instancia	Óptimo	Fitness AG	Fitness ABRA	RPD AG	RPD ABRA	RPD AG - ABRA
A-n32-k5	784	797	798	1,65816327	1,78571429	-0,12755102
A-n33-k5	661	662	662	0,15128593	0,15128593	0
A-n33-k6	742	743	743	0,13477089	0,13477089	0
A-n34-k5	778	787	783	1,15681234	0,64267352	0,514138817
A-n36-k5	799	802	802	0,37546934	0,37546934	0
A-n37-k5	669	672	675	0,44843049	0,89686099	-0,448430493
A-n37-k6	949	951	951	0,21074816	0,21074816	0
A-n38-k5	730	738	734	1,09589041	0,54794521	0,547945205
A-n39-k5	822	832	830	1,21654501	0,97323601	0,243309002
A-n39-k6	831	835	835	0,48134777	0,48134777	0
A-n44-k6	937	948	941	1,17395945	0,42689434	0,747065101
A-n45-k6	944	981	953	3,91949153	0,95338983	2,966101695
A-n45-k7	1146	1164	1155	1,57068063	0,78534031	0,785340314
A-n46-k7	914	925	918	1,20350109	0,43763676	0,765864333
A-n48-k7	1073	1074	1097	0,09319664	2,23671948	-2,143522833
A-n53-k7	1010	1038	1038	2,77227723	2,77227723	0
A-n54-k7	1167	1190	1182	1,97086547	1,28534704	0,685518423
A-n55-k9	1073	1098	1081	2,32991612	0,74557316	1,584342964
A-n60-k9	1354	1388	1375	2,51107829	1,55096012	0,960118168
A-n61-k9	1034	1054	1076	1,93423598	4,06189555	-2,127659574
A-n62-k8	1288	1334	1327	3,57142857	3,02795031	0,543478261
A-n63-k10	1314	1349	1335	2,66362253	1,59817352	1,065449011
A-n63-k9	1616	1700	1647	5,1980198	1,91831683	3,27970297
A-n64-k9	1401	1464	1436	4,49678801	2,49821556	1,998572448
A-n65-k9	1174	1221	1220	4,00340716	3,91822828	0,085178876
A-n69-k9	1159	1211	1193	4,4866264	2,93356342	1,553062985
A-n80-k10	1763	1865	1839	5,78559274	4,31083381	1,474758934
PROMEDIOS				2,09682042	1,54301362	0,5538068

Tabla 9: Comparación de las mejores soluciones encontradas por ABRA y AG para las instancias tipo A.

Instancia	Óptimo	Fitness AG	Fitness ABRA	RPD AG	RPD ABRA	RPD AG - ABRA
B-n31-k5	672	676	676	0,5952381	0,5952381	0
B-n34-k5	788	790	790	0,25380711	0,25380711	0
B-n35-k5	955	959	957	0,41884817	0,20942408	0,209424084
B-n38-k6	805	809	809	0,49689441	0,49689441	0
B-n39-k5	549	554	554	0,91074681	0,91074681	0
B-n41-k6	829	846	836	2,05066345	0,84439083	1,206272618
B-n43-k6	742	747	749	0,67385445	0,94339623	-0,269541779
B-n44-k7	909	932	916	2,53025303	0,77007701	1,760176018
B-n45-k5	751	756	755	0,66577896	0,53262317	0,133155792
B-n45-k6	678	695	682	2,50737463	0,5899705	1,91740413
B-n50-k7	741	748	746	0,94466937	0,67476383	0,269905533
B-n50-k8	1312	1332	1320	1,52439024	0,6097561	0,914634146
B-n51-k7	1032	1027	1036	-0,48449612	0,3875969	-0,872093023
B-n52-k7	747	751	752	0,53547523	0,66934404	-0,133868809
B-n56-k7	707	720	722	1,8387553	2,12164074	-0,282885431
B-n57-k7	1153	1151	1221	-0,17346054	5,89765828	-6,07111882
B-n57-k9	1598	1630	1620	2,00250313	1,3767209	0,625782228
B-n63-k10	1496	1564	1549	4,54545455	3,54278075	1,002673797
B-n64-k9	861	901	883	4,64576074	2,55516841	2,090592334
B-n66-k9	1316	1347	1339	2,3556231	1,74772036	0,607902736
B-n67-k10	1032	1080	1075	4,65116279	4,16666667	0,484496124
B-n68-k9	1272	1303	1305	2,43710692	2,59433962	-0,157232704
B-n78-k10	1221	1281	1259	4,91400491	3,11220311	1,801801802
PROMEDIOS				1,77566994	1,54795339	0,227716555

Tabla 10: Comparación de las mejores soluciones encontradas por ABRA y AG para las instancias tipo B.

