Universidad Autónoma
del Estado de Hidalgo
Área Académica de Ingeniería

El Problema del Agente Viajero resuelto con Algoritmos Genéticos: Un método para encontrar una solución al Problema de Secuenciación de Trabajos.

T E S I S

QUE PARA OBTENER EL GRADO DE
Maestro en Ciencias en Ingeniería Industrial
P R E S E N T A :

I.I. Gustavo Erick Anaya Fuentes

DIRECTOR DE TESIS:
Dra. Eva Selene Hernández Gress

NOVIEMBRE DE 2012

Agradecimientos

Gracias Dios por las bendiciones y el aprendizaje que me otorgas todos los días. Por la salud y la valiosa oportunidad que me has dado para seguir estudiando, por eso y mil cosas más gracias.

Gracias a mis padres por hacer de nuestra familia un verdadero hogar. Gracias Ariel por el ejemplo de esfuerzo, trabajo y valores que siempre me has enseñado. Gracias Juliana por la disciplina que me has heredado. Gracias Doris por ser una buena hermana y enseñarme tantas cosas, por hacerme tío de una personita que llena de felicidad a nuestras vidas. Gracias Dulce por ser una hermana de la que me siento orgulloso, pues cuando te desvelas haciendo tarea me motivas a trabajar de la misma forma. Gracias Paola por tu compañía y paciencia.

Gracias Dra. Eva Selene Hernández Gress por su valioso apoyo, conocimiento, atención, paciencia y tiempo que me concedió.

Gracias a los Doctores Juan Carlos Seck Tuoh Mora y Joselito Medina Marín por su apoyo, el cual siempre tendré presente.

Gracias a todos los profesores y compañeros.

Resumen

En este proyecto se estudia el Problema de Secuenciación de Trabajos resuelto a través del Problema del Agente Viajero mediante Algoritmos Genéticos; en donde se propone una codificación del Problema de Secuenciación al del Agente Viajero. Se revisan las diferentes metodologías y enfoques utilizados para resolver el Problema del Agente Viajero para posteriormente proponer dos versiones de torneo en el algoritmo genético; el primero llamado determinístico y el segundo llamado aleatorio. Se realizan diferentes pruebas para la solución del Problema del Agente Viajero, utilizando ambos algoritmos bajo diferentes parámetros para los operadores de número de individuos, número de iteraciones a realizar, probabilidad de cruce y probabilidad de mutación; a partir de estos se obtienen los parámetros y el algoritmo adecuado a realizar.

Posteriormente se realiza una decodificación para transformar al Problema de Secuenciación en un Problema del Agente Viajero en donde se experimenta la solución de este último en búsqueda de su optimización, para después volver a la codificación del Problema de Secuenciación de Trabajos. Se realizan diferentes experimentos tomados de la literatura para su comparación con los resultados obtenidos en los diferentes artículos.

Abstract

This project studies the Job Sequencing Problem solved through the Traveling Salesman Problem by Genetic Algorithms, which proposes an encoding of the scheduling problem to the traveling salesman. We review various methodologies and approaches used to solve the Traveling Salesman Problem and subsequently propose two versions tournament genetic algorithm, the first call and the second call deterministic random. Tests are conducted for solving the Traveling Salesman Problem, using both algorithms under different parameters for operators of number of individuals, number of iterations to perform, crossover probability and mutation probability, since these parameters are obtained and to perform the appropriate algorithm.

Later a decoding to transform the scheduling problem into a Traveling Salesman Problem where we experience the latter solution in search optimization, and then return to the code of the scheduling problem of jobs. They perform different experiments from the literature for comparison with the results obtained in the different articles.

Contenido

Lista de Figuras

Lista de Tablas

Introducción

El problema del agente viajero es un problema de optimización combinatoria en la que una persona visita sólo en una ocasión cada una de las ciudades; se deberá localizar la ruta que presente la distancia más corta y a ésta se le conoce como la ruta óptima.

A medida que crece el número de ciudades a visitar por el agente viajero, no es factible resolverlo por programación entera debido a que el tiempo de solución computacional crece de forma exponencial de acuerdo con el número de ciudades visitadas. A este tipo de problemas se les conoce como no polinomiales (NP). En esta investigación se modeló el problema del agente viajero a través de programación entera analizando hasta dónde fue factible de resolverse por este método. Posteriormente se propuso un algoritmo genético el cual fue probado con algunos ejemplos en donde la solución óptima fue encontrada a través de programación entera. También se utilizó el algoritmo genético para resolver algunos ejemplos en donde la solución óptima no pudo ser encontrada por programación entera. Por último, el problema del agente viajero resuelto por algoritmos genéticos se utilizó para resolver un problema de secuenciación de trabajos.

El algoritmo genético generado sirvió para resolver el problema del agente viajero. El enfoque innovador radicó en crear una codificación adecuada, trabajar con los operadores del algoritmo y determinar los mejores parámetros asociados para resolver el problema del agente viajero. Las operaciones son cruza, mutación, selección etc. y los parámetros son: tamaño de la población, criterio de paro, entre otros. Una vez que se obtuvieron resultados, se propuso una forma de transformar un Problema de Secuenciación de Trabajos a un Problema del Agente Viajero. La tesis está organizada de la siguiente manera: El Capítulo 1 sirve para contextualizar y dar un sentido de pertenencia al problema en cuestión, indicando trabajos existentes en la literatura que sirvieron como base a lo que se propone. El capítulo 2 proporciona un marco

teórico para el Problema del Agente Viajero, Algoritmos Genéticos y El Problema de Secuenciación de Trabajos. El Capítulo 3 presenta el lenguaje computacional que permite el desarrollo de los algorimos. En el Capítulo 4 se muestra la experimentación realizada a diversos problemas del agente viajero de diferentes artículos. El Capítulo 5 muestra la experimentación de la solución de problemas de secuenciación de trabajos a través del problema del agente viajero, mediante algoritmos genéticos, por último se incorpora una conclusión general del trabajo.

Capítulo 1

Antecedentes para la formulación del Problema del Agente Viajero

El Problema del Agente Viajero al que en lo sucesivo denominamos PAV; no fue en sus orígenes como se conoce hoy en día; éste evolucionó con el paso del tiempo conforme los investigadores encontraron una infinidad de aplicaciones del mismo. El PAV ha sido utilizado para resolver diversos problemas como los que se mencionan a continuación: en robótica permite resolver problemas de fabricación para minimizar el número de desplazamientos al realizar una serie de perforaciones en una plancha o en un circuito impreso, en control y operativa optimizada de semáforos o como en nuestro caso, se utiliza para problemas de rutas y secuenciación de trabajos [15].

1.1 Consulta Referencial

Fue en el año de 1800 cuando el matemático irlandés William Rowand Hamilton [23], estudió por primera vez el PAV, al diseñar un juego entre dos competidores en un icosaedro. Este juego consiste en hacer un recorrido por los veinte puntos del icosaedro usando las conexiones de la figura e iniciando y terminando en el mismo punto sin repetir los lugares visitados.

El 5 de febrero de 1930, en un coloquio en Viena, el matemático austriaco Karl Menger [8] plantea por primera vez el PAV en lenguaje matemático. En 1931 Menger

tuvo una estancia en Harvard donde dictó varios seminarios y en uno de ellos mencionó el problema de las rutas mínimas y el PAV. Whitney [8] se interesó en el PAV, proponiendo encontrar la solución entre las 48 capitales de los estados en la parte continental de la Unión Americana. Con este simple ejemplo inició la divulgación del PAV en los centros de investigación de la Unión Americana.

El PAV ya había surgido desde 1759, en este problema se pretendia mover al caballo del juego de ajedrez en todas las posiciones exactamente una vez. En 1832 el alemán Voigt [25] escribe un libro sobre como tener éxito al resolver el PAV. Poste- riormente, Homaifar [2] propone un enfoque en el cual se puede encontrar con certeza la solución óptima. El procedimiento consiste en la generación de todos los tours posibles en donde se evalúan sus correspondientes distancias. El tour con la distancia más pequeña es considerada como la mejor.

El PAV muestra todos los aspectos de optimización combinatoria. Durante la década de 1950, la Programación Lineal se estaba convirtiendo en una fuerza vital en la informática de soluciones a problemas de optimización combinatoria. Esto se debió a la financiación proporcionada por la Fuerza Aérea de los EE.UU. y en el interés de obtener soluciones óptimas a problemas de transporte combinatoria, esta es una de las razones por las cuales el PAV se encontraba en el interés de todos. Los intentos de resolver el PAV fueron inútiles hasta mediados de la década de 1950 cuando Dantzig, Fulkerson y Johnson [14] presentaron un método para resolver el PAV. Ellos mostraron la eficacia de su método de resolución de una instancia de 49 de la ciudades. Posteriormente, se hizo evidente a mediados de 1960, que el PAV no puede ser resuelto en tiempo polinomial usando técnicas de programación lineal, por lo que se plantea la complejidad computacional, lo que significa que cualquier esfuerzo programable para resolver problemas tales crecería de forma no polinomial con el tamaño del problema. Estas categorías de problemas que se conoce como NP-completos. Existe un gran progreso en el tratamiento de problemas NP-completos, como el PAV. Se han encontrado soluciones a los casos de PAV con tamaños de entrada muy pequeños, debido a la complejidad de su solución. También algunas soluciones en tiempos polinomiales se han encontrado en casos especiales del PAV. Los investigadores incluso han recurrido a la búsqueda de algoritmos de aproximación polinomial para variaciones NP-completo del PAV mencionados en [14]. Hasta el día de hoy, una solución eficiente para el caso general PAV, o incluso a cualquiera de sus variantes NP-completos, no se ha encontrado. A continuación se describen algunos de los principales trabajos para

resolver el PAV.

Dorigo [22] propone un heurístico llamado colonia de hormigas artificiales capaces de resolver el PAV, en la cual las hormigas de las colonias artificiales son capaces de generar recorridos más cortos mediante el uso de información acumulada en forma de un rastro de feromona depositada en los bordes de las rutas del PAV. Las hormigas van dejando rastros de feromona conforme van completando recorridos, al que tenga mayor cantidad de feromona le siguen el resto de las hormigas, en este caso las siguientes iteraciones. La cantidad de feromona depositada en cada nodo (ciudades) visitados por la mejor hormiga es inversamente proporcional a la longitud del recorrido; el recorrido más corto tendrá la mayor cantidad de feromona depositada en los nodos. Esta manera de depositar feromona con la intención de emular la característica de la acumulación de ruta de feromonas, que en el caso de las hormigas reales se debió a la interacción entre la longitud de la ruta de acceso y continuidad del tiempo.

Cerny [33] utilizó el método Montecarlo para resolver el PAV. El algoritmo genera aleatoriamente permutaciones de nodos, en donde su probabilidad depende de la distancia de la ruta; haciendo una analogía con la Termodinámica estática. En este caso la temperatura es utilizada como parámetro arbitrario.

Tolga [32] estudió la posibilidad de resolver el problema de multiagente viajero mPAV en donde se permite más de un agente viajero puesto que según el autor se presentan en la vida común con mayor frecuencia que el PAV. El problema se define de acuerdo a lo siguiente: Dado un conjunto de ciudades hay m vendedores ubicados en una sola ciudad, el resto de las ciudades que serán visitadas son llamadas ciudades intermedias. Por lo tanto el mPAV consiste en encontrar tours para todos los vendedores, en donde todos comienzan y terminan en la ciudad inicial, tal que cada ciudad intermedia es visitada solo una vez, minimizando el costo total de todos los recorridos. El autor informa que los mejores resultados obtenidos con 100 ciudades a visitar han requerido de 173 ordenadores para su solución. Una de las soluciones propuestas consiste en transformar el mPAV en un PAV normal integrando m vendedores.

William Cook [31] profesor de la Escuela Técnica Industrial e Ingeniería de Sistemas en Georgia es considerado el más avanzado en cuanto a resultados en el PAV según la comunidad científica. Cook puso en marcha su primer proyecto del PAV en 1988 cuando era profesor en la Universidad de Columbia en Nueva York junto con Vasek Chvátal un profesor de matemáticas de la Universidad de Rutgers. Más tarde los

dos investigadores reclutaron a David Applegate universitario graduado de Carnegie Mellon y a Robert Bixby, profesor de la Universidad Rice en Houston. En 1992, los cuatro encontraron la ruta óptima para 3 038 ciudades [31], rompiendo el récord anterior de 2 392 ciudades [31] establecido en 1987 por Manfred Padberg Universidad de Nueva York y Giovanni Rinaldi, del Instituto de Análisis de Sistemas y Ciencias de la Computación en Roma. La revista Discover considera a la solución de 3038 ciudades [31] como una de las 50 mejores historias de ciencia en 1992. Desde entonces este equipo ha recibido dicha mención en cinco ocasiones. Los investigadores resolvieron el PAV para 4 461 ciudades de Estados Unidos en 1993 [31], 7 397 ciudades en 1994 [31], 13 505 ciudades en 1998 [31] y un tour en Alemania de 15 113 [31] ciudades en el año 2001. En abril de 2004, que establece su registro actual con un recorrido óptimo de 24 978 ciudades suecas [31]. Sin embargo no han especificado como trabaja su algoritmo.

El PAV puede ser utilizado como una herramienta para resolver el Problema de Secuenciación de Trabajos [24] el cual según la definición de Ruiz [18] contiene un número de máquinas y un conjunto de trabajos con restricciones, el problema consiste en investigar, si existe una programación de trabajos que ayude a mejorar y hacer eficaz el uso de las máquinas, eliminando el tiempo ocioso, ya que no existe computadora alguna, que logre obtener una solución óptima en corto tiempo.

La metodología en la investigación de secuencias y de programación de tareas es muy amplia; conviene revisar los procedimientos a los que han acudido diversos autores en la búsqueda de soluciones que se plantean para problemas de optimización de recursos en la realización de tareas. Los primeros esfuerzos han sido desarrollados por Johnson, Kusiac (en Delgado [10]) y tomados por Kusiac [10] en donde incluyen las tripletas; número de operación, tiempo de proceso, número de máquina. En el año 2000 Onwubolu [10] toma los conceptos de los algorimos genéticos para hallar la combinación óptima de una celda de manufactura.

Con respecto al problema de programación de tareas [10], éste trata sobre la asignación de recursos limitados a ciertas áreas u operaciones a través de un determinado período de tiempo. La solución de este problema consiste en la optimización de uno o más objetivos, esto se presenta con frecuencia en sistemas de producción convencionales automatizados, donde está involucrado el tomar decisiones con respecto a la mejor asignación de recursos a procesos de información donde se tienen restricciones

de tiempo.

La programación de tareas y el control de flujo de trabajos a través de un ambiente de producción es escencial en los procesos de manufactura [10]. Una programación adecuada puede reducir significativamente los costos de producción y reducir los tiempos de proceso, permitiendo cumplir con los compromisos de entrega a tiempo. En este problema, se presenta un problema de optimización combinatoria y gran parte de ellos pertenecen a la clase de problemas NP-completos. El problema de secuenciación de trabajos es más difícil de resolver que el PAV, puesto que el problema de secuenciación puede ser mostrado como una versión especializada del PAV [24].

En los últimos años se han puesto un gran número de enfoques para modelar y solucionar los diferentes problemas de programación de tareas. Entre estos enfoques podemos mencionar a la programación matemática, reglas de despacho, sistemas expertos, redes neuronales, algoritmos genéticos, búsqueda Tabú, recocido simulado, lógica difusa, mencionados en [10].

En la presente investigación se eligieron a los algoritmos genéticos como heurístico para resolver problemas de secuenciación de trabajos debido a que además de ser robustos no proporcionan una única solución, haciéndolos flexibles para la toma de decisiones, y por que como se expone a continuación, los algoritmos genéticos han tenido resultados notables en este tipo de problemas; por lo que solucionan el problema de secuenciación como un PAV. A continuación se muestran también los antecedentes de los algoritmos genéticos.

Los primeros ejemplos de lo que hoy podríamos llamar algoritmos genéticos que en lo sucesivo se denomina AG; aparecieron a finales de 1950 y principios de 1960, programados en computadoras por biólogos evolutivos que buscaban explícitamente realizar modelos de aspectos de la evolución natural [23].

En 1962, investigadores como Box, G.J. Friedman, W.W. Bledsoe y H.J. Bremermann (mencionados en Haupt [11]) habían desarrollado independientemente algoritmos inspirados en la evolución para optimización de funciones y aprendizaje automático, pero sus trabajos generaron poca reacción. En 1965 surgió un desarrollo más exitoso, cuando Ingo Rechenberg mencionado en [11], quién entonces trabajaba en la Universidad Técnica de Berlín, introdujo una técnica que llamó estrategia evolutiva, aunque no se parecía a los AG actuales. En esta técnica no había población ni cruzamiento;

un padre mutaba para producir un descendiente, y se conservaba el mejor de los dos, convirtiéndose en el padre de la siguiente ronda de mutación.

Chatterjee [30] propone un algoritmo genético con un plan de reproducción a través de la mutación para un operador evolutivo que puede ser directamente aplicado para una permutación de n números para una aproximación de la solución del PAV. El análisis de esquema del algoritmo muestra que una reproducción con operadores de mutación, preserva la propiedad de convergencia global de un algoritmo que establece el teorema fundamental de los algoritmos mencionado en [30], en este algoritmo se evita un paso intermedio de codificación a través de las llaves que preservan el cruzamiento o permutación n usando un estado de ajuste, lo que hace que la propuesta sea interesante.

Algunos autores han trabajado con el PAV resuelto por AG. A continuación se exponen brevemente algunos trabajos:

Fogel [9] propone la evolución natural como un método para generar aprendizaje en máquinas. Una simulación de la evolución natural es conducida utilizando el PAV como un medio ambiente artificial; para una solución exacta de un PAV, los algoritmos conocidos exigen un número de pasos para crecer lo menor posible en forma exponencial de acuerdo con los elementos del problema. En problemas complejos, la ruta final se estimó alrededor del 99.9999999999 % de aproximación.

Larrañaga [27] presenta operadores de cruza y mutación, desarrollándolos para abordar el PAV con AG en diferentes representaciones como: binaria, de rutas, por proximidad, ordinaria y matriz en diferentes intentos por resolver el PAV con algoritmos genéticos.

Un AG eficiente para resolver el PAV es presentado por Moon [7]; en el que los resultados muestran la propuesta de los algoritmos genéticos al generar una solución óptima y un rendimiento superior comparado con los algoritmos tradicionales.

Jog [26], comenta en su artículo que los AG basan su progreso sobre el rendimiento de los elementos candidatos a ser las soluciones. Los autores se concentraron en el PAV. Para este problema existen varios algoritmos que van desde los AG puros hasta los AG que incorporan información heurística. Los AG mostraron una gran desventaja puesto que son ineficientes cuando se implementan sobre una computadora secuencialmente,

sin embargo, debido a sus propiedades paralelas, los AG pueden ser implementados exitosamente en varias computadoras de forma simultánea, aumentando la aceleración al encontrar la solución óptima. Holland [16], comenta que puesto que tratamos de imitar lo que ocurre en la naturaleza, se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto la selección del individuo estará relacionada con su valor de ajuste. Sin embargo no se deben eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población se volvería homogénea.

Snyder [21], presenta un heurístico efectivo para el PAV. El método de combinación de un AG con un tour heurístico local. Las soluciones son codificadas usando entradas aleatorias. Snyder [21] trabaja sobre un conjunto de 442 nodos; en donde el heurístico muestra que el resultado se encontró en muchos casos dentro del 1% del óptimo, con un tiempo computacional promedio de 10 segundos. El heurístico es comparado con otros, publicando los datos en ambas soluciones, así como también su tiempo computacional.

La mejora continua en el valor del AG ha sido atractivo para muchos tipos de métodos de resolución de problemas de optimización, particularmente los AG trabajan muy bien en mezcla (discreto continuo) de problemas combinatorios; además son menos suceptibles a que los resultados obtenidos sean los mismos de una generación a otra. Sin embargo tienden a ser computacionalmente caros. Una población inicial llamada Genoma o Cromosoma es generado aleatoriamente en poblaciones sucesivas, o generaciones, son obtenidos mediante operadores genéticos como la selección, cruza y mutación para evolucionar las soluciones a fin de encontrar la mejor. El operador de selección escoge a dos miembros de la presente generación a fin de participar en las siguientes operaciones: cruza y mutación. El operador de cruza intercambia a los alelos de dos padres para obtener dos hijos. La mutación se crea en un corto periodo después del cruce, intercambiando alelos de forma aleatoria Buthainah [12].

Applegate [8] comenta que en el año de 1998 se resolvió el PAV con una cantidad de 13,509 ciudades de los Estados Unidos. Este fue el mayor ejemplo que se había resuelto hasta el momento.

David Applegate, Robert Bixby, Chvatal Vasek y William Cook [8] anunciaron que habían resuelto el PAV para un total de 15,112 ciudades de Alemania. El recorrido tiene una longitud óptima de 1,573,084 en las unidades utilizadas, lo que se traduce

en un viaje de unos 66,000 kilómetros a través de Alemania. El total de tiempo de computadora utilizados para el cálculo fue de 22.6 años, a escala en un procesador de Compaq EV6 Alfa funcionando a 500 MHz.

En el año de 2003 Rodríguez [34] indica la forma de aplicar un AG al PAV tomando en consideración los operadores de selección, cruza y mutación; el problema consistió en encontrar la ruta más corta para visitar a diez ciudades de la República Mexicana tan solo en una ocasión y volviendo a la ciudad de partida. El problema se programó en lenguaje C, Borland 3.1 realizando los recorridos en una computadora Pentium III a 766 MHz en un tiempo de dos horas, en donde el criterio de paro fue por iteraciones al realizarse mil de estas, se encontró que la mejor solución se obtuvo en la generación 420.

En 2004 Buthainah [12] menciona un PAV con 24 978 ciudades en Suecia, el cual obtuvo un resultado de 855 597 kilometros de longitud y se informó que no existe recorrido más corto. Éste es actualmente el más grande ejemplo del PAV resuelto, superando el récord anterior de 15 112 ciudades en Alemania, fijado en 2001.

El PAV Suecia fue atacado por una serie de grupos con algunos de los métodos de búsqueda que se han desarrollado hasta la fecha. Lo más importante es la etapa final que mejora el límite inferior de 855,595 hasta 855,597 el valor óptimo requiere aproximadamente 8 años de tiempo de cálculo que se ejecuta en paralelo en una red de estaciones de trabajo Linux. [8]

El trabajo computacional es vital al resolver este tipo de algorimos por lo que a continuación se proporcionan algunas características de la teoría de la complejidad computacional.

1.2 Teoría de la complejidad computacional

La historia de la complejidad es bastante reciente, basta revisar algunos institutos dedicados a estudiar este tema, como lo menciona Maldonado [6], los primeros institutos se crean a finales de los años 1970s: en 1978, se crea el Centro de Estudios para la Dinámica No-Lineal en el Instituto La Jolla; a comienzos de los años 1980 se crea el Instituto Santa Cruz para la Ciencia No-Lineal, en el año 1980 surge el Centro

para Estudios No-Lineales en el Laboratorio Nacional de los Álamos; posteriormente, en 1981 se crea el Instituto para la Ciencia No-Lineal en la Universidad de California en Davis.

En 1984 [31] surge el Instituto Santa Fe (SFI), ampliando significativamente la comprensión del tipo de ciencia que hacían los anteriores Centros e Institutos, el SFI está consagrado a las Ciencias de la Complejidad. Nacen, la complejidad organizativa, administrativa y financieramente, las Ciencias de la Complejidad. Posteriormente, en Estados Unidos y en Europa primero, y luego también en Japón y China, surgen otros centros e institutos similares.

Para hablar de complejidad computacional, es importante mencionar a Alan Turing [6] quién introdujo el concepto de máquina de Turing en el trabajo publicado por la Sociedad Matemática de Londres en 1936, en el que se estudiaba la cuestión planteada por David Hilbert [6] sobre si las matemáticas son decidibles, es decir, si hay un método definido que pueda aplicarse a cualquier sentencia matemática y que nos diga si es cierta o no. Turing ideó un modelo formal de computador (algoritmo), la máquina de Turing, y demostró que existían problemas que una máquina no podía resolver. Con este aparato extremadamente sencillo es posible realizar cualquier cómputo que un computador digital sea capaz de realizar. La máquina propuesta es un dispositivo relativamente simple, pero capaz de resolver cualquier operación matemática. Turing tenía la ilusión de que su máquina tenía una capacidad tal que, potencialmente, podría realizar cualquier cosa realizable por el cerebro humano, incluyendo la capacidad de poseer conciencia de sí mismo. En contra parte Penrose [29] afirma la imposibilidad de que una máquina tenga tales capacidades puesto que una computadora puede hacer uso de reglas formales fijas, pero nunca será capaz de *ver* la coherencia de sus propias reglas. Lo que distingue a los humanos, es su capacidad para ver esta coherencia.

Para la realización del trabajo de Turing se tiene como precedente el trabajo de Church y Kleene llamado el *Cálculo Lamda*, en el cual se define el término de función, la noción de aplicación de funciones lo que sirvió para resolver *El Entscheidungsproblem* (problema de decisión) [1] fue el reto en lógica simbólica de encontrar un algoritmo general que decidiera si una fórmula del cálculo de primer orden es un teorema. En 1936, de manera independiente, Alonzo Church [1] y Alan Turing [6] demostraron que es imposible escribir tal algoritmo. Como consecuencia, es también imposible decidir con un algoritmo si ciertas frases concretas de la aritmética son ciertas o falsas.

