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

RESOLUCIÓN DEL PROBLEMA DE PLANIFICACIÓN DE TURNOS CONSIDERANDO EL MANEJO DE IMPREVISTOS UTILIZANDO ALGORITMOS GENÉTICOS

GIGLIOLA FRANCESCA TAVERNINI PIETRASANTA

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVILEN INFORMÁTICA

DICIEMBRE 2008

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

RESOLUCIÓN DEL PROBLEMA DE PLANIFICACIÓN DE TURNOS CONSIDERANDO EL MANEJO DE IMPREVISTOS UTILIZANDO ALGORITMOS GENÉTICOS

GIGLIOLA FRANCESCA TAVERNINI PIETRASANTA

Profesor Guía: **Guillermo Cabrera Guerrero**

Profesor Correferente: **José Miguel Rubio León**

Carrera: **Ingeniería Civil en Informática**

Diciembre 2008

A mi madre que con sus consejos me ha dado la seguridad para seguir mis sueños.

A mi padre que con su soporte ilimitado y su ejemplo ha hecho que mi sueño sea posible.

A mi hermano que siempre ha sido un modelo a seguir facilitando mi camino.

A mi hermana que es mi compañera, asesora y amiga.

A mis tios con los que siempre he contado, y en especial a Marisol que ha sido una amiga incondicional.

A mis primos, mis amigos queridos y familia en general, a todos los quiero mucho.

Agradezco a Dios por darme la oportunidad de existir y permitirme llegar a este momento tan especial y dichoso, y darme además padres tan especiales que han sido guía en mi vida y me han orientado a través de valores y apoyado en la senda de la vida.

A mis hermanos y a toda mi familia por el apoyo y amor absoluto e ilimitado. Al profesor Guillermo Cabrera por su apoyo permanente y oportunos comentarios que me han permitido acceder a este importante logro. A mis amigos por su preocupación y soporte.

RESUMEN

Las compañías de transporte enfrentan diversos desafíos en la planificación de los turnos primordialmente porque existen requerimientos legales y de servicios de calidad como son las leyes del tránsito que respectan a los horarios de trabajo, horas conducidas, salidas puntuales, etc. que apelan al buen uso de los recursos. El planeamiento de los recursos, particularmente el humano, toma importancia pues tiene una relación directa con los costos, los que con una buena asignación causan el ahorro de grandes sumas de dinero.

Se busca entonces un algoritmo que sea capaz de dar respuestas eficientes y oportunas a la planificación. Pocos son los que lo logran, pues es un problema muy complejo a causa de su naturaleza combinatorial.

Se propone el uso de Algoritmos Genéticos porque es capaz de generar soluciones óptimas a problemas pequeños y de alta calidad para grandes, tipo en que se encuentra incluido el problema a resolver.

Palabras Claves: Crew Scheduling Problem, Set Covering Problem, Algoritmos Genéticos.

ABSTRACT

The transport companies face several challenges in the crew scheduling, mainly because of the demand on legal and high quality services as the transit laws, for example hours of work, hours of driving, starts of jurneys at the pointed hour, etc. that appeal to the good use of the resources. The planning of these resources, specially the human one, is relevant due to its direct effect on the costs, whose right asignation can cause big amounts of money savings.

The purpose of this work is to create an algorithm able to provide efficient and opportune answers to the planification. No many have achived this purpose, due to the complexity of the problem because of this combinatorial nature.

In this work is proposed use the Metaheuristic of genetic algorithms for its ability to generate optimum solutions for small problems and high quality solutions for big problems, in which type is included the problem to solve.

ÍNDICE

GLOSARIO DE TÉRMINOS

Conductor Fijo: Aquel que tendrá asignada una única jornada de trabajo por cada tipo de periodo y en consecuencia todos los periodos del mismo tipo realizará el mismo trabajo.

Expedición: Es el trayecto de un vehículo entre su salida y su llegada al garaje.

Jornada de Trabajo: Conjunto de partes de trabajo efectuados por un mismo conductor en un día. Dentro de una jornada de trabajo los distintos trabajos tienen una secuencia determinada.

Leg: Corresponde a la unidad básica en la que puede ser descompuesto un pairing, es decir, un conjunto sucesivo conforman un pairing. Conjunto de tareas sucesivas, o también se puede definir como una parte de trabajo de un conductor en un día.

Línea: Una línea es un servicio regular de vehículos que recorren un itinerario determinado.

Punto de relevo: Se denomina punto de relevo al lugar donde un conductor puede ser reemplazado por otro.

Turno de Horario: Se define como una hora de inicio, una hora de finalización, y ocasionalmente un periodo de descanso entre ambas. En un sentido más general y dado que las jornadas de trabajo pueden empezar cada una de ellas en horas diferentes, se agrupan las que tienen un horario de trabajo similar formando lo que denomina también turnos horarios en un sentido más amplio. Por ejemplo turno de mañana, tarde o noche.

Viaje (Trip, pairing): Se denomina viaje o trayecto de una expedición comprendida entre dos puntos de relevo consecutivos.

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Capítulo 1

Descripción del Proyecto

1. Descripción del Proyecto

1.1 Introducción

Los métodos de programación matemática y los modelos matemáticos nacieron para dar solución a la necesidad de mejorar los procesos productivos y se han venido aplicando mayoritariamente a la distribución y organización de los recursos físicos y humanos. **[GC05]**

Desde hace unos años a este tiempo es que se han podido corroborar los resultados que esas mismas técnicas aportan cuando son empleadas para optimizar la eficacia de los recursos humanos, en específico dicha aplicación es particularmente interesante en las empresas de servicios, donde el potencial humano constituye el factor más importante y por lo tanto el coste más relevante.

Es en este escenario que se sitúa el problema de la organización de turnos, en particular en la organización de turnos en aquellas tareas donde, por sus características específicas la variedad de posibilidades alcanza una magnitud que lo hacen intratable de forma manual.

Dentro de esta problemática se enmarcan los organismos que tienen que cubrir unos servicios que se prolongan en el tiempo más allá de lo que es una jornada laboral, las que tienen una demanda de servicios fluctuante, las que deben ajustar su servicio a la demanda de un público incierto y a horarios extensos. Especialmente todas aquellas que disponen de la capacidad de mejorar la calidad de su servicio y la satisfacción de sus clientes y

colaboradores a base de distribuir de forma más eficiente el potencial humano de su empresa.

El transporte colectivo es un claro ejemplo de las organizaciones donde ocurren los eventos que hacen especialmente útil este tipo de planteamientos.

En primer lugar deben ajustar sus servicios al nivel de demanda de los usuarios, tanto en lo que respecta a itinerario de transporte como a la frecuencia del mismo. Posteriormente se deben ajustar los recursos físicos como son los buses, y por último es necesario compaginar todo esto con el calendario y horario laboral de los empleados.

La disposición de horarios en empresas de transporte colectivo es un problema muy complejo dadas las muchas variables que intervienen como son los horarios de los chóferes, sus horas trabajadas (con respecto a la ley de trabajo), la frecuencia de los recorridos, etc. Por lo tanto son muchas las combinaciones posibles que pueden formar parte de la solución buscada, es por esto que se busca que por medio de Algoritmos Genéticos se logre llegar a una solución tal, que considere en ella los imprevistos (sucesos que no han sido considerados y que al sucederse reiteradamente se hacen conocidon) y que se constituya en forma óptima.

Para una mejor comprensión un algoritmo genético (AG) es una técnica de programación que imita a la evolución biológica como estrategia para resolver problemas. Dado un problema específico a resolver, la entrada del AG es un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera, y una métrica llamada función de aptitud que permite evaluar cuantitativamente a cada candidata. Estas candidatas pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el AG las mejore, o que se generen aleatoriamente. Luego el AG evalúa cada candidata de acuerdo con la función de aptitud. En un acervo de candidatas generadas aleatoriamente la mayoría no funcionarán en absoluto, y serán eliminadas. Sin embargo, por puro azar, unas pocas pueden ser prometedoras y mostrar actividad, aunque sólo sea actividad débil e imperfecta, hacia la solución del problema. **[AM04]**

Esta problemática generalmente se plantea en cuatro etapas (las cuales pueden o no resolverse basadas en un AG) sucesivas, lo hacen así las compañías que simultáneamente a la resolución manual utilizan la automatización de algunas partes o el empleo de sistemas de soporte a la decisión.

Se enumeran a continuación las cuatro etapas **[FL97]** en que se divide el problema:

1. Organización de Líneas
2. Horarios de Vehículos o Vehicule Scheduling
3. Organización de las Jornadas de Trabajo o Crew Scheduling
4. Asignación de Conductores o Rostering

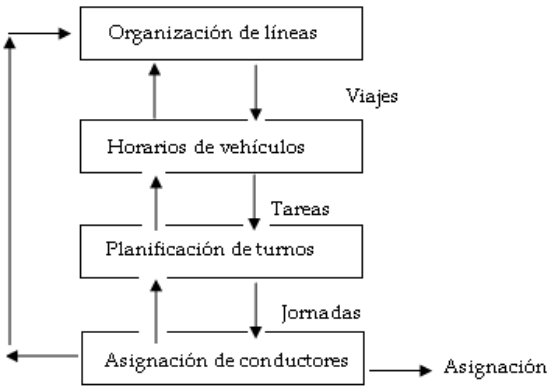


Ilustración 1.1: Plan de Transporte

El presente proyecto pretende abordar la penúltima de estas cuatro fases, es decir el problema de Crew Scheduling, planeación de turnos.

El problema, por lo tanto, consiste en dado un conjunto de jornadas de trabajo factibles se debe escoger un subconjunto de jornadas diarias que: permitan que todas las tareas sean cubiertas al menos una vez y que tenga un costo mínimo. Es decir encontrar una partición del conjunto de jornadas factible, que tenga el costo mínimo y cubra todas las tareas de forma tal que no se excedan las barreras legales como son el máximo de horas conducidas, vacaciones, descansos, etc. **[RP04]**

1.2 Descripción del Proyecto

El problema de planificación de turnos es una parte extremadamente compleja en el proceso de planeación en las compañías de transporte, no es un problema trivial, requiere de un análisis profundo, en el cual se tomen en cuenta una infinidad de factores que influyen sobre el modelado que se haga del problema. **[GC05]**

Su naturaleza combinatorial y la gran cantidad de problemas reales ha llevado al desarrollo de un vasto número de modelos y técnicas, que se aplican en la práctica acorde a la complejidad y a las características particulares de cada compañía.

El proceso de planificación comienza por la definición del horario de los vehículos, dirigiendo y minimizando el número de vehículos requeridos. Posteriormente el trabajo diario de cada vehículo se divide en trozos de trabajo que comienzan y finalizan en puntos de descanso, es decir; lugares en dónde los conductores pueden reemplazarse. Un conjunto de trozos de trabajo que satisfacen todas las restricciones es un pairing factible, puesto que un pairing es un trayecto entre dos puntos de relevo consecutivos y esta constituido asimismo por legs que son un conjunto de tareas sucesivas a cubrir al menos una vez. Una solución para el problema de BDSP (Bus Driver Scheduling Problem) es un conjunto de labores factibles que se espera cubran todos los viajes de los vehículos que han sido programados para una ruta o un conjunto de rutas. Varios objetivos y metas permitirán criterios para los costos y la calidad, y guiarán los procesos de construcción de la solución.

Para este tipo de problemas los AG son interesantes, porque no imponen una restricción particular en la estructura de la función objetivo, esto quiere decir que abarca diferentes características que son usualmente muy difíciles de manejar por los algoritmos tradicionales, condiciones lineales o diferenciales que no se aplican a la función objetivo, la que puede expresar varios criterios complejos, como un conjunto de objetivos para amplios rangos de labores. Pueden permitirse soluciones infactibles, pero la función objetivo debe incluir penalidades que traten de evadir soluciones con características indeseables, o

generar una función que permita arreglar la solución infactible de manera que cumpla con los requisitos. **[DS01]**

1.2.1 Descripción de Objetivos

1.2.1.1 Objetivo General

- Resolver mediante Algoritmos Genéticos el problema de Planificación de Turnos (Crew Scheduling) considerando el manejo de imprevistos.

1.2.1.2 Objetivos Específicos

- Realizar un estudio del estado del arte con respecto al problema, a fin de comprender la problemática que representa dentro de los servicios actuales.

- Especificar las implicancias del Término “Imprevisto” en la formulación del problema.

- Estudiar y lograr un conocimiento acabado de la Metaheurística (método heurístico para resolver problemas computacionales generales, que no tienen un algoritmo u heurística específico) definida, y elegir la forma más adecuada de esta para la resolución del problema de planificación de turnos.

- Obtener datos reales del problema a solucionar, con el objetivo de poder aplicar la solución propuesta, presentando finalmente los resultados obtenidos y contrastándolos con los resultados alcanzados actualmente.

- Creación de pautas de pruebas relativas a la Metaheurística elegida.

- Aplicación de las pautas de pruebas en los casos de estudio.

1.2.2 Organización del Texto

La presente memoria de Título se divide en 8 capítulos, los cuales a su vez se subdividen en secciones. Adicionales a estos se incluye una conclusión y las referencias que se utilizan a lo largo del desarrollo del presente documento. Finalmente existe un anexo que contiene al código fuente en C++ y que permite un mejor entendimiento técnico de la herramienta.

El presente capítulo exhibe los distintos objetivos que busca esta memoria, además de cómo se organizó el texto.

Los capítulos 3 y 4 dan una introducción a los términos abarcados en el presente documento, como también la distribución de este.

Los capítulos 5 y 6 exponen el modelo del problema, además de las fases que lo conforman. De la misma forma explican el algoritmo propuesto que soluciona el problema propuesto.

El capítulo 7 revela las distintas pruebas con datos de diversas empresas y sus respectivos resultados y comparaciones con otros estudios.

El capítulo 8 adiciona los datos de los imprevistos a los datos generados en el capítulo 7 y los prueba nuevamente en la herramienta.

Finalmente el capítulo 9 entrega las conclusiones generadas, como también una propuesta para un trabajo futuro.

Capítulo 2

Estado del Arte

2. Estado del Arte

2.1 Problema Crew Scheduling

Corresponde a la asignación de tareas bien definidas (pareamiento y construcción de turnos de trabajo) a un grupo de personas respetando un conjunto de reglas complicadas legalmente, restricciones y distribución de los recursos. La mayoría de las reglas legales son no-lineales y con evolución en el tiempo.

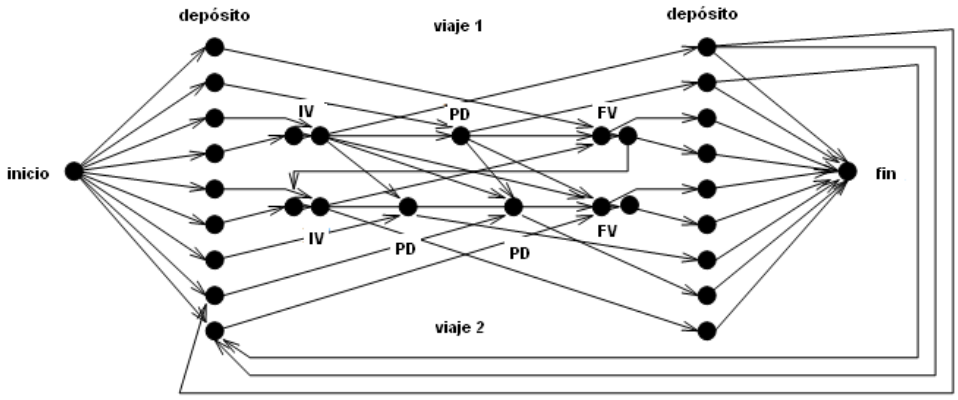


Ilustración 2.1: Gráfico Simple CSP

Es decir el problema de la planificación de equipos, tomando de manera genérica, busca encontrar una solución óptima para la asignación de recursos escasos (en particular mano de obra y/o tiempo) sujeto a una serie de restricciones que por lo general obedecen a restricciones del tipo laboral, capacidades de las personas, etc. Como muestra la figura 2.1 existen varias rutas, las cuales deben ser cubiertas por los viajes al menos una vez, ademas que deben cumplir un horario establecido y las restricciones de horario de los choferes. Sin

embargo no debe confundirse esta problemática con la que plantean los problemas de programación lineal (de tipo triviales), en donde ya existen suficientes técnicas y herramientas computacionales para encontrar la solución óptima que satisfaga una serie de restricciones de tipo entera o binaria. **[GC05]**

Este problema es del tipo NP-Complejo **[RK72] [GJ79]** por varias razones, una de las cuales es que tienen un número grande de posibles soluciones (sobre 10 soluciones), además se tienen soluciones objetivo de naturaleza no lineal, complicando aún más el problema.