A.2. Instrucciones del programa implementación de ABRA

A continuación se muestra la interfaz del programa creado para implementar ABRA y se explica cómo utilizarla.

A.2.1. Sobre el programa

El algoritmo acá presentado, fue implementado usando el lenguaje de programación Java y se le dotó de una sencilla interfaz gráfica, desde la cual se puede ingresar el nombre de las instancias de Augerat que se desean probar, además de seleccionar el valor de los parámetros principales del algoritmo. Una vez ejecutada una prueba, dentro de la misma interfaz se muestran

los resultados obtenidos, los cuales corresponden a las rutas tomadas por la mejor solución, su fitness con la cantidad de vehículos utilizados y el tiempo de ejecución que tomó.

A.2.2. Requisitos mínimos

- Cualquier Sistema Operativo con Java 1.8.0 instalado (no se garantiza compatibilidad con otras versiones).
- 128 MB Memoria Ram.
- Procesador Pentium 2 a 266 MHz.
- 1 MB Disco Duro libre.

A.2.3. Instalación

El programa no requiere ninguna instalación. Sólo se debe descomprimir el archivo **PrototipoABRA.rar** y dejar en el mismo directorio la carpeta **Augerat** (que tiene los archivos de las instancias) y la carpeta **Programa**, que contiene el ejecutable **ABRA.jar**. Para ejecutar el programa basta con hacer doble clic en este último archivo.

A.2.4. Uso del Programa

Al abrir el programa, aparecerá una ventana que vendrá con valores por default, como la de la figura 38. En ella, con el botón **Añadir** se pueden agregar los nombres de las instancias que se desean resolver (los nombres aparecen documentados repetidas veces en el Anexo A.1.). Para ello, se deben rellenar los 3 recuadros bajo la etiqueta **Instancia**, con el tipo de instancia, número de clientes (incluyendo depósito) y vehículos utilizados por la solución óptima (según el formato T-nCC-V mencionado en el punto 4.1.).