Mediante este modelo teórico y el análisis de la complejidad de los algoritmos, fue posible la categorización de problemas computacionales de acuerdo a su comportamiento, apareciendo así, el conjunto de problemas denominados Polinomiales (P) y No Polinomiales (NP), cuyas soluciones pueden encontrarse en tiempo polinómico por máquinas de Turing deterministas y no deterministas, respectivamente. Una razón para aceptar la máquina de Turing como un modelo general de cómputo es que el modelo definido es equivalente a muchas versiones modificadas que en principio pareciera incrementar el poder computacional.

Para comprender NP-completo, se debe considerar la teoría que se utiliza para clasificar todos los problemas de cálculo y los algoritmos utilizados para resolverlos, la cual es llamada teoría de complejidad computacional [31]. Recordemos que un algoritmo es un conjunto de instrucciones que van paso a paso, y al ejecutarse en el orden especificado, resolverá un problema determinado. Los problemas básicamente se clasificaron dentro de dos categorías: Uno considerado *fácil* si puede ser resuelto por un algoritmo que se ejecuta en tiempo polinómico; y un problema se considera como *completo* si no puede ser resuelto en tiempo polinómico. Estas definiciones son la base de la teoría de la complejidad computacional. Una nota importante es que cualquier problema que se ha demostrado ser completo, en el sentido de que sólo existe una solución exponencial al problema, o que no hay solución algorítmica para el problema, no está comprendida dentro de una clase de complejidad. La razón de esto es porque no se sabe cuánto tiempo se tardará en resolver un problema completo. Incluso las computadoras que existen en la actualidad, pueden tardar años en llegar a una solución, y los recursos de la computadora probablemente no alcanzarían en ese momento.

1.3 Planteamiento del problema

Actualmente los métodos convencionales como la programación entera reportan una frontera en cuanto al tiempo computacional para determinar la secuenciación de trabajos que permitan reducir tiempos de producción en la industria manufacturera. A través de la resolución del PAV con AG se busca validar un método para la resolución de éste, el cual servirá como un medio para solucionar el Problema de Secuenciación de Trabajos.

1.4 Objetivo general

Resolver el PAV a través de AG, posteriormente utilizar el PAV para resolver un Problema de Secuenciación de Trabajos.

1.5 Objetivos Específicos

1. Modelar el PAV a través de Ramificación y Acotamiento y Programación Entera.
2. Analizar el tamaño máximo del problema (número de ciudades a visitar) donde se encuentra la solución óptima a través de Programación Entera en Lingo.
3. Revisar la literatura para estudiar trabajos hechos con AG que resuelvan el problema.
4. Generar el AG, considerando criterios (búsqueda, paro, entre otros, etc.) y operaciones (cruza, mutación, etc.).
5. Programar el AG en Matlab.
6. Utilizar el AG para resolver con ejemplos en donde la solución óptima haya sido encontrada a través de Programación Entera con la finalidad de comparar las soluciones.
7. Utilizar el AG para resolver ejemplos con más ciudades donde la solución no puede ser encontrada a través de Programación Entera en Lingo.
8. Utilizar el PAV resuelto por AG para resolver un Problema de Secuenciación de Trabajos.

1.6 Pregunta de investigación

¿Puede el Problema del Agente Viajero ser utilizado para resolver un Problema de Secuenciación de Trabajos, si el Agente Viajero ha sido codificado y resuelto por un metaheurístico como los Algoritmos Genéticos?

1.7 Hipótesis

El Problema del Agente Viajero puede ser codificado y resuelto por un metaheurístico como los Algoritmos Genéticos, una vez que esto se ha hecho, el Poblema del Agente Viajero puede servir como un método para llegar a una solución del Problema de Secuenciación de Trabajos.

Capítulo 2

Métodos para resolver el Problema del Agente Viajero

El PAV [35] consiste en visitar n ciudades y regresar a su punto de origen. La solución óptima consiste en encontrar el orden de las ciudades que minimizan la distancia total que el vendedor debe recorrer antes de regresar a su punto de origen. Existen algunos métodos para resolver el PAV que a continuación se revisan.

2.1 Método de ramificación y acotamiento para resolver el problema del agente viajero

Muchos de los Problemas Enteros (PE) son resueltos mediante la técnica de ramificación y acotamiento, los cuales encuentran la solución enumerando los puntos factibles de una parte del problema (subproblema). Ahora bien, si la relajación del Problema Lineal (PL) es resuelta con un enfoque de PE pura, entonces la solución óptima para la relajación del PL es también solución óptima del PE.

El método de ramificación y acotamiento consiste básicamente en dos pasos según Winston [35]:

1. El problema estará terminado cuando sea innecesario ramificar un subproblema. Entonces se presentaran 3 posibles situaciones que han de resolver el problema:

(a) El subproblema no es factible.

(b) El subproblema da una solución óptima en la cual todas las variables tienen valores enteros.

(c) El valor óptimo de la función objetivo para el subproblema no excede (al maximizar) el mejor valor de la función objetivo obtenido hasta el momento.

2. Se puede dejar de considerar un subproblema en las siguientes situaciones:

(a) El subproblema no es factible.

(b) El mejor valor de la función objetivo obtenido hasta el momento es por lo menos igual al valor de la función objetivo para el subproblema.

Ahora vamos a ejemplificar el método de ramificación y acotamiento basándonos en el ejemplo de Winston [35] que enuncia lo siguiente: Joe State vive en Gary, Indiana. Es dueño de agencias de seguros en Gary, Fort Wayne, Evansville, Terre Haute y South Bend. Todos los meses de diciembre visita cada una de sus agencias. La distancia entre cada agencia (en millas), se muestra en la Tabla 2.1. ¿Qué orden debe seguir al visitar sus agencias que minimice la distancia total recorrida?

Tabla 2.1: Matriz 5X5 Winston

	Gary	Fort Wayne	Evansville	Terre Haute	South Bend
Ciudad 1 Gary	0	132	217	164	58
Ciudad 2 Fort Wayne	132	0	290	201	79
Ciudad 3 Evansville	217	290	0	113	303
Ciudad 4 Terre Haute	164	201	113	0	196
Ciudad 5 South Bend	58	79	303	196	0

Como observamos se trata de un PAV el cual cuenta con las ciudades $1, 2, 3, \ldots, N$. Para $i \neq j$, C_{ij} es la distancia desde la ciudad i hasta la ciudad j y sea $C_{ii} = M$, donde M es un número muy grande (en relación con las distancias reales del problema) [35]. Se define también:

$X_{ij} = 1$, si la solución al PAV va de la ciudad i a la ciudad j.
$X_{ij} = 0$, si no sucede así.

Así mismo para $i \neq j$,

C_{ij}= distancia entre las ciudades i y j.

C_{ii}= M, donde M es un número positivo muy grande para evitar tener rutas de una ciudad a ella misma.

Podríamos deducir que la respuesta al problema del agente viajero se logrará al resolver un problema de asignación que tiene una matriz de costos cuyo $ij-$ésimo elemento es C_{ij}. Por ejemplo, si resolvemos este problema de asignación y se llega a la solución X_{12}=X_{24}=X_{45}=X_{53}=X_{31}= 1. Esta solución se puede escribir como 1-2-4-5-3-1. Un recorrido que empieza y termina en la misma ciudad y pasa una vez por cada ciudad, se llama tour. La solución óptima para el problema de asignación no siempre será un tour. Por ejemplo, la solución óptima para el problema de asignación podría ser X_{15}=X_{21}=X_{34}=X_{43}=X_{52}= 1. Encontramos ahora que la solución óptima tiene dos subtours. Un subtour es un viaje redondo que no pasa por todas las ciudades. La asignación tiene dos: 1-5-2-1 y 3-4-3. Si pudieramos excluir todas las soluciones factibles que contienen subtours y luego resolver el problema de asignación, se podría llegar a la solución óptima del PAV. En muchos casos un método de ramificación y acotamiento es lo más eficaz para resolver el problema.

Para aplicar el método de ramificación y acotamiento, iniciamos por resolver el problema de asignación de la Tabla 2.1, el cual denominaremos subproblema 1 mediante el método húngaro, el cual se explica más adelante. La solución óptima es X_{15}=X_{21}=X_{34}=X_{43}=X_{52}= 1, $Z = 495$.
Como se muestra en (Fig.2.1). Esta solución contiene dos subtours (1-5-2-1 y 3-4-3) por lo que no puede ser la solución óptima.

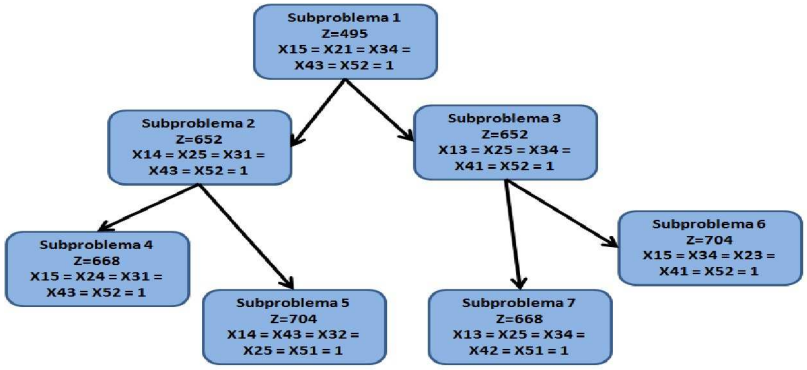

Figura 2.1: Subproblemas Winston

Ahora ramificamos sobre el problema 1 en tal manera que evitemos que uno de los subtours del subproblema 1 recurra o se repita en las soluciones para los subproblemas posteriores. Elegimos al subtour 3-4-3 para excluirlo. La solución óptima para el problema debe tener X_{34}= 0 o X_{43}= 0 (si X_{34}=X_{43}=1, la solución óptima tendría el subtour 3-4-3). Por lo tanto, es posible ramificar sobre el subproblema 1 al sumar los dos subproblemas siguientes:

$Subproblema2 = subproblema1+$ (X_{34}=0, ó C_{34}=M).
$Subproblema3 = subproblema1+$ (X_{43}=0, ó C_{43}=M).

Elegimos de forma arbitraria el subproblema 2 para resolverlo, aplicando el método húngaro a la matriz de costo como se muestra en la Tabla 2.2. La solución óptima para el subproblema 2 es $Z = 652$, X_{14}=X_{25}=X_{31}=X_{43}=X_{52}= 1. Esta solución incluye los

subtours 1-4-3-1 y 2-5-2. Así que no puede ser la solución óptima.

Tabla 2.2: Matriz costos subproblema 2

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	132	217	164	58
Ciudad 2	132	M	290	201	79
Ciudad 3	217	290	M	M	303
Ciudad 4	164	201	113	M	196
Ciudad 5	58	79	303	196	M

Se ramifica sobre el subproblema 2 buscando excluir el subtour 2-5-2. Se debe tener la seguridad de que X_{25} ó X_{53} sean igual a 0. Por lo que se suman los dos subproblemas siguientes:

$Subproblema4 = Subproblema2+($ X_{25}=0, ó C_{25}= M).
$Subproblema5 = Subproblema2+$ (X_{52}=0, ó C_{52}=M).

Tomamos al subproblema 4 para resolverlo como se muestra en la Tabla 2.3 genera una solución óptima de $z = 668$ con $X_{15} = X_{24} = X_{31} = X_{43} = X_{52} = 1$; Aquí no se presenta un subtour, por lo tanto la solución es probablemente óptima.

Tabla 2.3: Matriz costos subproblema 4

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	132	217	164	58
Ciudad 2	132	M	290	201	M
Ciudad 3	217	290	M	M	303
Ciudad 4	164	201	113	M	196
Ciudad 5	58	79	303	196	M

Posteriormente resolvemos el subproblema 5, al aplicar el método húngaro a la matriz de la Tabla 2.4. La solución óptima para el subproblema 5 es $z = 704$, X_{14}=X_{43}=X_{32}= X_{25}=X_{51}= 1. Esta solución es un tour, pero $z = 704$ no es tan buena como la de la solución probable del subproblema 4 de $z = 668$. Por consiguiente, se podría dejar de considerar al subproblema 5.

Tabla 2.4: Matriz costos subproblema 5

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	132	217	164	58
Ciudad 2	132	M	290	201	M
Ciudad 3	217	290	M	M	303
Ciudad 4	164	201	113	M	196
Ciudad 5	58	M	303	196	M

Por último tenemos el subproblema 3. Encontramos la solución óptima para el problema de asignación de la Tabla 2.5 que se genera para este subproblema, la cual proporciona la ruta: X_{13}=X_{25}=X_{34}=X_{41}=X_{52}= 1, con un resultado para la función objetivo de $z = 652$.

Tabla 2.5: Matriz costos subproblema 3

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	132	217	164	58
Ciudad 2	132	M	290	201	79
Ciudad 3	217	290	M	113	303
Ciudad 4	164	201	M	M	196
Ciudad 5	58	79	303	196	M

Esta solución contiene los subtours 1-3-4-1 y 2-5-2. Como $652 < 668$, todavía es posible que el subproblema 3 genere una solución sin subtours y que llegue a $z = 668$. Por lo que ramificamos sobre el subproblema 3. Cualquier solución factible para el PAV que provenga del subproblema 3 debe tener X_{25}=0 ó X_{52}=0; por lo que podemos generar los subproblemas 6 y 7.

$Subproblema6 = Subproblema3+$ (X_{25}=0 ó C_{25}= M).
$Subproblema7 = Subproblema3+$ (X_{52}=0 ó C_{52}= M).

Elegimos al subproblema 6 para resolverlo. La solución óptima para el subproblema 6 es X_{15}=X_{34}=X_{23}=X_{41}=X_{52}= 1, $z = 704$. La solución no tiene subtours aunque su

valor de z es de 704 es mayor al de la solución probable del subproblema 4, por lo que el subproblema 6 no puede generar la solución óptima para el problema.

Únicamente tenemos al subproblema 7 por resolver. La solución óptima para este subproblema es X_{13}=X_{25}=X_{34}=X_{42}=X_{51}=1, con un valor para la función objetivo de $z = 668$. Por lo tanto, el subproblema 4 y el subproblema 7 generan la misma solución óptima como se observa en (Fig. 2.1). En donde el subproblema 4 tiene la ruta 1-5-2-4-3, mientras el subproblema 7 muestra la ruta 5-1-3-4-2-5. Por lo tanto los subproblemas 7 y 8 presentan la misma ruta con diferentes puntos de inicio y en orden inverso.

El método de ramificación y acotamiento requiere la aplicación del método húngaro el cual se describe a continuación:

2.2 Método Húngaro

El Método Húngaro es un algoritmo que se usa para resolver problemas de minimización, ya que es más eficaz que el empleado para resolver el problema del transporte por el alto grado de degeneración que pueden presentar los problemas de asignación. Para esto ejemplificamos retomando el subproblema 1 de la ramificación y acotamiento de Winston [35], según la Tabla 2.1. Las fases para la aplicación del método húngaro según [35] son:

Paso 1: Encontrar primero el elemento más pequeño en cada fila de la matriz de costos $m * m$; ese elemento debe restarse a cada uno de los costos de su fila para obtener una nueva matriz que se observa en la Tabla 2.6; encontrar para esta nueva matriz, el costo mínimo en cada columna. A continuación se debe buscar el mínimo de cada columna y restarlo de cada elemento de su columna correspondiente y con ello se contruye la nueva matriz de la Tabla 2.7.

Paso 2: Consiste en trazar el número mínimo de líneas horizontales,verticales o ambas que se requieren para cubrir las filas y/o las columnas donde se encuentren los ceros; se debe de dar prioridad a aquellas que tengan mayor número de ceros. Esto se hace sobre la matriz de la Tabla 2.7; si se necesitan m líneas para cubrir todos los ceros, se tiene una solución óptima entre los ceros cubiertos de la matriz. Si se requieren

Tabla 2.6: Matriz reducida

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	74	159	106	0
Ciudad 2	53	M	211	122	0
Ciudad 3	104	177	M	0	190
Ciudad 4	51	88	0	M	83
Ciudad 5	0	21	245	138	M

Tabla 2.7: Matriz de costos reducidos

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	53	159	106	0
Ciudad 2	53	M	211	122	0
Ciudad 3	104	156	M	0	190
Ciudad 4	51	67	0	M	83
Ciudad 5	0	0	245	138	M

menos de m líneas para cubrir todos los ceros, se debe continuar con el paso 3. El número de líneas para cubrir los ceros es igual a la cantidad de asignaciones que hasta ese momento se pueden realizar.

Tabla 2.8: Matriz de costos reducidos ceros cubiertos

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	53	~~159~~	~~106~~	~~0~~
Ciudad 2	53	M	~~211~~	~~122~~	~~0~~
Ciudad 3	104	156	~~M~~	~~0~~	~~190~~
Ciudad 4	51	67	~~0~~	~~M~~	~~83~~
Ciudad 5	~~0~~	~~0~~	~~245~~	~~138~~	~~M~~

Paso 3: Encontrar el menor elemento diferente de cero (llamado k) en la Tabla 2.8 y que no está cubierto por las líneas dibujadas en el paso 2; a continuación se debe restar k de cada elemento no cubierto por las líneas dibujadas y sumar k a cada elemento cubierto por dos líneas (intersecciones); ésto en la misma tabla. Por último se debe regresar al paso 2.

Para nuestro ejemplo se forman únicamente cuatro líneas, por lo tanto es necesario aplicar el paso 3, en donde el valor de $k = 51$. Aplicando este paso tenemos la Tabla

2.9.

Tabla 2.9: Matriz de costos reducidos con $k = 51$

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	M	2	159	106	~~0~~
Ciudad 2	2	M	211	122	~~0~~
Ciudad 3	~~53~~	~~105~~	~~M~~	~~0~~	~~190~~
Ciudad 4	~~0~~	~~16~~	~~0~~	~~M~~	~~83~~
Ciudad 5	~~0~~	~~0~~	~~296~~	~~189~~	~~M~~

Al aplicar el paso 2 observamos que nuevamente se forman cuatro líneas en la Tabla 2.9 por lo que es necesario aplicar nuevamente el paso 3, en donde el valor de $k = 2$. Al hacerlo obtenemos la Tabla 2.10.

Tabla 2.10: Matriz de costos reducidos con $k = 2$

	Ciudad 1	Ciudad 2	Ciudad 3	Ciudad 4	Ciudad 5
Ciudad 1	~~M~~	~~0~~	157	104	~~0~~
Ciudad 2	~~0~~	~~M~~	209	120	~~0~~
Ciudad 3	~~53~~	~~105~~	~~M~~	~~0~~	~~192~~
Ciudad 4	~~0~~	~~16~~	~~0~~	~~M~~	~~85~~
Ciudad 5	~~0~~	~~0~~	296	189	~~M~~

Para obtener la solución se trazan las 5 líneas buscando cubrir todos los ceros y dando prioridad a aquellas asignaciones que tengan más ceros; en cada uno de ellos podemos encontrar una solución por ejemplo, en la fila uno tomamos de la ciudad 1 a la 5 puesto que tenemos un cero en ese punto, para tener $X_{15} = 1$; posteriormente en la fila 2 tomamos de la ciudad 2 a la 1 para tener $X_{21} = 1$; en la fila 3 tomamos al cero que se encuentra de la ciudad 3 a la 4, por lo tanto $X_{34} = 1$; ahora en la fila 4 tomamos al cero de la ciudad 4 a la 3; para decir que $X_{43} = 1$; finalmente en la fila 5 tomamos al cero de la ciudad 5 a la 2, de tal manera que $X_{52} = 1$. Por lo que tenemos que la solución es $X_{15} = X_{21} = X_{34} = X_{43} = X_{52}$, el cual forma el *subproblema*1 del ejemplo de ramificación y acotamiento; el cual presenta dos subtours, por lo que se continua con el método de ramificación y acotamiento mostrado en la sección en que se presenta el mismo.

2.3 Programación entera

2.3.1 El Problema del Agente Viajero

El PAV es un problema de optimización combinatoria que tiene un número finito de soluciones factibles; y en el caso que se llegara a tener muchas soluciones factibles se requiere una gran cantidad de tiempo computacional para enumerar de forma explícita todas las soluciones posibles.

El PAV puede definirse de la siguiente manera [12] : Sea una red G = [N, A, D] que está definida por un conjunto de N nodos, y A el conjunto de arcos, y D = $[d_{ij}]$ la matriz de costos. Eso es, d_{ij} el costo de moverse desde el nodo i al nodo j. El PAV requiere un ciclo Halmiltoniano en G de mínimo costo (un ciclo Hamiltoniano es uno que pasa a través de cada nodo i de N exactamente una vez). El PAV es un problema de permutación que tiene como objetivo encontrar el camino de menor longitud o mínimo costo en un grafo no dirigido que representa las ciudades o nodos a ser visitados; en el cual el se pretende elegir el resultado tal que la suma de todas las distancias Euclidianas entre cada nodo y su sucesor se reducen al mínimo.

El PAV puede resolverse mediante programación entera el cual consiste en elegir la ruta que minimice la distancia entre las ciudades $1, 2, 3, \ldots, N$. Para $i \neq j$, C_{ij} es la distancia desde la ciudad i hasta la ciudad j y sea $C_{ii} = M$, donde M es un número muy grande (en relación con las distancias reales del problema). Se define también:

$X_{ij} = 1$, si la solución al PAV va de la ciudad i a la ciudad j.
$X_{ij} = 0$, si no sucede así.

Entonces, la solución al problema se encuentra al resolver:

$$
\begin{aligned}
&\min Z = \textstyle\sum_i \sum_j C_{ij}\ X_{ij} ....(a)\\
&\text{sujeto a}\\
&\textstyle\sum_{i=1}^{i=N} X_{ij} = 1\ (\forall\ j = 1, 2, \ldots N) ....(b)\\
&\\
&\textstyle\sum_{j=1}^{j=N} X_{ij} = 1\ \ (\forall\ i = 1, 2, \ldots N) ....(c)\\
&\\
&u_i - u_j + N x_{ij} \leq N - 1(\ \ \forall\ i \neq j; i = 2, 3, \ldots N;\\
&j = 2, 3, \ldots N) ....(d)\\
&\forall X_{ij} = 0\, o\, 1, \forall\ u_j \geq 0
\end{aligned}
\tag{2.1}
$$

La función objetivo (a) proporciona la distancia total de los arcos incluidos en un tour. Las restricciones en (b) dan certeza de que uno llega una vez a cada ciudad. Las restricciones en (c) aseguran que uno abandona la ciudad cada vez. Las restricciones en (d) son clave para el planteamiento. Ellas aseguran los siguiente:

1. Cualquier conjunto de X_{ij} que forma un subtour será no factible, es decir, se incumple (d).

2. Cualquier conjunto de X_{ij} que forma un tour será factible, habrá un conjunto de u_j que satisfaga (d).

Uno de los enfoques utilizados para resolver el PAV es la Programación Entera PE; y para poder definir a la PE es necesario interpretar a la Programación Lineal PL, la cual es una herramienta para resolver problemas de optimización. Dantzing [14] desarrolló el método simplex para resolver problemas lineales los cuales se basan en la función objetivo que busca maximizar o minimizar la función de decisión, sin dejar de lado las restricciones o limitaciones de recursos para resolver el problema.

Un problema de Programación Entera (PE) es un Problema Lineal en el cual se requiere que algunas variables o todas sean enteros no negativos.

Los PE que requieren que todas las variables sean enteras son llamados Problemas Puros de Programación con Enteros [35]; por ejemplo:

$$
\begin{aligned}
&max\ \ Z = \ \ 3X_1 + 2X_2\\
&s.a.\\
&\qquad\qquad X_1 + X_2 \geq 6\\
&\qquad\qquad X_1, X_2 \geq 0,\ \ \forall,\ \ X_1, X_2\ \ enteros
\end{aligned}
$$

Aunque también existen problemas combinados de programación con enteros [35]; en donde no se requiere que X_2 sea un entero. Por ejemplo:

$$\begin{array}{lll} max & Z = & 3X_1 + 2X_2 \\ s.a. & & \\ & & X_1 + X_2 \geq 6 \\ & & X_1, X_2 \geq 0, \ \forall, \ X_1 \ enteros \end{array}$$

Así como la Programación Entera binaria[35], en donde las variables pueden ser solamente $0 - 1$.

$$\begin{array}{lll} max & Z = & X_1 - X_2 \\ s.a. & & \\ & & X_1 + 2X_2 \geq 2 \\ & & 2X_1 - X_2 \geq 1 \\ & & X_1, X_2 = 0 \text{ ó bien } 1 \end{array}$$

Un PE puede considerarse como la relajación de un PL adecuando las restricciones pertinentes en donde la región factible para cualquier PE debe estar contenida en la región factible para la relajación del PL correspondiente.