Un problema se encuentra dentro del conjunto NP, si y sólo si, puede resolverse mediante un algoritmo no determinístico en tiempo polinomial. El nombre NP proviene de "non-deterministic polynomial-bounded" (polinomialmente acotado no determinístico).

2.2 Problema de Set Covering

El Set Covering Problems (SCP) es un problema clásico que consiste en encontrar el número mínimo de conjuntos que contengan todos los elementos de todos los conjuntos dados al menor costo posible. El SCP al igual que otros problemas como: el Set Packing Problem (SPP) y El Set Partitioning Problem (SPaP) son problemas pertenecientes a la clase *NP-completo*, esto ya que las soluciones son demasiado complejas y no poseen la certeza que logren encontrar una respuesta exacta en un tiempo razonable. Se puede decir que los problemas de NP-completo son los problemas más difíciles de NP y muy probablemente no formen parte de la clase de complejidad P. La razón es que de tenerse una solución polinómica para un problema de NP-completo, todos los problemas de NP tendrían también una solución en tiempo polinómico.

Dada la dificultad de determinar la solución óptima y el gran tamaño de los problemas reales, es que se han ideado varios algoritmos de optimización y algoritmos basados en heurística como son simulated anneling, tabu search, GRASP, y los algoritmos genéticos entre otros. Aunque se ha probado que la existencia de un algoritmo polinomial

que se aproxime a la solución óptima en un valor cercano a $1/4\,\ell og_m$, implica que P = NP **[LY92]**, en la práctica la mayoría de las heurísticas que aparecen en la literatura se acercan muy eficientemente a las soluciones óptimas **[FC95].** En general, dada la estructura del problema de los casos del mundo real y al considerable esfuerzo en desarrollar algoritmos que rindan más y mejor en esos casos, el actual estado del arte del SCP es que los casos con unos cientos de filas y unas miles de columnas pueden resolverse obteniendo resultados óptimos y los casos con unos miles de filas y unos millones de columnas pueden resolverse obteniendo un valor dentro del 1% alrededor del óptimo en un tiempo de ejecución computacional razonable. La entrada esta dada por varios conjuntos de elementos o datos que tienen algún elemento en común. Estos tres problemas poseen la misma función objetivo y se diferencian por las restricciones que deben cumplir. **[PI05]**

Existen muchas aplicaciones de este tipo de problema, siendo las principales localización de servicios, selección de archivos en una base de datos, simplificación de expresiones booleanas, balanceo de líneas de producción, entre otros. El termino coverage (cobertura) es asociado con la capacidad que poseen los candidatos Ni de cubrir la demanda de sus clientes. Este sistema puede ser también especificado en términos de coeficientes binarios a_{ij}, el cual toma un valor 1 si el sitio candidato de locación puede cubrir la demanda del nodo i, en caso contrario tendrá el valor cero.

Este problema Set Covering se relaciona con el proyecto ya que la segunda fase, optimización de pairings, se modela como tal, y puesto que como ya se ha indicado anteriormente este busca encontrar el número mínimo de conjuntos que contengan todos los elementos de todos los conjuntos dados al menor costo posible, esto se enlaza con el problema en que todos los legs que conforman los pairings deben estar cubiertos al menos una vez y en la menor cantidad de pairings, puesto que los pairings se encuentran asociados a un costo el cual se busca minimizar.

En la literatura existente se ha podido comprobar que la efectividad de los resultados depende de la heurística aplicada y por el número de filas y columnas que se

están utilizando (demanda, candidatos). En donde a medida que aumenta la cantidad de filas y columnas que se desean examinar aumenta la complejidad y tiempo de respuesta.

Las múltiples opciones que cubren toda la demanda se diferencian por los diferentes costos que significa ubicar el servicio o elemento en un lugar u otro. Es por esta razón que la misión del problema Set Covering busca encontrar aquellas posiciones estratégicas que minimicen los costos totales.

Para un mejor entendimiento de lo que plantea este problema se presenta el siguiente ejemplo: Una ciudad ha sido dividida en 20 sectores. Dentro de esta ciudad se han elegido 10 lugares posibles donde sería posible instalar algún tipo de servicio como una sede de cruz roja, hospital, estación de carabineros, colegios, centros de Internet, etc. Esta decisión estratégica donde ubicar este servicio en particular, es un claro ejemplo de los problemas de Set Covering.

Para este caso va existir un vector de costos C_j = [C_1, C_2, C_3 …, C_{10}] que representan los costos que significa instalar en cada sitio el servicio, M = [1,…,20] y N = [1,…,10]. Los valores a_{ij} de la matriz tendrán el valor 1 si existe una cobertura entre la columna j y la fila i, por ejemplo la columna j=9 cubre la fila i=14, i=15, i=18, i=19.

Luego tomando en cuenta que x_j, será igual a 1 cuando instala el servicio en la locación j y cero en caso contrario se buscará encontrar el vector binario X= [x_1, x_2, …., x_{10}] que cumpla:

Minimice $\sum_{1}^{10} C_j * X_j$

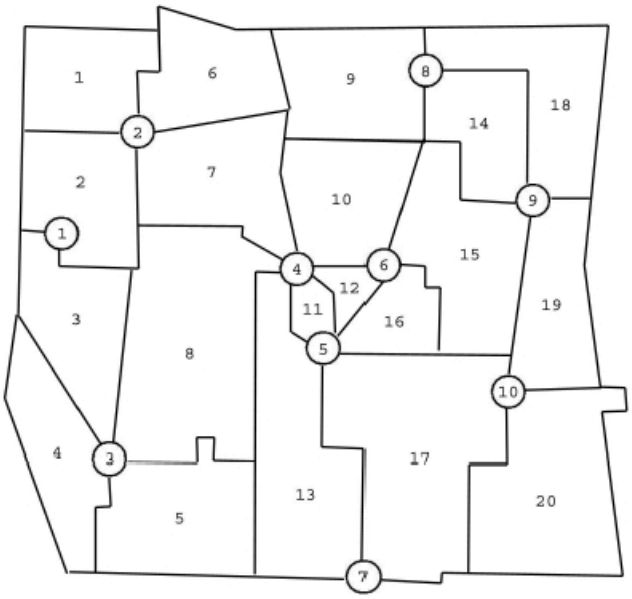


Ilustración 2.2: Sectores de una ciudad y posibles lugares de instalación de servicios

La solución de este problema indicará dónde se deben ubicar los servicios para que cubran los sectores a un costo mínimo. **[PI05]**

Es trabajo de una adecuada heurística encontrar aquella solución que cubra toda la demanda al menor costo posible.

2.3 Imprevistos

Un acontecimiento es imprevisto cuando ocurre por primera vez, pero si se repite insistentemente (aún bajo formas aparentemente distintas), se convierte en previsible. En un primer instante, el acontecimiento imprevisto es considerado (y visto) como tal, pero luego de un tiempo (para comprender) se puede llegar a (el momento de) concluir que no lo habrá sido tanto como parecía al principio.

Como ejemplo de imprevistos relacionados con el problema se pueden mencionar el tema de las condiciones de niebla o tormenta, o accidentes de tipo técnico o mejor dicho fallas mecánicas ya sea que se reviente un neumático, se rompa la dirección del vehículo, se averíen los frenos, se quiebre una rotula, etc.

Los imprevistos son importantes para el presente problema ya que posibilita un nivel superior de beneficios para el cliente, que finalmente es quien hace la elección de cual empresa preferir en función de los beneficios entregados por esta. El presente trabajo busca entregar un nivel en el cual no sólo se resuelva la asignación de los turnos, sino más bien un plus adicional a la resolución clásica del problema, como es el considerar los imprevistos. Es necesario adaptar los datos de los imprevistos a los datos del problema, a los pairings asociados, aunque cabe destacar que el modelo propuesto para resolver la asignación de los turnos no varía con esta variable, sino más bien alcanza un aumento en la cantidad de datos ingresados, esto tiene sentido pues con los imprevistos mejoran el servicio aumentando los datos, pero no varía el modelo, sino que los datos incrementados se encausan en él.

2.4 Algoritmos Genéticos

Un algoritmo genético (AG) es una técnica de programación que imita a la evolución biológica como estrategia para resolver problemas. Dado un problema específico a resolver, la entrada del AG es un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera, y una métrica llamada función de aptitud que permite evaluar cuantitativamente a cada candidata. Estas candidatas pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el AG las mejore, pero se suelen generar aleatoriamente.

Luego el AG evalúa cada candidata de acuerdo con la función de aptitud. En un conjunto de candidatas generadas aleatoriamente, por supuesto, la mayoría no funcionarán en absoluto, y serán eliminadas. Sin embargo, por puro azar, unas pocas pueden ser prometedoras; pueden mostrar actividad, aunque sólo sea actividad débil e imperfecta, hacia la solución del problema.

Estas candidatas prometedoras se conservan y se les permite reproducirse. Se realizan múltiples copias de ellas, pero las copias no son perfectas; se introducen cambios aleatorios durante el proceso de copia. Luego, esta descendencia digital prosigue con la siguiente generación, formando una nueva acumulación de soluciones candidatas, y son

sometidas a una ronda de evaluación de aptitud. Las candidatas que han empeorado o no han mejorado con los cambios en su código son eliminadas de nuevo; pero, de nuevo, por puro azar, las variaciones aleatorias introducidas en la población pueden haber mejorado a algunos individuos, convirtiéndolos en mejores soluciones del problema, más completas o más eficientes. De nuevo, se seleccionan y copian estos individuos vencedores hacia la siguiente generación con cambios aleatorios, y el proceso se repite. Las expectativas son que la aptitud media de la población se incrementará en cada ronda y, por tanto, repitiendo este proceso cientos o miles de rondas, pueden descubrirse soluciones muy buenas del problema.

Los algoritmos genéticos han demostrado ser una estrategia enormemente poderosa y exitosa para resolver problemas, demostrando de manera espectacular el poder de los principios evolutivos. Se han utilizado algoritmos genéticos en una amplia variedad de campos para desarrollar soluciones a problemas tan difíciles o más difíciles que los abordados por los diseñadores humanos. Además, las soluciones que consiguen son a menudo más eficientes, más elegantes o más complejas que nada que un ingeniero humano produciría. **[AM04]**

Esto nos lleva a que el AG sea el elegido para la fase de *optimización de pairings,* de forma tal que cada leg este incorporada en al menos un pairing.
La exploración es la capacidad de mostrar diferentes partes del espacio de búsqueda en la población del algoritmo, la explotación corresponde a la capacidad de tunning y combinación de las soluciones óptimas. Los algoritmos genéticos poseen operadores genéticos como son la mutación (pequeña modificación de los genes del individuo) y el cruzamiento (recombinación de las carácterísticas de los cromosomas para generar nuevos organismos), los cuales tienden a explorar el espacio de solución. El cruzamiento explota la vecindad de cada solución. La exploración es útil para no estancarse en óptimos locales, y la explotación es útil para obtener el óptimo global una vez que se ha aproximado a él lo suficiente.

2.4.1 Métodos de Representación

Antes de que un algoritmo genético pueda ponerse a trabajar en un problema, se necesita un método para codificar las soluciones potenciales del problema de forma que una computadora pueda procesarlas. Un enfoque común es codificar las soluciones como cadenas binarias: secuencias de 1s y 0s, donde el dígito de cada posición representa el valor de algún aspecto de la solución. Otro método similar consiste en codificar las soluciones como cadenas de enteros o números decimales, donde cada posición, de nuevo, representa algún aspecto particular de la solución. Este método permite una mayor precisión y complejidad que el método comparativamente restringido de utilizar sólo números binarios, y a menudo "está intuitivamente más cerca del espacio de problemas". **[FP02]** Un tercer método consiste en representar a los individuos de un AG como cadenas de letras, donde cada letra representa un aspecto específico de la solución.

La virtud de estos tres métodos es que facilitan la definición de operadores (cruzamiento y mutación) que causen los cambios aleatorios en las candidatas seleccionadas: cambiar un 0 por un 1 o viceversa, sumar o restar al valor de un número una cantidad elegida al azar, o cambiar una letra por otra.

2.4.2 Métodos de Selección

Un algoritmo genético puede utilizar muchas técnicas diferentes para seleccionar a los individuos que deben copiarse hacia la siguiente generación. Algunos de estos métodos son mutuamente excluyentes, pero otros pueden utilizarse en combinación, lo que se hace a menudo.

Selección elitista: se garantiza la selección de los miembros más aptos de cada generación. (La mayoría de los AGs no utilizan elitismo puro, sino que usan una forma modificada por la que el individuo mejor, o algunos de los mejores, son copiados hacia la siguiente generación en caso de que no surja nada mejor). En el presente documento se utiliza este

método de selección, ya que el algoritmo elige a los dos individuos con mejor fitness (función objetivo que comprende el costo), o sea los con menor costo.

Selección proporcional a la aptitud: los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.

Selección por rueda de ruleta: una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. (Conceptualmente, esto puede representarse como un juego de ruleta cada individuo obtiene una sección de la ruleta, pero los más aptos obtienen secciones mayores que las de los menos aptos. Luego la ruleta se hace girar, y en cada vez se elige al individuo que "posea" la sección en la que se pare la ruleta).

Selección escalada: al incrementarse la aptitud media de la población, la fuerza de la presión selectiva también aumenta y la función de aptitud se hace más discriminadora. Este método puede ser útil para seleccionar más tarde, cuando todos los individuos tengan una aptitud relativamente alta y sólo les distingan pequeñas diferencias en la aptitud.

Selección por torneo: se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción. A cada individuo de la población se le asigna un rango numérico basado en su aptitud, y la selección se basa en este ranking, en lugar de las diferencias absolutas en aptitud. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.

Selección generacional: la descendencia de los individuos seleccionados en cada generación se convierte en toda la siguiente generación. No se conservan individuos entre las generaciones.

Selección por estado estacionario: la descendencia de los individuos seleccionados en cada generación vuelven al acervo genético preexistente, reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.

Selección jerárquica: los individuos atraviesan múltiples rondas de selección en cada generación. Las evaluaciones de los primeros niveles son más rápidas y menos discriminatorias, mientras que los que sobreviven hasta niveles más altos son evaluados más rigurosamente. La ventaja de este método es que reduce el tiempo total de cálculo al utilizar una evaluación más rápida y menos selectiva para eliminar a la mayoría de los individuos que se muestran poco o nada prometedores, y sometiendo a una evaluación de aptitud más rigurosa y computacionalmente más costosa sólo a los que sobreviven a esta prueba inicial.

2.4.3 Métodos de Cambio

Una vez que la selección ha elegido a los individuos aptos, éstos deben ser alterados aleatoriamente con la esperanza de mejorar su aptitud para la siguiente generación. Existen dos estrategias básicas para llevar esto a cabo. La primera y más sencilla se llama ***mutación.*** Al igual que una mutación en los seres vivos cambia un gen por otro, una mutación en un algoritmo genético también causa pequeñas alteraciones en puntos concretos del código de un individuo.

El segundo método se llama ***cruzamiento***, e implica elegir a dos individuos para que intercambien segmentos de su código, produciendo una "descendencia" artificial cuyos individuos son combinaciones de sus padres. Este proceso pretende simular el proceso análogo de la recombinación que se da en los cromosomas durante la reproducción sexual. Las formas comunes de cruzamiento incluyen al cruzamiento de un punto, en el que se establece un punto de intercambio en un lugar aleatorio del genoma de los dos individuos, y uno de los individuos contribuye todo su código anterior a ese punto y el otro individuo contribuye todo su código a partir de ese punto para producir una descendencia, y al cruzamiento uniforme, en el que el valor de una posición dada en el genoma de la

descendencia corresponde al valor en esa posición del genoma de uno de los padres o al valor en esa posición del genoma del otro padre, elegido con un 50% de probabilidad.