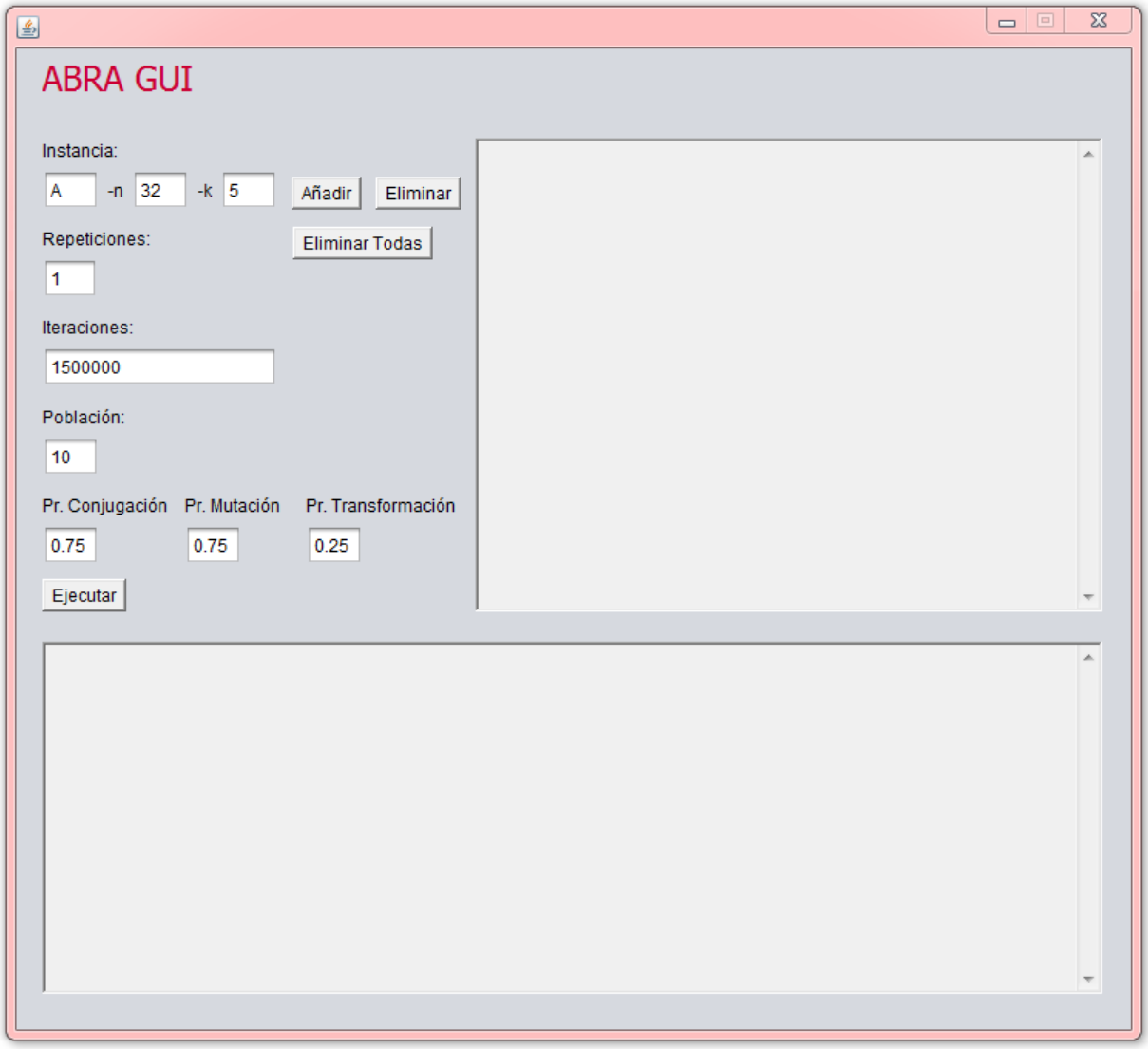

Figura 38: Pantalla inicial del programa.

Una vez agregadas algunas instancias, éstas aparecerán en el recuadro del costado derecho. Para eliminar alguna de las instancias ingresadas, se debe volver a escribir su nombre y luego presionar el botón **Eliminar**. Si se desean eliminar todas las instancias ingresadas, basta con presionar el botón **Eliminar Todas**. La imagen 39 muestra una lista de instancias ya ingresadas.