Al retomar el ejemplo de 5 ciudades de Winston [35] Tabla 2.1, podemos escribir el problema de programación entera de la siguiente manera:

La función objetivo (a) es como sigue:

$\min Z = M * X_{11} + 132 * X_{12} + 217 * X_{13} + 164 * X_{14} + 58 * X_{15} + 132 * X_{21} + M*$
$X_{22} + 290 * X_{23} + 201 * X_{24} + 79 * X_{25} + 217 * X_{31} + 290 * X_{32} + M * X_{33} + 113 * X_{34}$
$+303 * X_{35} + 164 * X_{41} + 201 * X_{42} + 113 * X_{43} + M * X_{44} + 196 * X_{45} + 58 * X_{51} + 79$
$X_{52} + 303 * X_{53} + 196 *_{X54} + M * X_{55}$
sujeto a

Las restricciones del tipo (b) son:

$X11 + X12 + X13 + X14 + X15 = 1$
$X21 + X22 + X23 + X24 + X25 = 1$
$X31 + X32 + X33 + X34 + X35 = 1$
$X41 + X42 + X43 + X44 + X45 = 1$
$X51 + X52 + X53 + X54 + X55 = 1$

Las restricciones del tipo (c) son:

$X11 + X21 + X31 + X41 + X51 = 1$
$X12 + X22 + X32 + X42 + X52 = 1$
$X13 + X23 + X33 + X43 + X53 = 1$
$X14 + X24 + X34 + X44 + X54 = 1$
$X15 + X25 + X35 + X45 + X55 = 1$

(2.2)

Las restricciones del tipo (d) son:

$U2 - U3 + 5 * X23 \leq 4$
$U2 - U4 + 5 * X24 \leq 4$
$U2 - U5 + 5 * X25 \leq 4$
$U3 - U2 + 5 * X32 \leq 4$
$U3 - U4 + 5 * X34 \leq 4$
$U3 - U5 + 5 * X35 \leq 4$
$U4 - U2 + 5 * X42 \leq 4$
$U4 - U3 + 5 * X43 \leq 4$
$U4 - U5 + 5 * X45 \leq 4$
$U5 - U2 + 5 * X52 \leq 4$
$U5 - U3 + 5 * X53 \leq 4$
$U5 - U4 + 5 * X54 \leq 4$

Por último se muestran las restricciones de no negatividad:

$X11, X12, X13, X14, X15, X21, X22, X23, X24, X25, X31, X32, X33, X34, X35, X41, X42, X43, X44, X45, X51, X52, X53, X54, X55 \geq 0$

En donde para evitar que una ciudad tenga como destino a sí misma se asigna en la función objetivo un valor relativamente grande al que llamamos M. Aunado a esto, con las restricciones del tipo (d) se impide la creación de subtours para el resultado del PAV.

La solución completa de este problema puede observarse en la Figura 3 del Capítulo de experimentación, se resolvió con Lingo obteniéndose la siguiente ruta óptima $1 - 5 - 2 - 4 - 3 - 1$; que es la misma obtenida a través del método de ramificación y acotamiento cuyo resultado es $Z = 668$.

2.4 Algoritmos Genéticos

Cuando el número n de ciudades a visitar en el PAV crece, la complejidad computacional lo hace también, hasta un punto en el cual resulta imposible esperar demasiado tiempo para obtener un resultado; por lo que es conveniente utilizar un heurístico como los AG, que permiten encontrar un resultado muy cercano al óptimo mediante operadores de torneo, cruza y mutación.

Según Chambers [20] la teoría de evolución de Charles Darwin fue utilizada para ejemplificar la evolución genética; Jean Baptiste Lamarck en [20] propuso una teoría multifacética de la evolución, lo que especifica su teoría es que las características adquiridas por un individuo a lo largo de su vida son heredadas a sus descendientes. La evolución propuesta por Lamarck [20] ha sido probada dentro de la aplicación de la evolución artificial en las computadoras debido a que en el plano biológico ha sido descartada esta teoría.

Un AG trabaja de la misma forma que la evolución natural descrita por Lamarck en [20]; en la cual tendremos una población de soluciones candidatas a resolver el problema. Se seleccionan las soluciones basándose en qué tan buenas son éstas y posteriormente se crean nuevas soluciones desde aquellas que fueron seleccionadas.

Los AG buscan proporcionar una calificación para la supervivencia en la búsqueda de un buen resultado. En la naturaleza, los individuos más aptos tienen más probabilidades de sobrevivir y aparearse, por lo que la próxima generación debe ser mejor. Esta misma idea se aplica al PAV, en donde primero tratan de encontrar las regiones de soluciones y luego combinar a las más aptas para crear una nueva generación de soluciones que deben ser mejor que la generación anterior. También incluyen un elemento mutación al azar para evitar la degradación entre los elementos, evitando caer en un óptimo local. Una codificación adecuada encuentra la solución al problema de manera que cada posible solución tiene una codificación única [19]. La población inicial se genera de manera aleatoria utilizando una o varias técnicas predeterminadas conjuntamente. Para evaluación se toma una simple solución como parámetro y devuelve un número que indica lo buena que la solución es. Por sí mismo el número devuelto no significa nada, solamente al ser comparado con otro valor podemos seleccionar a la mejor solución.

Cuando se trabaja con AG generalmente se compara una población generada al azar con una inicial de mejores encontrados [3], funcionando mejor la población de mejores [3] en este caso entendiéndose como mejores aquellos que cumplen las restricciones del modelo de programación entera, es decir, aquellos cuya función objetivo tenga adaptabilidad y que desde ahora llamaremos (fitness).

Existen operadores genéticos que son la base de la búsqueda del valor óptimo en los cuales no puede determinarse una secuencia de pasos para una solución. Existen dos tipos de comportamiento de búsqueda: búsqueda aleatoria y búsqueda local. La búsqueda aleatoria explora alrededor de todas las soluciones; y la búsqueda local explora las mejores soluciones y es capaz de escalar hacia arriba acercándose a un óptimo local.

Los operadores utilizados por los algoritmos genéticos son selección, cruza y mutación. A continuación se da una descripción de ellos.

Selección:

1. Selección por ruleta: Este consiste en que cada individuo de la población tiene una probabilidad de ser seleccionada analógicamente como en una ruleta. Entre mejor sea la solución se tendrán secciones más grandes por lo tanto existen mayores posibilidades de ser seleccionadas. La idea básica es determinar la probabilidad de selección o probabilidad de supervivencia para cada individuo al evaluar su solución. El proceso de selección está basado en girar la ruleta en tiempos iguales para la población, cada tiempo selecciona una sola solución para una nueva población. El método es un proceso de muestreo estocástico. Baker en [20] propuso un método universal de muestreo estocástico que utiliza solo un impulso para la rueda. La estrategia básica esencial del enfoque es mantener un número de copias de cada cromosoma en la siguiente generación.

2. Selección por torneo: escoge aleatoriamente a un conjunto de soluciones y saca al mejor individuo para su reproducción. El número de individuos en el conjunto es llamado tamaño del torneo.

   Este método toma regularmente 2 o más soluciones al azar y utiliza una estrategia de torneo, donde se utiliza un determinado número de corridas. En cada corrida un número de soluciones compiten entre sí y solamente la mejor solución ganará y será seleccionada [15].

Cruzamiento: El cruzamiento de individuos toma normalmente a dos padres y crea hijos con la mezcla de información genética de ambos padres. Las formas comunes de cruzamiento son de un punto, de dos puntos, y de n puntos y cruzamiento uniforme [15].

En el cruzamiento de un punto al azar se elige a lo largo de cada conjunto de soluciones de padres, en donde cada mitad de cromosomas es intercambiado. Por ejemplo si tenemos dos inviduvios para realizar el cruce y los llamamos $Padre1$ y $Padre2$ con sus respectivos valores:
$Padre1 = 1234|5678$ $Padre2 = 8765|4321$

Sean sección 1 del $Padre1 = 1234$ y sección 2 del $padre1 = 5678$
Mientras sección 1 del $Padre2 = 8765$ y sección 2 del $Padre2 = 4321$

Como observamos contamos con un punto de cruce en cada uno de los padres o individuos, los cuales se dividen justamente por la mitad.

Al realizar el cruzamiento obtendremos dos hijos; ambos se formaran a partir de los cortes hechos en los padres. La primera sección del $Hijo1$ se forma a partir de la sección 1 del $Padre1$, posteriormente la sección 2 del $Hijo1$ se forma con la totalidad de la sección 2 del $Padre2$. Por lo tanto obtenemos:

$Hijo1 = 12344321$

El $Hijo2$ lo obtenemos tomando la sección 1 del $Padre2$ y colocándolo en la sección 1 del $Hijo2$, depues tomando la sección 2 del $Padre1$ y colocándolo en la sección 2 del $Hijo2$ tenemos:

$Hijo2 = 87655678$

Por lo tanto decimos que los dos hijos son resultado de la cruza de los dos padres.

Mutación: Despues de realizar cierta cantidad de veces el proceso de cruza, se corre el riesgo de caer en un óptimo local o se dice que la población se ha degradado. Por lo que es importante recurrir a la mutación como lo hace la evolución natural. Cuando creamos hijos independientemente de la técnica se deberá copiar la información de los hijos desde los padres. Debido a esto, cada parte del individuo tiene cierta probabilidad de ser mutado. Si un valor es mutado, es reemplazado con un valor factible

para la parte de un gen que llamaremos (alelo)[20].

Potvin [17] propone los siguientes pasos para desarrollar un algoritmo genético:

1. Crear una población inicial de cromosomas P (generación 0).
2. Evaluar la aptitud de cada cromosoma.
3. Seleccione P padres de la población actual a través de selección proporcional (es decir, la probabilidad de selección es proporcional a la aptitud física).
4. Elegir al azar un par de padres para el apareamiento. Cambio de cadenas de bits con el cruce de un punto para crear dos hijos.
5. Proceso de cada descendiente por el operador de mutación, e introduzca el resultado hijos en la nueva población.
6. Repita los pasos 4 y 5 hasta que todos los padres son seleccionados y apareados (descendientes P se crean).
7. Reemplace la población vieja por la nueva.
8. Evaluar la aptitud de cada cromosoma en la nueva población.
9. Ir de nuevo al paso 3 si el número de generaciones es menos que algunos superiores consolidados. De lo contrario, el resultado final es el mejor cromosoma creado durante la búsqueda.

En este proyecto se propone un algoritmo genético que funciona de la siguiente manera:

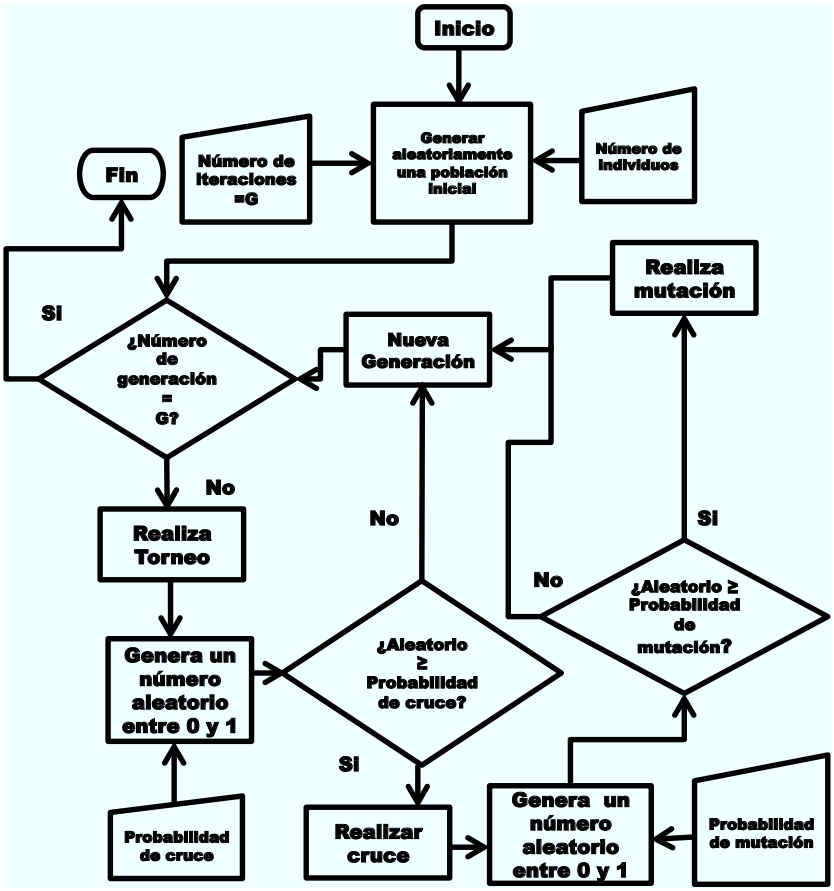

Figura 2.2: Diagrama de Flujo

Ahora ejemplificamos el funcionamiento del AG tomando el caso del problema de 5 ciudades de Winston [35]. Para iniciar se requiere tener una matriz cuadrada que represente el costo de la distancia al viajar de la ciudad i a la ciudad j; se genera una población inicial de un determinado número de individuos con rutas aleatorias, por ejemplo 10 individuos con 5 alelos y su respectiva aptitud (Fitness), como se observa en la Tabla 2.11, a partir del ejemplo de 5 ciudades de Winston [35], que se muestra en la Tabla 2.1.

Tabla 2.11: Tabla PNC

Individuo	Ruta	Costos	Aptitud(Fitness)
1	1-2-3-4-5	132+290+113+196+58	789
2	5-3-1-2-4	303+217+132+201+196	1049
3	3-4-5-2-1	113+196+79+132+217	737
4	2-4-5-1-3	201+196+58+217+290	962
5	3-1-4-2-5	217+164+201+79+303	964
6	5-2-3-1-4	79+210+217+164+196	946
7	2-1-5-3-4	132+58+303+113+201	807
8	3-2-1-5-4	290+132+58+196+113	789
9	1-5-2-4-3	58+79+201+113+217	668
10	5-4-1-3-2	196+164+217+290+79	946

Generamos dos permutaciones aleatorias de tamaño igual al número de individuos para poder llevar a cabo el torneo; por ejemplo la primera permutación $P1 = 6 - 3 - 7 - 8 - 5 - 1 - 2 - 4 - 9 - 10$; depués se obtiene la segunda permutación mediante el desplazamiento de las posiciones en $P1$, por ejemplo desplazando 5 unidades obtenemos $P2 = 2 - 4 - 9 - 10 - 6 - 3 - 7 - 8 - 5 - 1$.

Posteriormente se realiza el torneo poniendo a competir a los individuos de la permutación $P1$ contra los individuos de la permutación 2 que se encuentran en la misma posición. En este caso el individuo 6 con 2; el 3 con 4; 7 con 9; 8 con 10; 5 con 6; 1 con 3; 2 con 7; 4 con 8; 9 con 5; 10 con 1. En donde los vencedores son aquellos que presenten una menor aptitud (distancia más corta) y con ellos se forma una nueva matriz que se muestra en la Tabla 2.12.

Tabla 2.12: Competidores

Competidores	Vencedor	Ruta	Aptitud(Fitness)
6 contra 2	6	5-2-3-1-4	946
3 contra 4	3	3-4-5-2-1	737
7 contra 9	9	1-5-2-4-3	668
8 contra 10	8	3-2-1-5-4	789
5 contra 6	6	5-2-3-1-4	946
1 contra 3	3	3-4-5-2-1	737
2 contra 7	7	2-1-5-3-4	807
4 contra 8	8	3-2-1-5-4	789
9 contra 5	9	1-5-2-4-3	668
10 contra 1	1	1-2-3-4-5	789

Ordenando de menor a mayor formamos con a la población vencedora mostrada en la Tabla 2.13.

Para realizar el cruce entre individuos se genera aleatoriamente un arreglo del tamaño de el número de individuos entre dos renglones por dos columnas. En nuestro ejemplo contamos con 10 individuos por lo que este arreglo es de tamaño 5 renglones por 2 columnas. En donde la columna uno a la que llamaremos columna de POSIBLES PADRES 1 toma para este caso los valores aleatorios: $9, 10, 2, 10, 7$, mientras la columna dos a la que llamaremos POSIBLES PADRES 2 toma los valores aleatorios: $1, 3, 6, 8, 10$ que servirán como posibles padres.

Con anterioridad se determina la probabilidad de cruce con valores entre 0 y 1, por ejemplo (0.6); y se genera un número aleatorio entre cero y uno, en donde si el aleatorio es menor o igual que el número predeterminado (0.6) se realiza el cruce entre la pareja formada por el elemento del primer renglón de la columna POSIBLES PADRES 1 y el elemento de la segunda columna POSIBLES PADRES 2. Para este caso suponga que se obtiene un aleatorio de 0.5 por lo que se toman como padres para cruce a los individuos 9 y 1.

$Padre1 = Individuo9 = 5 - 2 - 3 - 1 - 4$
$Padre2 = Individuo1 = 1 - 5 - 2 - 4 - 3$

Tabla 2.13: Vencedores

Individuo	Ruta	Aptitud(Fitness)
1	1-5-2-4-3	668
2	1-5-2-4-3	668
3	3-4-5-2-1	737
4	3-4-5-2-1	737
5	3-2-1-5-4	789
6	3-2-1-5-4	789
7	1-2-3-4-5	789
8	2-1-5-3-4	807
9	5-2-3-1-4	946
10	5-2-3-1-4	946

Aleatoriamente se generan dos puntos de corte para dividir a ambos padres en tres segmentos cada uno, por ejemplo en la posición dos y en la posición 4 contando siempre de izquierda a derecha. En donde el $hijo1$ que al terminar el cruce será llamado $Individuo9$ se forma con el primer segmento del $Padre1$ (5-2) y el segundo segmento del $Padre2$ (2-4). Finalmente el tercer segmento del $Hijo1$ forma con el tercer segmento del $Padre1$ (4). Para formar el $Hijo2$ que al realizar el cruce será llamado $Individuo1$ se toma el primer segmento del $Padre2$ (1-5), posteriormente se toma al segundo segmento del $Padre1$ (3-1) y finalmente se tomamos al tercer segmento del $Padre2$(3). Formándose así los individuos 9 y 1.

$Hijo1 = Individuo9 = 5 - 2 - 2 - 4 - 4$
$Hijo2 = Individuo1 = 1 - 5 - 3 - 1 - 3$

Posteriormente se identifica la existencia de valores faltantes en cada individuo, si esto sucede entonces existe un subtour. Si no existen valores faltantes los individuos en realidad son tours que formarán la nueva generación. En nuestro caso el $Individuo9$ tiene los elementos faltantes (1-3), mientras que en el $Individuo1$ los elementos faltantes son (2-4), por lo que existe un subtour en cada uno de ellos. Para generar el tour en el $Individuo9$ se colocan los valores faltantes (1-3) en las posiciones en las que

estos se encuentran en el subtour del $Individuo1$ y que además tienen valores repetidos en el subtour del $Individuo9$. De la misma forma para obtener al $Individuo1$ se colocan los valores faltantes (2-4) en las posiciones que estos tienen en el subtour del $Individuo9$ y que además se repiten en el subtour del $Individuo1$. Con lo que se obtiene:

$Individuo9 = 5 - 2 - 3 - 1 - 4$
$Individuo1 = 1 - 5 - 2 - 4 - 3$

El proceso se repite para cada una de las parejas del mismo renglón formada por las columnas de POSIBLES PADRES 1 y POSIBLES PADRES 2. En nuestro caso ahora generamos un número aleatorio para ver si existe cruce entre los individuos 10 y 3. El número generado es de 0.7 por lo que no existirá cruce entre ellos; de tal manera que los individuos 10 y 3 toman sus propios valores.

$Individuo10 = 5 - 2 - 3 - 1 - 4$
$Individuo3 = 3 - 4 - 5 - 2 - 1$

Ahora generamos un número aleatorio para ver si existe cruce entre los individuos 2 y 6. El valor obtenido es 0.6 y puesto que este es un valor igual al determinado para la probabilidad de cruce se llevará a cabo el cruce. Para ello se define como $Padre1$ al individuo del renglón tres de la columna POSIBLES PADRES 1, el cual es el $Individuo2$; y se define como $Padre2$ al individuo del tercer renglón de la columna POSIBLES PADRES 2, el cual es el $Individuo6$. Se generan aleatoriamente los puntos de corte y estos son (1 y 4) para nuestro caso.

$Padre1 = Individuo2 = 1 - 5 - 2 - 4 - 3$
$Padre2 = Individuo6 = 3 - 2 - 1 - 5 - 4$

El $hijo1$ que al terminar el cruce será llamado $Individuo2$ se forma con el primer segmento del $Padre1$ (1) y el segundo segmento del $Padre2$ (2-1-5). Finalmente el tercer segmento del $Hijo1$ forma con el tercer segmento del $Padre1$ (3). Para formar el $Hijo2$ que al realizar el cruce será llamado $Individuo6$ se toma el primer segmento del $Padre2$ (3), posteriormente se toma al segundo segmento del $Padre1$ (5-2-4) y finalmente tomamos al tercer segmento del $Padre2$(4). Formándose así los individuos 2 y 6.

$Hijo1 = Individuo2 = 1 - 2 - 1 - 5 - 3$

$Hijo2 = Individuo6 = 3 - 5 - 2 - 4 - 4$

Como se puede observar los dos hijos obtenidos presentan subtours por lo que debemos identificar cuales son los elementos faltantes en cada individuo. En nuestro caso el $Individuo2$ tiene como elemento faltante al (4), mientras que en el $Individuo6$ tiene como elemento faltante al (1). Para generar el tour en el $Individuo2$ se coloca el valor faltante (4) en la posición en la que se tiene el valor repetido en el subtour del $Individuo2$. De la misma forma para obtener al $Individuo6$ se coloca el valor faltante (1) en la posición que se repite en el subtour. Con lo que se obtiene:

$Hijo1 = Individuo2 = 4 - 2 - 1 - 5 - 3$
$Hijo2 = Individuo6 = 3 - 5 - 2 - 1 - 4$

Posteriormente generamos un número aleatorio para ver si existe cruce entre los individuos 10 y 8, los cuales son los elementos de las columnas POSIBLES PADRES 1 y POSIBLES PADRES 2 respectivamente en el cuarto renglón. El valor obtenido es 0.9 y puesto que este es superior al determinado para la probabilidad de cruce (0.6); no se realizará el cruce. De tal manera que hasta el momento las rutas de los individuos 10 y 8 siguen siendo las mismas como se muestran a continuación:

$Individuo10 = 5 - 2 - 3 - 1 - 4$
$Individuo8 = 2 - 1 - 5 - 3 - 4$

Nuevamente generamos un número aleatorio para ver si existe cruce entre los individuos 7 y 10. El valor obtenido es 0.1 y puesto que este es un valor menor al determinado para la probabilidad de cruce (0.6) se llevará a cabo el cruce. Para ello se define como $Padre1$ al individuo del renglón cinco de la columna POSIBLES PADRES 1, el cual es el $Individuo7$; y se define como $Padre2$ al individuo del quinto renglón de la columna POSIBLES PADRES 2, el cual es el $Individuo10$. Se generan aleatoriamente los puntos de corte y estos son (2 y 3) para nuestro caso.

$Padre1 = Individuo7 = 1 - 2 - 3 - 4 - 5$
$Padre2 = Individuo10 = 5 - 2 - 3 - 1 - 4$

El $hijo1$ que al terminar el cruce será llamado $Individuo7$ se forma con el primer segmento del $Padre1$ (1-2) y el segundo segmento del $Padre2$ (3). Finalmente el tercer segmento del $Hijo1$ forma con el tercer segmento del $Padre1$ (4-5). Para formar el $Hijo2$ que al realizar el cruce será llamado $Individuo10$ se toma el primer

segmento del $Padre2$ (5-2), posteriormente se toma al segundo segmento del $Padre1$ (3) y finalmente tomamos al tercer segmento del $Padre2$(1-4). Formándose así los individuos 7 y 10.

$Hijo1 = Individuo7 = 1 - 2 - 3 - 4 - 5$
$Hijo2 = Individuo10 = 5 - 2 - 3 - 1 - 4$

Posteriormente se identifica la existencia de valores faltantes en cada individuo, si esto sucede entonces existe un subtour. Si no existen valores faltantes los individuos en realidad son tours que formarán la nueva generación como sucede en nuestro caso.

A pesar de la existencia del cruce los individuos 7 y 10 tienen finalmente las mismas rutas que en un principio, por lo que los individuos se han degradado. Es preciso entonces adoptar una probabilidad de mutación que evite la degeneración en toda la población. Para iniciar con ello escribimos la población actual en la Tabla 2.14.

Tabla 2.14: Nueva Pob

Individuo	Ruta	Costos	Aptitud(Fitness)
1	1-5-2-4-3	58+79+201+113+217	668
2	4-2-1-5-3	201+132+58+303+113	807
3	3-4-5-2-1	113+196+79+132+217	737
4	2-4-5-1-3	201+196+58+217+290	962
5	3-2-1-5-4	290+132+58+196+113	789
6	3-5-2-1-4	303+79+132+164+113	791
7	1-2-3-4-5	132+290+113+196+58	789
8	2-1-5-3-4	132+58+303+113+201	807
9	2-4-5-1-3	201+196+58+217+290	962
10	5-2-3-1-4	79+290+217+164+196	946

Predeterminadamente definimos la probabilidad de mutación que para nuestro caso será de 0.1. Posteriormente generamos un aleatorio por cada individuo de toda la población para ser candidato a mutación.

Para el $Individuo1$ se genera un aleatorio que marca 0.2 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo2$ se genera un aleatorio que marca 0.1 por lo que podemos realizar la mutación para lo cual se genera un punto de corte del individuo de manera aleatoria, para nuestro caso el punto de corte vale 3 por lo que el $Individuo2$ se divide en el tercer nodo teniendo: (4-2-1) (5-3) en donde la mutación consiste en intercambiar las posiciones de los dos segmentos obtenidos para formar así al nuevo $Individuo2 = 5-3-4-2-1$

Para el $Individuo3$ se genera un aleatorio que marca 0.5 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo4$ se genera un aleatorio que marca 0.7 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo5$ se genera un aleatorio que marca 0.6 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo6$ se genera un aleatorio que marca 0.9 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo7$ se genera un aleatorio que marca 0.4 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo8$ se genera un aleatorio que marca 0.5 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo9$ se genera un aleatorio que marca 0.2 por lo que no es candidato a ser mutado y su ruta seguirá siendo la misma.