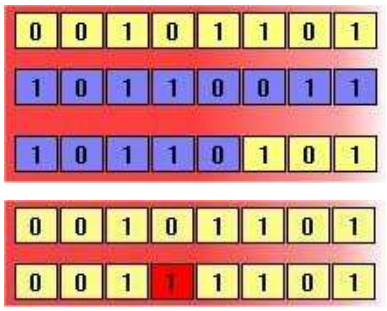


Ilustración 2.3: Cruzamiento y Mutación

La ilustración de arriba muestra el efecto de estos dos operadores genéticos en los individuos de una población de cadenas de 8 bits. El diagrama superior muestra a dos individuos llevando a cabo un cruzamiento de un punto; el punto de intercambio se establece entre las posiciones quinta y sexta del genoma, produciendo un nuevo individuo que es híbrido de sus progenitores. El segundo diagrama muestra a un individuo sufriendo una mutación en la posición 4, cambiando el 0 de esa posición de su genoma por un 1. **[AM04]**

En general el porcentaje de mutación es bajo (del orden de 0,1 a 5 %) mucho menor al porcentaje de cruzamiento. Esto permite asegurar la diversidad de la población, y el cruzamiento debe ser superior al 80%, si embargo en algunos problemas puede ser menor.

2.4.4 Ventajas

El primer y más importante punto es que los algoritmos genéticos son intrínsecamente paralelos. La mayoría de los otros algoritmos son en serie y sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al mismo tiempo, y si la solución que descubren resulta sub-óptima, no se puede hacer otra cosa que abandonar todo el trabajo hecho y empezar de nuevo. Sin embargo, ya que los AGs tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez. Si un camino resulta ser un callejón sin salida, pueden eliminarlo fácilmente y continuar el trabajo en avenidas más prometedoras, dándoles una mayor probabilidad en cada ejecución de encontrar la solución. **[AM04]**

Sin embargo, la ventaja del paralelismo va más allá de esto. Considere lo siguiente: todas las cadenas binarias (cadenas de ceros y unos) de 8 dígitos forman un espacio de búsqueda, que puede representarse como ******** (donde * significa “0 o 1”). La cadena 01101010 es un miembro de este espacio. Sin embargo, también es un miembro del espacio 0*******, del espacio 01******, del espacio 0******0, del espacio 0*1*1*1*, del espacio 10*01**0, etcétera. Evaluando la aptitud de esta cadena particular, un algoritmo genético estaría sondeando cada uno de los espacios a los que pertenece. Tras muchas evaluaciones, iría obteniendo un valor cada vez más preciso de la aptitud media de cada uno de estos espacios, cada uno de los cuales contiene muchos miembros. Por tanto, un AG que evalúe explícitamente un número pequeño de individuos está evaluando implícitamente un grupo de individuos mucho más grande de la misma manera que un encuestador que le hace preguntas a un cierto miembro de un grupo étnico, religioso o social espera aprender algo acerca de las opiniones de todos los miembros de ese grupo, y por tanto puede predecir con fiabilidad la opinión nacional sondeando sólo un pequeño porcentaje de la población. De la misma manera, el AG puede dirigirse hacia el espacio con los individuos más aptos y encontrar el mejor de ese grupo. En el contexto de los algoritmos evolutivos, esto se conoce como teorema del esquema, y es la ventaja principal de los AGs sobre otros métodos de resolución de problemas. **[HMG]**

Debido al paralelismo que les permite evaluar implícitamente muchos esquemas a la vez, los algoritmos genéticos funcionan particularmente bien resolviendo problemas cuyo espacio de soluciones potenciales es realmente grande, demasiado vasto para hacer una búsqueda exhaustiva en un tiempo razonable. En un problema lineal, la aptitud de cada componente es independiente, por lo que cualquier mejora en alguna parte dará como resultado una mejora en el sistema completo. No es necesario decir que hay pocos problemas como éste en la vida real. La no linealidad es la norma, donde cambiar un componente puede tener efectos en cadena en todo el sistema, y donde cambios múltiples que, individualmente, son perjudiciales, en combinación pueden conducir hacia mejoras en la aptitud mucho mayores.

Otra ventaja notable de los algoritmos genéticos es que se desenvuelven bien en problemas con un paisaje adaptativo complejo aquéllos en los que la función de aptitud es discontinua, ruidosa, cambia con el tiempo, o tiene muchos óptimos locales. La mayoría de los problemas prácticos tienen un espacio de soluciones enorme, imposible de explorar exhaustivamente; el reto se convierte entonces en cómo evitar los óptimos locales soluciones que son mejores que todas las que son similares a ella, pero que no son mejores que otras soluciones distintas situadas en algún otro lugar del espacio de soluciones. Muchos algoritmos de búsqueda pueden quedar atrapados en los óptimos locales: si llegan a lo alto de una colina del paisaje adaptativo, descubrirán que no existen soluciones mejores en las cercanías y concluirán que han alcanzado la mejor de todas, aunque existan picos más altos en algún otro lugar del mapa. **[HMG]**

Los algoritmos evolutivos, por otro lado, han demostrado su efectividad al escapar de los óptimos locales y descubrir el óptimo global incluso en paisajes adaptativos muy escabrosos y complejos. (Debe decirse que, en la realidad, a menudo no hay manera de decir si una cierta solución a un problema es el óptimo global o sólo un óptimo local muy alto. Sin embargo, aunque un AG no devuelva siempre una solución perfecta y demostrable a un problema, casi siempre puede devolver al menos una muy buena solución). Todos los cuatro componentes principales de los AGs: paralelismo, selección, mutación y cruzamiento trabajan juntos para conseguir esto. Al principio, el AG genera una población inicial diversa, lanzando una "red" sobre el paisaje adaptativo. Pequeñas mutaciones permiten a cada individuo explorar sus proximidades, mientras que la selección enfoca el progreso, guiando a la descendencia del algoritmo cuesta arriba hacia zonas más prometedoras del espacio de soluciones. **[HMG]**

Sin embargo, el cruzamiento es el elemento clave que distingue a los algoritmos genéticos de los otros métodos como el Hill Climbing y el Simulated Anneling. Sin el cruzamiento, cada solución individual va por su cuenta, explorando el espacio de búsqueda en sus inmediaciones sin referencia de lo que el resto de individuos puedan haber descubierto. Sin embargo, con el cruzamiento en juego, hay una transferencia de información entre los candidatos prósperos, los individuos pueden beneficiarse de lo que

otros han aprendido, y los esquemas pueden mezclarse y combinarse, con el potencial de producir una descendencia que tenga las virtudes de sus dos padres y ninguna de sus debilidades. En una generación se seleccionaron dos circuitos progenitores para llevar a cabo el cruzamiento; un padre tenía una buena topología (componentes como inductores y condensadores colocados en el sitio correcto) pero malos tamaños (valores demasiado bajos de inductancia y capacidad para los componentes). El otro padre tenía mala topología pero buenos tamaños. El resultado de aparearlos mediante cruzamiento fue una descendencia con la buena topología de un padre y los buenos tamaños del otro, dando como resultado una mejora sustancial de la aptitud sobre sus dos padres.

El problema de encontrar el óptimo global en un espacio con muchos óptimos locales también se conoce como el dilema de la exploración versus explotación, "un problema clásico de todos los sistemas que pueden adaptarse y aprender" **[HMG]**. Una vez que un algoritmo (o un diseñador humano) ha encontrado una estrategia para resolver problemas que parece funcionar satisfactoriamente, ¿debería centrarse en hacer el mejor uso de esa estrategia, o buscar otras? Abandonar una estrategia de probada solvencia para buscar otras nuevas casi garantiza que supondrá una pérdida y degradación del rendimiento, al menos a corto plazo. Pero si uno se queda con una estrategia particular excluyendo a todas las demás, corre el riesgo de no descubrir estrategias mejores que existen pero no se han encontrado. De nuevo, los algoritmos genéticos han demostrado ser muy buenos en dar con este equilibrio y descubrir buenas soluciones en un tiempo y esfuerzo computacional razonables.

Otra área en el que destacan los algoritmos genéticos es su habilidad para manipular muchos parámetros simultáneamente. Muchos problemas de la vida real no pueden definirse en términos de un único valor que hay que minimizar o maximizar, sino que deben expresarse en términos de múltiples objetivos, a menudo involucrando contrapartidas: uno sólo puede mejorar a expensas de otro. Los AGs son muy buenos resolviendo estos problemas: en particular, su uso del paralelismo les permite producir múltiples soluciones, igualmente buenas, al mismo problema, donde posiblemente una solución candidata optimiza un parámetro y otra candidata optimiza uno distinto y luego un

supervisor humano puede seleccionar una de esas candidatas para su utilización. Si una solución particular a un problema con múltiples objetivos optimiza un parámetro hasta el punto en el que ese parámetro no puede mejorarse más sin causar una correspondiente pérdida de calidad en algún otro parámetro, esa solución se llama óptimo paretiano o no dominada. **[CC00]**

Finalmente, una de las cualidades de los algoritmos genéticos que, a primera vista, puede parecer un desastre, resulta ser una de sus ventajas: a saber, los AGs no saben nada de los problemas que deben resolver. En lugar de utilizar información específica conocida a priori para guiar cada paso y realizar cambios con un ojo puesto en el mejoramiento, como hacen los diseñadores humanos, son "relojeros ciegos" **[RD96]**; realizan cambios aleatorios en sus soluciones candidatas y luego utilizan la función de aptitud para determinar si esos cambios producen una mejora.

La virtud de esta técnica es que permite a los algoritmos genéticos comenzar con una mente abierta, por así decirlo. Como sus decisiones están basadas en la aleatoriedad, todos los caminos de búsqueda posibles están abiertos teóricamente a un AG; en contraste, cualquier estrategia de resolución de problemas que dependa de un conocimiento previo, debe inevitablemente comenzar descartando muchos caminos a priori, perdiendo así cualquier solución novedosa que pueda existir **[JK99]**. Los AGs, al carecer de ideas preconcebidas basadas en creencias establecidas sobre "cómo deben hacerse las cosas" o sobre lo que "de ninguna manera podría funcionar", los AGs no tienen este problema. De manera similar, cualquier técnica que dependa de conocimiento previo fracasará cuando no esté disponible tal conocimiento, pero los AGs no se ven afectados negativamente por la ignorancia. Mediante sus componentes de paralelismo, cruzamiento y mutación, pueden viajar extensamente por el paisaje adaptativo, explorando regiones que algoritmos producidos con inteligencia podrían no haber tenido en cuenta, y revelando potencialmente soluciones de asombrosa e inesperada creatividad que podrían no habérseles ocurrido nunca a los diseñadores humanos. Un ejemplo muy gráfico de esto es el redescubrimiento, mediante la programación genética, del concepto de retroalimentación negativa, un principio crucial para muchos componentes electrónicos importantes de hoy en día, pero un

concepto que, cuando fue descubierto en primera instancia, se le denegó una patente de nueve años porque el concepto era demasiado contrario a las creencias establecidas **[JK99]**. Por supuesto, los algoritmos evolutivos no están enterados ni preocupados de si una solución va en contra de las creencias establecidas sólo de si funciona la solución.

2.4.5 Limitaciones

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima, del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia. El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridándolas con los Algoritmos Genéticos.

Tienen la desventaja de requerir de mucho tiempo de procesador. La evaluación de la función de desempeño suele tomar en muchos casos un tiempo considerable. Esta se debe evaluar para cada nuevo cromosoma generado, y esto se debe hacer por varias generaciones.

Pueden converger prematuramente sin llegar al óptimo, debido a una serie de problemas de diversa índole, es decir; a diferencia de otras técnicas evolutivas, en general no es posible asegurar la convergencia de los algoritmos genéticos al conjunto de soluciones óptimas.

Capítulo 3

Desarrollo del Problema

3. Desarrollo del Problema

3.1 Descripción del Problema

El problema consiste en dado un conjunto de jornadas de trabajo factibles escoger un subconjunto de jornadas que:

- Permitan que todas las tareas sean cubiertas al menos una vez y,
- Que tenga un costo mínimo.

Es decir encontrar una partición del conjunto de jornadas factibles, que tenga el costo mínimo y cubra todas las tareas, o la partición de costo mínimo. **[FG06]**

Asimismo se puede decir que el problema puede definirse como encontrar el mínimo costo de un conjunto de duties o Jornadas de trabajo diarias (periodo de tiempo durante el cual el trabajador trabaja). Los costos de las jornadas de trabajo generalmente dominan los costos totales de la operación que cubren todos los viajes o los bloques de vehículos (sucesión de asignaciones del Vehicle Schedule entre dos depósitos). Son los componentes básicos de los vehicle Schedule, itinerario de un vehículo entre el departamento al depósito y hasta que vuelva de nuevo al depósito. Cada bloque de un vehículo puede dividirse en porciones o piezas de trabajo, esa división sólo puede ocurrir en un punto de descanso (punto espacio-tiempo, donde se puede realizar o no un relevo de equipo, tomarse un break o finalizar la jornada de trabajo.), es decir donde es posible que los conductores se puedan cambiar. Una jornada de trabajo es un conjunto de porciones de trabajo y que pueden asignarse a un conductor.

3.1.1 Restricciones

- Los conductores no deben trabajar más de 5 horas continuas.
- La estación de inicio de un pairing (viaje) debe ser la misma de llegada, y debe ser una estación base.
- Los descansos deben ser superiores a media hora (30 minutos) e inferiores a 2 horas (120 minutos).
- Aquellos pairings que comiencen primeros han de terminar antes de otro que haya comenzado posterior a él.
- El equipo perteneciente a un vehículo esta compuesto sólo por el conductor.

3.1.2 Estrategia de Resolución

A causa de la complejidad del problema en relación a la cantidad de restricciones, espacio de búsqueda, etc. es que se hace necesaria su división, pues permite abordar el problema de manera modular y con detallismo. El problema será dividido en dos partes:

La primera será la ***generación de pairings,*** en donde actúan las restricciones de la función objetivo y que tienen relación con los costos del problema. El número de pairings factibles generados debe ser suficientemente grande como para surtir las soluciones necesarias. Los pairings (viajes; Pareja (Origen, tiempo salida) - (Destino, tiempo llegada)) están compuestos por los legs y son representados en el timetable (vehicle schedule).
En esta actividad la entrada son todas las legs del timetable.

Los pairings utilizados para hacer o probar el software serán obtenidos de los archivos de la OR-Library (es una biblioteca que posee archivos con una colección de conjuntos de datos de prueba para una variedad de problemas de Investigación de Operaciones), ya que esta posee archivos con los pairings factibles, asociado además el costo del pairing. **[OR90]**

Se genera una ***optimización de las soluciones*** obtenidas en la primera etapa (subconjunto de pairings) y que dan como resultado la solución factible que cubre todas las

restricciones del problema, de forma tal que cada leg en la lista este incorporada en al menos un pairing. Tradicionalmente esta etapa para en el problema de la planificación de turnos se formula como un problema de Set Covering y es independiente de la primera parte que relaciona a los costos.

3.2 Presentación del Modelo

El problema se Crew Scheduling formulado como Set covering es un problema clásico del tipo NP-Completo, y en donde la entrada esta dada por unos varios conjuntos de elementos o datos que tienen algún elemento en común. En general, estos problemas consisten en encontrar el número mínimo de conjuntos que contengan un cierto número de elementos de todos los conjuntos como se menciono anteriormente. En otras palabras, consiste en encontrar un conjunto de soluciones que permitan cubrir un conjunto de necesidades al menor costo posible. Un conjunto de necesidades corresponde a las filas, y el conjunto solución es la selección de columnas que cubren en forma óptima el conjunto de filas.

Este puede ser formulado matemáticamente con la siguiente notación:

Entrada: a_{ij} = 1, si la vecindad del nodo i es adyacente al nodo j

a_{ij} = 0, si no.

C_j = costo de ubicar el candidato en el sitio j.

Se puede explicar también el término a_{ij} como un valor igual a 1 si el sitio candidato j puede cubrir la demanda del nodo i. Y un valor cero en caso contrario.

Variables de decisión: X_j = 1 si localizamos en el sitio j al candidato.

X_j = 0 si no

Con esta notación, podremos formular el problema de Set covering como sigue:

Mínimo $\sum_j C_j * X_j$ **(1)**

En donde $\sum_j a_{ij} * X_j > 1 \quad \forall i$ **(2)**

$X_j = 0,1 \qquad \forall j$ **(3)**

(1): Función objetivo minimiza el costo total de seleccionar la locación j,

(2): Restricción, estipula que cada nodo debe ser cubierto por al menos una localización. Nótese que la parte izquierda de las función (2) entrega el número de puntos que cubre la demanda del nodo i. En el caso de que el costo de todas las instalaciones sea semejante (por ejemplo C_j =1), la función objetivo a minimizar se simplifica a:

Mínimo $\sum_j X_j$ **(4)**

(4): En este caso la función lo que nos responde, es la cantidad de puntos que minimizan la función objetivo.