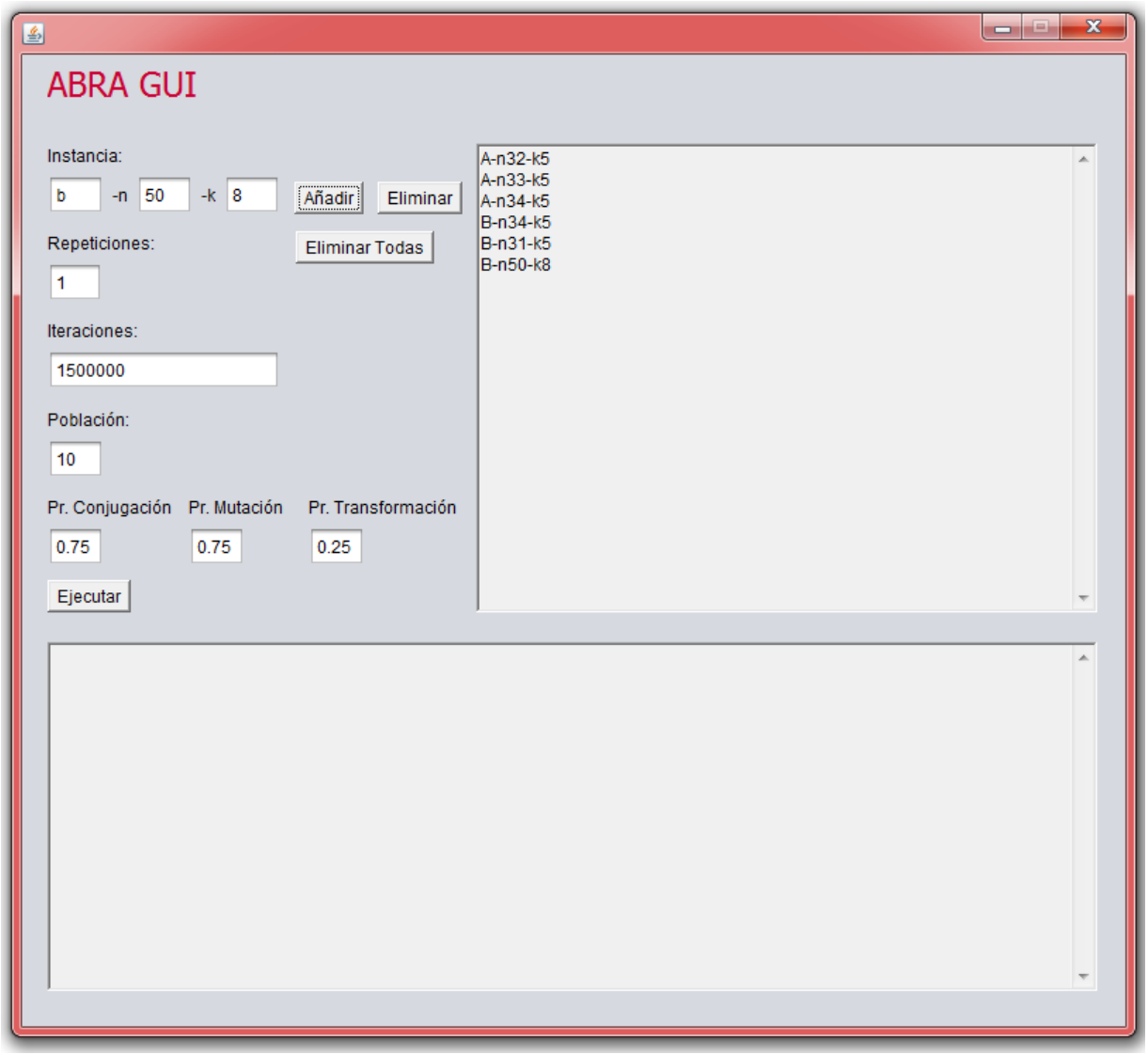

Figura 39: Pantalla del programa con instancias ya ingresadas.

En el recuadro etiquetado como **Repeticiones** se puede elegir el número de veces que se ejecutará el algoritmo por instancia.

Los recuadros restantes representan a cada uno de los parámetros principales de ABRA (se dicen principales a los que fueron ajustados en el punto 4.2.), y cada uno va acompañado de su respectiva etiqueta para saber a cual se refiere. Ningún parámetro puede ser negativo, y además, **Iteraciones** y **Población** deben ser números enteros.

Una vez ingresadas las instancias, elegido el número de repeticiones, y seleccionada la configuración de parámetros con que se desea probar el algoritmo, se debe presionar el botón **Ejecutar** para iniciar las pruebas.

Una vez pulsado el botón Ejecutar, en el recuadro inferior de la ventana aparece un texto informando de qué instancia y cual repetición se está ejecutando actualmente, como en la figura 40. Mientras el programa se encuentre ejecutando alguna prueba, la interfaz queda bloqueada.