Para el $Individuo10$ se genera un aleatorio que marca 0.1 por lo que podemos realizar la mutación para lo cual se genera un punto de corte del individuo de manera aleatoria, para este caso el punto de corte vale 2 por lo que el $Individuo10$ se divide en el segundo nodo teniendo: (5-2) (3-1-4) en donde la mutación consiste en intercambiar las posiciones de los dos segmentos obtenidos para formar así al nuevo $Individuo10 = 3-1-4-5-2$. Con esto obtenemos la nueva generación mostrada en la Tabla 2.15.

Predeterminadamente se elige el número de iteraciones o generaciones a realizar, es decir este procedimiento se repite cuantas veces se haya decidido. Para este caso en

Tabla 2.15: Nueva Generación

Individuo	Ruta	Costos	Aptitud(Fitness)
1	1-5-2-4-3	58+79+201+113+217	668
2	5-3-4-2-1	303+113+201+132+58	807
3	3-4-5-2-1	113+196+79+132+217	737
4	2-4-5-1-3	201+196+58+217+290	962
5	3-2-1-5-4	290+132+58+196+113	789
6	3-5-2-1-4	303+79+132+164+113	791
7	1-2-3-4-5	132+290+113+196+58	789
8	2-1-5-3-4	132+58+303+113+201	807
9	2-4-5-1-3	201+196+58+217+290	962
10	3-1-4-5-2	217+164+196+79+290	946

donde solamente ejemplificamos una generación obtenemos como mejor resultado al $Individuo1$ el cual tiene la ruta: $1-5-2-4-3$ con una distancia total recorrida de 668 unidades.

Capítulo 3

Implementación del Problema del Agente Viajero

La mayor parte de los problemas de Programación Entera se resuelven en la práctica mediante la técnica de ramificación y acotamiento. Como vimos en el Capitulo 2, estos métodos encuentran la solución óptima de un Problema con Enteros mediante la enumeración exhaustiva de los puntos de una región factible de un subproblema. Según Winston[35]; si se resuelve la relajación del Problema Lineal de un Problema con Enteros puro y obtiene una solución en la cual todas las variables son números enteros, entonces la solución óptima para la relajación del Problema Linea es también la solución óptima del Problema con Enteros.

3.1 Programación entera en Lingo

Una forma de resolver el Problema del Agente Viajero utilizando programación entera es a través del software Lingo, el cual es permite programar el método de resolución al problema consistente en lo siguiente:

1. Declaramos las variables dentro de la instrucción $SETS$ y acontinuación $CITY/1..N/:U$ en donde $CITY$, que es la ciudad del Problema del Agente Viajero, la cual inicia en 1 y termina en N; donde N es el número de elementos que tiene el problema de asignación. Posteriormente se declara $LINK(CITY, CITY):DIST, X$ en donde se indica que la ruta será de $CITY$ a $CITY$ tomando como

el costo o distancia a $DIST, X$. Indicamos la terminación de esta instrucción con $ENDSETS$.

2. Guardamos en $DATA$ Los valores de entrada, es decir, las distancias existentes entre cada elemento del problema de asignación e indicamos la finalización de este con $ENDATA$.

3. La función objetivo consiste en minimizar el producto de la distancia y el costo $MIN = @SUM(LINK : DIST * X)$;

4. Con $@FOR(CITY(K) : @SUM(CITY(I) : X(I, K)) = 1;)$ $@FOR(CITY(K) : @SUM(CITY(J) : X(K, J)) = 1;)$ aseguramos que los resultados obtenidos serán enteros.

5. Finalmente $@FOR(CITY(K) : @FOR(CITY(J)|J\#GT\#1\#AND\#K\#GT\#1 : U(J) - U(K) + N * X(J, K) < N - 1;))$ asegura que ningún resultado será un subtour.

3.2 Algortimo Genético programado en Matlab

Como hemos visto los algoritmos genéticos son heurísticos que permiten obtener resultados aproximados al óptimo con un tiempo computacional relativamente corto por lo que se programa dicho algoritmo en MATLAB (MATrix LABoratory) funcionando de la siguiente manera:

1. Iniciamos creando un aleatorio semilla a través de la instrucción:

   $RandStream.setDefaultStream\ (RandStream('mt19937ar','seed', sum(100 * clock)))$;

2. Creamos la población inicial en una matriz de ceros con $P = zeros(numInd, n+2)$;, en donde el arreglo es de tamaño $numInd$ (número de individuos) por $n+2$ (número de nodos más dos). Además, la matriz $PN = zeros(numInd, n + 2)$ servirá para guardar la población en forma temporal.

3. Las rutas aleatorias son formadas de manera aleatoria desde el renglón $i = 1$, hasta el número de individuos $numInd$ mediante el número aleatorio formado en $randperm(n)$ y guardado en $P(i, 1 : n)$.

4. Se determina el ciclo del algoritmo de acuerdo al número de iteraciones $forj = 1 : numIter$

5. Evaluamos las distancias para cada uno de los individuos con $P(i, n+1) = costo(P(i, 1:n), M)$

6. Para realizar el TORNEO definimos una función objetivo con sus respectivas variables $PN = torneo(Pob,\ num, num)$. Donde: Pob = Población original; num = Número de individuos; $numc$ =Número de ciudades.

7. Posteriormente se genera aleatoriamente P_1 = que es una permutación de $rango = num$.

8. Se realiza un corrimiento aleatorio para la permutación: $corr = round(rand * (num - 2)) + 1$

   Donde *corr* es un valor redondeado a partir de un aleatorio fraccionario de rango $(num - 2)) + 1$.

9. Se forma la segunda permutación con $P_2 = circshift(P_1, [0, corr])$

   Donde P_2 se origina a partir de P_1 = recorriendo cero posiciones en las filas [0] y *corr* posiciones en las columnas.

10. $PN = zeros(num, numc + 2)$

    Donde PN es una matriz de ceros de tamaño $(num, numc+2)$, el cual permitirá inicializar la posición de los resultados.

    El torneo se lleva a cabo de la siguiente manera:

11. Revisar desde la fila 1 hasta *num* a cada individuo, si el valor de la distancia del elemento fila 1 es mayor que la distancia de P_2 en la misma columna. Mientras *aux* toma el valor de P_2. Los resultados vencedores son guardados en $PN(i, 1 : numc + 1) = Pob(aux, 1 : numc + 1)$ mediante las instrucciones:

    $For\ i = 1 : num\ aux = P_1(i); if Pob(aux, numc+1)\ aux = P_2(i)\ end\ PN(i, 1 : numc + 1) = Pob(aux, 1 : numc + 1);\ end\ return\ end$

12. Se realiza el cruce entre los padres:

    function $PNC = cruce(PN, numInd, n, probCruce)$

    El cual permite el cruce de dos posibles rutas con probabilidad de cruce predeterminada.

13. La lista de posiciones se genera mediante un arreglo llamado Pos desde uno hasta numInd: $Pos = 1 : numInd$;

14. Inicializa la lista con:

    $lista = zeros(numInd/2, 2)$; En donde la lista es una matriz de ceros del tamaño de $numInd/2$ filas por 2 columnas.

15. Ahora se inicializa la población en donde PNC genera una matriz de $numInd$ por $n + 2$ columnas de ceros.

    $PNC = zeros(numInd, n + 2)$; PNC toma valores de PN con la instrucción: $PNC(:, :) = PN(:, :)$;

16. Para la lista de cruces se toma para i, desde 1 hasta $numInd/2$ con la instrucción: $for\ i = 1 : numInd/2$.

    Y para j desde 1 hasta 2 con la instrucción: $for\ j = 1 : 2$

    Se nombra como ln al tamaño de la posición (Pos) con la instrucción: $ln = length(pos)$;

    Se genera un número aleatorio entero de rango (ln) y es guardado en Aux mediante la instrucción: $aux = randi(ln)$;

    Posteriormente se hace una lista de valores de rango (ln) con la instrucción: $lista(i, j) = pos(aux)$;

17. Ahora generamos los arreglos en donde serán guardados los resultados del cruce. Para ello se elabora un arreglo de ceros llamada $listaC$. La cual toma los valores de $zeros(1, 2)$ indicando una fila con dos columnas de ceros.

Se nombra como $hijo1$ a $h1 = zeros(1, n)$, que es el arreglo de ceros de una fila por n columnas.

Y se nombra como $hijo2$ a $h1 = zeros(1, n)$, que es el arreglo de ceros de una fila por n columnas.

18. Se genera un número aleatorio *probCruce* para ver la probabilidad de la realización de un cruce entre los padres. Para ello se utiliza la instrucción: for $i = 1 : numInd/2$. Verificando sí el número aleatorio es menor que la probabilidad de cruce $if(rand \leq probCruce)$. Sí esto es así, entonces se procede a seleccionar a los padres: El $Padre1$ $P_1 = PN(lista(1, 1), :)$, y el $Padre2$ $P_2 = PN(lista(1, 2), :)$.

19. Se generan las posiciones de cruce mediante un arreglo llamado *pos* que toma valores desde 1 hasta $n-1$ para asegurar que durante el cruce existan dos cortes. $pos = 1 : n - 1$ y para j desde 1 hasta 2 $forj = 1 : 2$ $ln = lenght(pos)$, en donde ln es el tamaño de la posición *pos*. Después se genera un aleatorio con distribución entera cuyo rango es ln.

    El número aleatorio de rago (tamaño de cada individuo) es guardado en la lista $listaC(1, j)$.

20. Procedemos a revisar el orden de la lista; sí $C(1, 1) > listaC(1, 2)$ se intercambian las posiciones. Por lo tanto : $if(listaC(1, 1) > listaC(1, 2))$ $aux = listac(1, 1)$, $listaC(1, 1) = listaC(1, 2)$ y $listaC(1, 2) = aux$.

21. Se lleva a cabo el cruce de los padres:

    El $hijo1$ en la posición $(1, 1 : listaC(1, 1))$ toma los valores del $Padre1$ en la posición $(1, 1 : listaC(1, 1))$.

    El $hijo1$ en la posición $(1, 1 : listaC(1, 1) + 1)$ toma los valores del $Padre2$ en la posición $(1, listaC(1, 1) + 1 : listaC(1, 2))$.

    El $hijo1$ en la posición $(1, listaC(1, 2) + 1 : n)$ toma los valores del $Padre1$ en la posición $(1, listaC(1, 2) + 1 : n)$.

El $hijo2$ en la posición $(1, 1 : listaC(1, 1))$ toma los valores del $Padre2$ en la posición $(1, 1 : listaC(1, 1))$.

El $hijo2$ en la posición $(1, listaC(1, 1) + 1 : listaC(1, 2))$ toma los valores del $Padre1$ en la posición $(1, listaC(1, 1) + 1 : listaC(1, 2))$.

El $hijo2$ en la posición $(1, listaC(1, 2) + 1 : n)$ toma los valores del $Padre2$ en la posición $(1, listaC(1, 2) + 1 : n)$.

22. Buscamos la existencia de algún subtour para ello utilizamos la instrucción: $In = intersect(h1(1, lista(1, 1) + 1 : listaC(1, 2)), h2(1, listaC(1, 1) + 1 :$ $listaC(1, 2))$; la cual permite detectar la intersección de $h1$ con $h2$, de tal manera que si algún valor falta, esto indica que existen subtours.

23. Ahora se extraen los valores de $h1$ que no están en In mediante la instrucción:

    $S_1 = setdiff(h1(1, listaC(1, 1) + 1 : listaC(1, 2)), In)$ y guardamos el tamaño de $S1$: $aux = length(S1)$.

24. Se obtienen los valores del $hijo2$ que no están en In en el segmento determinado en $listaC(1, 1) + 1$ hasta $C(1, 2)$con la instrucción:

    $S_2 = setdiff(h2(1, listaC(1, 1) + 1 : listaC(1, 2)), In.$

25. Se busca la posición de la columna j que tiene S_1 en P_1 con la instrucción:

    $auxp1 = find(p1 == S1(j)$ y $auxp2 = find(p2 == S2(j)$; $h1$ toma los valores en la posición $(1, auxp1)$ a partir de $P2$ en la posición $(1, auxp2)$; mientras que $h2$ toma los valores en la posición $(1, auxp2)$ a partir de $p1$ en la posición $(1, auxp1)$.

26. Posteriormente se sobre escribe el arreglo final de $h1$ y $h2$ en $PNC(lista(1, 1), 1 : n)$ y $PNC(lista(1, 2), 1 : n)$ respectivamente.

27. Posteriormente para realizar la mutación escribimos a la función $functionPNM = mutacion(PNC, numInd, n, probMut)$.

28. Inicializamos una lista de ceros $lista = zeros(1, 2)$.

29. Se propone un arreglo de listas mutadas mediante $PNM = zeros(numInd, n+2); PNM(:,:) = PNC(:,:)$.

30. Recorremos las rutas a través de $fori = 1 : numInd$.

31. Generando una probabilidad de mutación aleatoria $if rand <= probMut$.

32. Llamamos Pos al arreglo de 1 renglón hasta n que es el número de nodos. $pos = 1 : n$.

33. Para crear las posiciones aleatorias para la mutación tenemos tomamos para las columnas de 1 a 2 $forj = 1 : 2$.

34. Generamos un auxiliar aleatorio de rango del tamaño de Pos a través de la instrucción: $aux = randi(length(pos))$.

35. En donde la lista(1,j) toma los valores del auxiliar $lista(1, j) = pos(aux)$.

36. Los valores son guardados en en arreglo pos (aux) mediante la instrucción: $pos(aux) = [\ ]$.

37. Para realizar la mutación se nombra al primer segmento del individuo como un auxiliar con la instrucción: $aux = PNC(i, lista(1, 1))$.

38. En la posición del primer segmento del individuo a mutar se colocan los valores del segundo segmento de este individuo mediante la instrucción:

    $PNM(i, lista(1, 1)) = PNM(i, lista(1, 2))$.

39. Finalmente los elementos de la segunda sección para el nuevo individuo mutado se toman a partir del elemento auxiliar que originalmente fue la primera sección a través de la instrucción $PNM(i, lista(1, 2)) = aux;$. Intercambiando así las posiciones lo cual es la esencia de la mutación.

En el capítulo 2 y sección titulada Algoritmos Genéticos se muestra un ejemplo del funcionamiento del código presentado aquí.

3.3 Algorimos Genéticos determinísticos y aleatorios

Se estructuraron dos algoritmos genéticos a los cuales llamamos determinístico y aleatorio.

El AG determinístico es un algoritmo que, es completamente predictivo si se conocen sus entradas. Es decir, si se conocen las entradas del algoritmo siempre producirá la misma salida, y la secuencia interna pasará por la misma secuencia de estados. El AG determinístico comprende los siguientes pasos:

1. Se genera una lista aleatoria del tamaño del número de individuos que forman la población:

10	1	3	5	2	8	7	6	9	4

2. Se genera un número aleatorio para correr la lista generada en el paso anterior:

6	9	4	10	1	3	5	2	8	7

3. Compiten los dos individuos de la lista, se compara el fitness y se selecciona al menor.

4. Se forma la nueva población:

10	9	4	5	1	8	7	6	9	7

El AG Aleatorio (Cost weighting), es llamado así puesto que para la generación de la nueva población requiere de un número aleatorio, el cual comprende los siguientes pasos:

1. Se ordena la población con referencia al fitness de menor a mayor como se observa en la Tabla 3.1.

Tabla 3.1: Cost weighting 1

11631
11860
11872
12097
12359
12363
12588
13477

2. El mejor individuo se pasa directamente a la nueva población.

3. Se resta cada una de las distancias del mayor fitness, como en la Tabla 3.2.

Tabla 3.2: Cost weighting 2

1846
1617
1605
1380
1118
1114
889
0
Suma=9569

4. Se divide el resultado de la resta entre el total de la suma de las distancias (9569) y se realiza la suma acumulada, como en la Tabla 3.3.

Tabla 3.3: Cost weighting 3

1846/9569	0.19291462	0.19291462
1617/9569	0.16898317	0.36189779
1605/9569	0.16772913	0.52962692
1380/9569	0.1442157	0.67384262
1118/9569	0.11683562	0.79067823
1114/9569	0.1164176	0.90709583
889/9569	0.09290417	1
0/9569	0	1

5. Se genera un número aleatorio por ejemplo (0.3014); en este caso puesto que el aleatorio es menor que (0.3618) se elige al segundo, y así sucesivamente hasta completar la nueva población, como en la Tabla 3.4.

Tabla 3.4: Cost weighting 4

11631	11860	12359

A través de los lenguajes mostrados es posible comenzar a experimentar sobre el Problema del Agente Viajero en donde como podemos observar los lenguajes de programación siguen diferentes pasos, todos con el objetivo de optimizar.

Capítulo 4

Experimentación de la solución del Problema del Agente Viajero con Algoritmos Genéticos

Una vez que se ha explicado el funcionamiento que tienen los AG para resolver el PAV, aplicamos la metodología a diferentes problemas.

Para ello indicaremos la gráfica de promedio de valores obtenidos durante cada generación, a la cual llamaremos en adelante Aptitud Generacional. Mientras que también identificamos al menor valor obtenido durante cada generación, al cual llamaremos en adelante Fitness Generacional.

4.1 Problema con 5 ciudades (2005) [35]

A continuación se experimenta el PAV expuesto en Winston [35], el cual consiste en 5 ciudades de acuerdo a la matriz que se muestra en la Tabla 2.1.

La (Fig. 4.1) muestra la ruta encontrada a través de programación entera con el apoyo del programa en software Lingo descrito en el Capitulo 3; proporcionando un valor de 668 unidades para la totalidad del recorrido y que además es un valor óptimo para el problema.

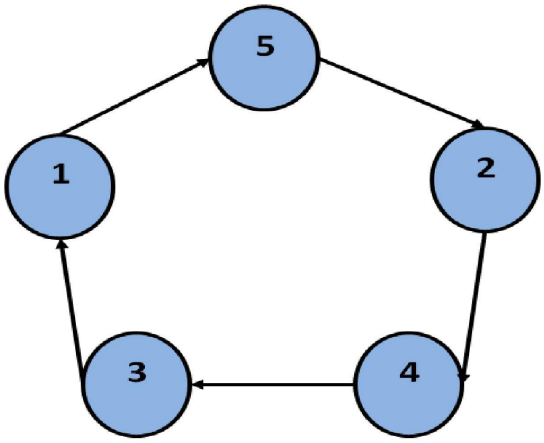

Figura 4.1: Ruta 5x5 Winston

El mismo problema fue aplicado para ser resuelto a través del AG determinístico descrito en el Capitulo 3, con parámetros de 100 individuos, 100 iteraciones, probabilidad de cruzamiento de 0.6 y una probabilidad de mutación del 0.1; obteniendo la misma ruta que en la Programación Entera (Fig. 4.1). La gráfica de Aptitud Generacional (Fig. 4.2), muestra el comportamiento de dichos promedios, en donde observamos una continua variación puesto que como podemos ver en la (Fig. 4.3), el mejor Fitness fue encontrado en la primera generación.

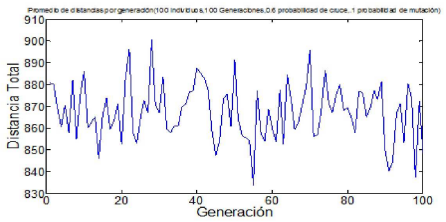

Figura 4.2: Aptitud Generacional 5X5 Winston

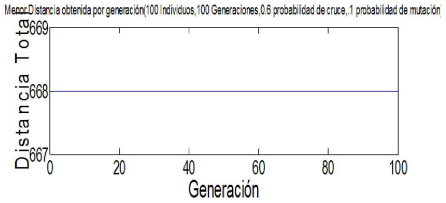

Figura 4.3: Fitness Generacional 5X5 Winston

4.2 Problema con 10 ciudades de Gerhard (2006) [28]

Se toma como ejemplo a la Matriz 10 X 10 Gerhard (2006) [28] en nuestra búsqueda de los parámetros que mejor resultados presenten al momento de realizar la experimentación del PAV.

Tabla 4.1: Matriz 10 X 10 Gerhard (2006) [28]

	1	2	3	4	5	6	7	8	9	10
1	0	107	241	190	124	80	316	76	152	157
2	107	0	148	137	88	127	336	183	134	95
3	241	148	0	374	171	259	509	317	217	232
4	190	137	374	0	202	234	222	192	248	42
5	124	88	171	202	0	61	392	202	46	160
6	80	127	259	234	61	0	386	141	72	167
7	316	336	509	222	392	386	0	233	438	254
8	76	183	317	192	202	141	233	0	213	188
9	152	134	217	248	46	72	438	213	0	206
10	157	95	232	42	160	167	254	188	206	0

A partir de la Tabla 4.1, y mediante programación entera a través de Lingo (Capítulo 3) se obtuvo una ruta de 1185 unidades en donde la secuencia es mostrada en (Fig. 4.4).

El AG determínistico utilizó parámetros de 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 1/10; con un tiempo computacional de 3 seg. Encontrando una ruta de 1185 unidades, en donde la secuencia se puede observar en (Fig. 4.4).

La gráfica de los promedios de la función objetivo en cada una de las generaciones que se presenta en (Fig. 4.5); muestra el comportamiento del AG en lo referente a la Aptitud Generacional.

En cuanto a la gráfica de (Fig. 4.6) muestra como el mejor Fitness Generacional fue encontrado antes de 20 iteraciones.

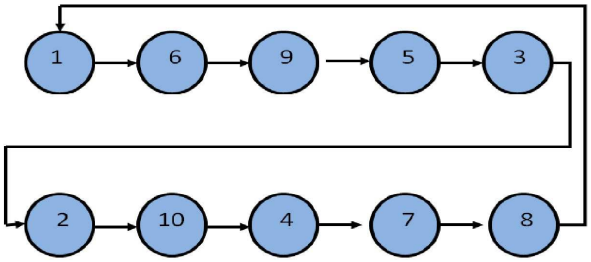

Figura 4.4: Ruta matriz 10X10 Gerhard (PE)

Para la experimentación del AG aleatorio se utilizan parámetros de 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 1/10; con un tiempo computacional de 3 segundos, encontrando una ruta con 1185 unidades de recorrido, en donde la secuencia se muestra en (Fig. 4.7).

La gráfica de Aptitud Generacional de la función objetivo en cada una de las generaciones se presenta en (Fig. 4.8), en donde se observa la mejora existente de generación en generación.

Mientras que la gráfica del Fitness Generacional (Fig. 4.9), permite observar que el mejor resultado fue obtenido antes de la generación número 20.

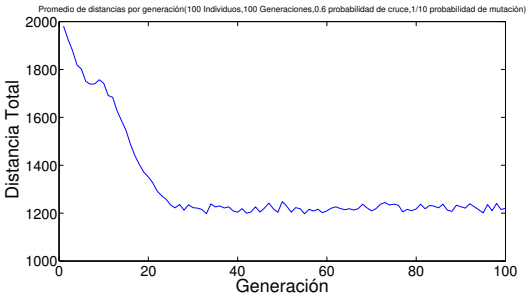

Figura 4.5: Aptitud Generacional Matriz 10X10 Determinista

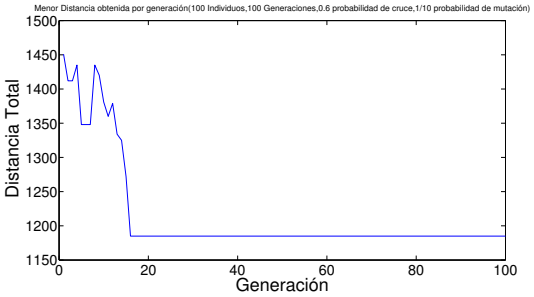

Figura 4.6: Fitness Generacional Matriz 10X10 Determinista

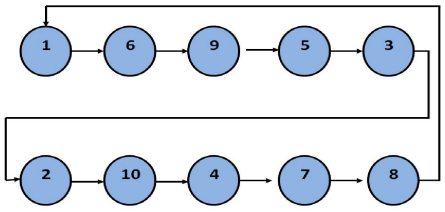

Figura 4.7: Ruta Matriz 10X10 Gerhard Aleatorio

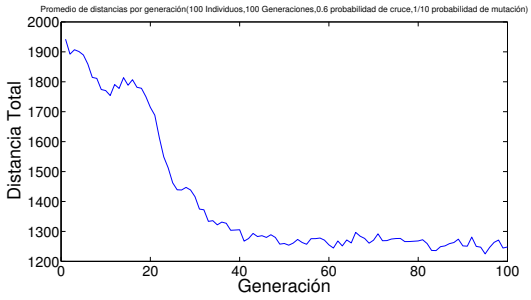

Figura 4.8: Aptitud Generacional Matriz 10X10 Gerhard Aleatorio

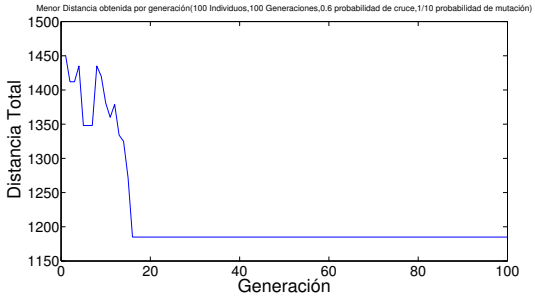

Figura 4.9: Fitness Generacional Matriz 10X10 Gerhard Aleatorio

4.3 Problema con 10 ciudades de Rodríguez (2003) [34]

Rodríguez [34] alumno del Instituto Politecnico Nacional en 2003 escribió una tesis en la cual planteaba la utilización de los AG para resolver el PAV, utilizando distancias de 10 ciudades del territorio mexicano: 1.- Cd. Victoria, 2.- Monterrey, 3.- Durango, 4.- Chihuahua, 5 Culiacán, 6 Guanajuato, 7 Chilpancingo, 8 Veracruz, 9 Campeche, 10 Tuxtla. Con ello se formuló la matriz presentada en la Tabla 4.2.