3.2.1 Fase 1 del Modelo. Generación de Pairings

En esta actividad la entrada son todas las legs del timetable (vehicle schedule). Las restricciones se definen como la función C.

C: $2^L \rightarrow \{0, 1\}$, 2^L conjunto potencia de L (legs)

Cuando la función toma el valor 0 no cumple todas las restricciones, en caso contrario es un pairing factible. Por lo tanto el conjunto de pairings factibles (F) se define como:

F: $\{f \rightarrow 2^L / \subseteq (f) = 1\}$,

La salida de esta actividad es la entrada a la siguiente (optimización de pairings).

Con una secuencia de legs entre dos puntos de descanso se forma un pairing. Los pairings compatibles se agrupan en piezas de trabajo. Las jornadas de trabajo por otro lado son periodos de tiempo durante el cual el trabajador trabaja y deben estar sujetos a restricciones. Durante la jornada de trabajo de un empleado este realiza legs o en forma mas general un pairing que no es más que un conjunto de piezas de trabajo factibles.

Dado el anterior razonamiento se puede formular la siguiente estrategia para encontrar el conjunto de pairings factibles y que expone en la siguiente ilustración.

FactibleDutiesGeneration (Vehicle Schedule){
1. Encuentre los puntos de descanso consecutivos del Vehicle Schedule.

2. Cree todas las legs a partir las secuencias de trips entre dos puntos de descanso.

3. Cree pairings con las secuencias de legs sin breaks.

4. Filtre los pairings factibles es decir aquellos que cumplen con la regulación.

5. Cree factible jornadas de trabajo con todas las combinaciones de legs del vehicle block. Estas combinaciones deben cumplir las restricciones y además ser factibles entre si.
}

Ilustración 3.1: Estrategia Pairings Factibles.

3.2.2 Fase 2 del Modelo. Optimización de Pairings (Utilización de Algoritmos Genéticos)

Cuando ya se ha generado una cantidad grande de pairings (salida fase anterior), comienza la fase de optimización de pairings que se modela como un problema Set Covering y se define como:

Dado un conjunto M con m elementos y n subconjuntos $Mj \subseteq M$ con costos asociados Cj, $j = 1,2\ldots.n$ encontrar un subconjunto S de {1,2…n} tales que:

$$\bigcup_{j \in S} M_j = M \qquad \textbf{(1)}$$

Y que…

$$\sum_{j \in S} C_j$$ Sean mínimos.

Por lo tanto, se busca una colección de subconjuntos tales que cada elemento de M esta contenido en al menos un subconjunto seleccionado y el costo total de todos los subconjuntos seleccionados es mínimo.

Al modelar el problema como Set Covering se permite que en las soluciones se acepte que los empleados puedan viajar como pasajeros en algunos pairings, posibilitando que se puedan obtener soluciones mejores. También existe la posibilidad de que no exista ninguna solución factible bajo la condición de que cada leg esté cubierto sólo una vez.

3.2.3 Modelado de los Imprevistos

El primer modelo que se propone para lograr abordar el problema de los imprevistos consiste en caso de ocurrir un imprevisto ajustar el horario de los buses posteriores en salidas, prorrateando el tiempo de atraso. La solución debe tomar en cuenta el nivel de calidad de servicio que la empresa persigue ofrecer, pero que en esta propuesta no es alto,

ya que los clientes que son transportados quedan botados sin solución y perdiendo el tiempo, asimismo los clientes que esperan por tomar el bus siguiente deben esperar más de lo habitual, aumentando el número de clientes insatisfechos.

El segundo modelo a proponer consiste en mantener un recurso adicional (bus y chofer fijo) en un lugar estratégico, de forma tal que cuando ocurra el imprevisto este pueda acudir y asistir a los clientes rápidamente, perjudicando lo menos posible el itinerario normal de las personas. Este modelo se ajusta con la forma en que se ha formulado el problema, ya que el recurso adicional se configura como una tarea más, es decir un leg que finalmente conforma un pairing, y que además posee un costo asociado y que corresponde al costo a pagar por la mejora en el nivel del servicio, que es lo que se busca en la mayoría de las veces.

Finalmente se pretende modelar los imprevistos como lo indica la propuesta 2, ya que esta mejora la calidad del servicio, que es lo que se busca e interfiere en la menor cantidad de personas en ser siniestradas por el incidente.

De la misma manera deja a los clientes del bus desastrado más satisfechos generando una preferencia futura que incide directamente en el éxito financiero de la empresa.

3.3 Algoritmo Propuesto

En base al estudio realizado anteriormente se propone un algoritmo genético que resuelve el problema de Set Covering (segunda fase “Optimización de Pairings” se modela como Set Covering) teniendo los datos de los archivos de la OR-library para la primera parte “Generación de Pairings” como entrada. **[OR90]**. Para efectos de un mejor entendimiento del algoritmo propuesto es que se definen inicialmente las variables y formatos para posteriormente exponer su funcionamiento:

El archivo de entrada:

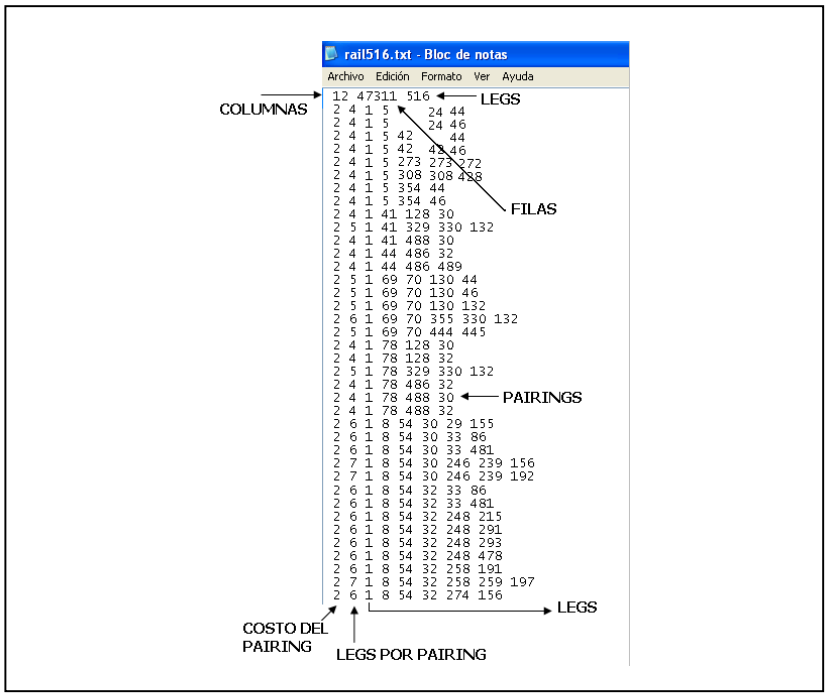


Ilustración 3.2: Archivo de entrada algoritmo

El formato del archivo de entrada esta compuesto por: en la primera fila se encuentran los datos de la cantidad de filas (legs) y columnas del archivo (pairings), las que corresponden a la cantidad de legs que posee el archivo y la cantidad de pairings que se forman con la combinación de los legs.

La primera columna del archivo representa el costo asociado al pairing, que generalmente es de valor 1 o 2. La segunda columna representa a la cantidad de legs que posee el pairing. Todas las filas conforman un pairing desde la tercera columna en adelante.

Representación: La representación elegida fue la no-binaria pues se ha visto que los estudios anteriores realizados por Beasley **[BC94]** indicaron que el rendimiento de un algoritmo genético utilizando una representación no-binaria no fue mucho más bajo que el mismo utilizando una binaria, además es necesaria para no perder la representación del pairing**.**

Función Fitness: Esta relacionada a un individuo, y a los costos de los pairings que contiene el individuo, se pretende que esta a medida que se itera vaya disminuyendo, ya que esta corresponde a la función objetivo del problema set covering.

$$\textbf{Minimice} \qquad \sum_{1}^{pairings} C_j * X_j$$

Población: La población en este algoritmo esta compuesta por seis individuos a los cuales se les será aplicada las funciones del algoritmo genético para alcanzar la solución factible óptima.

La población inicial será aleatoria, en donde los seis individuos se rellenaran con pairings elegidos del archivo de entrada en forma randómica.

Individuo: Un individuo es un conjunto de pairings que cubren algunas legs, esta es representada por una matriz, la cual tiene como número de filas un valor tal que sea mayor al largo mínimo (legs total / leg más grande) de la población, por ejemplo si la cantidad de legs es 507 y el pairing más largo esta compuesto por 12 legs, el número mínimo de la población es 507/12, por lo tanto el tamaño del individuo debe ser superior a este valor. El número de columnas corresponde a la cantidad de legs que posea el pairing más largo, como en el ejemplo anterior esto correspondería a 12, ya que es la opción más critica en la que el pairing cubra la mayoría de los legs, o sea que los legs sean todos del mismo tamaño que el máximo.

Un individuo factible o ***solución factible*** es aquel que cubre todas las legs al menos una vez y al menor costo.

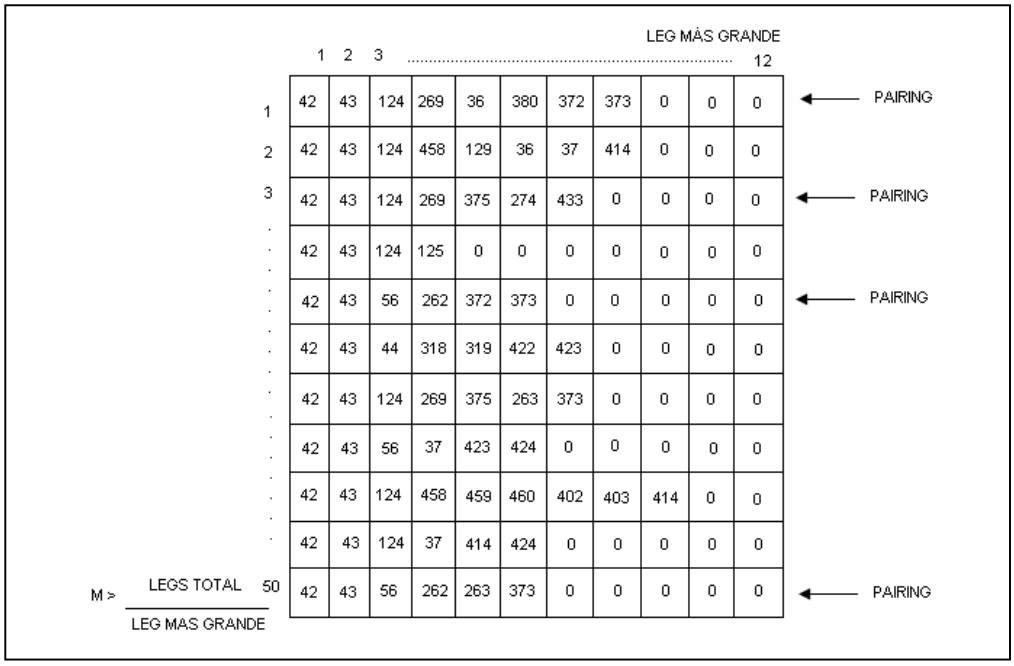


Ilustración 3.3: Individuo

Vector Cobertura: El vector cobertura es aquel que guarda los legs que son cubiertos por el individuo. Este vector tiene como largo la cantidad de legs, y en cada posición guarda la cantidad de pairings que contienen al leg. Entiéndase que no guarda el número del pairing que lo cubre sino la cantidad, con el fin de que posteriormente permita determinar cuál es el leg que más veces esta cubierto y determinar la estrategia para disminuirlo, logrando que la función fitness del algoritmo genético se minimice.

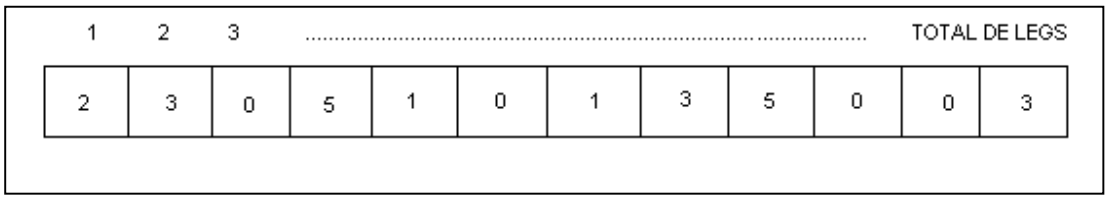


Ilustración 3.4: Vector Cobertura

La Ilustración 3.4 muestra el vector cobertura en donde el leg número 3 no esta cubierto por ningún pairing en el individuo (por lo tanto el individuo no es factible), y el leg número 1 esta cubierto por dos pairings. La labor de este vector es garantizar que el leg este cubierto al menos una vez.

Condición de Término: Como condición de término de un algoritmo existen dos opciones: una es que se espere a que este converja, la otra es dar un número definido de iteraciones y después de cumplido esto entregar la solución o individuo que se tiene en ese momento. Se ha decidido que en esta implementación dar un número determinado de iteraciones.

Operadores Genéticos: Como ya se mencionó anteriormente el algoritmo genético posee dos operadores para hacer las combinaciones, como son el cruzamiento y la mutación:

<u>Cruzamiento:</u> Toma dos individuos y forma dos hijos mediante la combinación de los componentes de los individuos padres, permitiendo de esta forma que el algoritmo efectúe una explotación del espacio de búsqueda. Se elige un número aleatorio entre el 0 y el número de filas de la matriz individuo, el cual determina desde qué pairing se hará el intercambio con el otro individuo. Además se elegirá un número aleatorio entre el 1 y el cero, si este nùmero esta entre 0 y 0,5 (incluido el 0,5) entonces se cambiará desde el pairing hacia arriba, si el número esta entre 0,5 y 1 el intercambio será hacia abajo.

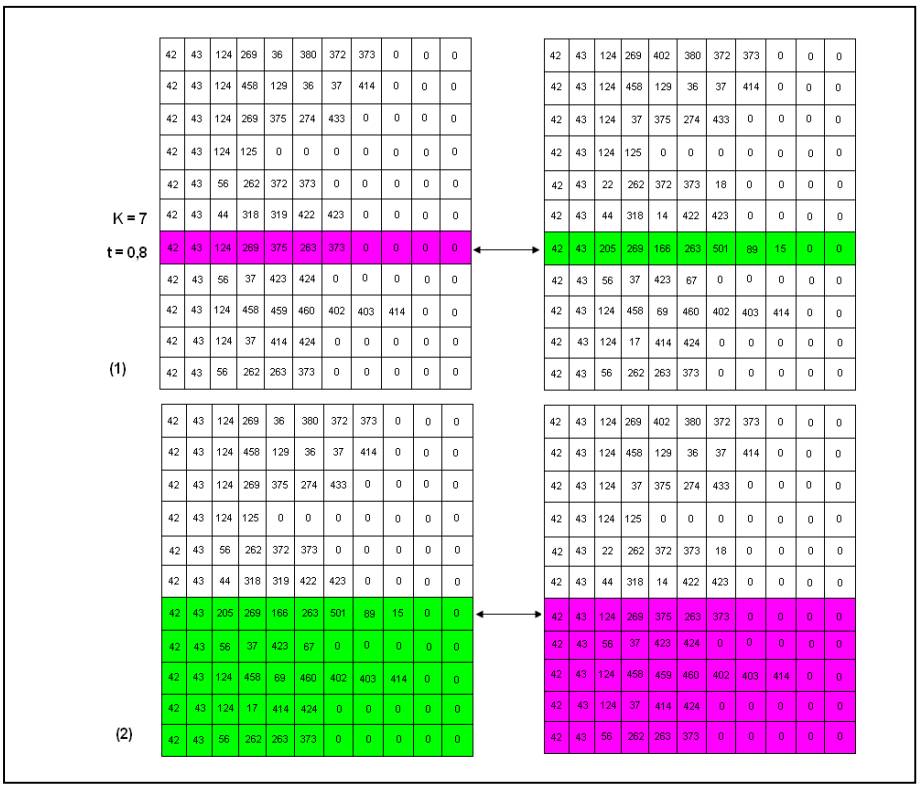


Ilustración 3.5: Cruzamiento

Mutación: Es el intercambio de una fila completa del individuo (pairing) por una fila del archivo de entrada.

Se elige un número al azar desde 0 al número de filas del individuo, posteriormente se elige un número al azar entre 0 y el número total de filas del archivo de entrada, y se intercambian.