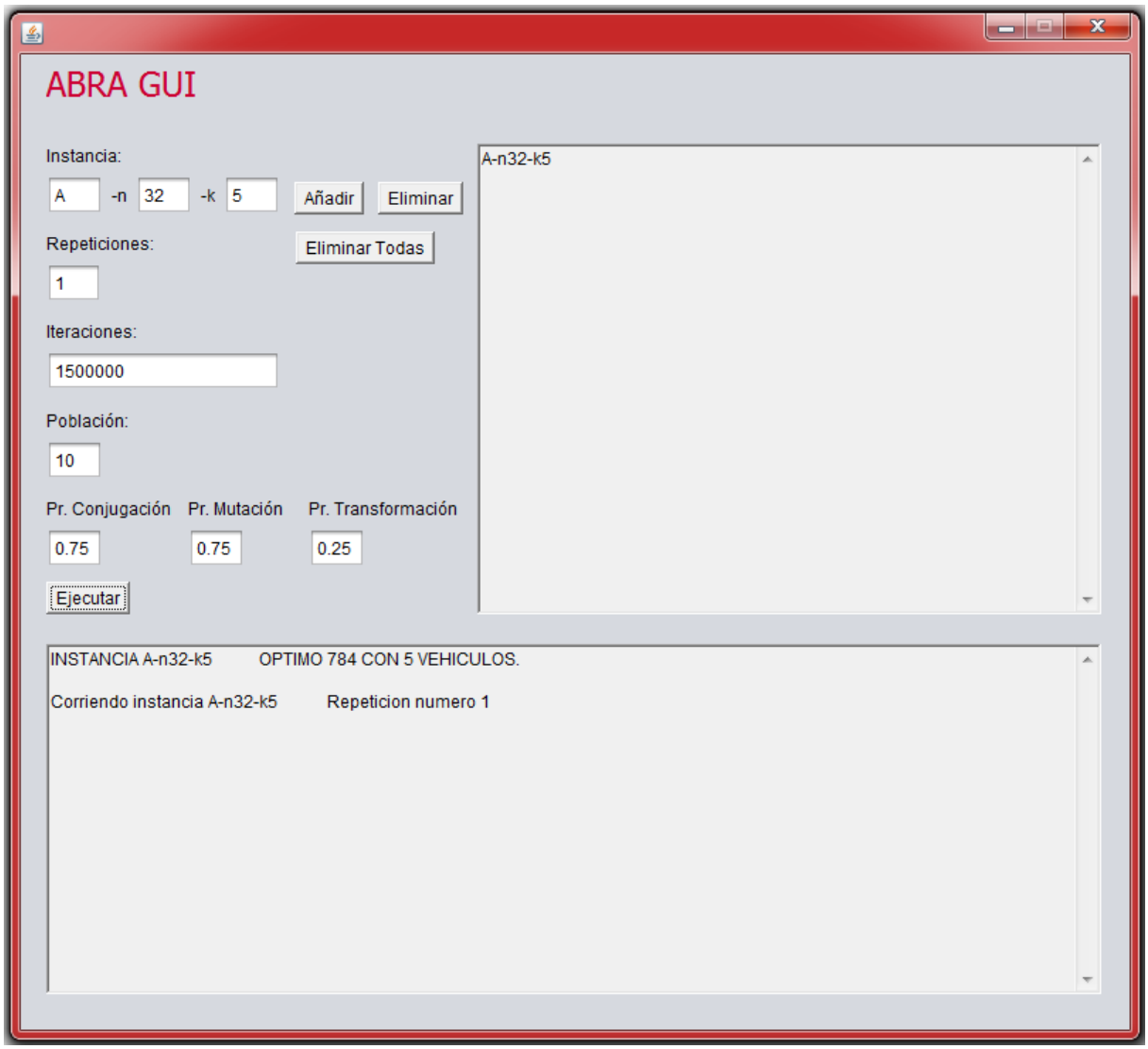

Figura 40: Pantalla del programa mientras ejecuta una prueba.

Cada vez que se termina una repetición, aparece en el recuadro inferior un resumen de los resultados obtenidos, el cual contiene el fitness de la mejor solución encontrada, número de vehículos, tiempo de ejecución y las rutas escogidas. Esto se puede apreciar en la figura 41 (el resumen está dentro del cuadro verde).