Tabla 4.2: Matriz 10 ciudades de Rodríguez (2003) [34]

	1	2	3	4	5	6	7	8	9	10
1	0	287	822	1131	1361	4804	975	749	1610	1522
2	287	0	615	818	1154	662	1271	1036	1897	1809
3	822	615	0	709	539	601	1295	1327	2227	1921
4	1131	818	709	0	1248	1143	1747	1854	2497	2473
5	1361	1154	539	1248	0	1033	1584	1797	2610	2391
6	4804	662	601	1143	1033	0	633	866	1677	1460
7	975	1271	1295	1747	1584	633	0	716	1572	1089
8	749	1036	1327	1854	1797	866	716	0	861	773
9	1610	1897	2227	2497	2610	1677	1572	861	0	674
10	1522	1809	1921	2473	2391	1460	1089	773	674	0

Rodríguez [34] encontró una solución con una distancia total recorrida de 8914 Km a través de AG, en donde la secuencia se muestra en la (Fig. 4.10).

Utilizando programación entera con el software Lingo, y en la búsqueda por solucionar el PAV encontramos una distancia total recorrida de 7392 Km de acuerdo con la secuencia (Fig. 4.11).

A través de los AG determínisticos con parámetros de 10 individuos, 100 iteraciones, 0.6 como probabilidad de cruce y 0.1 como probabilidad de mutación; encontramos una solución con una distancia total recorrida de 7392 Km de acuerdo a la secuencia mostrada en la (Fig. 4.11).

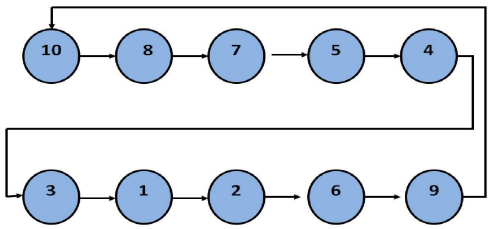

Figura 4.10: Secuencia Tesis Rodríguez [34]

Como se observa el resultado obtenido a través de programación entera y también por AG determínisticos es mejor que el resultado que Rodríguez [34] encontro en su tesis.

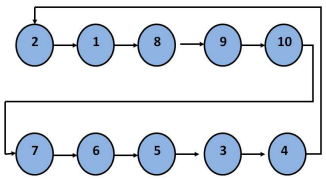

Figura 4.11: Secuencia Genético Rodríguez [34]

4.4 Problema con 15 ciudades de Gerhard (2006) [28]

Se experimenta con un PAV de 15 ciudades a visitar de un problema expuesto por Gerhard [28] mostrada en la Tabla 4.3.

Tabla 4.3: Matriz 15 X 15 Gerhard (2006) [28]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	107	241	190	124	80	316	76	152	157	283	133	113	297	228
2	107	50	148	137	88	127	336	183	134	95	254	180	101	234	175
3	241	148	0	374	171	259	509	317	217	232	491	312	380	391	412
4	190	137	374	50	202	234	222	192	248	42	117	287	79	107	38
5	124	88	171	202	0	61	392	202	46	160	319	112	163	322	240
6	80	127	259	234	61	0	386	141	72	167	351	55	157	331	272
7	316	336	509	222	392	386	0	233	438	254	202	439	235	254	210
8	76	183	317	192	202	141	233	0	213	188	272	193	131	302	233
9	152	134	217	248	46	72	438	213	0	206	365	89	206	368	286
10	157	95	232	42	160	167	254	188	206	0	159	220	57	149	80
11	283	254	491	117	319	351	202	272	365	159	0	404	176	106	79
12	133	180	312	287	112	55	439	193	89	220	404	0	210	384	225
13	113	101	280	79	163	157	235	131	209	57	156	210	0	186	117
14	297	234	391	107	322	331	254	302	368	149	106	384	186	0	69
15	228	175	412	38	240	272	210	233	286	80	79	325	117	69	0

Los resultados obtenidos por programación entera desarrollada en Lingo para el Problema del Agente Viajero de la Tabla 4.3 muestran la ruta de la (Fig. 4.12) en donde la función objetivo es tiene un valor de 1513 unidades.

El AG determínistico utilizó parámetros de 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 0.05; con un tiempo computacional de 5 segundos, encontrando una ruta de 1692 de acuerdo a la secuencia (Fig. 4.13).

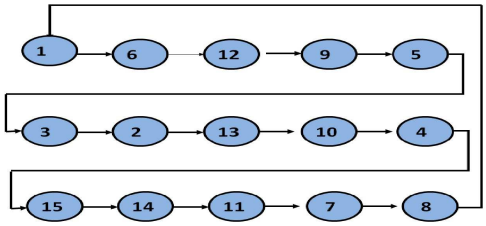

Figura 4.12: Ruta Matriz 15 X 15 Gerhard (PE)

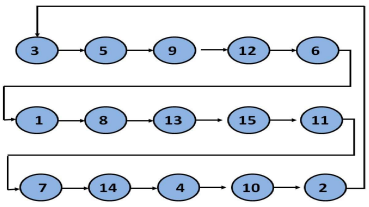

Figura 4.13: Ruta Matriz 15 X 15 Gerhard Determinista

La gráfica de Aptitud Generacional mostrada en (Fig. 4.14), permite identificar la mejora que proporciona el algoritmo de acuerdo a la gráfica decreciente.

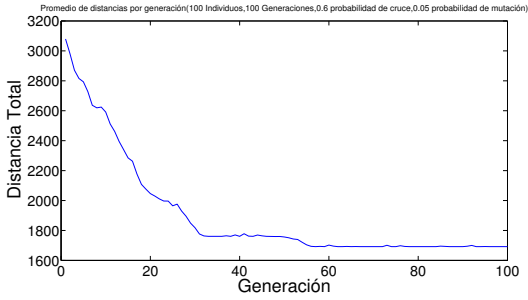

Figura 4.14: Aptitud Generacional determinista

La gráfica del Fitness Generacional en (Fig. 4.15) documenta que el mejor resultado fue alcanzado después de la iteración número 45.

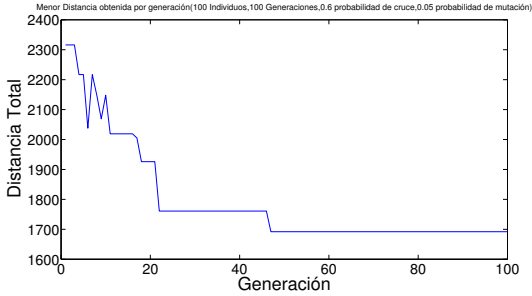

Figura 4.15: Fitness Generacional Matriz 15X15 Deter

El AG aleatorio utilizó los siguientes parámetros de 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 0.05; con un tiempo computacional de 5 segundos, encontrando una distancia total recorrida de 1757 unidades de acuerdo a la ruta (Fig. 4.16).

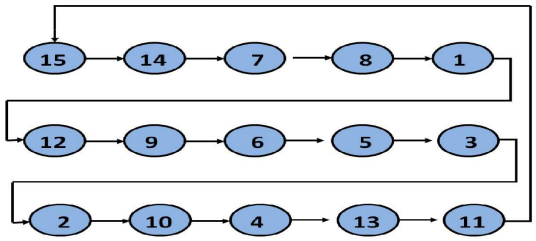

Figura 4.16: Ruta Matriz 15 X 15 Gerhard Aleatorio

El AG aleatorio presenta la Aptitud Generacional de la (Fig. 4.17) que muestra como el algoritmo encuentra mejores resultados durante cada generación puesto que la gráfica es decreciente.

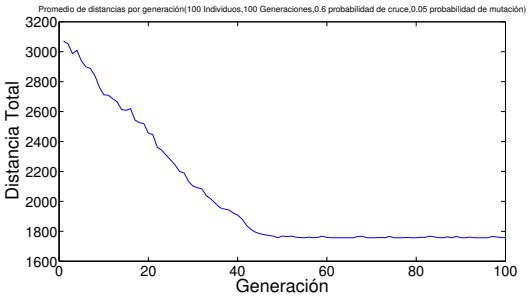

Figura 4.17: Aptitud Generacional Matriz 15 X 15 Gerhard Aleatorio

El Fitness Generacional (Fig. 4.18) del algoritmo define que el mejor resultado obtenido se encontró y en las cercanías de la generación 40.

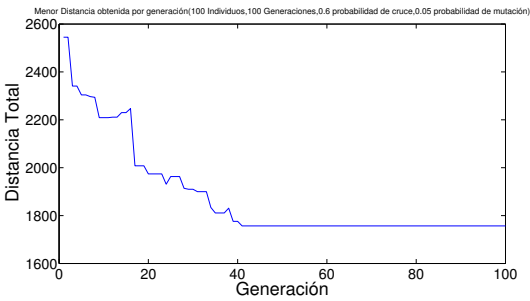

Figura 4.18: Fitness Generacional Matriz 15 X 15 Gerhard Aleatorio

4.5 Problema con 20 ciudades Gerhard (2006) [28]

La Tabla 4.4 Matriz 20 X 20 Gerhard (2006) [28] es utilizada para probar los AG en comparación con la programación entera.

Tabla 4.4: Matriz 20 X 20 Gerhard (2006) [28]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	107	241	190	124	80	316	76	152	157	283	133	113	297	228	129	348	276	188	150
2	107	0	148	137	88	127	336	183	134	95	254	180	101	234	175	176	265	199	182	67
3	241	148	0	374	171	259	509	317	217	232	491	312	380	391	412	349	422	356	355	204
4	190	137	374	0	202	234	222	192	248	42	117	287	79	107	38	121	152	86	68	70
5	124	88	171	202	0	61	392	202	46	160	319	112	163	322	240	232	314	287	238	155
6	80	127	259	234	61	0	386	141	72	167	351	55	157	331	272	226	362	296	232	164
7	316	336	509	222	392	386	0	233	438	254	202	439	235	254	210	266	154	282	321	298
8	76	183	317	192	202	141	233	0	213	188	272	193	131	302	233	98	234	289	177	216
9	152	134	217	248	46	72	438	213	0	206	365	89	206	368	286	278	360	333	284	201
10	157	95	232	42	160	167	254	188	206	0	159	220	57	149	80	132	193	127	100	28
11	283	254	491	117	319	351	202	272	365	159	0	404	176	106	79	161	165	141	95	187
12	133	180	312	287	112	55	439	193	89	220	404	0	210	384	225	279	415	349	287	217
13	113	101	280	79	163	157	235	131	209	57	156	210	0	186	117	75	231	165	81	85
14	297	234	391	107	322	331	254	302	368	149	106	384	186	0	69	191	59	35	125	167
15	228	175	412	38	240	272	210	233	286	80	79	325	117	69	0	122	122	56	56	108
16	129	176	349	121	232	226	187	98	278	132	161	279	75	191	122	50	244	178	66	160
17	348	265	422	152	314	362	313	344	360	193	165	415	231	59	122	244	0	66	178	198
18	276	199	356	86	287	296	266	289	333	127	141	349	165	35	56	178	66	50	112	132
19	188	182	355	68	238	232	154	177	284	100	96	285	81	125	56	66	178	112	0	128
20	150	67	204	70	155	164	282	216	201	28	187	217	85	167	108	160	198	132	128	0

La ruta del PAV con 20 ciudades muestra la secuencia en la (Fig. 4.19) obtenida a través de programación entera (Lingo) en donde la distancia recorrida por el Agente Viajero es de 1688.

El AG determínistico utilizó los siguientes parámetros: 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 1/20; con un tiempo

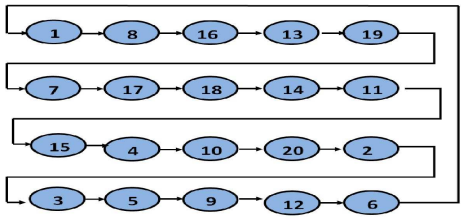

Figura 4.19: Ruta Matriz 20X20 PE

computacional de 7 segundos, recorriendo una distancia de 1700 de acuerdo a la secuencia de la (Fig. 4.20).

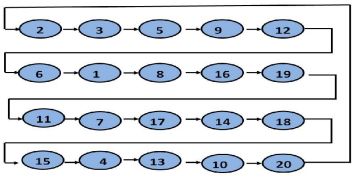

Figura 4.20: Ruta Matriz 20x20 determinista

El AG determínistico para 20 ciudades presenta una Aptitud Generacional mostrada en la (Fig. 4.21), en donde se observa una mejora en cada una de las generaciones que transcurren.

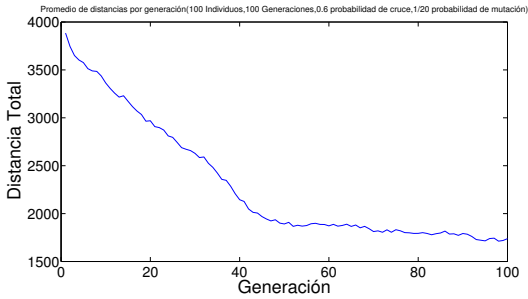

Figura 4.21: Aptitud Generacional Matriz 20x20 determinista

Así también el Fitness Generacional de la (Fig. 4.22) en la cual se muestra como disminuye su valor en cada generación mejorando en cada una de ellas.

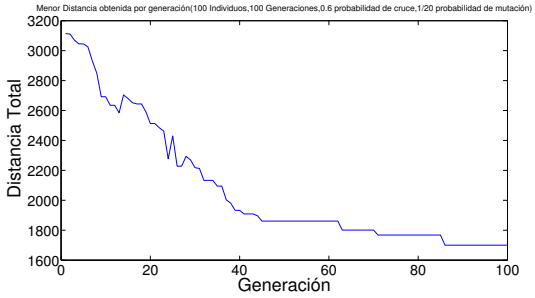

Figura 4.22: Fitness Generacional Matriz 20x20 determinista

El AG aleatorio para 20 ciudades utilizó los siguientes parámetros: 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 1/20; con un tiempo computacional de 7 segundos, encontrando una distancia recorrida de 1854 unidades de acuerdo a la ruta presentada en la (Fig. 4.23).

El AG aleatorio para 20 ciudades obtiene una Aptitud Generacional en la (Fig. 4.24) que presenta la mejora en cada una de las generaciones.

Mientras que el Fitness Generacional de la (Fig. 4.25) permite observar como el mejor valor fue alcanzado cerca de la generación número 60.

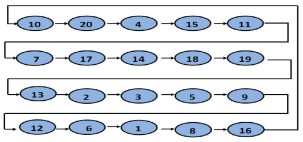

Figura 4.23: Ruta Matriz 20x20 Gerhard Aleatorio

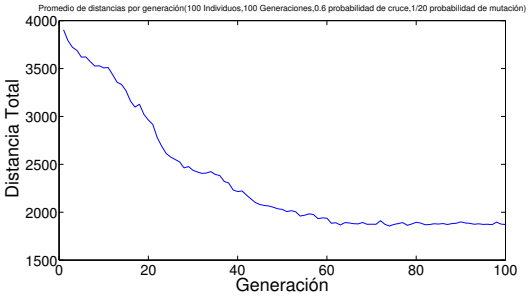

Figura 4.24: Aptitud Generacional Matriz 20x20 Gerhard Aleatorio

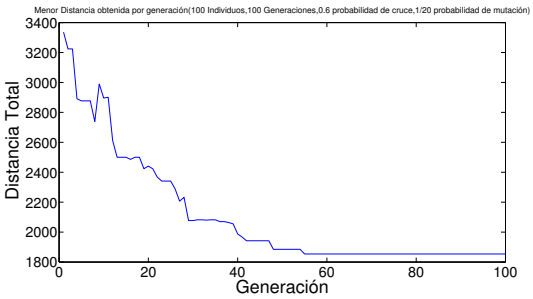

Figura 4.25: Fitness Generacional Matriz 20x20 Gerhard Aleatorio

4.6 Problema con 21 ciudades Gerhard (2006) [28]

Hasta el momento se ha tenido éxito al conseguir un valor cercano al óptimo y en algunos casos alcanzando al resultado óptimo en los problemas presentados anteriormente. Ahora experimentamos con el PAV propuesto en Matriz 21 X 21 Gerhard (2006) [28] el cual se muestra en la Tabla 4.5.

Tabla 4.5: Matriz 21 X 21 Gerhard (2006) [28]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	107	241	190	124	80	316	76	152	157	283	133	113	297	228	129	348	276	188	150	65
2	107	0	148	137	88	127	336	183	134	95	254	180	101	234	175	176	265	199	182	67	42
3	241	148	0	374	171	259	509	317	217	232	491	312	380	391	412	349	422	356	355	204	182
4	190	137	374	0	202	234	222	192	248	42	117	287	79	107	38	121	152	86	68	70	137
5	124	88	171	202	0	61	392	202	46	160	319	112	163	322	240	232	314	287	238	155	65
6	80	127	259	234	61	0	386	141	72	167	351	55	157	331	272	226	362	296	232	164	85
7	316	336	509	222	392	386	0	233	438	254	202	439	235	254	210	266	154	282	321	298	168
8	76	183	317	192	202	141	233	0	213	188	272	193	131	302	233	98	234	289	177	216	141
9	152	134	217	248	46	72	438	213	0	206	365	89	206	368	286	278	360	333	284	201	111
10	157	95	232	42	160	167	254	188	206	0	159	220	57	149	80	132	193	127	100	28	95
11	283	254	491	117	319	351	202	272	365	159	0	404	176	106	79	161	165	141	95	187	254
12	133	180	312	287	112	55	439	193	89	220	404	0	210	384	225	279	415	349	287	217	138
13	113	101	280	79	163	157	235	131	209	57	156	210	0	186	117	75	231	165	81	85	92
14	297	234	391	107	322	331	254	302	368	149	106	384	186	0	69	191	59	35	125	167	255
15	228	175	412	38	240	272	210	233	286	80	79	325	117	69	0	122	122	56	56	108	175
16	129	176	349	121	232	226	187	98	278	132	161	279	75	191	122	50	244	178	66	160	161
17	348	265	422	152	314	362	313	344	360	193	165	415	231	59	122	244	0	66	178	198	286
18	276	199	356	86	287	296	266	289	333	127	141	349	165	35	56	178	66	50	112	132	220
19	188	182	355	68	238	232	154	177	284	100	96	285	81	125	56	66	178	112	0	128	167
20	150	67	204	70	155	164	282	216	201	28	187	217	85	167	108	160	198	132	128	0	88
21	65	42	182	137	65	85	321	141	111	95	254	138	92	255	175	161	286	220	167	88	0

Gerhard [28] obtuvo una distancia de 2042 unidades de acuerdo a la secuencia presentada en la (Fig. 4.26).

En la búsqueda de la ruta más corta de la matriz presentada en la Tabla 4.5, el software Lingo no logró realizar la corrida.

El AG determínistico para resolver el PAV con 21 ciudades utilizaron los siguientes parámetros: 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 0.05; con un tiempo computacional de 8 segundos. Encontrando una distancia recorrida de 2042 unidades de acuerdo a la secuencia presentada en la (Fig. 4.26).

El AG determínistico muestra una Aptitud Generacional adecuada, en donde podemos observar la funcionalidad del algoritmo a través de la (Fig. 4.27).

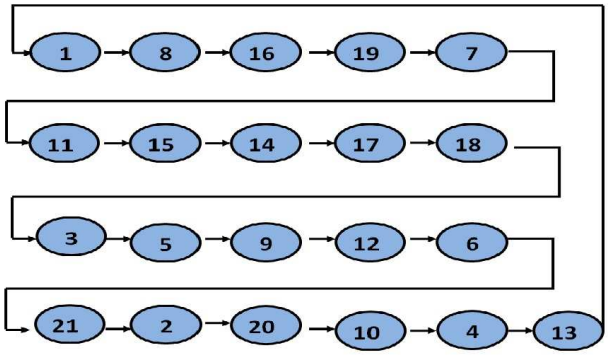

Figura 4.26: Ruta Matriz 21X21 Gerhard Determinista

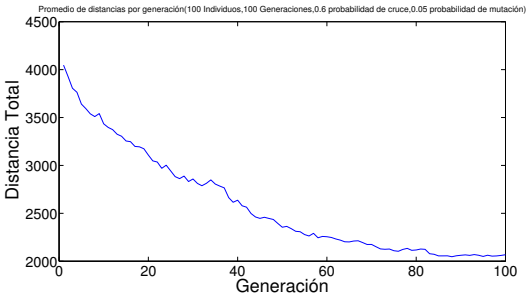

Figura 4.27: Aptitud Generacional Matriz 21X21 Determinista

Mientras que el Fitness Generacional muestra mejoras de generación en generación, alcanzando su mejor valor en las cercanías de la generación número 80, como lo podemos constatar en la (Fig. 4.28).

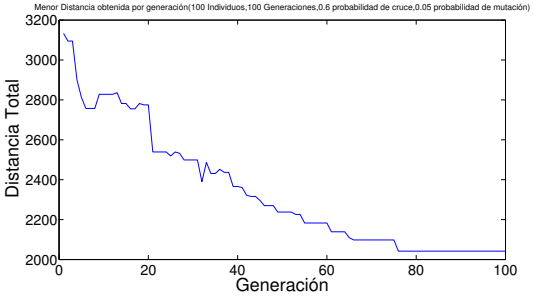

Figura 4.28: Fitness Generacional Matriz 21X21 Gerhard Determinista

Posteriormente experimentamos con el AG aleatorio utilizando los siguientes parámetros: 100 individuos, 100 generaciones, probabilidad de cruce de 0.6, probabilidad de mutación de 0.05; con un tiempo computacional de 8 segundos, encontrando una distancia de recorrido de 2077 unidades de acuerdo a la secuencia mostrada en la (Fig. 4.29).

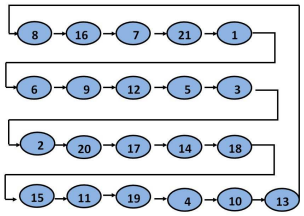

Figura 4.29: Ruta Matriz 21X21 Gerhard Aleatorio

El AG aleatorio para resolver el PAV de 21 ciudades muestra un comportamiento en

su Aptitud Generacional (Fig. 4.30), en donde nuevamente observamos la mejora en cada generación.

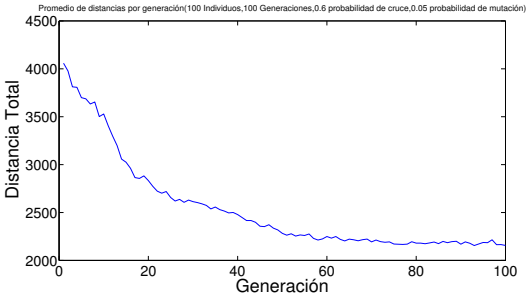

Figura 4.30: Aptitud Generacional Matriz 21X21 Gerhard Aleatorio

El Fitness Generacional puede observarse en la (Fig. 4.31), comprobando que el comportamiento fue bueno alcanzando su mejor valor en las cercanías de la generación número 80, aunque no fue mejor que el realizado a través del AG determínistico.

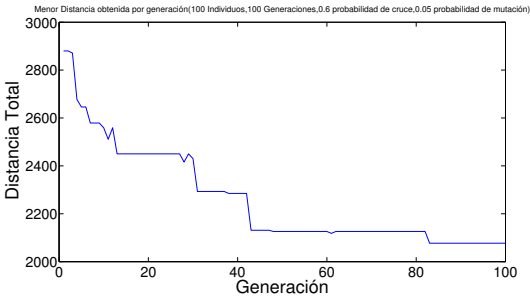

Figura 4.31: Fitness Generacional Matriz 21X21 Gerhard Aleatorio

4.7 Problema con 280 ciudades Gerhard (2006) [28]

Ahora buscamos experimentar con un problema de un número mayor de ciudades a considerar que en los ejemplos propuestos anteriormente, por lo que tomamos al PAV para 280 ciudades [28], el cual está basado en ciudades de Alemania y es presentado mediante coordenadas para la localización de los nodos o ciudades. Estas coordenadas se muestran en las columnas de la Tabla 4.6,.

A partir de las coordenadas se elaboró la matriz de costos mediante la determinación de las distancias, de acuerdo a la expresión $d = \sqrt{(X_2 - X_1) + (Y_2 - Y_1)}$ [5].

Gerhard [28] llegó a una solución de 2579 unidades utilizando algoritmos genéticos y paralelismo computacional.

Debido al volumen de nodos que presenta este problema, no es sencillo resolverlo a través de Programación Entera; por lo tanto se utilizan los AG determinísticos para resolverlo, puesto que como observamos en los ejemplos resueltos previamente, fueron estos los que mejores resultados arrojaron en comparación con los AG aleatorios. Por lo que utilizando 100 individuos, 20000 iteraciones, una probabilidad de cruce de 0.6 y 0.1 como probabilidad de mutación como parámetros, se busca encontrar un buen resultado.