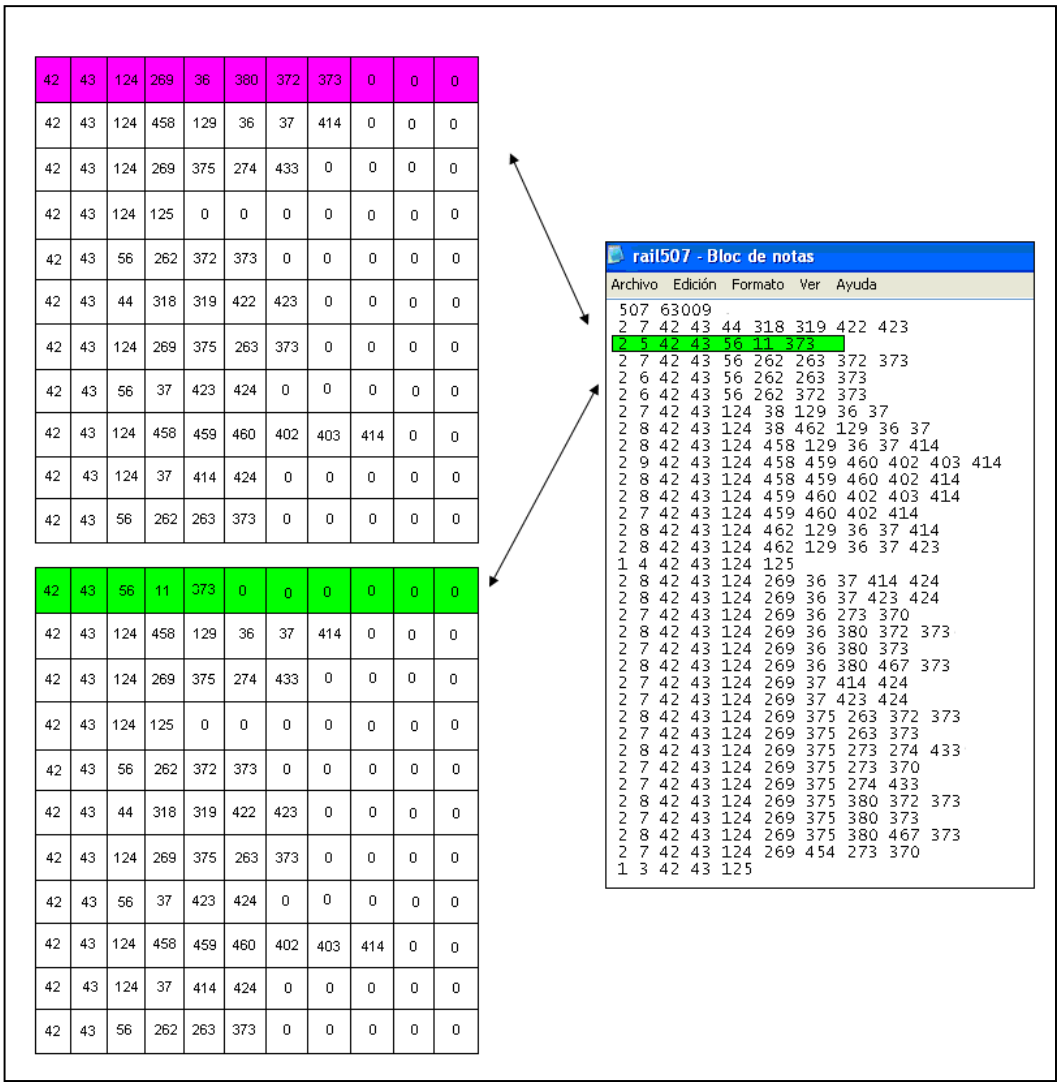


Ilustración 3.6: Mutación en un Punto

Se menciona que se han dispuesto de dos tipos de cruzamientos, uno que consta de cruzar un solo punto del individuo con el archivo y el segundo es aquel en que se mutan 3 puntos distintos del individuo con 3 pairings del archivo de entrada.

Tratamiento de los Infactibles: El software trabaja siempre con individuos factibles, es decir preocuparse de que todos los legs estén cubiertos al menos una vez como lo indica el problema, y se logra al determinar primero cuales son los legs que se encuentran sólo en un pair en el archivo de entrada, estos serán ingresados al individuo obligatoriamente, posteriormente se determinan cuales son los individuos que faltan por cubrir (que no están en los pairings obligatorios) y se busca entre los pairings que los contienen aleatoriamente un pair, permitiendo que se fuerce a encontrarlo, pero a la vez de forma sorteada. Esto se repite hasta que todos los legs sean cubiertos. Subsiguiente a esto se verifica si existen pairings que estén repetidos o que redunden datos incluidos en otros pairs y se eliminan, disminuyendo el fitness total del individuo.

Es necesario el tratamiento de los infactibles, pues la solución encontrada debe cumplir con todas las restricciones del problema, asimismo es imprescindible que se trabaje con individuos factibles con el fin de determinar cuales son los mejores candidatos para las iteraciones del algoritmo genético en la busqueda de aquel con mejor fitness. El 91 % de los individuos después de ser cruzados y/o mutados cambian su estado a infactible, y puesto que este porcentaje es alto se hace imperioso el tratamiento para lograr soluciones válidas.

3.3.2 Funcionamiento Del Algoritmo Propuesto

Con todo lo descrito anteriormente se propone el siguiente algoritmo preliminar:

1. Definir de cantidad de iteraciones que realice el algoritmo.

2. Luego se rellena el vector cobertura, y que posee la cantidad pairings que contienen un leg. Este vector se crea primeramente con el fin de determinar cuales son los legs que se encuentran sólo en un pair, y que deben encontrarse obligatoriamente en el individuo, posteriormente se utiliza para saber el rango pairings que posee un leg determinado y asi elegir un pairing del rango, consiguiendo que siempre se cubra el leg.

3. Seguidamente se determinan los obligatorios como se menciona anteriormente y se ingresan al individuo.

4. Ya logrado esto se itera durante 8 veces para ir generando los individuos y creando la población con 8 individuos. Aunque se destaca que anterior a esto a cada individuo se le verifica cuales son los legs que faltan, luego se busca aleatoriamente el pairing que lo posee y se ingresa al individuo hasta que este los cubre a todos, finalmente se le verifican los repetidos y se anulan.

5. Buscan los pairings que pueden estar contenidos en otros y se anulan, en un ciclo hasta que no se puedan anular más.

6. Calculo del fitness de cada individuo.

7. Ahora ya factibles los individuos comienzan las iteraciones: Se selecciona un par de individuos de la población.

8. Se determina la PC (probabilidad de cruzamiento) y que corresponde a un número entre el 0 y el 1. Si esta probabilidad (PC) es inferior o igual a 0.8 valor que generalmente se utiliza en este tipo de problema y que ha sido probado como óptimo en otros estudios **[BC94]**, entonces se aplica el cruzamiento.

    A. Determinar punto de cruce. (número entre donde terminan los obligatorios y el fin del individuo).
    B. Determinar dirección del cruce. (número entre 0 y 1). Si la dirección del cruce es inferior o igual a 0.5 el cruce será desde el número en donde terminan los obligatorios y el punto de cruce. Si es superior a 0.5 el cruce es desde el punto de cruce hasta el fin del individuo.

9. Se determina el PM (probabilidad de mutación) y que corresponde a un número entre el 0.0 y el 1. Si este número (PM) es inferior o igual a 0.02, entonces se aplica la mutación (que dependiendo del algoritmo puede ser en un punto o en 3).
    A. Determino el punto de mutación en cada individuo (entre 0 y las filas del individuo).
    B. Determino el pairing a mutar para cada uno de los individuos (entre 0 y el número de filas del archivo).

10. Ya finalizado el cruzamiento y la mutación. Se vuelve a verificar que el individuo sea factible, y se valida también que no existan repetidos o redundantes, y se calcula el fitness de los nuevos individuos generados.

11. Luego se hace un ranking con los fitness de la población y se determinan cuales son los más altos. Se compara con los nuevos individuos generados y se intercambian en caso de que su fitness sea menor.

12. Repetir esto la cantidad de veces definida (iteraciones).

13. Finalmente de la población se elige el individuo con mejor fitness (menor fitness) y que corresponde a la solución arrojada por el algoritmo.

Se menciona que la solución final corresponderá al conjunto de pairings que cubren al menos una vez las legs, y además este pairing esta asociado en la práctica a un equipo de trabajo (chofer fijo), además del costo asociado a esta solución.

En caso de necesitar información más particular del funcionamiento y composición del algoritmo referirse al Apéndice A- Código Fuente.

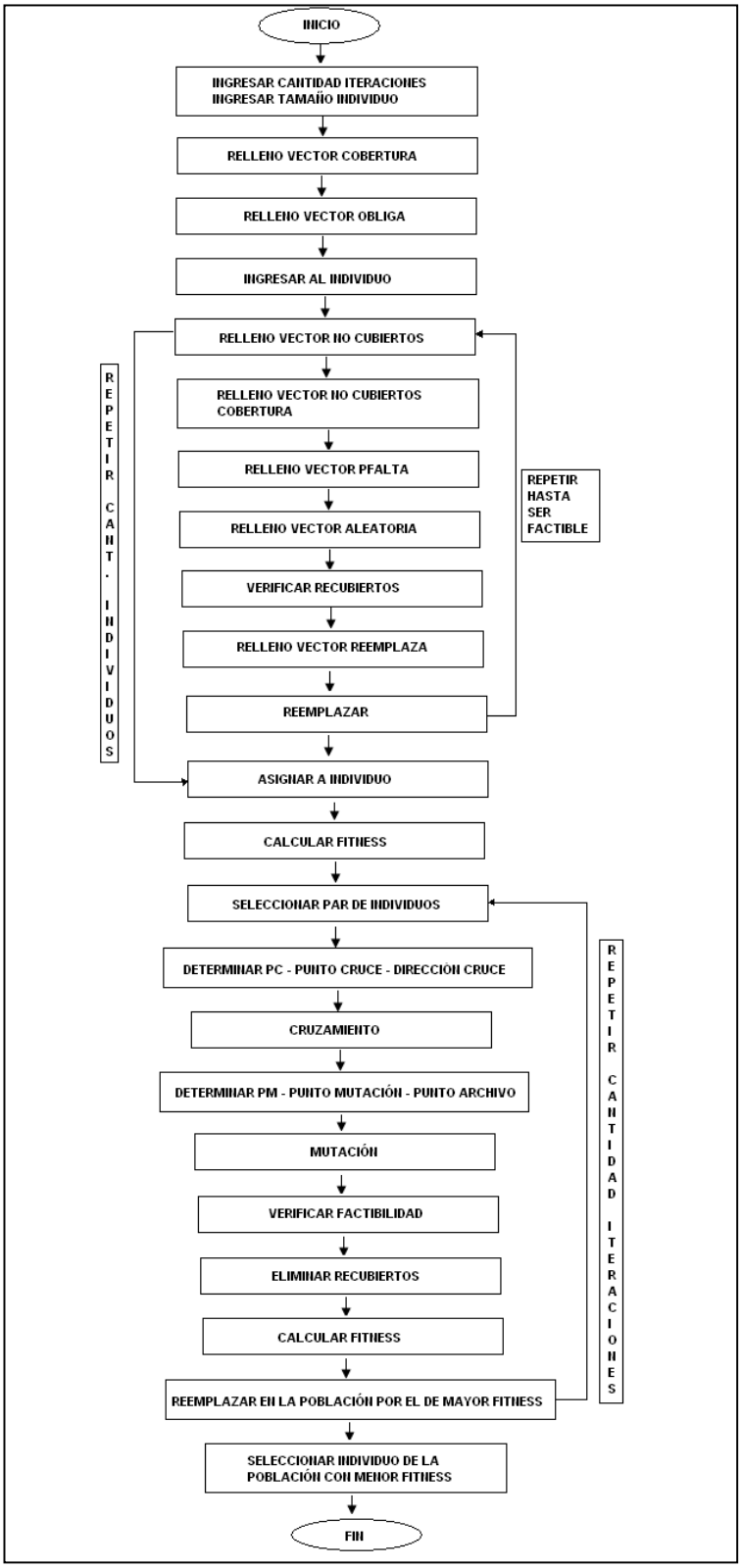


Ilustración 3.7: Diagrama Flujo Algoritmo Genético Propuesto

Capítulo 4

Implementación

4. Implementación

En el presente capitulo se presenta la aplicación construida y los resultados logrados con ella en tres escenarios: 2 casos teóricos, el primero Rail516 obtenido de **[OR90]**, y el segundo teórico C1 de la librería CSPLib **[CS08]**, y finalmente el caso especifico o problema real correspondiente a los datos de la empresa de buses Pullman bus entregados por la alumna tesista de la carrera Ingeniería en Transporte, Paola Morris, quién tiene como profesor guía a la Sra. Cecilia Montt.

4.1 Presentación de la Herramienta

En esta sección se exhibe la interfaz gráfica que posee la herramienta, con el fin de poder entender su lógica y funcionamiento.

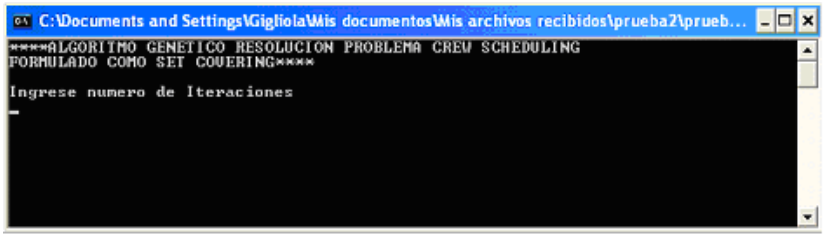


Ilustración 4.1: Interfaz Gráfica del Sistema

El parámetro pedido corresponde a la cantidad de veces que el algoritmo debe iterarse.

Ya ingresado el dato correspondiente al número de iteraciones comienza la ejecución, en donde a medida que se realiza esta el sistema expone el número de la

iteración pertinente, el individuo con mayor fitness (candidato a ser reemplazado) y su posición correspondiente.

Asimismo exhibe el número de los individuos que fueron elegidos para la iteración y su fitness correspondiente, además del vector fitness, el que posee los fitness de los individuos de la población en orden correspondiente.

Ya terminadas las iteraciones la herramienta expone solución o fitness final, y la cantidad de equipos asociados a la solución, como también el tiempo de ejecución.

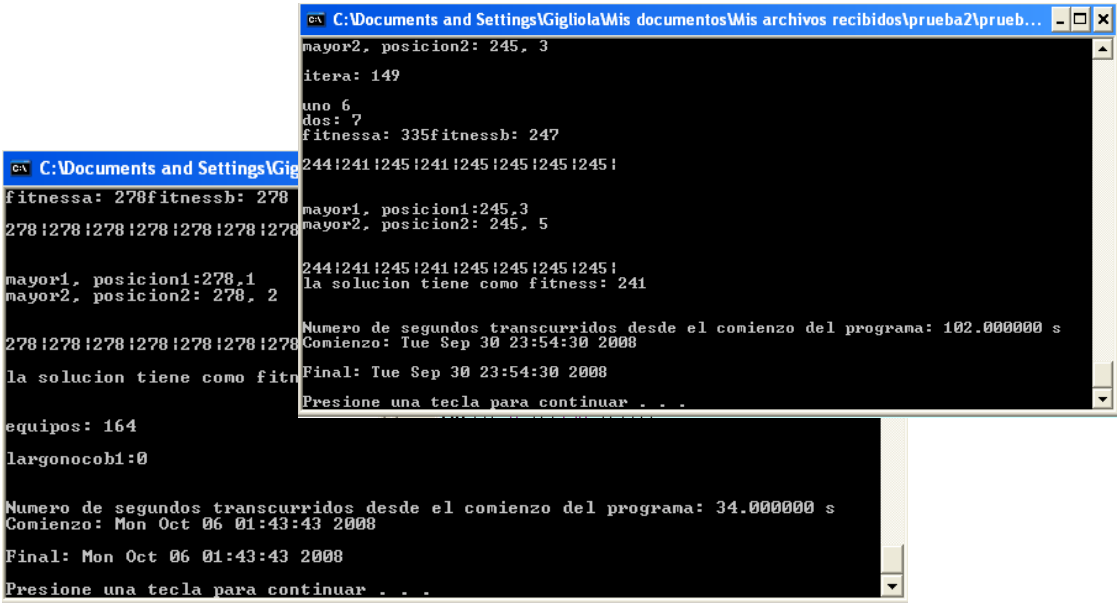


Ilustración 4.2: Interfaz Gráfica del Sistema. Ejecución

4.2 Presentación de Resultados

En el siguiente apartado se exponen los resultados obtenidos al aplicar la herramienta desarrollada en primer lugar en el problema Rail516, luego en el problema R1 de la CSPlib del tipo Bus driver scheduling y finalmente el problema real de Pullman Bus.