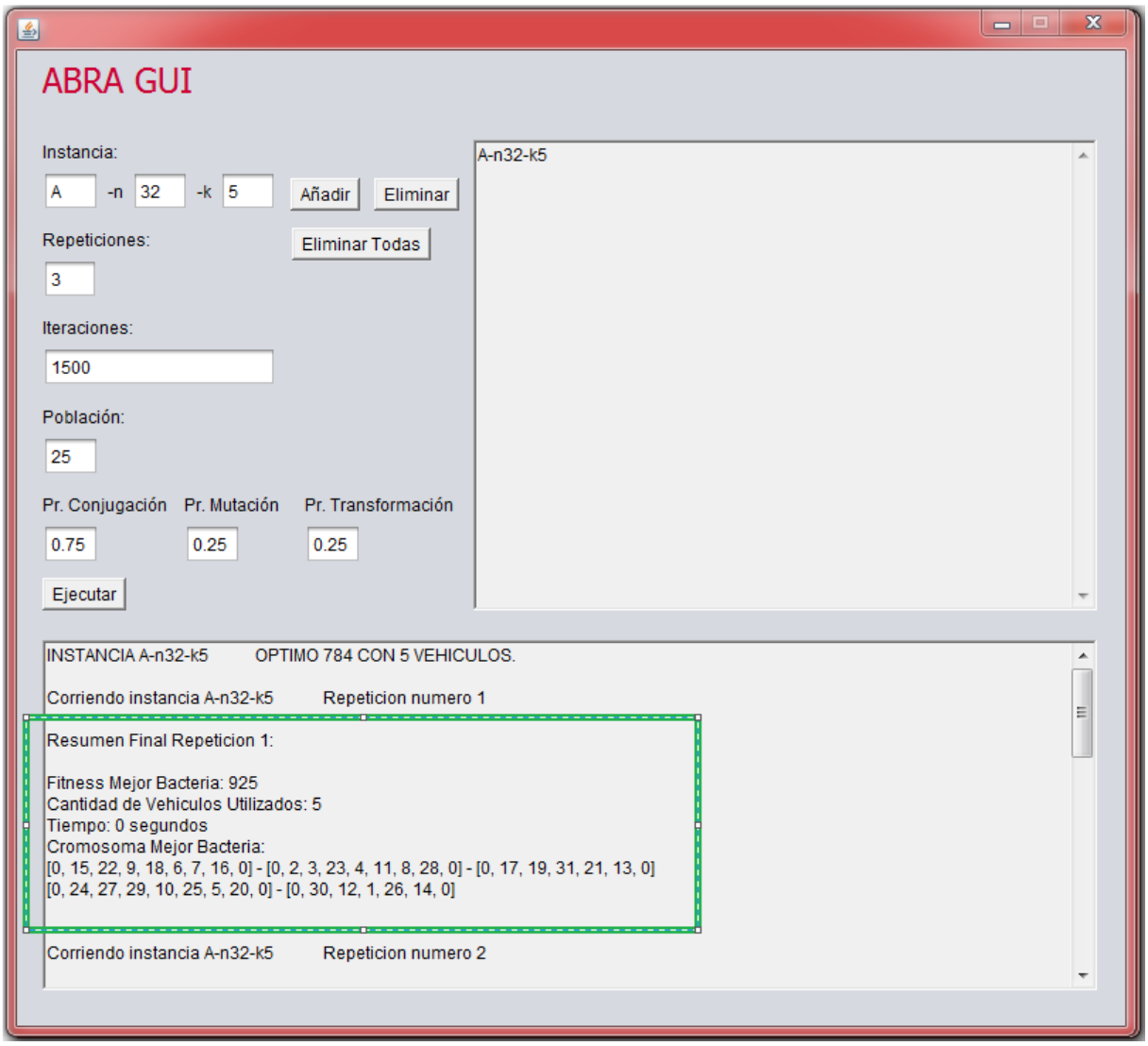

Figura 41: Pantalla del programa con los resultados de una prueba, mientras las siguientes continúan en ejecución.

Una vez ejecutadas las pruebas para todas las instancias y sus repeticiones, se muestra en el recuadro inferior el texto “EJECUCION FINALIZADA!!!”, además de desplegarse una ventana de mensaje avisando, como en la figura 42. Una vez finalizadas las pruebas, la interfaz queda desbloqueada para ser utilizada nuevamente. Cuando se lancen nuevas pruebas, el contenido en el recuadro con los resultados anteriores será borrado y reemplazado, por lo que puede ser conveniente copiarlos antes de ejecutar más experimentos.