Una vez realizado este experimento al que denominamos AO, el algoritmo utilizó un tiempo de 420 segundos en llevar a cabo el proceso, proporcionando un resultado para la función objetivo de 7560.7 unidades. La ruta que proporciona este valor es la siguiente: $18-207-212-211-213-143-111-114-115-117-155-136-135-269-262-261-272-16-17-133-134-270-264-206-208-253-254-259-260-276-277-4-250-249-255-252-265-156-157-118-63-64-67-71-70-48-49-36-11-8-7-275-5-6-3-280-1-2-243-242-241-237-236-235-234-195-194-197-196-217-221-220-200-175-112-88-87-84-85-57-56-45-40-39-38-35-140-227-226-228-209-266-138-182-161-94-95-96-97-102-173-174-160-159-178-203-218-222-223-149-153-120-119-108-104-105-106-191-192-198-199-146-148-131-21-128-31-32-33-34-30-125-154-139-204-205-210-229-231-238-239-244-9-10-12-13-14-$

Tabla 4.6: Coordenadas 280 ciudades Gerhard(2006)

Eje X	Col.1	Col.2	Col.3	Col.4	Col.5	Col.6	Col.7	Col.8	Col.9	Col.10
	288	116	32	32	48	180	64	172	260	228
	288	104	32	40	56	180	72	172	260	228
	270	104	40	40	56	180	80	172	260	236
	256	104	56	40	48	180	80	180	260	236
	256	90	56	40	56	172	80	180	260	228
	246	80	48	44	56	172	88	188	260	228
	236	64	40	44	104	172	104	196	276	228
	228	64	32	44	104	172	124	204	276	228
	228	56	32	32	104	164	124	212	276	220
	220	56	24	24	104	148	132	220	276	212
	212	56	16	16	104	124	140	228	284	204
	204	56	16	16	104	124	132	228	284	196
	196	56	8	24	104	124	124	236	284	188
	188	56	8	32	116	124	124	236	284	180
	196	56	8	44	124	124	124	236	284	180
	188	40	8	56	132	124	124	228	284	180
	172	40	8	56	132	104	124	228	284	180
	164	40	8	56	140	104	132	236	288	180
	156	40	8	56	148	104	124	236	280	196
	148	40	16	56	156	104	120	228	276	204
	140	40	8	64	164	104	128	228	276	212
	148	40	8	72	172	104	136	236	276	220
	164	32	24	72	172	104	148	236	268	228
	172	32	32	56	172	104	162	228	260	236
	156	32	32	48	172	104	156	228	252	246
	140	32	32	56	172	92	172	236	260	252
	132	32	32	56	172	80	180	252	260	260
	124	32	32	48	180	72	180	260	236	280
Eje Y	Col.1	Col.2	Col.3	Col.4	Col.5	Col.6	Col.7	Col.8	Col.9	Col.10
	149	161	121	81	73	77	21	29	37	85
	129	153	113	83	73	69	25	37	45	93
	133	161	113	73	81	61	25	45	53	93
	141	169	113	63	83	53	25	45	61	101
	157	165	105	51	89	53	41	37	69	101
	157	157	99	43	97	61	49	41	77	109
	169	157	99	35	97	69	57	49	77	117
	169	165	97	27	105	77	69	57	69	125
	161	169	89	25	113	81	77	65	61	125
	169	161	89	25	121	85	81	73	53	117
	169	153	97	25	129	85	65	69	53	109
	169	145	109	17	137	93	61	77	61	101
	169	137	109	17	145	109	61	77	69	93
	169	129	97	17	145	125	53	69	77	93
	161	121	89	11	145	117	45	61	85	101
	145	121	81	9	145	101	37	61	93	109
	145	129	73	17	137	89	29	53	101	117
	145	137	65	25	137	81	21	53	109	125
	145	145	57	33	137	73	21	45	109	145
	145	153	57	41	137	65	9	45	101	145
	145	161	49	41	137	49	9	37	93	145
	169	169	41	41	125	41	9	37	85	145
	169	169	45	49	117	33	9	29	97	145
	169	161	41	49	109	25	9	29	109	145
	169	153	49	51	101	17	25	21	101	141
	169	145	57	57	93	9	21	21	93	125
	169	137	65	65	85	9	21	21	85	129
	169	129	73	63	85	9	29	29	85	133

$15-271-142-193-186-187-185-184-147-141-247-245-240-246-232-233-225-224-219-144-145-180-179-258-278-279-248-256-183-171-170-100-99-101-169-172-201-202-216-215-214-230-251-257-26-27-28-29-124-113-83-76-75-77-78-79-81-110-151-268-267-137-150-152-116-65-66-72-73-74-80-91-90-89-109-121-122-24-23-25-22-107-103-92-98-93-82-86-62-61-129-130-127-126-37-50-51-52-53-55-44-60-123-132-273-274-263-190-189-188-166-167-168-165-164-163-162-181-176-177-158-43-46-47-54-69-68-58-59-42-41-20-19$

La gráfica de la figura (Fig. 4.32) muestra la Aptitud Generacional de la función objetivo del experimento AO durante cada generación, en donde el promedio es menor conforme el número de generaciones aumenta.

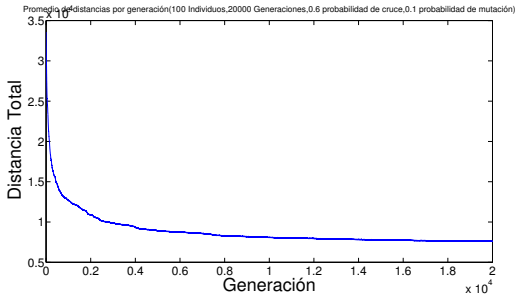

Figura 4.32: Aptitud Generacional del modelo determinístico experimento AO

La figura (Fig. 4.33) muestra el Fitness Generacional del experimento AO, el cual mejora de generación en generación, lo que nos hace pensar que aumentando el número de generaciones o el número de individuos o ambos en los parámetros, podremos tener mejores resultados, aunque el costo computacional será mayor.

Modificamos los parámetros al AG determínistico para llevar a cabo el experimento que denominaremos AP. En donde los parámetros a considerar son: 500 individuos, 20000 iteraciones, 0.6 como probabilidad de cruce y 0.1 como probabilidad de mutación.

El experimento AP utilizó un costo computacional de 2225 segundos, arrojando una distancia recorrida de 6750.22 unidades, siendo este resultado mejor que el experimento AO; en donde la secuencia encontrada es la siguiente: $144-199-167-168-$

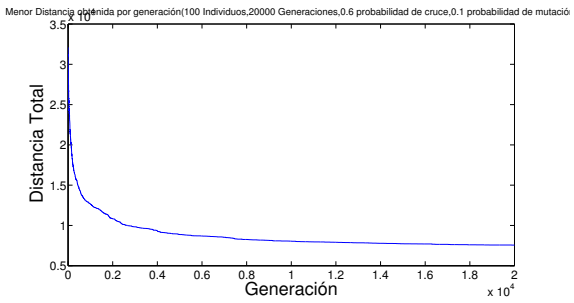

Figura 4.33: Fitness Generacional del modelo determínistico AO

100 – 99 – 98 – 93 – 82 – 83 – 88 – 112 – 165 – 188 – 189 – 190 – 186 – 187 – 163 – 107 – 111 – 114 – 115 – 117 – 154 – 18 – 17 – 13 – 14 – 24 – 23 – 20 – 21 – 128 – 37 – 50 – 51 – 52 – 141 – 216 – 215 – 231 – 238 – 239 – 240 – 245 – 247 – 250 – 249 – 256 – 262 – 16 – 25 – 22 – 26 – 27 – 28 – 156 – 152 – 137 – 268 – 267 – 149 – 179 – 180 – 150 – 15 – 12 – 11 – 246 – 237 – 236 – 235 – 233 – 234 – 226 – 227 – 230 – 251 – 255 – 257 – 133 – 127 – 126 – 29 – 32 – 31 – 30 – 125 – 129 – 130 – 148 – 142 – 147 – 182 – 106 – 91 – 92 – 97 – 96 – 95 – 94 – 101 – 172 – 164 – 185 – 193 – 197 – 196 – 203 – 204 – 207 – 208 – 209 – 229 – 232 – 228 – 210 – 213 – 195 – 194 – 176 – 177 – 120 – 124 – 34 – 35 – 39 – 48 – 47 – 46 – 66 – 84 – 87 – 158 – 138 – 266 – 263 – 264 – 265 – 140 – 139 – 157 – 116 – 86 – 85 – 65 – 64 – 59 – 44 – 57 – 58 – 63 – 62 – 61 – 118 – 161 – 183 – 162 – 90 – 80 – 78 – 77 – 75 – 74 – 73 – 71 – 70 – 69 – 45 – 40 – 41 – 119 – 151 – 178 – 136 – 269 – 270 – 275 – 274 – 273 – 272 – 271 – 89 – 81 – 79 – 76 – 72 – 67 – 68 – 42 – 123 – 155 – 253 – 254 – 258 – 259 – 276 – 6 – 5 – 243 – 242 – 241 – 244 – 248 – 277 – 9 – 10 – 8 – 7 – 1 – 2 – 280 – 3 – 19 – 33 – 36 – 38 – 49 – 53 – 54 – 55 – 56 – 113 – 171 – 166 – 191 – 192 – 198 – 135 – 132 – 131 – 159 – 160 – 175 – 174 – 170 – 169 – 102 – 103 – 60 – 43 – 122 – 121 – 153 – 134 – 261 – 260 – 4 – 279 – 278 – 252 – 206 – 145 – 184 – 173 – 105 – 104 – 108 – 109 – 110 – 181 – 146 – 143 – 205 – 212 – 211 – 214 – 217 – 218 – 219 – 225 – 224 – 223 – 222 – 221 – 220 – 202 – 201 – 200

La figura (Fig. 4.34) muestra la Aptitud Generacional que tuvo el AG determinístico, en donde la mejora de generación en generación se hace bastante notoria de acuerdo a la gráfica decreciente.

El Fitness Generacional del experimento AP mostrado en la (Fig. 4.35) muestra la gran mejora de cada generación, y dicha mejora es menos notoria a partir de la generación 10000, por lo que para mejorar los resultados requerimos experimentar

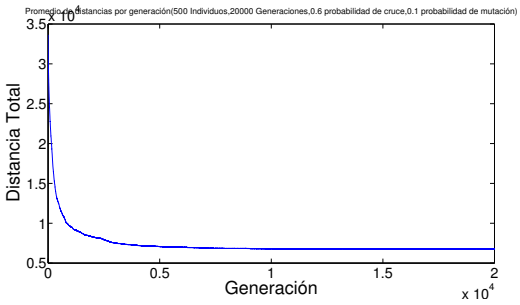

Figura 4.34: Aptitud Generacional del modelo determinístico experimento AP

con el AG determínistico con un mayor número de individuos y no incrementando las iteraciones.

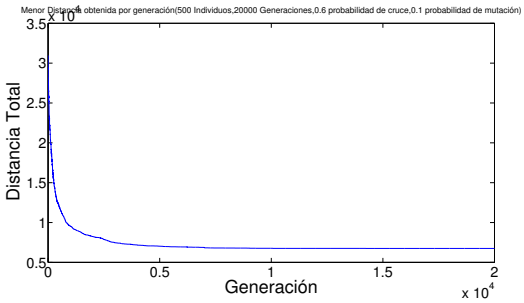

Figura 4.35: Fitness Generacional del modelo determínistico AP

Debido a lo anterior experimentamos ahora con el AG determínistico al que nombraremos AR, para lo cual asignamos los siguientes parámetros: 2000 individuos, 20000 iteraciones, probabilidad de cruza de 0.6 y una probabilidad de mutación de 0.1.

El algoritmo genético determínistico con los parámetros mencionados utilizó un tiempo computacional de 8405 segundos, arrojando una distancia recorrida de 6424.03 unidades de acuerdo a la secuencia: $186-193-145-146-147-148-139-131-20-21-128-127-34-35-39-40-44-59-63-62-42-41-125-130-255-230-233-236-235-234-225-226-227-211-208-253-254-264-265-200-201-198-161-174-107-114-115-113-110-108-105-170-171-172-162-163-164-188-189-$

190−191−192−197−199−140−135−134−155−121−120−119−157−152−132−13−12−11−278−246−239−238−237−210−207−205−143−142−141−129−126−124−123−122−65−66−74−75−96−97−98−99−100−166−165−206−248−243−242−2−1−280−3−276−275−22−26−43−60−151−178−144−219−223−224−228−229−232−231−279−6−7−274−149−177−87−83−82−81−90−104−179−256−249−244−241−240−245−247−250−251−209−252−257−258−262−263−268−267−266−138−156−153−154−30−31−33−37−50−51−52−53−54−46−61−118−116−86−85−84−103−102−101−168−167−169−67−57−45−25−23−24−14−15−272−204−203−202−196−194−195−221−222−220−217−218−216−133−18−17−270−212−213−215−214−4−5−277−260−259−261−271−16−19−29−38−47−55−56−68−69−70−71−72−73−76−80−91−92−93−94−95−78−77−79−160−150−137−136−269−273−9−8−10−27−28−32−36−49−48−58−64−117−158−159−176−180−181−173−106−109−89−88−112−111−175−182−183−184−185−187

El AG determínistico muestra un adecuado comportamiento en cuanto a su Aptitud Generacional (Fig. 4.36) se refiere, puesto que la gráfica es decreciente, lo cual indica que conforme avanzan las generaciones se presentan mejores resultados.

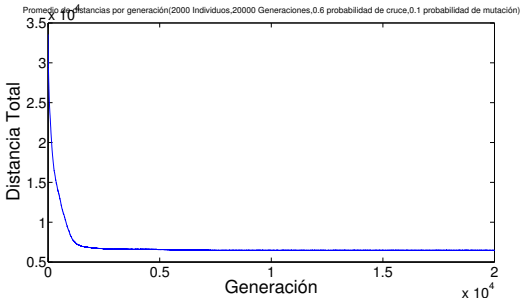

Figura 4.36: Aptitud Generacional del modelo determínistico AR

El Fitness Generacional (Fig.4.37) del AG determínistico muestra que los parámetros considerados para el experimento AR arroja mejores resultados que los experimentos AO y AP. Por lo que parece ser recomendable incrementar el número de individuos para obtener mejores resultados a costa del tiempo computacional.

Ahora experimentamos con el AG determínistico modificando sus parámetros con 20000 individuos, 1000 iteraciones, una probabilidad de cruce de 0.6 y una probabi-

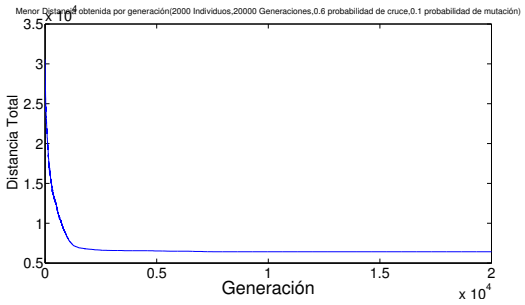

Figura 4.37: Fitness Generacional del modelo determínistico AR

lidad de mutación de 1/280. A cuyo experimento nombramos como AT.

El experimento AT necesitó un tiempo computacional de 21600 segundos otorgando una distancia total recorrida de 5509.59 unidades, de acuerdo a la secuencia: 157 – 158 – 111 – 110 – 104 – 90 – 91 – 92 – 93 – 80 – 81 – 82 – 88 – 112 – 114 – 87 – 113 – 86 – 85 – 66 – 65 – 64 – 63 – 62 – 61 – 60 – 43 – 42 – 41 – 46 – 45 – 56 – 59 – 44 – 57 – 58 – 67 – 84 – 83 – 89 – 109 – 108 – 173 – 172 – 165 – 164 – 193 – 202 – 216 – 213 – 214 – 215 – 218 – 217 – 221 – 219 – 223 – 226 – 227 – 228 – 229 – 245 – 21 – 280 – 243 – 242 – 241 – 246 – 231 – 238 – 237 – 232 – 233 – 230 – 251 – 247 – 278 – 279 – 3 – 5 – 6 – 276 – 275 – 272 – 16 – 132 – 122 – 118 – 116 – 79 – 76 – 77 – 75 – 71 – 70 – 68 – 55 – 54 – 53 – 47 – 40 – 125 – 150 – 142 – 141 – 144 – 220 – 222 – 224 – 225 – 234 – 235 – 236 – 239 – 240 – 244 – 274 – 15 – 17 – 130 – 154 – 171 – 167 – 188 – 166 – 168 – 169 – 98 – 94 – 78 – 74 – 73 – 72 – 69 – 52 – 48 – 39 – 35 – 29 – 23 – 24 – 269 – 138 – 135 – 134 – 131 – 129 – 128 – 127 – 33 – 36 – 37 – 50 – 51 – 49 – 38 – 34 – 14 – 13 – 12 – 259 – 249 – 256 – 255 – 257 – 263 – 264 – 265 – 268 – 267 – 266 – 139 – 140 – 143 – 145 – 199 – 201 – 200 – 262 – 7 – 9 – 273 – 271 – 270 – 133 – 21 – 117 – 115 – 103 – 102 – 100 – 99 – 95 – 96 – 97 – 101 – 105 – 107 – 106 – 170 – 185 – 187 – 189 – 192 – 194 – 197 – 11 – 10 – 8 – 261 – 260 – 258 – 252 – 209 – 210 – 207 – 212 – 211 – 250 – 248 – 4 – 277 – 254 – 253 – 208 – 206 – 205 – 204 – 203 – 149 – 155 – 123 – 124 – 32 – 31 – 30 – 126 – 28 – 27 – 26 – 22 – 25 – 20 – 19 – 18 – 136 – 137 – 148 – 147 – 146 – 198 – 196 – 195 – 191 – 190 – 186 – 161 – 174 – 162 – 182 – 181 – 163 – 184 – 183 – 175 – 159 – 160 – 151 – 178 – 179 – 180 – 176 – 177 – 152 – 156 – 153 – 121 – 120 – 119

Como podemos observar en la (Fig. 4.38), la Aptitud Generacional muestra un comportamiento.

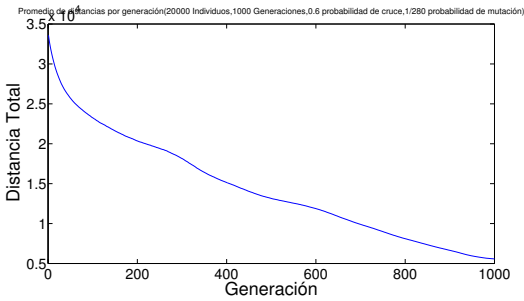

Figura 4.38: Aptitud Generacional del modelo determínistico AT

El AG determínistico muestra en la (Fig. 4.39) el comportamiento del Fitness Generacional, en donde la gráfica mostrada es decreciente por lo que el algoritmo muestra ser funcional para nuestros objetivos.

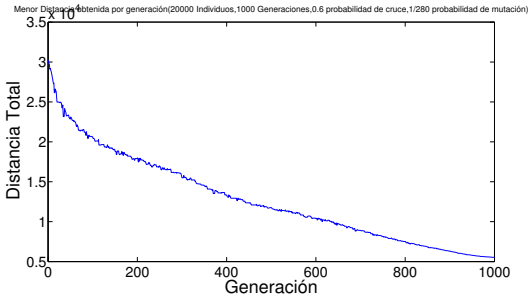

Figura 4.39: Fitness Generacional del modelo determínistico AT

En un intento por mejorar los resultados obtenidos para el PAV con 280 ciudades, modificamos nuevamente los parámetros del AG determínistico, teniendo ahora 5000 individuos, 20000 iteraciones, una probabilidad de cruce de 0.6 y una probabilidad de mutación de 1/280. Como podemos darnos cuenta utilizamos en este experimento al cual denominaremos como AS, a las cantidades más grandes hasta el momento empleadas para el rubro de individuos y generaciones de la presente investigación.

El experimento AS utilizó un tiempo computacional de 46810 segundos arrojando una distancia total recorrida de 5325.67 unidades de acuerdo a la secuencia: $163-161-175-160-159-158-114-113-86-116-61-60-30-31-32-33-34-35-41-$

66 – 76 – 75 – 74 – 73 – 72 – 71 – 67 – 64 – 63 – 62 – 118 – 117 – 115 – 107 – 106 – 105 – 108 – 88 – 84 – 85 – 65 – 58 – 57 – 56 – 45 – 46 – 42 – 43 – 44 – 59 – 68 – 70 – 69 – 55 – 54 – 47 – 40 – 39 – 38 – 37 – 49 – 48 – 53 – 52 – 51 – 50 – 36 – 28 – 25 – 23 – 24 – 16 – 261 – 259 – 232 – 233 – 234 – 235 – 236 – 237 – 238 – 239 – 240 – 241 – 244 – 247 – 250 – 251 – 231 – 246 – 245 – 249 – 248 – 278 – 4 – 5 – 6 – 7 – 8 – 128 – 127 – 126 – 123 – 103 – 99 – 100 – 101 – 173 – 124 – 125 – 29 – 27 – 26 – 22 – 14 – 13 – 15 – 271 – 272 – 273 – 260 – 256 – 225 – 224 – 223 – 222 – 221 – 220 – 253 – 254 – 270 – 133 – 134 – 269 – 209 – 230 – 229 – 228 – 227 – 226 – 205 – 141 – 156 – 121 – 122 – 21 – 20 – 19 – 18 – 1 – 280 – 2 – 242 – 243 – 3 – 279 – 277 – 276 – 275 – 132 – 155 – 111 – 112 – 81 – 79 – 80 – 104 – 176 – 139 – 140 – 266 – 267 – 268 – 135 – 136 – 137 – 138 – 149 – 179 – 180 – 181 – 92 – 93 – 98 – 97 – 94 – 96 – 95 – 78 – 77 – 170 – 172 – 171 – 168 – 167 – 166 – 188 – 189 – 265 – 274 – 9 – 10 – 11 – 12 – 17 – 153 – 157 – 110 – 109 – 90 – 91 – 102 – 169 – 165 – 187 – 186 – 185 – 162 – 174 – 89 – 82 – 83 – 87 – 119 – 120 – 154 – 129 – 130 – 131 – 142 – 143 – 144 – 200 – 202 – 203 – 204 – 208 – 252 – 255 – 257 – 258 – 262 – 263 – 264 – 206 – 207 – 210 – 211 – 212 – 213 – 214 – 215 – 218 – 219 – 217 – 216 – 201 – 196 – 195 – 194 – 192 – 191 – 190 – 193 – 197 – 198 – 199 – 145 – 146 – 147 – 148 – 150 – 178 – 152 – 151 – 177 – 182 – 183 – 184 – 164

El experimento AS obtuvo los mejores resultados hasta el momento encontrados en la presente investigación, en donde la Aptitud Generacional de la (Fig. 4.40) presenta el buen funcionamiento del algoritmo debido a que la gráfica es decreciente.

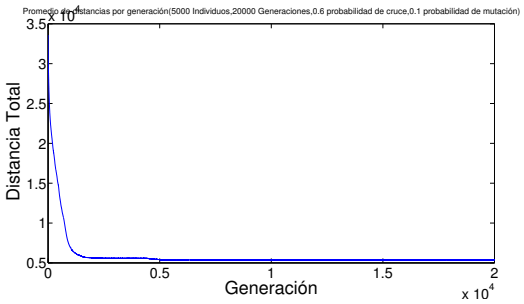

Figura 4.40: Aptitud Generacional del modelo determínistico AS

El AG determínistico para el experimento AS muestra en su Fitness Generacional (Fig. 4.41), la mejora de los resultados obtenidos conforme transcurren las generaciones, en donde debido a los parámetros utilizados se obtuvo el mejor resultado encontrado para este problema en la presente investigación.

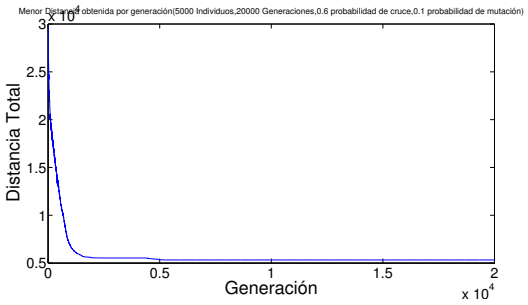

Figura 4.41: Fitness Generacional del modelo determínistico AS

Experimentamos una vez más modificando los parámetros con 50000 individuos, 1000 generaciones, 0.6 como probabilidad de cruce y 1/280 como probabilidad de mutación. A este experimento lo nombramos BJ.

El AG determínistico del experimento BJ utilizó un tiempo computacional de 81140 segundos y una distancia total recorrida de 4724.52 unidades con la secuencia: $152-151-178-179-149-148-147-146-142-141-140-139-138-131-21-128-27-26-20-132-135-136-137-180-181-182-183-184-185-186-187-188-189-190-191-192-193-198-145-143-144-197-194-195-196-201-200-267-19-22-25-23-18-133-269-257-254-255-256-258-259-260-274-273-12-13-14-24-17-268-266-265-264-263-262-275-9-8-10-11-272-253-252-209-227-226-225-224-215-214-211-210-208-207-212-213-216-217-218-219-220-221-222-223-228-229-271-15-16-270-134-129-154-125-30-31-33-36-35-39-38-37-50-51-49-52-59-63-86-110-107-174-173-170-171-172-106-105-104-108-155-130-177-176-150-261-5-1-2-242-244-246-231-232-237-236-235-234-233-280-279-3-4-278-248-247-245-243-241-240-239-238-230-251-250-249-277-6-7-276-206-205-204-203-202-199-157-119-60-43-47-48-53-54-58-74-94-93-90-91-92-101-98-95-96-97-99-100-102-103-169-168-166-165-164-167-80-78-77-75-76-83-82-79-81-89-109-88-112-113-116-87-84-85-65-66-73-72-71-70-69-68-67-64-62-61-118-117-115-114-111-162-163-161-175-160-159-158-120-121-127-29-28-32-34-40-41-46-55-56-57-44-45-42-123-126-124-122-153-156$

El experimento BJ presenta una Aptitud Generacional (Fig. 4.42) en donde la gráfica

decreciente muestra el buen funcionamiento que tuvo el AG.