El archivo Rail516 corresponde a una instancia real proporcionada por el investigador italiano Paolo Nobili, y que fue obtenida de empresas de ferrocarriles italianas.

El benchmark utilizado es catalogado como “pequeño” **[OR90]** para la realidad italiana, pues es el con menos filas en la **[OR90]**, ya que este país posee un gran desarrollo con respecto a lo ferroviario. Sin embargo este problema (rail516) es lo suficientemente grande como para validar la herramienta desarrollada.

El archivo R1 fue obtenido de la biblioteca virtual CSPLib **[CS08]** que posee problemas de diferentes tipos enfocados a la investigación. Este archivo obtuvó sus datos de la empresa de buses Reading, los cuales permiten dar una instancia para la comparación.

El problema teórico 1 Rail 516 posee 516 legs, 12 columnas, y 47311 filas.
El problema teórico 2 R1 posee 53 legs, 11 columnas, y 2503 filas.

El computador en que fueron realizadas las pruebas corresponde a un Procesador Turion64 de 2.2 GHz, memoria RAM de 1 GB y 120 G de capacidad. Utiliza como sistema operativo “Windows XP Professional Edition”.

El compilador utilizado para el desarrollo de la herramienta es Dev-C++ y como editor el bloc de Notas.

4.2.1 Resultados Benchmark (Rail516)

Se destaca que se hicieron dos tipos de mutaciones (A. en 1 punto, B. en 3 puntos) por lo que los resultados corresponden a estos dos tipos.

En un inicio se muestran las distintas pruebas que se le realizaron al algoritmo con las dos versiones (mutación A y B) en un plan de pruebas de 500 - 1000 - 1500 y 2000 iteraciones con individuos de tamaño 250 - 300 y 350. A través de estas pruebas y a medida que se desarrollaba el algoritmo se observó que las mejores medidas de individuo y cantidad de iteraciones ideales eran de 300 y 1000 iteraciones entregando los mejores

resultados con 8 individuos, es por esto que las pruebas con estas magnitudes son analizadas a profundidad y comparadas ulteriormente con otros estudios.

La solución entregada por la herramienta es aquella del individuo con el menor fitness, como el fitness esta asociado directamente con la cantidad de equipos, es decir el individuo con menos equipos, y cada equipo esta asociado a un costo que es el que debe minimizar la función objetivo, menor fitness.

El tiempo mostrado en las soluciones viene dado en segundos. El intento es la cantidad de veces que se prueba a herramienta con una cierta configuración de parámetros, como por ejemplo con un individuo de tamaño 8 en la versión A de la herramienta, se harán 10 intentos de prueba.

Se destaca que para mayor información sobre las pruebas realizadas a las dos versiones de la herramienta dirigirse al apéndice B en donde se encuentran las tablas de pruebas.

	Version A			Versión B		
Individuos	**Equipos**	**Costo**	**Tiempo**	**Equipos**	**Costo**	**Tiempo**
6	143	213	35	135	196	88
10	136	199	55	133	194	67
12	136	199	68	133	194	107

Tabla 1: Resumen mejores soluciones en A y B con distintos individuos

Se puede advertir que cuando se utiliza el algoritmo en su versión A los resultados son inferiores a los arrojados por la versión B, que genera soluciones mejores, ya que al poseer 3 puntos de mutación existen más puntos de refrescamiento en la solución, al intercambiar más equipos desde el archivo de entrada. A medida que aumentan las iteraciones las soluciones mejoran, asimismo el tiempo de procesamiento aumenta, aunque muy escasamente.

Los tiempos de procesamiento comparando ambas versiones son similares y no alcanzan un volumen excesivo en comparación con otros estudios que serán mencionados posteriormente. Al incrementar el tamaño de los individuos las soluciones mejoran considerablemente (en un 72,5%), y en menos iteraciones arrojan resultados similares a los que se obtendrían al iterar más veces con individuos de menor tamaño.

Siempre que se amplian las dimensiones y las iteraciones, existen mejoras en las soluciones, hasta que se llega a un punto en que el algoritmo converge y no encuentra resultados superiores. Es por esto que a continuación se señalan las pruebas realizadas con las dimensiones convenientes.

Inicialmente se presenta el plan de pruebas para los dos tipos de herramientas (A y B).

Iteraciones	Tamaño Individuo	Intento
200	300	10
400	300	10
500	300	10
1000	300	10

Tabla 2: Plan de Pruebas Rail516

La determinación del tamaño del individuo a probar como se ha indicado anteriormente fundamentalmente fue obtenida a través de la experiencia obtenida al desarrollar la herramienta, ya que al elegir un individuo de un tamaño grande produce que el software se ejecute en forma rápida y entregando soluciones malas por ejemplo de valores superiores a 300 si se itera pocas veces, es por esto que es conveniente que a medida que aumenta el tamaño del individuo aumente también el número de iteraciones.

Cada vez que se incrementa la cantidad de individuos en la población los resultados mejoran proporcionadamente hasta llegar a la medida de 8 individuos con un individuo de tamaño 300 y con 1000 iteraciones en donde los resultados se mantienen, es decir convergen en este punto.

Con esto se quiere dar a entender que el mejor resultado en la versión A fue de 199 y que al acrecentar los individuos por ejemplo en 10 y con 1500 iteraciones el producto de estas pruebas son los mismos, sólo se produce un aumento en el tiempo de procesamiento.

Como se puede notar en la tabla 3 a medida que aumentan las iteraciones, los resultados presentan una mejoría, obteniéndose en la categoría con más iteraciones (1000) la mejor solución. De la misma forma se percibe que el tiempo en ejecutar las soluciones no necesariamente aumenta en gran medida con las iteraciones, no existe una relación directa, ya que gran parte de la herramienta trabaja con valores aleatorios.

Iteraciones	Tiempo	Equipos	Costo	Intento
200	10	176	303	1
200	16	178	307	2
200	17	175	300	3
200	10	177	297	4
200	15	175	303	5
200	26	174	296	6
200	10	180	312	7
200	30	174	302	8
200	18	177	303	9
200	14	177	304	10
400	37	172	290	1
400	14	175	304	2
400	23	173	297	3
400	42	170	288	4
400	26	174	296	5
400	24	169	291	6
400	17	173	299	7
400	17	171	290	8
400	18	173	298	9
400	27	172	292	10
500	41	173	297	1
500	37	169	287	2
500	20	172	287	3
500	76	158	266	4
500	16	140	240	5
500	22	172	288	6
500	55	170	277	7
500	34	171	290	8
500	76	169	286	9
500	84	169	285	10
1000	106	150	241	1
1000	101	156	257	2
1000	67	136	199	3
1000	43	135	256	4
1000	89	160	273	5
1000	99	164	276	6
1000	37	169	286	7
1000	46	145	229	8
1000	76	148	206	9
1000	59	136	244	10

Tabla 3: Pruebas con Mutación en 1 sólo punto (A) 8 individuos.

La tabla 4 exhibe una mejora en comparación con las pruebas en la herramienta con mutación en un solo punto, logrando que la mejor solución obtenida en las pruebas anteriores fuera superada, debido a que posee más puntos de intercambio de equipos con el archivo de entrada.

Nº Iteraciones	Tiempo	Equipos	Costo	Intento
200	10	174	296	1
200	13	177	305	2
200	17	177	305	3
200	12	174	301	4
200	15	166	287	5
200	11	178	307	6
200	11	168	294	7
200	12	175	299	8
200	16	168	294	9
200	16	177	299	10
400	25	172	290	1
400	34	169	286	2
400	21	169	290	3
400	19	172	291	4
400	27	171	293	5
400	20	172	292	6
400	44	170	291	7
400	33	174	295	8
400	38	169	292	9
400	16	175	292	10
500	27	169	289	1
500	33	171	286	2
500	23	173	292	3
500	27	168	285	4
500	29	170	290	5
500	19	169	286	6
500	27	173	290	7
500	20	169	286	8
500	32	168	284	9
500	60	168	284	10
1000	57	162	270	1
1000	108	151	241	2
1000	57	149	205	3
1000	39	162	275	4
1000	32	166	275	5
1000	**54**	**133**	**194**	**6**
1000	33	160	273	7
1000	82	154	205	8
1000	73	130	199	9
1000	106	133	272	10

Tabla 4: Pruebas a Herramienta con Mutación en 3 puntos (B) 8 individuos.

	Versión A			Versión B		
Iteración	Tiempo	Equipos	Costo	Tiempo	Equipos	Costo
200	26	174	296	15	166	287
400	42	170	288	34	169	286
500	16	140	240	32	168	284
1000	67	136	199	54	133	194

Tabla 5: Resumen Mejores Soluciones por Cantidad de Iteraciones Rail516.

El cuadro resumen muestra las mejores soluciones. Se concluye que a medida que aumentan las iteraciones mejores son las soluciones, además que la herramienta con 3 puntos de mutación es mejor que la con sólo un punto, ya que permite que haya un refrescamiento en el individuo al ingresar pairings del archivo de entrada en forma aleatoria.

Se destaca que existen resultados en donde el costo es distinto y la cantidad de equipos es igual, esto debido a que los costos asociados a los pairings (un equipo cubre un pairing) varia entre 1 y 2, lo que provoca esta diferencia.

También se enfatiza que en comparación con la mejor solución encontrada a este problema, como se muestra en la tabla 7.11 los resultados obtenidos si bien no son los mejores, se acercan bastante **[JO07],** ya que la mejor solución corresponde a 182 y la encontrada en el presente estudio es 194, lo que reporta un -6,5934% de diferencia, indicando que el resultado no esta a mucha distancia de la mejor solución. Se menciona asimismo, que la solución encontrada es bastante óptima, ya que el tiempo que toma en encontrarse es de 54 segundos a diferencia de la mejor encontrada que es de 217 segundos en el mejor de los estudios **[JO07]**.

Instance	Rows	columns	Zlp	CNS	CFT	YKI	CA	CPLEX
E.1-E.5	500	5000	21.38		28.4	28.4		28.6
E.1-E.5	500	5000	8.92		14.0	14.0		14.2
G.1-G.5	1000	10000	149.48	167.4	166.4	166.4		168.6
H.1-H.5	1000	10000	45.67	60.4	59.6	59.6		61.8
RAIL507	507	63009	172.15	174	175	175	174	175
RAIL516	516	47311	182.00	182	182	182	182	*182
RAIL582	582	55515	209.71	211	211	211	211	*211
RAIL2536	2436	1081841	688.40	692	691	691	691	*689
RAIL2586	2586	920683	935.92	951	948	945	948	979
RAIL4284	4284	1092610	1054.05	1070	1065	1064	1063	1089
RAIL4872	4872	968672	1509.64	1534	1534	1528	1532	1570

Tabla 6: Estado del arte mejores soluciones caso Rail516

CNS: Relajación Lagrangiana basado en un algoritmo heuristico de Ceria et al. **[CN98]**

CFT: Relajación Lagrangiana basado en un algoritmo heuristico de Caprara et al. **[FC95]**

YKI: Búsqueda local por vecindario 3-flip de Yagiura et al. **[YK06]**

CA: Búsqueda tabú con algoritmo heuristico primal-dual de Caserta. **[CA07]**

Instance	Rows	columns	GR	BPD	PU-UI	PD-LP	RND
4.1-4.10	200	1000	528.3	602.2	534.0	521.0	518.6
5.1-5.10	200	2000	270.4	292.5	271.2	277.6	265.7
6.1-6.5	200	1000	156.4	176.4	155.0	161.2	156.6
A.1-A.5	300	3000	253.2	302.8	254.4	255.6	261.0
B.1-B.5	300	3000	78.2	89.4	82.2	85.0	84.6
C.1-C.5	400	4000	234.8	278.4	238.0	244.4	249.8
D.1-D.5	400	4000	70.4	71.8	69.6	71.4	72.8
E.1-E.5	500	5000	31.0	38.4	32.0	31.8	34.8
F.1-F.5	500	5000	16.2	18.8	17.2	17.0	17.4
G.1-G.5	1000	10000	178.8	212.0	177.4	187.2	192.6
H.1-H.5	1000	10000	66.6	82.8	66.8	69.0	73.0
RAIL507	507	63009	210	355	222	201	198
RAIL516	516	47311	202	368	230	185	185
RAIL582	582	55515	247	432	285	247	225
RAIL2536	2436	1081841	882	1412	987	750	748
RAIL2586	2586	920683	1157	1755	1290	1057	1057
RAIL4284	4284	1092610	1358	2095	1453	1185	1186
RAIL4872	4872	968672	1868	2853	2015	1685	1685

Tabla 7: Mejores soluciones caso Rail516 distintas Heurísticas

GR: Heurística Greedy.

BPD: Algoritmo básico primal-dual.

PD-UI: Algoritmo primal-dual con regla de incremento uniforme.

PD-LP: Algoritmo primal-dual con utilizando una solución óptima dual para el problema de relajación LP.
RND: Algoritmo Rounding.

Como se exhibe en el cuadro de la tabla 7.12 se advierte que la solución encontrada con la heurística Algoritmos Genéticos no es la mejor, pero sí se aloja entre las superiores, ya que ni en sus soluciones con menor iteración se obtuvo un resultado tan alto como el de la heurística BPD **[JO07]**.

Se destaca que existen muchos más estudios sobre heurísticas que no han sido mencionadas en el presente documento, pero cabe destacar que el objetivo de las tablas mostradas anteriormente es dar a entender que los resultados obtenidos por el algoritmo propuesto en la solución del problema teórico Rail516 se acerca a las mejores encontradas.

4.2.2 Resultados Benchmark (R1 de la CSPLib)

Existen diversos problemas de planificación de Turnos de Chóferes para la locomoción colectiva, por lo que para dar un análisis al algoritmo más amplio y puesto ya que ha sido testeado el software en un problema de trenes, es que se presentan pruebas a un problema de otro tipo como son los buses y que permite dar más veracidad y alinamiento al estudio.

Este problema (R1) fue adquirido a través de **[CS08]** y corresponde a los datos de una pequeña empresa europea de buses Reading. Como ya se mencionó anteriormente este archivo posee 11 columnas, 2503 filas y 53 legs.

Los datos han sido probados en las dos versiones, la primera con mutación en 1 punto (A) y la segunda versión con mutación en 3 puntos (B). Con variaciones en las iteraciones de 250-500-1000 y 4000, y con combinaciones de individuos de 33, 53 y 103 en

tamaño. Asimismo el algoritmo se modificó de manera tal que existen 3 magnitudes de individuos 6, 8, y 10 para demostrar como incide este aspecto en las soluciones.

Los tiempos obtenidos están en segundos, y en el caso en que este tiempo es 0 es porque el tiempo es inferior a 1 segundo.

Los mejores resultados son obtenidos cuando se ocupa la mutación en 3 puntos, de la misma forma se nota que al aumentar las iteraciones se mejoran las soluciones, hasta llegar a un cierto punto en donde no mejoran. Asimsimo si se incrementa el tamaño de los individuos también mejoran las soluciones, aunque se genera un estancamiento después de las 1000 iteraciones.

Si las iteraciones aumentan en demasía al igual que el tamaño de los individuos las soluciones no mejoran, sólo aumenta el tiempo de procesamiento, lo mismo sucede con los individuos, se cree que siempre es mejor que existan más individuos para que la piscina de soluciones sea mayor, y generalmente lo es, pero en este caso como la masa de datos es pequeña los distintos individuos convergen en valores de costo total similares.

Con la versión B se logró obtener el mejor resultado y que corresponde a 11 equipos con un costo asociado de 11, puesto que en este problema los costos de todos los equipos son iguales y es 1. En comparación con lo expuesto en **[CS08]** como mejor solución esta versión obtuvó los mismos resultados que el mejor que se ha encontrado y que corresponde como se mencionó a 11. En los datos obtenidos en la librería **[CS08]** no se indicaba el tiempo de procesamiento, pero se presume que el obtenido por la herramienta es conveniente ya que sólo tomó en el mejor de sus tiempos 5 segundos en entregar la respuesta con 1000 iteraciones y un individuo de tamaño 103.