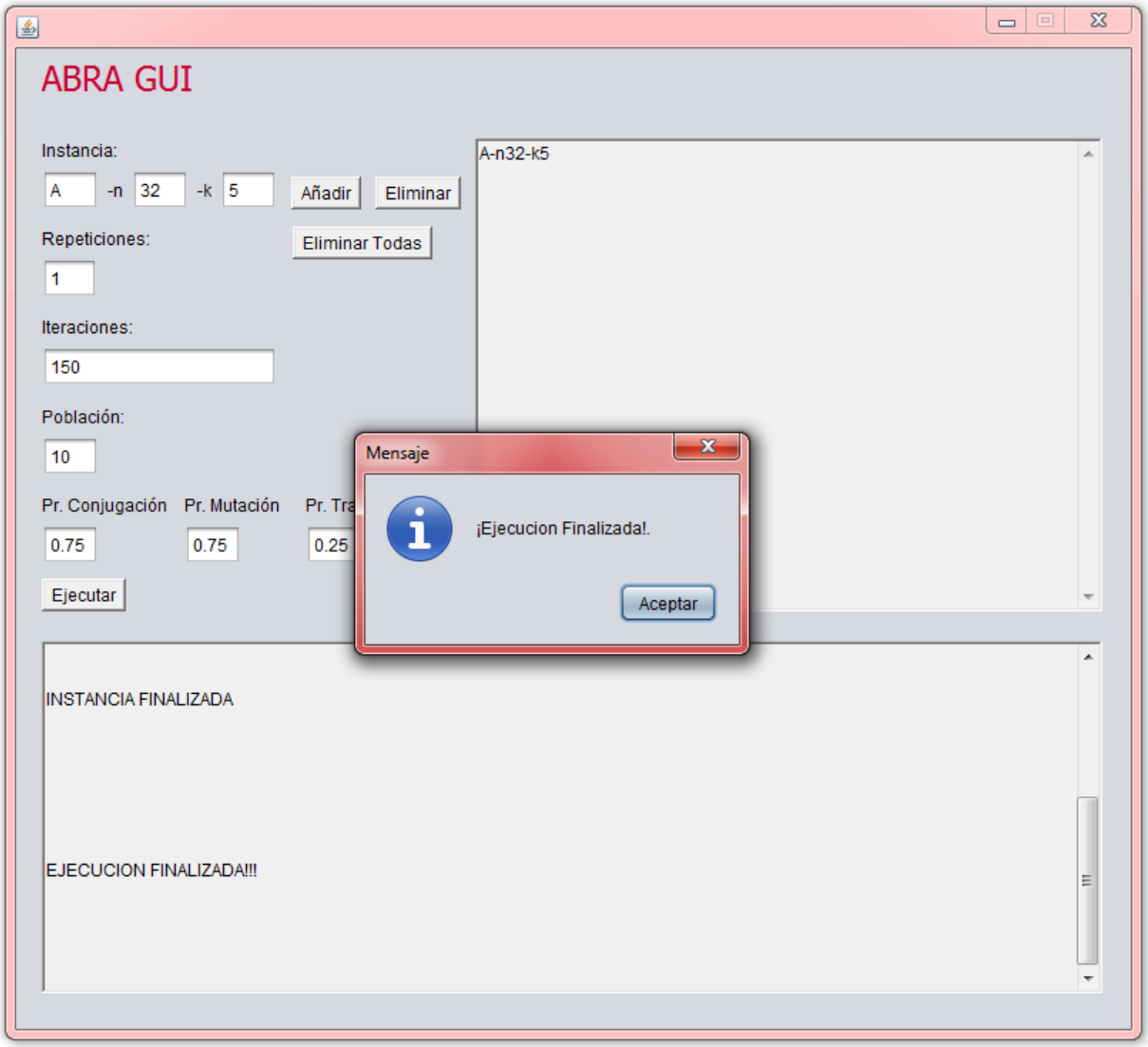

Figura 42: Pantalla y mensaje del programa al terminar todas las pruebas.