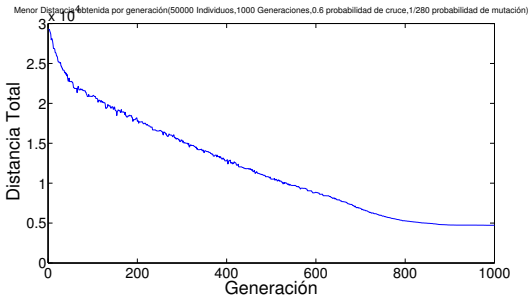

Figura 4.42: Aptitud Generacional del modelo determínistico BJ

El Fitness Generacional (Fig. 4.43) del experimento BJ muestra el mejor resultado obtenido a través del AG determínistico de la presente investigación.

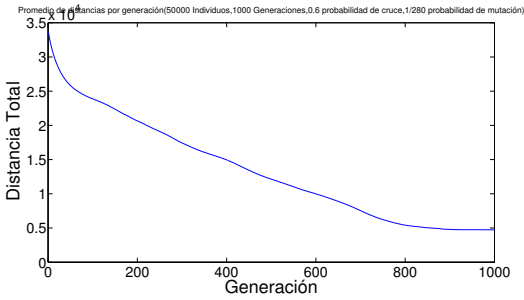

Figura 4.43: Fitness Generacional del modelo determínistico BJ

Podemos concluir el capítulo analizando los parámetros que proporcionan mejores resultados para el AG determínistico, en donde podemos deducir que es conveniente considerar un número relativamente grande de individuos con un número de iteraciones más moderado que el primero como lo podemos observar en los experimentos BZ,AT y BJ. Además de la probabilidad de cruza cercana a 0.6 y una probabilidad de mutación relativamente baja $(1/n)$. Esto nos permite utilizar el algoritmo con dichos parámetros para aplicarlos a problemas de la industria como el Problema de Secuenciación de Trabajos.

En la Tabla 4.7 se muestra la comparación de los resultados obtenidos para el PAV de 280 ciudades de Gerhard [28]; en donde el tiempo es dado en segundos.

Tabla 4.7: Tabla comparativa de resultados del PAV

Experimento	Torneo	Individuos	Generaciones	Cruce	Mutación	Tiempo	Resultado	Resultado Art.
5 Cds. [35]	Determinístico	100	100	0.6	0.1	2	668	668
5 Cds. [35]	Aleatorio	100	100	0.6	0.1	2	668	668
10 Cds. [28]	Determinístico	100	100	0.6	1/10	3	1185	1185
10 Cds. [28]	Aleatorio	100	100	0.6	1/10	3	1185	1185
10 Cds. [34]	Determinístico	100	100	0.6	0.1	3	7392	8914
10 Cds. [34]	Aleatorio	100	100	0.6	0.1	3	7392	8914
15 Cds. [28]	Determinístico	100	100	0.6	0.05	5	1692	1513
15 Cds. [28]	Aleatorio	100	100	0.6	0.05	5	1757	1513
20 Cds. [28]	Determinístico	100	100	0.6	1/20	7	1700	1688
20 Cds. [28]	Aleatorio	100	100	0.6	1/20	7	1854	1688
21 Cds. [28]	Determinístico	100	100	0.6	0.05	8	2042	2042
21 Cds. [28]	Aleatorio	100	100	0.6	0.05	8	2077	2042
280 Cds. [28]								
AO	Determinístico	100	20000	0.6	0.1	420	6213.25	2579
BT	Aleatorio	100	20000	0.6	0.1	900	7968.55	2579
AP	Determinístico	500	20000	0.6	0.1	2220	6750.22	2579
BU	Aleatorio	1000	20000	0.6	0.1	13680	8686.33	2579
AQ	Determinístico	1000	20000	0.6	0.1	4200	5656.54	2579
CA	Aleatorio	1000	20000	0.6	0.1	13980	8686.33	2579
AR	Determinístico	2000	20000	0.6	0.1	8400	6424.03	2579
BX	Aleatorio	2000	20000	0.6	0.1	28860	10820.19	2579
AS	Determinístico	5000	20000	0.6	1/280	46810	5325.67	2579
BI	Aleatorio	5000	20000	0.6	1/280	76680	7433.69	2579
BZ	Determinístico	20000	1000	0.6	1/280	3600	7205.13	2579
AT	Aleatorio	20000	1000	0.6	1/280	21600	5509.59	2579
BJ	Determinístico	50000	1000	0.6	1/280	81140	4724.52	2579

Capítulo 5

Experimentación de la solución del problema de secuenciación de trabajos resuelto a través del Problema del agente viajero con Algoritmos Genéticos

Como observamos en el capítulo anterior los AG puden ser utilizados para resolver el PAV, y este último a su vez permite resolver el Problema de Secuenciación de Trabajos mediante su decodificación.

Podemos describir al Problema de Secuenciación de Trabajos como la disposición de un conjunto de máquinas que realizan diversas tareas cuando en Ingeniería Industrial se pretende producir cierto producto a partir de un insumo. Para lo cual, seguimos una serie de pasos, donde cada paso consiste en aplicar una máquina determinada durante un período de tiempo. A cada uno de esos pasos le llamamos operación y a esa secuencia de operaciones necesarias para terminar el producto le llamamos trabajo. Si pretendemos obtener varios productos distintos, tendremos asociados un trabajo con sus correspondientes operaciones para cada uno de esos productos. De tal manera que al tener un conjunto de máquinas y un conjunto de trabajos, una secuencia es una asignación que fija a cada operación con un intervalo de tiempo para ser efectuada.

En donde el problema consiste en encontrar una secuencia que minimice el tiempo necesario para completar todas las operaciones a la cual se denomina como makespan.

El Problema de Secuenciación de Trabajos se define según [13], de la siguiente manera: Sean dados M número de máquinas, J un conjunto de trabajos y O un conjunto de operaciones. Para cada operación $i \in O$ y una máquina $m_i \in M$ en la que debe realizarse consumiendo un tiempo $p_i \in \mathbb{R}$ ininterrumpido y positivo. Encontrar un tiempo de comienzo S_i para cada operación $i \in O$ tal que minimiza el makespan, definido como $min_{i \in O}(s_i + p_i)$, sujeto a las restricciones:

a) $s_i \geq 0$ para todo $i \in O$
b) $s_j \geq s_i + p_i$ para todo $i, j \in O$ con $i \prec j$
c) $s_j \geq s_i + p_i$ para todo $i, j \in O$ con $m_i = m_j$

En donde: la restricción a) implica que ninguna máquina está disponible antes del instante 0; la restricción b) da cuenta de la relación de precedencia $\prec$. Esta relación refleja el hecho de que un trabajo se compone de operaciones que tienen que efectuarse en un orden preciso y no en otro. Esto resulta claro si consideramos un trabajo que consista en forjar, perforar y empaquetar una pieza. De tal manera que si existe una relación de precedencia $i \prec j$, entonces j no puede comenzar antes que i finalice. Finalmente la restricción c) implica que ninguna máquina puede procesar dos operaciones al mismo tiempo. Si suponemos que una máquina no puede realizar más de una operación a la vez, es preciso fijar una secuencia para procesar las operaciones correspondientes a una misma máquina. Es decir, la secuencia debe imponer un orden entre las operaciones que se efectúen en la misma máquina, quedando determinada una secuencia de operaciones en cada una de dichas máquinas.

A continuación se experimenta con algunos problemas descritos en artículos cientificos de actualidad [4][18] .

5.1 Problema de Secuenciación de Trabajos con 3 Máquinas y 3 Trabajos de Tamilarasi(2010) [4]

Se experimentó con el problema que presenta Tamilarasi [4] sobre la secuenciación de trabajos de tres máquinas con tres trabajos por realizar, los tiempos de operación se

muestran en la Tabla 5.1:

Tabla 5.1: Tamilarasi(2010)[4] Tiempos

$O_1J_1M_1 = 2$	$O_2J_1M_3 = 3$	$O_3J_1M_2 = 4$
$O_4J_2M_2 = 1$	$O_5J_2M_1 = 5$	$O_6J_2M_3 = 2$
$O_7J_3M_3 = 4$	$O_8J_3M_1 = 6$	$O_9J_3M_2 = 4$

Donde:
J_n= Trabajo n ; M_n= Máquina n
O_1= Es la operación definida para construir la matriz de costos del PAV.

El resultado obtenido por Tamilarasi [4] a través de Recocido Simulado es de 17 unidades. En nuestro caso a partir de los tiempos mostrados en la Tabla 5.1 generamos la matriz de costos Tabla 5.2. En donde O_1 es la operación de realizar el trabajo 1 en la máquina 1; O_2 es cuando se realiza el trabajo 1 en la máquina 3 como se muestra en la Tabla 5.1.

Tabla 5.2: Matriz de costos Tamilarasi(2010)[4]

	O1	O2	O3	O4	O5	O6	O7	O8	O9
O1	0	5	6	2	7	2	4	8	4
O2	5	0	7	3	5	5	7	6	4
O3	6	7	0	5	5	4	4	6	8
O4	2	3	5	0	6	3	4	6	5
O5	7	5	5	6	0	7	5	11	5
O6	2	5	4	3	7	0	6	6	4
O7	4	7	4	4	5	6	0	10	8
O8	8	6	6	6	11	6	10	0	10
O9	4	4	8	5	5	4	8	10	0

La matriz de costos se construye de la siguiente manera: De O_1 a O_2 el trabajo J_1 debe ser procesado por ambas máquinas M_1 y M_3, de tal manera que la operación O_2 no puede empezar hasta terminar O_1 es decir:

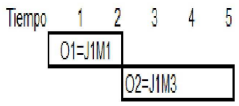

Figura 5.1: Trabajos

Por lo que de O_1 a O_2 existe un tiempo de 5 unidades; de la misma forma que de O_2 a O_1 como se observa en la Tabla 5.2.

En caso de que la máquina y/o el trabajo no sean los mismos las operaciones pueden realizarse simultáneamente, por ejemplo O_1 y O_6:

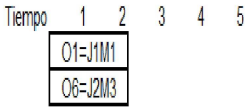

Figura 5.2: Trabajos 2

Por lo que de O_1 a O_6 existe un tiempo de 2 unidades; de la misma forma que de O_6 a O_1 como se observa en la Tabla 5.2.

Una vez construida la matriz de costos se utiliza programación entera resuelta en Lingo y con AG se resuelve en Matlab encontrando una distancia óptima de 37 unidades de acuerdo con la ruta mostrada en (Fig. 5.2), lo cual nos permite obtener la secuencia de trabajos como se muestra en la (Fig. 5.3) con un tiempo total de 17 unidades.

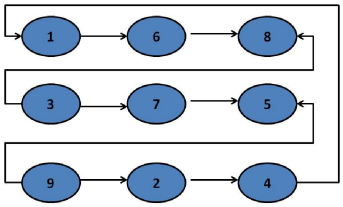

Figura 5.3: Tamilarasi 3x3 ruta Lingo

La secuencia y tiempo obtenidos a través del AG determinístico es similar al obtenido a través de programación entera y puede ser regresado al Problema de Secuenciación de Trabajos como se observa en (Fig. 5.4).

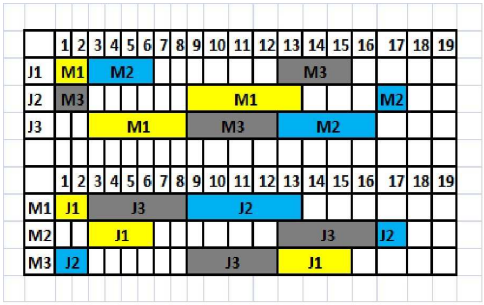

Figura 5.4: Tamilarasi 3x3 makespan

5.2 Problema de Secuenciación de Trabajos con 6 Máquinas y 6 Trabajos de Ruiz(2011) [18]

De la misma forma se trabaja con el problema de Ruiz J.A.[18](2011), quien presenta un Problema de Secuenciación de Trabajos de 6 máquinas con 6 trabajos por realizar, cuyo código es llamado por Ruiz con las literales y números (FT06,de la misma manera en que se encuentra en el Benchmarking del Problema de Secuenciación de Trabajos [18]), los tiempos de operación se muestran en la Tabla 5.3.

Tabla 5.3: Tiempos de trabajos de Ruiz J.A.[18](2011) caso FT06

$O_1J_1M_1=3$	$O_2J_1M_2=6$	$O_3J_1M_3=1$	$O_4J_1M_4=7$	$O_5J_1M_5=6$	$O_6J_1M_6=3$
$O_7J_2M_1=10$	$O_8J_2M_2=8$	$O_9J_2M_3=5$	$O_{10}J_2M_4=4$	$O_{11}J_2M_5=10$	$O_{12}J_2M_6=10$
$O_{13}J_3M_1=9$	$O_{14}J_3M_2=1$	$O_{15}J_3M_3=5$	$O_{16}J_3M_4=4$	$O_{17}J_3M_5=7$	$O_{18}J_3M_6=8$
$O_{19}J_4M_1=5$	$O_{20}J_4M_2=5$	$O_{21}J_4M_3=5$	$O_{22}J_4M_4=3$	$O_{23}J_4M_5=8$	$O_{24}J_4M_6=9$
$O_{25}J_4M_1=3$	$O_{26}J_4M_2=3$	$O_{27}J_4M_3=9$	$O_{28}J_4M_4=1$	$O_{29}J_4M_5=5$	$O_{30}J_4M_6=4$
$O_{31}J_4M_1=10$	$0_{32}J_4M_2=3$	$O_{33}J_4M_3=1$	$O_{34}J_4M_4=3$	$O_{35}J_4M_5=4$	$O_{36}J_4M_6=9$

A partir del tiempo necesario para cada máquina se elaboró la Tabla 37 del CD anexo, representado como un PAV como se realizó en el problema de Tamilarasi [4].

El mejor tiempo encontrado por Ruiz J.A.[18](2011) es de 55 unidades aunque no especifica la secuencia a seguir.

La ruta obtenida por AG determinísticos muestran una función objetivo de 229 unidades de acuerdo con la secuencia de (Fig. 5.4).

Al transformar el PAV al Problema de Secuenciación de actividades se encontró un valor de 55 unidades igualando al obtenido por Ruiz J.A.(2011),[18] pero con una secuencia distinta, la cual se muestra en (Fig. 5.5).

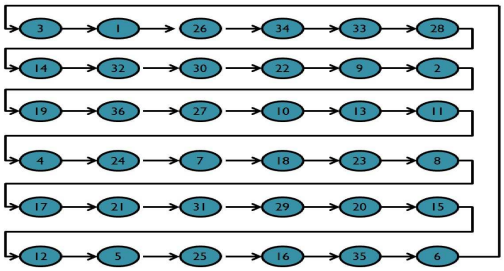

Figura 5.5: Ruta FT06 Determinista

Figura 5.6: Secuencia FT06 Determinista

5.3 Problema de Secuenciación de Trabajos con 10 Máquinas y 5 Trabajos de Ruiz(2011) [18]

Ruiz J.A.[18](2011), presenta un problema de secuenciación de trabajos de 10 máquinas con 5 trabajos por realizar, a la cual identifica como (LA04), los tiempos de operación se muestran en la Tabla 5.4.

Tabla 5.4: Tiempos de trabajos de Ruiz J.A.[18](2011) caso LA04

$O_1J_1M_1 = 29$	$O_2J_1M_2 = 78$	$O_3J_1M_3 = 9$	$O_4J_1M_4 = 36$	$O_5J_1M_5 = 49$
$O_6J_2M_1 = 11$	$O_7J_2M_2 = 62$	$O_8J_2M_3 = 56$	$O_9J_2M_4 = 44$	$O_{10}J_2M_5 = 21$
$O_{11}J_3M_1 = 4$	$O_{12}J_3M_2 = 28$	$O_{13}J_3M_3 = 90$	$O_{14}J_3M_4 = 69$	$O_{15}J_3M_5 = 75$
$O_{16}J_4M_1 = 46$	$O_{17}J_4M_2 = 46$	$O_{18}J_4M_3 = 72$	$O_{19}J_4M_4 = 30$	$O_{20}J_4M_5 = 11$
$O_{21}J_5M_1 = 85$	$O_{22}J_5M_2 = 91$	$O_{23}J_5M_3 = 74$	$O_{24}J_5M_4 = 39$	$O_{25}J_5M_5 = 33$
$O_{26}J_6M_1 = 10$	$O_{27}J_6M_2 = 89$	$O_{28}J_6M_3 = 12$	$O_{29}J_6M_4 = 90$	$O_{30}J_6M_5 = 45$
$O_{31}J_7M_1 = 71$	$O_{32}J_7M_2 = 81$	$O_{33}J_7M_3 = 95$	$O_{34}J_7M_4 = 98$	$O_{35}J_7M_5 = 99$
$O_{36}J_8M_1 = 43$	$O_{37}J_8M_2 = 9$	$O_{38}J_8M_3 = 85$	$O_{39}J_8M_4 = 52$	$O_{40}J_8M_5 = 22$
$O_{41}J_9M_1 = 6$	$O_{42}J_9M_2 = 22$	$O_{43}J_9M_3 = 14$	$O_{44}J_9M_4 = 26$	$O_{45}J_9M_5 = 69$
$O_{46}J_{10}M_1 = 61$	$O_{47}J_{10}M_2 = 53$	$O_{48}J_{10}M_3 = 49$	$O_{49}J_{10}M_4 = 21$	$O_{50}J_{10}M_5 = 10$

A partir del tiempo requerido para cada máquina mostrado en la Tabla 5.4, se elaboró la transformación de la matriz Ruiz J.A. [18] a otra que permita resolver el problema a través del enfoque de un PAV, que se puede observar en la Tabla 38 del CD anexo.

Se conoce que el mejor tiempo obtenido por Ruiz J.A.(2011) para el problema LA04 es de 590 unidades. Mientras que se experimenta a través del AG determinístico para buscar una forma distinta de obtener una secuencia que nos proporcione un buen resultado respecto al tiempo de secuenciación de actividades utilizando como parámetros: 60000 individuos, 300 iteraciones, 0.6 como probabilidad de cruce y 1/50 como probabilidad de mutación. Con la ruta obtenida para el PAV se obtiene una secuencia de trabajos mostrada en la Fig. 56 del CD anexo, con un tiempo requerido de 621 unidades.

5.4 Problema de Secuenciación de Trabajos con 10 Máquinas y 10 Trabajos de Ruiz(2011) [18]

Ruiz J.A.[18](2011), presenta un Problema de Secuenciación de Trabajos de 10 máquinas con 10 trabajos por realizar, cuyo al cual Ruiz nombra como (FT10). Los tiempos de

operación se muestran en la Tabla 5.5.

Tabla 5.5: Tiempos de trabajos de Ruiz J.A.[18](2011) caso FT10

$O_1J_1M_1 = 29$	$O_2J_1M_2 = 78$	$O_3J_1M_3 = 9$	$O_4J_1M_4 = 36$	$O_5J_1M_5 = 49$	$O_6J_1M_6 = 11$
$O_7J_1M_7 = 62$	$O_8J_1M_8 = 56$	$O_9J_1M_9 = 44$	$O_{10}J_1M_{10} = 21$	$O_{11}J_2M_1 = 43$	$O_{12}J_2M_2 = 28$
$O_{13}J_2M_3 = 90$	$O_{14}J_2M_4 = 69$	$O_{15}J_2M_5 = 75$	$O_{16}J_2M_6 = 46$	$O_{17}J_2M_7 = 46$	$O_{18}J_2M_8 = 72$
$O_{19}J_2M_9 = 30$	$O_{20}J_2M_{10} = 11$	$O_{21}J_3M_1 = 85$	$O_{22}J_3M_2 = 91$	$O_{23}J_3M_3 = 74$	$O_{24}J_3M_4 = 39$
$O_{25}J_3M_5 = 33$	$O_{26}J_3M_6 = 10$	$O_{27}J_3M_7 = 89$	$O_{28}J_3M8 = 12$	$O_{29}J_3M_9 = 90$	$O_{30}J_3M_{10} = 45$
$O_{31}J_4M_1 = 71$	$O_{32}J_4M_2 = 81$	$O_{33}J_4M_3 = 95$	$O_{34}J_4M_4 = 98$	$O_{35}J_4M_5 = 99$	$O_{36}J_4M_6 = 43$
$O_{37}J_4M_7 = 9$	$O_{38}J_4M_8 = 85$	$O_{39}J_4M_9 = 52$	$O_{40}J_4M_{10} = 22$	$O_{41}J_5M_1 = 6$	$O_{42}J_5M_2 = 22$
$O_{43}J_5M_3 = 14$	$O_{44}J_5M_4 = 26$	$O_{45}J_5M_5 = 69$	$O_{46}J_5M_6 = 61$	$O_{47}J_5M_7 = 53$	$O_{48}J_5M_8 = 49$
$O_{49}J_5M_9 = 21$	$O_{50}J_5M_{10} = 10$	$O_{51}J_6M_1 = 47$	$O_{52}J_6M_2 = 2$	$O_{53}J_6M_3 = 84$	$O_{54}J_6M_4 = 95$
$O_{55}J_6M_5 = 6$	$O_{56}J_6M_6 = 52$	$O_{57}J_6M_7 = 65$	$O_{58}J_6M_8 = 25$	$O_{59}J_6M_9 = 48$	$O_{60}J_6M_{10} = 72$
$O_{61}J_7M_1 = 37$	$O_{62}J_7M_2 = 46$	$O_{63}J_7M_3 = 13$	$O_{64}J_7M_4 = 61$	$O_{65}J_7M_5 = 55$	$O_{66}J_7M_6 = 21$
$O_{67}J_7M_7 = 32$	$O_{68}J_7M_8 = 30$	$O_{69}J_7M_9 = 89$	$O_{70}J_7M_{10} = 32$	$O_{71}J_8M_1 = 86$	$O_{72}J_8M_2 = 46$
$O_{73}J_8M_3 = 31$	$O_{74}J_8M_4 = 79$	$O_{75}J_8M_5 = 32$	$O_{76}J_8M_6 = 74$	$O_{77}J_8M_7 = 88$	$O_{78}J_8M_8 = 36$
$O_{79}J_8M_9 = 19$	$O_{80}J_8M_{10} = 48$	$O_{81}J_9M_1 = 76$	$O_{82}J_9M_2 = 69$	$O_{83}J_9M_3 = 85$	$O_{84}J_9M_4 = 76$
$O_{85}J_9M_5 = 26$	$O_{86}J_9M_6 = 51$	$O_{87}J_9M_7 = 40$	$O_{88}J_9M_8 = 89$	$O_{89}J_9M_9 = 74$	$O_{90}J_9M_{10} = 11$
$O_{91}J_{10}M_1 = 13$	$O_{92}J_{10}M_2 = 85$	$O_{93}J_{10}M_3 = 61$	$O_{94}J_{10}M_4 = 52$	$O_{95}J_{10}M_5 = 90$	$O_{96}J_{10}M_6 = 47$
$O_{97}J_{10}M_7 = 7$	$O_{98}J_{10}M_8 = 45$	$O_{99}J_{10}M_9 = 64$	$O_{100}J_{10}M_{10} = 76$		

A partir del tiempo necesario para cada máquina se elaboró la matriz Ruiz J.A. [18] de la Tabla 39 del CD anexo. El mejor tiempo obtenido por Ruiz J.A.(2011) es de 930 unidades.

Para resolver el PAV que se generá se utilizó el AG descrito en capítulos anteriores con los siguientes parámetros: 50000 individuos, 1000 iteraciones, 0.6 como probabilidad de cruce y 1/100 como probabilidad de mutación, en donde se obtuvo un tiempo de 1156 unidades para la secuenciación de trabajos con la ruta mostrada en Fig.55 del CD anexo.