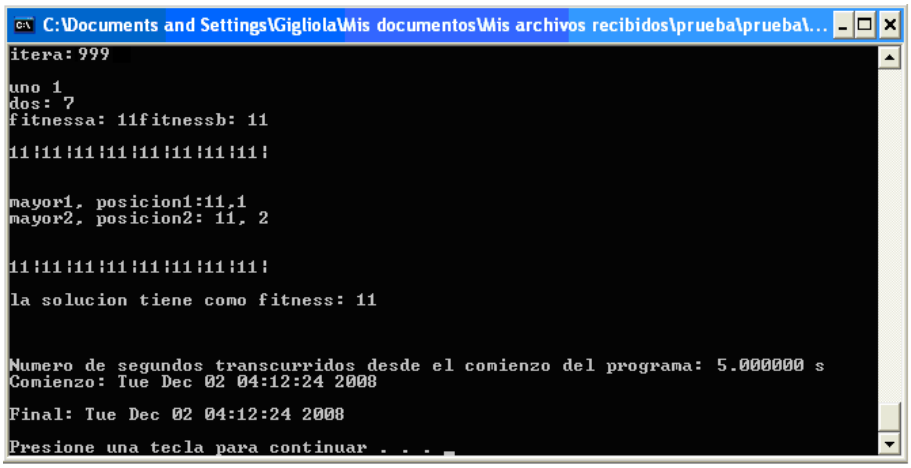


Ilustración 4.3: Mejor solución versión B con un individuo de tamaño 103

Esto indica que las dimensiones de esta solución son en las que el algoritmo converge, puesto que a medida que aumentan las iteraciones no aparecen soluciones superiores sólo aumenta el tiempo de procesamiento.

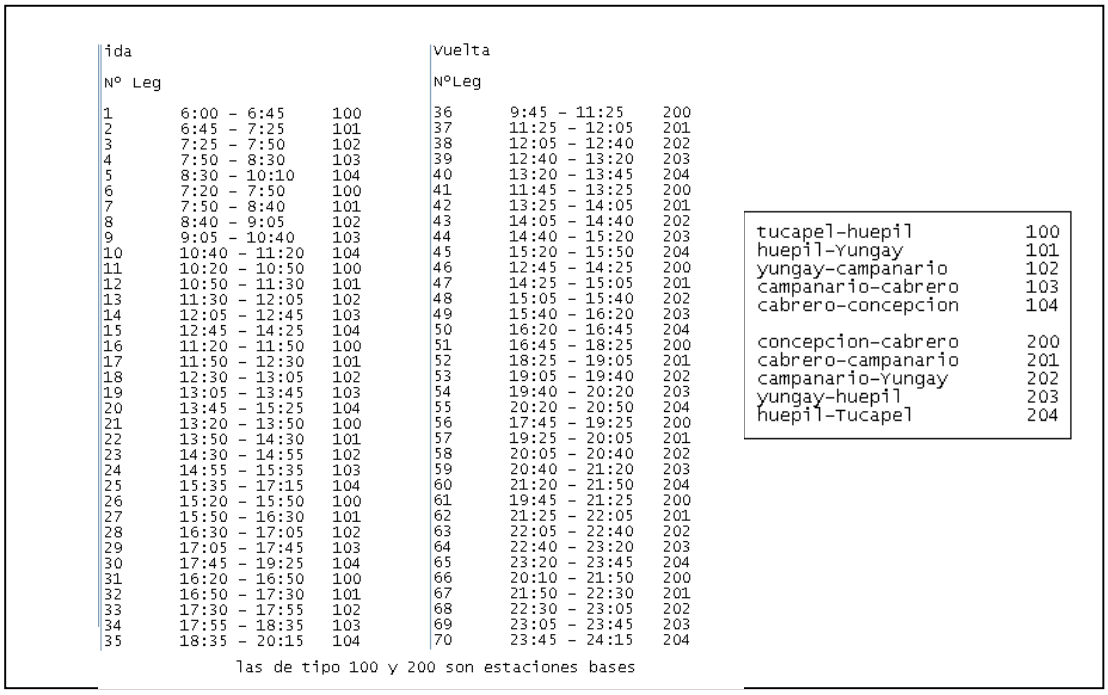
Se distingue que al ser este un problema de pocos datos al procesador se le simplifica el procesamiento de los datos, es por esto que existen soluciones que tienen como tiempo de respuesta 0 en donde no se alcanza ni siquiera un segundo de procesamiento. Es aún más, las soluciones no fluctuan en exceso en sus valores de respuesta a diferencia del problema de los trenes, en donde la peor solución es 307 y la mejor 194 donde se observa una distancia significativa. En este caso la peor solución es 17 y la mejor 11, insistiendo a causa de que la cantidad de datos es menor. Igualmente este bajo volumen de los datos incide escasamente en las dos versiones, en donde los resultados no son muy distintos generandose una diferencia minina de distancia 1 entre las 2 mejores soluciones.

4.2.3 Resultados Problema Real (Pullman Bus)

En esta sección se expondrán los resultados obtenidos al aplicar la herramienta desarrollada en el problema real correspondiente a la empresa Pullman Bus.

Como introducción se puede decir que la historia de Pullman se remonta hacia la mitad del siglo pasado cuando en la zona central de Chile dio sus primeros pasos en el transporte de pasajeros cubriendo la ruta entre Cartagena y San Antonio. En los años setenta comienza un crecimiento sostenido, que en la actualidad posiciona a Pullman como una de las mayores empresas de transporte terrestre en Chile, con más de doscientos destinos y la mejor flota de buses del país.

Los resultados expuestos son resultado de la prueba de los datos entregados por la empresa de manera simple y con los datos que incluyen los imprevistos. Asimismo, los datos entregados por esta empresa se muestran en la siguiente Ilustración y concierne a los horarios de salida de los buses en la ruta Tucapel- Concepción , ubicada en la Provincia de Bíobío, en la VIII Región del Bíobío, estos horarios concuerdan con las salidas desde los dos puntos ida y vuelta y que corresponden a las estaciones bases. La ida corresponde a la dirección Tucapel-Concepción y la vuelta Concepción-Tucapel. Con estos datos es que se han conformado los legs, que posteriormente constituyen a los pairings que están asociados a un equipo, y que han de ser planificados. Este problema posee 70 legs, 16 columnas y 1258 filas.



ida			Vuelta		
N° Leg			N°Leg		
1	6:00 - 6:45	100	36	9:45 - 11:25	200
2	6:45 - 7:25	101	37	11:25 - 12:05	201
3	7:25 - 7:50	102	38	12:05 - 12:40	202
4	7:50 - 8:30	103	39	12:40 - 13:20	203
5	8:30 - 10:10	104	40	13:20 - 13:45	204
6	7:20 - 7:50	100	41	11:45 - 13:25	200
7	7:50 - 8:40	101	42	13:25 - 14:05	201
8	8:40 - 9:05	102	43	14:05 - 14:40	202
9	9:05 - 10:40	103	44	14:40 - 15:20	203
10	10:40 - 11:20	104	45	15:20 - 15:50	204
11	10:20 - 10:50	100	46	12:45 - 14:25	200
12	10:50 - 11:30	101	47	14:25 - 15:05	201
13	11:30 - 12:05	102	48	15:05 - 15:40	202
14	12:05 - 12:45	103	49	15:40 - 16:20	203
15	12:45 - 14:25	104	50	16:20 - 16:45	204
16	11:20 - 11:50	100	51	16:45 - 18:25	200
17	11:50 - 12:30	101	52	18:25 - 19:05	201
18	12:30 - 13:05	102	53	19:05 - 19:40	202
19	13:05 - 13:45	103	54	19:40 - 20:20	203
20	13:45 - 15:25	104	55	20:20 - 20:50	204
21	13:20 - 13:50	100	56	17:45 - 19:25	200
22	13:50 - 14:30	101	57	19:25 - 20:05	201
23	14:30 - 14:55	102	58	20:05 - 20:40	202
24	14:55 - 15:35	103	59	20:40 - 21:20	203
25	15:35 - 17:15	104	60	21:20 - 21:50	204
26	15:20 - 15:50	100	61	19:45 - 21:25	200
27	15:50 - 16:30	101	62	21:25 - 22:05	201
28	16:30 - 17:05	102	63	22:05 - 22:40	202
29	17:05 - 17:45	103	64	22:40 - 23:20	203
30	17:45 - 19:25	104	65	23:20 - 23:45	204
31	16:20 - 16:50	100	66	20:10 - 21:50	200
32	16:50 - 17:30	101	67	21:50 - 22:30	201
33	17:30 - 17:55	102	68	22:30 - 23:05	202
34	17:55 - 18:35	103	69	23:05 - 23:45	203
35	18:35 - 20:15	104	70	23:45 - 24:15	204

las de tipo 100 y 200 son estaciones bases

tucapel-huepil	100
huepil-Yungay	101
yungay-campanario	102
campanario-cabrero	103
cabrero-concepcion	104
concepcion-cabrero	200
cabrero-campanario	201
campanario-Yungay	202
yungay-huepil	203
huepil-Tucapel	204

Ilustración 4.4: Legs formadas por los horarios. Total 70. Nombre de las rutas que forman los legs

4.2.3.1 Problema Pullman Bus sin incluir imprevistos

El plan de pruebas se indica en la siguiente tabla, y que a diferencia del presentado en el problema teórico posee un individuo de menor tamaño, ya que la cantidad de filas del archivo de entrada es menor, al igual que la cantidad de legs.

Iteraciones	Tamaño Individuo	Intento
200	50	10
400	50	10
500	50	10

Tabla 8: Plan de Pruebas Problema Real Pullman Bus

Los datos escogidos en el plan de pruebas se fundamentan principalmente en el análisis efectuado sobre los resultados de las instancias teóricas, asimismo como del conocimiento adquirido de manera empírica durante el desarrollo de la herramienta. El tamaño del individuo siempre ha de ser menor que la cantidad de legs, y las columnas del tamaño del pairing más largo. Se destaca también que el costo de cada pairing es 10.

Iteraciones	Tiempo	Equipos	Costo	Intento
200	33	19	190	1
200	22	24	240	2
200	29	24	240	3
200	19	21	210	4
200	11	22	220	5
200	17	21	210	6
200	11	22	220	7
200	56	21	210	8
200	16	19	190	9
200	77	29	290	10
400	73	23	230	1
400	28	21	210	2
400	29	20	200	3
400	47	20	200	4
400	61	20	200	5
400	92	21	210	6
400	18	27	270	7
400	72	19	190	8
400	64	22	220	9
400	34	18	180	10
500	36	17	170	1
500	34	15	150	2
500	39	16	160	3
500	72	18	180	4
500	55	19	190	5
500	54	21	210	6
500	37	16	160	7
500	18	15	150	8
500	39	20	200	9
500	10	21	210	10

Tabla 9: Resultados Pullman Herramienta con Herramienta A.

La siguiente tabla muestra los resultados obtenidos al probar los datos del problema real en la segunda herramienta, la que en el problema teórico Rail516 da mejores resultados que la herramienta A, pero que en esta ocasión no sucedió, ya que el mejor valor obtenido de equipos es de 15 y costo 150, y es el mismo de la primera herramienta (A).

Iteraciones	Tiempo	Equipos	Costo	Intento
200	53	22	220	1
200	45	26	260	2
200	48	25	250	3
200	91	20	200	4
200	18	21	210	5
200	73	22	220	6
200	34	22	220	7
200	28	23	230	8
200	90	27	270	9
200	61	26	260	10
400	72	21	210	1
400	73	21	210	2
400	26	24	240	3
400	27	20	200	4
400	83	19	190	5
400	72	20	200	6
400	74	19	190	7
400	56	21	210	8
400	18	18	180	9
400	49	23	230	10
500	108	22	220	1
500	54	21	210	2
500	74	20	200	3
500	64	20	200	4
500	38	20	200	5
500	23	17	170	6
500	82	19	190	7
500	67	15	150	8
500	63	18	180	9
500	92	19	190	10

Tabla 10: Resultados Pullman Herramienta con Herramienta B.

Se desprende la misma conclusión que en el problema rail516 y que en R1, y es que a medida que aumentan las iteraciones los resultados son mejores, ya que como se va trabajando con una población que ha medida que van mejorando los individuos desecha los malos.

Los datos entregados por la organización para comparación indican que la empresa utiliza sólo 11 equipos para lograr la asignación total de los chóferes en los trayectos, lo que indicaría que los resultados arrojados por la herramienta en sus dos versiones están

muy alejados de la realidad. Este distanciamiento tiene un fundamento muy simple, ya que la empresa no cumple estrictamente con la restricción de la ley Orgánica del trabajo de un máximo de 8 horas diarias, sino más bien paga horas extras a los mismos equipos, los que generalmente trabajan cerca de 9 a 10 horas diarias, es decir la empresa relaja la restricción del problema que indica que se deben trabajar máximo 8 horas diarias. Si la empresa siguiera las restricciones de la ley orgánica al pie de la letra le significaría utilizar una cantidad de equipos similar obtenida por el algoritmo (14 o 15 equipos). Se destaca que no hay que confundir con la ley 20.271, la que apunta a que no se deben conducir más de 5 horas continuas, ley que la empresa si cumple.

Para comprobar cual sería el resultado que arroja el algoritmo al relajar las restricciones iniciales a este estudio, se puede hacer una comparación real, lo que permitiría alinear las pruebas y determinar la eficacia del algoritmo.

Al volver a formular el problema con las restricciones de trabajo modificadas en donde las horas de la jornada de trabajo ahora son de 9,8 horas diarias, el archivo de entrada al algoritmo es modificado y ya no posee las mismas dimensiones, sino más bien ahora goza de 70 legs, 16 columnas y 1701 filas. Estas dimensiones han sufrido un aumento, ya que existen más pairings que cumplen con la nueva restricción.

El plan de pruebas que se indica en la siguiente tabla, es el mismo presentado anteriormente. La cantidad de individuos es 8.

Iteraciones	Tamaño Individuo	Intento
200	50	10
400	50	10
500	50	10

Tabla 11: Plan de Pruebas 2 Problema Real Pullman Bus

Los datos escogidos en el plan de pruebas se fundamentan principalmente en el análisis efectuado sobre los resultados de las instancias teóricas, asimismo como del conocimiento adquirido de manera empírica durante el desarrollo de la herramienta, y debe ser igual al anterior para poder comparar. El costo de cada pairing es 10.

Iteraciones	Tiempo	Equipos	Costo	Intento
200	45	24	240	1
200	48	24	240	2
200	12	22	220	3
200	13	18	180	4
200	57	20	200	5
200	33	19	290	6
200	38	21	210	7
200	25	22	220	8
200	27	18	180	9
200	38	19	190	10
400	58	18	180	1
400	36	23	230	2
400	29	21	210	3
400	37	20	200	4
400	61	19	290	5
400	46	16	160	6
400	48	20	200	7
400	30	21	210	8
400	39	17	170	9
400	42	22	220	10
500	58	21	210	1
500	47	19	190	2
500	39	22	220	3
500	57	15	150	4
500	48	21	210	5
500	37	20	200	6
500	36	14	140	7
500	54	20	200	8
500	55	19	190	9
500	48	21	210	10

Tabla 12: Resultados Pullman Restricciones relajadas con Herramienta A.

La siguiente tabla muestra los resultados obtenidos al probar los datos del problema real en la segunda herramienta, la que en las primeras pruebas con las restricciones originales genera los mismos resultados que la herramienta A, pero que en esta ocasión suscita una mejora, la cual se alinea con los datos entregados por la empresa.

Iteraciones	Tiempo	Equipos	Costo	Intento
200	59	23	230	1
200	43	19	290	2
200	39	18	280	3
200	38	21	210	4
200	60	23	230	5
200	55	19	190	6
200	56	17	170	7
200	47	18	180	8
200	45	21	210	9
200	58	22	220	10
400	51	20	200	1
400	46	21	210	2
400	53	17	170	3
400	57	23	230	4
400	62	22	220	5
400	47	19	190	6
400	60	15	150	7
400	65	18	180	8
400	49	21	210	9
400	59	14	140	10
500	73	12	120	1
500	68	19	190	2
500	55	18	180	3
500	63	14	140	4
500	39	16	160	5
500	43	17	170	6
500	71	19	190	7
500	46	15	150	8
500	62	13	130	9
500	70	12	120	10

Tabla 13: Resultados Pullman Restricciones Relajadas con Herramienta B.