5.5 Problema de Secuenciación de Trabajos con 5 Máquinas y 10 Trabajos de Ruiz(2011) [18] caso LA02

Ruiz J.A.[18](2011), presenta un Problema de Secuenciación de Trabajos de 5 máquinas con 10 trabajos por realizar, cuyo código es (LA02), los tiempos de operación se muestran en la Tabla 5.6:

Tabla 5.6: Tiempos de trabajos de Ruiz J.A.[18](2011) caso LA02

$O_1J_1M_1 = 20$	$O_2J_1M_2 = 31$	$O_3J_1M_3 = 17$	$O_4J_1M_4 = 87$	$O_5J_1M_5 = 76$
$O_6J_2M_1 = 24$	$O_7J_2M_2 = 18$	$O_8J_2M_3 = 32$	$O_9J_2M_4 = 81$	$O_{10}J_2M_5 = 25$
$O_{11}J_3M_1 = 58$	$O_{12}J_3M_2 = 72$	$O_{13}J_3M_3 = 23$	$O_{14}J_3M_4 = 99$	$O_{15}J_3M_5 = 28$
$O_{16}J_4M_1 = 45$	$O_{17}J_4M_2 = 76$	$O_{18}J_4M_3 = 86$	$O_{19}J_4M_4 = 90$	$O_{20}J_4M_5 = 97$
$O_{21}J_5M_1 = 42$	$O_{22}J_5M_2 = 46$	$O_{23}J_5M_3 = 17$	$O_{24}J_5M_4 = 48$	$O_{25}J_5M_5 = 27$
$O_{26}J_6M_1 = 98$	$O_{27}J_6M_2 = 67$	$O_{28}J_6M_3 = 62$	$O_{29}J_6M_4 = 27$	$O_{30}J_6M_5 = 48$
$O_{31}J_7M_1 = 80$	$O_{32}J_7M_2 = 12$	$O_{33}J_7M_3 = 50$	$O_{34}J_7M_4 = 19$	$O_{35}J_7M_5 = 28$
$O_{36}J_8M_1 = 94$	$O_{37}J_8M_2 = 63$	$O_{38}J_8M_3 = 98$	$O_{39}J_8M_4 = 50$	$O_{40}J_8M_5 = 80$
$O_{41}J_9M_1 = 75$	$O_{42}J_9M_2 = 41$	$O_{43}J_9M_3 = 50$	$O_{44}J_9M_4 = 55$	$O_{45}J_9M_5 = 14$
$O_{46}J_{10}M_1 = 61$	$O_{47}J_{10}M_2 = 37$	$O_{48}J_{10}M_3 = 18$	$O_{49}J_{10}M_4 = 79$	$O_{50}J_{10}M_5 = 72$

A partir del tiempo necesario para cada máquina se elaboró la matriz Ruiz J.A. [18](2011) Tabla 40 del CD anexo, la cual permite dar un enfoque de un PAV.

El mejor tiempo obtenido por Ruiz J.A.(2011) a través de una metodología no especificada es de 655 unidades.
Al resolver el enfoque del PAV dado a este problema a través de los AG, en los cuales se utilizaron parámetros de 50000 individuos, 1000 iteraciones, 0.6 como probabilidad de cruce y 1/50 como probabilidad de mutación; en donde se obtuvo un tiempo de 768 unidades con la ruta mostrada en la Fig. 57 del CD anexo.

5.6 Problema de Secuenciación de Trabajos con 5 Máquinas y 10 Trabajos de Ruiz(2011) [18] caso LA03

Ruiz J.A.[18](2011), presenta un Problema de Secuenciación de Trabajos de 5 máquinas con 10 trabajos por realizar, cuyo código es (LA03), los tiempos de operación se muestran a continuación en la Tabla 5.7 :

Tabla 5.7: Tiempos de trabajos de Ruiz J.A.[18](2011) caso LA03

$O_1J_1M_1 = 82$	$O_2J_1M_2 = 23$	$O_3J_1M_3 = 45$	$O_4J_1M_4 = 38$	$O_5J_1M_5 = 84$
$O_6J_2M_1 = 18$	$O_7J_2M_2 = 29$	$O_8J_2M_3 = 21$	$O_9J_2M_4 = 50$	$O_{10}J_2M_5 = 41$
$O_{11}J_3M_1 = 52$	$O_{12}J_3M_2 = 52$	$O_{13}J_3M_3 = 38$	$O_{14}J_3M_4 = 54$	$O_{15}J_3M_5 = 16$
$O_{16}J_4M_1 = 54$	$O_{17}J_4M_2 = 62$	$O_{18}J_4M_3 = 74$	$O_{19}J_4M_4 = 57$	$O_{20}J_4M_5 = 37$
$O_{21}J_5M_1 = 81$	$O_{22}J_5M_2 = 61$	$O_{23}J_5M_3 = 30$	$O_{24}J_5M_4 = 68$	$O25J5M5 = 57$
$O_{26}J_6M_1 = 79$	$O_{27}J_6M_2 = 89$	$O_{28}J_6M_3 = 89$	$O_{29}J_6M_4 = 11$	$O_{30}J_6M_5 = 81$
$O_{31}J_7M_1 = 91$	$O_{32}J_7M_2 = 66$	$O_{33}J_7M_3 = 2$	$O_{34}J_7M_4 = 33$	$O_{35}J_7M_5 = 20$
$O_{36}J_8M_1 = 32$	$O_{37}J_8M_2 = 84$	$O_{38}J_8M_3 = 55$	$O_{39}J_8M_4 = 8$	$O_{40}J_8M_5 = 24$
$O_{41}J_9M_1 = 7$	$O_{42}J_9M_2 = 39$	$O_{43}J_9M_3 = 64$	$O_{44}J_9M_4 = 54$	$O_{45}J_9M_5 = 56$
$O_{46}J_{10}M_1 = 19$	$O_{47}J_{10}M_2 = 83$	$O_{48}J_{10}M_3 = 8$	$O_{49}J_{10}M_4 = 7$	$O_{50}J_{10}M_5 = 40$

A partir del tiempo necesario para la realización de cada trabajo en cada máquina se elaboró la matriz de la Tabla 41 del CD anexo; la cual busca dar un enfoque del PAV. El mejor tiempo obtenido por Ruiz J.A.(2011) es de 597 unidades. A continuación se experimentó mediante AG con parámetros de 50000 individuos, 1000 iteraciones, 0.6 como probabilidad de cruce y 1/50 como probabilidad de mutación; en donde se obtuvo un tiempo de 699 unidades con la ruta Fig. 58 del CD anexo.

5.7 Problema de Secuenciación de Trabajos con 5 Máquinas y 20 Trabajos de Ruiz(2011) [18] caso LA12

Ruiz J.A.[18], presenta un Problema de Secuenciación de Trabajos de 5 máquinas con 20 trabajos por realizar, cuyo código es (LA12), los tiempos de operación se muestran en la Tabla 5.8.

Tabla 5.8: Tiempos de trabajos de Ruiz J.A.[18](2011) caso LA12

$O_1J_1M_1=82$	$O_2J_1M_2=23$	$O_3J_1M_3=45$	$O_4J_1M_4=38$	$O_5J_1M_5=84$
$O_6J_2M_1=18$	$O_7J_2M_2=29$	$O_8J_2M_3=21$	$O_9J_2M_4=50$	$O_{10}J_2M_5=41$
$O11J_3M_1=52$	$O_{12}J_3M_2=52$	$O_{13}J_3M_3=38$	$O_{14}J_3M_4=54$	$O_{15}J_3M_5=16$
$O_{16}J_4M_1=54$	$O_{17}J_4M_2=62$	$O_{18}J_4M_3=74$	$O_{19}J_4M_4=57$	$O_{20}J_4M_5=37$
$O_{21}J_5M_1=81$	$O_{22}J_5M_2=61$	$O_{23}J_5M_3=30$	$O_{24}J_5M_4=68$	$O_{25}J_5M_5=57$
$O_{26}J_6M_1=79$	$O_{27}J_6M_2=89$	$O_{28}J_6M_3=89$	$O_{29}J_6M_4=11$	$O_{30}J_6M_5=81$
$O_{31}J_7M_1=91$	$O_{32}J_7M_2=66$	$O_{33}J_7M_3=20$	$O_{34}J_7M_4=33$	$O_{35}J_7M_5=20$
$O_{36}J_8M_1=32$	$O_{37}J_8M_2=84$	$O_{38}J_8M_3=55$	$O_{39}J_8M_4=8$	$O_{40}J_8M_5=24$
$O_{41}J_9M_1=7$	$O_{42}J_9M_2=39$	$O_{43}J_9M_3=64$	$O_{44}J_9M_4=54$	$O_{45}J_9M_5=56$
$O_{46}J_{10}M_1=19$	$O_{47}J_{10}M_2=83$	$O_{48}J_{10}M_3=8$	$O_{49}J_{10}M_4=7$	$O_{50}J_{10}M_5=40$
$O_{51}J_{11}M_1=63$	$O_{52}J_{11}M_2=6$	$O_{53}J_{11}M_3=64$	$O_{54}J_{11}M_4=91$	$O_{55}J_{11}M_5=40$
$O_{56}J_{12}M_1=74$	$O_{57}J_{12}M_2=42$	$O_{58}J_{12}M_3=98$	$O_{59}J_{12}M_4=61$	$O_{60}J_{12}M_5=15$
$O_{61}J_{13}M_1=26$	$O_{62}J_{13}M_2=80$	$O_{63}J_{13}M_3=87$	$O_{64}J_{13}M_4=75$	$O_{65}J_{13}M_5=6$
$O_{66}J_{14}M_1=75$	$O_{67}J_{14}M_2=44$	$O_{68}J_{14}M_3=39$	$O_{69}J_{14}M_4=24$	$O_{70}J_{14}M_5=22$
$O_{71}J_{15}M_1=12$	$O_{72}J_{15}M_2=15$	$O_{73}J_{15}M_3=20$	$O_{74}J_{15}M_4=79$	$O_{75}J_{15}M_5=8$
$O_{76}J_{16}M_1=80$	$O_{77}J_{16}M_2=61$	$O_{78}J_{16}M_3=43$	$O_{79}J_{16}M_4=26$	$O_{80}J_{16}M_5=22$
$O_{81}J_{17}M_1=63$	$O_{82}J_{17}M_2=36$	$O_{83}J_{17}M_3=62$	$O_{84}J_{17}M_4=96$	$O_{85}J_{17}M_5=40$
$O_{86}J_{18}M_1=22$	$O_{87}J_{18}M_2=33$	$O_{88}J_{18}M_3=10$	$O_{89}J_{18}M_4=18$	$O_{90}J_{18}M_5=5$
$O_{91}J_{19}M_1=89$	$O_{92}J_{19}M_2=96$	$O_{93}J_{19}M_3=64$	$O_{94}J_{19}M_4=95$	$O_{95}J_{19}M_5=64$
$O_{96}J_{20}M_1=8$	$O_{97}J_{20}M_2=38$	$O_{98}J_{20}M_3=18$	$O_{99}J_{20}M_4=15$	$O_{100}J_{20}M_5=23$

A partir del tiempo necesario para cada máquina en cada una de las tareas a realizar, se elaboró la matriz con enfoque del PAV mostrado en la Tabla 42 del CD anexo. El mejor tiempo obtenido por Ruiz J.A.(2011) es de 1039 unidades. Al experimentar a través de los AG con parámetros de 70000 individuos, 1250 generaciones, 0.6 como probabilidad de cruce y 1/50 como probabilidad de mutación. Nuevamente retomando

el Problema de Secuenciación de Trabajos se obtuvo un tiempo de 1412 unidades con la ruta Fig. 59 del CD anexo.

5.8 Problema de Secuenciación de Trabajos con 5 Máquinas y 20 Trabajos de Ruiz(2011) [18] caso LA13

Ruiz J.A.[18](2011), presenta un Problema de Secuenciación de Trabajos de 5 máquinas con 20 trabajos por realizar, cuyo código es (LA13), los tiempos de operación se muestran en la Tabla 5.9.

Tabla 5.9: Tiempos de trabajos de Ruiz J.A.[18](2011) caso LA13

$O_1J_1M_1 = 87$	$O_2J_1M_2 = 72$	$O_3J_1M_3 = 66$	$O_4J_1M_4 = 60$	$O_5J_1M_5 = 95$
$O_6J_2M_1 = 48$	$O_7J_2M_2 = 54$	$O_8J_2M_3 = 39$	$O_9J_2M_4 = 35$	$O_{10}J_2M_5 = 5$
$O_{11}J_3M_1 = 97$	$O_{12}J_3M_2 = 46$	$O_{13}J_3M_3 = 21$	$O_{14}J_3M_4 = 20$	$O_{15}J_3M_5 = 55$
$O_{16}J_4M_1 = 59$	$O_{17}J_4M_2 = 34$	$O_{18}J_4M_3 = 37$	$O_{19}J_4M_4 = 19$	$O_{20}J_4M_5 = 46$
$O_{21}J_5M_1 = 28$	$O_{22}J_5M_2 = 24$	$O_{23}J_5M_3 = 73$	$O_{24}J_5M_4 = 25$	$O_{25}J_5M_5 = 23$
$O_{26}J_6M_1 = 45$	$O_{27}J_6M_2 = 78$	$O_{28}J_6M_3 = 83$	$O_{29}J_6M_4 = 28$	$O_{30}J_6M_5 = 5$
$O_{31}J_7M_1 = 53$	$O_{32}J_7M_2 = 37$	$O_{33}J_7M_3 = 12$	$O_{34}J_7M_4 = 71$	$O_{35}J_7M_5 = 29$
$O_{36}J_8M_1 = 38$	$O_{37}J_8M_2 = 55$	$O_{38}J_8M_3 = 87$	$O_{39}J_8M_4 = 33$	$O_{40}J_8M_5 = 12$
$O_{41}J_9M_1 = 48$	$O_{42}J_9M_2 = 40$	$O_{43}J_9M_3 = 49$	$O_{44}J_9M_4 = 83$	$O_{45}J_9M_5 = 7$
$O_{46}J_{10}M_1 = 90$	$O_{47}J_{10}M_2 = 23$	$O_{48}J_{10}M_3 = 65$	$O_{49}J_{10}M_4 = 17$	$O_{50}J_{10}M_5 = 27$
$O_{51}J_{11}M_1 = 62$	$O_{52}J_{11}M_2 = 66$	$O_{53}J_{11}M_3 = 84$	$O_{54}J_{11}M_4 = 85$	$O_{55}J_{11}M_5 = 19$
$O_{56}J_{12}M_1 = 25$	$O_{57}J_{12}M_2 = 64$	$O_{58}J_{12}M_3 = 46$	$O_{59}J_{12}M_4 = 59$	$O_{60}J_{12}M_5 = 13$
$O_{61}J_{13}M_1 = 41$	$O_{62}J_{13}M_2 = 73$	$O_{63}J_{13}M_3 = 53$	$O_{64}J_{13}M_4 = 80$	$O_{65}J_{13}M_5 = 88$
$O_{66}J_{14}M_1 = 14$	$O_{67}J_{14}M_2 = 67$	$O_{68}J_{14}M_3 = 57$	$O_{69}J_{14}M_4 = 74$	$O_{70}J_{14}M_5 = 47$
$O_{71}J_{15}M_1 = 84$	$O_{72}J_{15}M_2 = 78$	$O_{73}J_{15}M_3 = 41$	$O_{74}J_{15}M_4 = 84$	$O_{75}J_{15}M_5 = 64$
$O_{76}J_{16}M_1 = 63$	$O_{77}J_{16}M_2 = 46$	$O_{78}J_{16}M_3 = 26$	$O_{79}J_{16}M_4 = 28$	$O_{80}J_{16}M_5 = 52$
$O_{81}J_{17}M_1 = 64$	$O_{82}J_{17}M_2 = 11$	$O_{83}J_{17}M_3 = 17$	$O_{84}J_{17}M_4 = 10$	$O_{85}J_{17}M_5 = 73$
$O_{86}J_{18}M_1 = 85$	$O_{87}J_{18}M_2 = 97$	$O_{88}J_{18}M_3 = 67$	$O_{89}J_{18}M_4 = 95$	$O_{90}J_{18}M_5 = 38$
$O_{91}J_{19}M_1 = 59$	$O_{92}J_{19}M_2 = 65$	$O_{93}J_{19}M_3 = 95$	$O_{94}J_{19}M_4 = 93$	$O_{95}J_{19}M_5 = 46$
$O_{96}J_{20}M_1 = 60$	$O_{97}J_{20}M_2 = 85$	$O_{98}J_{20}M_3 = 43$	$O_{99}J_{20}M_4 = 32$	$O_{100}J_{20}M_5 = 85$

A partir del tiempo necesario para cada máquina se elaboró la matriz Tabla 43 del CD anexo para darle el enfoque del PAV. El mejor tiempo obtenido por Ruiz J.A.(2011) es de 1150 unidades. Mediante AG y con parámetros de 50000 individuos, 1000 iteraciones, 0.6 como probabilidad de cruce y 1/100 como probabilidad de mutación; se obtuvo un tiempo de 1311 unidades con la ruta Fig. 60 del CD anexo.

En la Tabla 5.10 se muestra la comparación de los resultados de los problemas del presente Capítulo.

Tabla 5.10: Tabla comparativa de resultados

Experimento	Individuos	Generaciones	P. Cruce	P. Mutación	Tiempo (seg)	Resultado del artículo	Resultado AG
Tamilarasi	1000	100	0.6	1/9	120	17	17
FT06	1000	100	0.6	1/36	180	55	55
LA04	60000	300	0.6	1/50	64800	590	621
FT10	50000	1000	0.6	1/100	41230	930	1156
LA02	50000	1000	0.6	1/50	40120	655	768
LA03	50000	1000	0.6	1/50	40100	597	699
LA12	70000	1250	0.6	1/50	110810	1039	1412
LA13	50000	1000	0.6	1/100	75900	1150	1311

Capítulo 6

Conclusiones

En la presente investigación se explicaron las diferentes metodologías que permiten resolver al PAV, las cuales posteriormente sirvieron para convertirlo a un problema de secuenciación de trabajos en uno de PAV. Uno de los procedimientos explicados para ello se hizo a través de la Programación Entera; los cuales son bastante comunes en diversas disciplinas como en la ingeniería industrial, al presentar la problemática cuando se busca su optimización, puesto que cuando estos problemas son demasiados robustos el tiempo computacional crece hasta un punto en que resulta infactible su utilización para tomar decisiones dentro de la industria, ya que requieren respuestas inmediatas. Es por ello que los heurísticos muestran ser una herramienta muy útil para la búsqueda de respuestas próximas al óptimo; puesto que a pesar de encontrar la respuesta óptima en problemas de poca complejidad y tamaño, no es posible encontrar el óptimo en problemas de un mayor tamaño en un tiempo relativamente corto.

Por ello se recurre a los AG, los cuales son heurísticos que permitieron encontrar buenos resultados a los PAV presentados en esta investigación, en donde logramos encontrar parámetros que registraron los mejores resultados en los diferentes problemas. Estos parámetros fueron un número de individuos mayor al número de iteraciones, en una proporción de 10 individuos para 1 iteración aproximadamente. La probabilidad de cruce muestra los mejores resultados en 0.6 y la probabilidad de mutación utilizada fue relativamente baja de 1/n; en donde n representaba el número de ciudades del problema.

También se utilizaron dos torneos para poner a prueba el desempeño de cada uno de

ellos. El primero fue llamado AG determinístico y el segundo AG aleatorio. Como se observó en los resultados del determinístico, se encontraron mejores resultados que aquellos problemas resueltos a través del AG aleatorio;

El PAV resuelto por AG determinístico mostró resultados iguales a los óptimos con tiempos computacionales cortos cuando el número de ciudades a visitar fue relativamente pequeño, por ejemplo, el problema de 5 ciudades de [35] encontró el resultado óptimo de 668 unidades en un tiempo computacional de dos segundos, favorecidos por los parámetros utilizados que permitieron encontrar los resultados con un menor número de generaciones. Así también para el problema de 21 ciudades de Gerhard [28], en donde la complejidad computacional complica la solución a través de Programación Entera, se logró encontrar una solución óptima de 2 042 unidades con los AG. El problema de 10 ciudades de la tesis Manipulación Genética aplicada al Problema del Agente Viajero del Instituto Politécnico Nacional [34] obtuvo una solución de 8 914 Km, mientras que en nuestra experimentación se obtuvo una solución de 7 392 Km utilizando AG con un tiempo computacional de 3 segundos. Mientras tanto para el problema de 280 ciudades obtuvieron en [28] una distancia de 2579, en tanto que el mejor resultado obtenido a través de nuestro AG fue de 4 724.52 unidades con un tiempo computacional de 81 140 segundos.

El PAV finalmente fue utilizado para resolver Problemas de Secuenciación de Trabajos a través de los AG; en donde, al experimentar con problemas planteados en diferentes artículos científicos [4] [18], se encontraron resultados muy próximos a los planteados por ellos.

El Problema de Secuenciación de Trabajos mostró la complicación de realizar la codificación de este a un PAV y más aun cuando el número de trabajos y máquinas era demasiado grade, aunque en un futuro se podría solucionar con el desarrollo de un programa que apoye a la codificación. Las ventajas de utilizar el PAV para resolver el Problema de Secuenciación de Trabajos fueron mayores puesto que una vez codificado a un PAV, este es una simplificación del Problema de Secuenciación de Trabajos facilitando su solución. En la experimentación realizada para la Secuenciación de Trabajos, logramos igualar ciertos resultados aunque en su mayoría tan sólo logramos aproximarnos a la solución encontrada en los artículos científicos.

Con ello podemos concluir que los Problemas de Secuenciación de Trabajos que se presentan en la industria, pueden ser resueltos en tiempos adecuados, como lo de-

mandan las empresas, ajustando los tiempos para la realización de tareas a un PAV, el cual al resolverlo mediante un heurístico como lo son los AG obtiene resultados próximos al óptimo.

Referencias

[1] Church A. *An unsolvable problem of elementary number theory.* Estados Unidos de América, 1936.

[2] Homaifar A. *Schema analysis of the traveling salesman problem using genetic algorithms Complex Systems.* Computer Engineering NC AT University, 1992.

[3] Smith A. Integrated Facility Design using an Evolutionary Approach with a Subordinate Network Algorithm. *IEEE Transactions on Evolutionary Computation*, 1999.

[4] Tamilarasi A. An enhanced genetic algorithm with simulated annealing for job-shop scheduling. *International Journal of Engineering, Science and Technology* .pp. 144-151, 2010.

[5] Lehmann C. *Geometría Analítica.* México D.F., 1989.

[6] Maldonado C. *El mundo de las ciencias de la complejidad.* Colombia, 2001.

[7] Moon C. An efficient genetic algorithm for the traveling salesman problem with precedence constraints. *European Journal of Operational Research,* pp. 606-617, 2002.

[8] Applegate D. *The Traveling Salesman Problem: Computational Study* . Princeton University Press, USA, 2006.

[9] Fogel D. *An evolutionary approach to the traveling salesman problem.* San Diego, USA, 2004.

[10] Delgado E. Aplicación de Algoritmos Genéticos para la Programación de Tareas en una celda de manufactura. *Ingeniería e investigación;Universidad Nacional de Colombia.*pp. 24-31, 2005.

[11] Haupt E. Practical Genetic Algorithms. *Wiley-Intersciencie,* , 1998.

[12] Buthainah F. Enhanced Traveling Salesman Problem Solving by Genetic Algorithm Technique. *World Academy of Science, Engineering and Technology 38.* pp. 296-300, 2008.

[13] Vicentini F. Algoritmos heurísticos y el problema de job shop scheduling. *Universidad de Buenos Aires,* , 2003.

[14] Dantzig G. *Solution of a large scale traveling salesman problem.* The Rand Corporation, Santa Monica, California, USA, 1954.

[15] Mitsuo G. *The first Lauren series coefficients for singulary perturbed stochastic matrices.* Ashikaga, Japan, 2000.

[16] Holland J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, USA, 1975.

[17] Potvin J. Genetic algorithms for the traveling salesman problem Annals of Operations Research. *Centre de Recherche sur les Transports, Universitd de Montréal* pp. 337-370, 1996.

[18] Ruiz J. Complexity Indicators applied to the Job Shop Scheduling Problem. *International Journal of Combinatorial Optimization Problems and Informatics* .pp. 25-31, 2011.

[19] Bryant K. Genetic Algorithms and the Traveling Salesman Problem. , pp. 001-029, 2000.

[20] Chambers L. *Genetic Algorithms.* University Western, Australia, 1998.

[21] Snyder L. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*pp. 000-000, 2004.

[22] Dorigo M. Ant colonies for the traveling salesman problem. *Universit Libre de Bruxelles.* pp. 001-009, 1997.

[23] Mitchell M. *An Introduction to Genetic Algorithms* . Cambridge, Massachusetts London, England, 1996.

[24] Yamamura M. An Efficient Genetic Algorithm for Job Shop Scheduling Problems. *Departament of Intelligence Science,Graduate School of Interdisciplinary Science and Engineering,Tokyo Institute of Technology.*pp. 506-511, 1995.

[25] Zbigniew M. *Genetic Algorithms Data Structures Evolution Programs* . Springer, New York, USA, 1994.

[26] Jog P. Parallel Genetic Algorithms Applied to the Traveling Salesman Problem. *Jornal on Optimitation,* pp. 515-529, 1991.

[27] Larranaga P. Genetic Algorithms for the Travelling Salesman Problem: A review of Representations and Operators. *Artificial Intelligence Review,* pp. 129-170, 2000.

[28] Gerhard R. Discrete and combinatioral optimization. *Universidad de Heidelberg Instituto de Ciencias de la ComputaciÛn, Heidelberg Alemania*, 2006.

[29] Penrose R. *The Emperorís New Mind.* Reino Unido, 1990.

[30] Chatterjee S. Genetic algorithms and traveling salesman problem. *European Journal of Operational Research,* pp. 490-510, 1996.

[31] Cook S. The P versus NP Problem. *Clay Mathematics Institute*pp. 000-000, 2000.

[32] Bektas T. The multiple traveling salesman problem: an overviewof formulations and solution procedures. *The International Journal of Management Sciencie.* pp. 210-217, 2006.

[33] Cerny V. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Universit Libre de Bruxelles.* pp. 041-051, 1985.

[34] Rodríguez V. Manipulación Genética aplicada al Problema del Agente Viajero. *Instituto Politécnico Nacional* pp. 337-370, 2003.

[35] Winston W. *Investigación de Operaciones: Aplicaciones y algoritmos* . Indiana University, USA, 2005.