Estas pruebas en general indican la tendencia de que los tiempos arrojados siempre por el algoritmo con mutación en 3 puntos son siempre más altos, esto debido a que se ha de procesar 2 puntos de intercambio más con el archivo de entrada lo que comúnmente provoca que sean más los cambios que el algoritmo debe hacer para volver a la factibilidad. De esta misma forma y como ha sido manisfestado en los análisis de las pruebas anteriores las soluciones entregadas por la herramienta en su versión B son mejores, puesto que poseen más cambios con el archivo de entrada lo que posibilita que no exista un estancamiento en los individuos, refrescando su contenido y disminuyendo su costo.

En particular al relajar la restricción existe un mejoramiento en la mejor solución entregada en las primeras pruebas, pues se avanza de una respuesta de 15 equipos a una de

12 equipos. En comparación con los datos obtenidos de la empresa esta solución no es la mejor, ya que ellos trabajan con sólo 11 equipos. La explicación que se le da a este hecho consiste en que al relajar la restricción de horario a 9,8 horas se hizó un aproximado de lo explicado por la empresa, ya que ellos al pagar horas extras tienen la posibilidad de manipular y extender este tiempo. De igual forma en este estudio se considero que el tiempo de descanso de los chóferes fluctuaba entre 30 y 120 minutos, pero ha indicado la empresa que este tiempo en general no excede los 60 minutos.

Por otra parte la solución entregada por la herramienta A con las restricciones iniciales arrojó como mejor solución 15 equipos y en la última prueba con la nueva restricción esta disminuyó a 14 equipos, no existiendo una diferencia muy grande como en el caso de rail516, ya que el volumen de los datos (nueva cantidad de pairings) no aumentó demasiado generando esta pequeña mejora.

Con la herramienta B se produjó una mejora mayor, de 3 equipos, esto debido a que al aumentar la cantidad de pairings en el archivo de entrada y que al haber una mutación en 3 puntos provocará que esta piscina de nuevas posibilidades aumentará el beneficio en la solución, ya que este proceso (mutación) tiene relación directa con el archivo de entrada a diferencia del cruzamiento en donde el traspaso de información es entre 2 individuos (Toma dos individuos y forma dos hijos mediante la combinación de los componentes de los individuos padres, permitiendo de esta forma que el algoritmo efectúe una explotación del espacio de búsqueda).

Se desprende de todas las pruebas es que a medida que aumentan las dimensiones ya sean de los individuos, como de la cantidad de individuos o de iteraciones los resultados mejoras, hasta llegar a un punto en el cual las soluciones no mejoran sólo aumentan su tiempo de procesamiento. También la herramienta con mutación en 3 puntos es mejor que con un solo punto ya que existen más intercambios con el archivo de entrada permitiendo que no se generen estancamientos en las soluciones e iterando sin beneficio.

Los resultados de las distintas pruebas demuestran que el comportamiento del algoritmo ante los distintos problemas es bueno, ya que al compararse con otras heurísticas los resultados que arrojan estan cercanos a las mejores soluciones.

4.2.3.1 Problema Pullman Bus incluyendo imprevistos

Como se ha referido anteriormente se ha decidido optar por la opción 2 de mantener un recurso adicional, puesto que esta alternativa ofrece una mejora en la calidad del servicio ofrecida por este tipo de empresas (de transporte colectivo).

Este recurso adicional se relaciona con el algoritmo en que se agrega como una tarea más, incidiendo en la generación de los pairings, lo que aumenta la cantidad de combinaciones y altera la cantidad de pairings que cumplen con las restricciones. No confundir con que el modelo propuesto varía, este queda inmóvil, sólo se renuevan los archivos de entrada al algoritmo y se hace una nueva generación de los pairings con más tareas.

Esta nueva versión que incluye los imprevistos se alinea con el archivo que posee las restricciones relajadas, aunque ni el archivo con las restricciones originales ni las relajadas pueden ser objeto de comparación, ya que la empresa en la actualidad no toma en consideración el factor imprevisto, es decir no posee ningún mecanismo o plan de contingencia.

El plan de prueba corresponde al mismo utilizado anteriormente en este caso, y se destaca que el nuevo archivo de entrada tiene una configuración reformada de 76 legs, 17 columnas y 1872 filas. Del mismo modo se mantienen la cantidad de individuos (8).

Iteraciones	Tamaño Individuo	Intento
200	50	10
400	50	10
500	50	10

Tabla 14: Plan de Pruebas considerando Imprevistos Problema Real Pullman Bus

La siguiente tabla muestra las pruebas realizadas a la primera herramienta A incluyendo en los datos las tareas de los imprevistos.

Iteraciones	Tiempo	Equipos	Costo	Intento
200	22	22	220	1
200	38	27	270	2
200	29	24	240	3
200	42	21	210	4
200	53	28	280	5
200	39	27	270	6
200	47	20	200	7
200	27	24	240	8
200	29	22	220	9
200	31	23	230	10
400	33	21	210	1
400	29	20	200	2
400	57	19	190	3
400	33	23	230	4
400	59	22	220	5
400	28	24	240	6
400	28	20	200	7
400	31	21	210	8
400	37	20	200	9
400	60	24	240	10
500	66	19	190	1
500	48	22	220	2
500	47	21	210	3
500	39	18	180	4
500	66	22	220	5
500	36	23	230	6
500	55	21	210	7
500	51	18	180	8
500	70	23	230	9
500	72	20	200	10

Tabla 15: Resultados Pullman considerando Imprevitos con Herramienta A.

La tabla muestra los resultados conseguidos al probar los datos del problema real en la segunda herramienta B.

Iteraciones	Tiempo	Equipos	Costo	Intento
200	31	26	260	1
200	40	22	220	2
200	28	25	250	3
200	44	23	230	4
200	37	24	240	5
200	33	25	250	6
200	45	25	250	7
200	39	22	220	8
200	34	24	240	9
200	48	24	240	10
400	44	22	220	1
400	56	19	290	2
400	46	21	210	3
400	59	23	230	4
400	43	18	180	5
400	50	23	230	6
400	49	16	160	7
400	51	20	200	8
400	39	19	190	9
400	46	13	130	10
500	63	14	140	1
500	68	16	160	2
500	59	19	190	3
500	53	13	130	4
500	58	14	140	5
500	69	15	150	6
500	55	18	180	7
500	58	13	130	8
500	62	16	160	9
500	62	17	170	10

Tabla 16: Resultados Pullman considerando Imprevitos con Herramienta B.

El considerar los imprevistos se ve reflejado en el aumento en la cantidad de tareas, y a su vez este incremento en las tareas incurren en un acrecentamiento en la cantidad de equipos a necesitar para cubrir al menos una vez todas las tareas. Este crecimiento en los valores entregados por el algoritmo como resultado corresponde a 2 equipos más, lo que significa una acentuación de un 16, 6% en la solución inicial, es decir en los costos.

El considerar en el problema los imprevistos mejora la calidad del servicio que es una de las palancas competitivas de los negocios en la actualidad. Prácticamente en todos los sectores de la economía se considera el servicio al cliente como un valor adicional, y corresponde a la esencia en los casos de empresas de servicios, como son las de locomoción colectiva. Las empresas se deben caracterizar por el altísimo nivel en la calidad de los servicios que entregan a los clientes que los contratan.

El personal en todos los niveles y áreas, y en especial los dueños de las empresas deben estar conscientes de que el éxito de las relaciones entre la empresa y cada uno de los clientes depende de las actitudes y conductas que observen en la atención de las demandas de las personas que son o representan al cliente, es por esto que en este proyecto se tomó en consideración la variable imprevistos, porque se estima que debe ser esencial para que una solución sea completa y genere un beneficio a la empresa.

La empresa en particular debe estimar si esta incorporación de 2 equipos produce un beneficio mayor al costo que se ha de ocasionar, pues esta inversión puede, y es lo más probable, generar beneficios en las preferencias de los clientes.

Capítulo 5

Conclusiones

5. Conclusiones

5.1 Conclusión General

A modo de conclusión de este trabajo de título, se puede poner hincapié en la gran importancia del problema de planificación de equipos (Crew Scheduling) para la gran mayoría de las empresas de transporte colectivo, cuyo funcionamiento depende en gran medida de la justa planeación y asignación que se haga de los equipos de trabajo que obran en ella.

Se hace imperioso entonces, encontrar las soluciones más eficientes que resuelvan esta planificación, aunque cabe destacar que no son necesariamente las óptimas.

Los problemas del tipo Crew Scheduling generan un espacio de soluciones factibles muy amplio, esto a causa de su naturaleza combinatorial. Es por esto que se hace necesario un modelo tal que permita encontrar una solución factible, modelo en que se distinguen todas las etapas de la generación de la solución, y que fue implementado en la herramienta desarrollada por este trabajo. Esta formulación del modelo y la siguiente implementación permitió entender de manera práctica el funcionamiento de la técnica (AG), el contrarrestar la herramienta con otras desarrolladas con otros algoritmos ampliamente reconocidos, validando de esta manera y a través del benchmark a la herramienta implementada. Finalmente se ratificó con la aplicación del problema real de Pullman Bus, el cual trae consigo un plus como es el considerar el manejo de los imprevistos, mejorando asi el beneficio generado para la empresa de transporte que lo utilice.

Se logró así, con esta información poner en práctica el proyecto de planificación de turnos, formulando el problema como Set Covering y utilizando la Metaheurística de Algoritmos Genéticos en su desarrollo, los cuales sin duda son una estrategia que logra implantar buenos resultados en múltiples problemas de optimización combinatorial, aunque se destaca que los resultados han dependido de la manera en que han sido implementadas sus características fundamentales, lo que ha permitido que se corrobore la eficacia de la herramienta desarrollada y su formulación. Esta comprobación fue a través de las pruebas que demuestran que el comportamiento del algoritmo ante los distintos problemas es bueno, ya que al compararse con otras heurísticas los resultados que arrojan estan cercanos a las mejores soluciones.

En cuanto al contenido académico y personal, se puede mencionar que el tema permitió visualizar un aspecto nuevo de la aplicación que tiene la informática en los distintos sectores de la sociedad. Así, el estudio realizado a lo largo de este trabajo de título posibilitó el interiorizarse en temas tan complejos como las metaheurísticas y los problemas de optimización, que vienen a complementar lo aprendido en todos los años de carrera.

Se espera que a través de este documento se haya traspasado el esfuerzo realizado durante estos años de carrera, tanto en el estudio del problema como en el desarrollo de la aplicación y las gestiones para obtener la información necesaria para validarla.

5.2 Trabajo Futuro

Como es de sospechar este tipo de trabajos relativos a la optimización estan sujetos a la mejora, en la medida que no se ha encontrado el óptimo global.

Es conveniente que se haga una implementación del problema formulado de la misma forma, pero que considere una alternativa nueva que resuelva el asunto de los imprevistos, utilizando las mismas pruebas de manera que sea posible el análisis y la comparación. De la misma forma es beneficioso el buscar nuevas técnicas para dar solución

a este problema que logren mejores resultados. De la misma forma, se puede mejorar el aspecto visual de la herramienta, el hacerla más amigable al usuario. También desde el punto de vista de los resultados obtenidos, un elemento importante es el código que puede ser optimizado.

Referencias

[AM04] Adam Marczyk, “Algoritmos genéticos y computación evolutiva”, (2004).

[BC94] J. E. Beasley y P. C Chu, “a genetic algorithm for a set covering problem”, (1994).

[CA07] M. Caserta: “Tabu search-based metaheuristic algorithm for large-scale set covering problems”. In K.F. Doerner, M. Gendreau, P. Greistorfer, W.J. Gutjahr, R.F. Hartl, and M. Reimann (eds.): Metaheuristics: Progress in Complex Systems Optimization (2007).

[CC00] Carlos Coello, “An updated survey of GA-based multiobjective optimization techniques”, ACM Computing Surveys, (2000).

[CN98] S. Ceria, P. Nobili, y A. Sassano: “A Lagrangian-based heuristic for large-scale set covering problems. Mathematical Programming”, (1998).

[DS01] Teresa G. Dias, Jorge P. Sousa, Joao F Cunha, “A Genetic Algorithm for the Bus Driver Scheduling Problem”, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal (2001).

[FC95] M Fischetti A Caprara y P Toth. “A heuristic method for the set covering problem”. Proc of the fifth IPCO Conference, lectura y notas de ciencias de computadores (1995).

[FG06] Francisco Gómez, “Crew scheduling para la cuenca de alimentadores del Sistema Transmilenio”, Movilife LTDA. (2006)

[FL97] R. Freling, “Models and techniques for integrating vehicle and crew scheduling”, Ph.D. Thesis, Erasmus University, Rotterdam, Holanda (1997).

[FP02] Peter Fleming y R.C. Purshouse. “Evolutionary algorithms in control systems engineering: a survey”. Control Engineering Practice, (2002).

[GC05] Guillermo Cabrera, “Resolución del Problema de Planificación de Equipos de Trenes Urbanos con Búsqueda tabú Hibridizada con Programación Entera”, Memoria Título Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile (2005).

[GJ79] M.R Garey y D.S Johnson, “Computers and Interectability” una guía para la teoría de NP- completo (exhaustivo), editor Victor Klee (1979).

[HMG] Holland, John. “Genetic algorithms” Scientific American, julio de 1992, Mitchell, Melanie. “An Introduction to Genetic Algorithms” MIT Press, 1996, Goldberg, David. “Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison-Wesley, (1989).

[JK99] John Koza, Forest Bennett, David Andre y Martin Keane. Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, (1999).

[JO07] Journal of the Operations Research Society of Japan. “Relaxation Heuristics For The Set Covering Problem”, Japon (2007).

[LY92] C. Lund y M. Yannakakis. “On the hardness of approximating minimization problems” 33rd IEEE Symposium on Foundations of Computer Science. (1992).

[OR90] J. E. Beasley, OR-Library está originalmente descrita en "OR-Library: distributing test problems by electronic mail", Journal of the Operational Research Society 41(11) pp1069-1072, (1990).

[PI05] Pablo Itaim Ananias, “Resolución del Problema de Set-Covering utilizando un Algoritmo Genético”, (2005).

[RD96] Richard Dawkins, The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design. W.W. Norton, (1996).

[RK72] R. M Karp “Reducebility among combinatorial problems” en “in complexity of computers computations”, Pleno de prensa, Nueva York, USA (1972).

[RP04] Helena Ramalhinho Louren, José Pinto Paixão, Rita Portugal, “Metaheuristics for The Bus-Driver Scheduling Problem”, Department of Economics and Management, Universitat Pompeu Fabra, Barcelona, España, (2004).

[CS08] CSPLib, librería con problemas de investigaciones tipo Crew Scheduling para comparaciones, http://www.csplib.org/, (2008).

[YK06] M. Yagiura, M. Kishida, and T. Ibaraki: “A 3-flip neighborhood local search for the set covering problem”. European Journal of Operational Research, (2006).

APÉNDICE A – CÓDIGO FUENTE

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


int main() {

  time_t comienzo, final;
   struct tm *tiempoComienzoPtr, *tiempoFinalPtr;
   comienzo = time( NULL );

//////////////////////////////definicion vector fit////////////////////////////////////
     int * fit;
   fit = (int*) malloc (sizeof(int) * 100);

////////////////////////////////////////////////////////////////////////////////////

srand((unsigned) time(NULL));

FILE *archivo;
FILE *idea3;
int columnas, filas, legs, ar, a, h, i, e, j ,r ,k, m, s, t, u, v, d, g=0, l, p ,may,lar, n, x, z,largo, large, filasind, c, b,
f, fil=0, fitness1=0, total=0, linea, costolinea, cantind, caracter, count, cont, conta=0, contar, counter=0,
contador=0, candidato, reemp, pair, pairing, y=50, max=20, largobliga,itera,kil, contado, cuenta, co, azar,
cone, par, la, flag=0, hi, xi, unos, largoreemplaza,  temp, largonocob, fitnessfinal=0, largorank, coni, pa,pape,
ric, rea, tes,remp, siete, ocho, pairin,erre, heo, mayor, er,ti, ki,cuentar, tr,usa, tuto, gigio ,ele, ere, de, ka, tu,
ye,gig, gigo, contadora0, conteo, ber, mer, teo, totale, al, ola, lon, contad, usados, resto, aux, pare,
rica,cost,jota, menor, num, au,ke,lo, li, ko, reem,vez, contan, kia, fin,dia,  has, sep, cien, preem,te,hey, fes, tot,
contigo, ceros, coint, listo, largoaleaind, largoaleapair, restoaleapair,
fitness2=0,fitness3=0,fitness4=0,fitness5=0,fitness6=0,fitness7=0,fitness8=0, fitnessa=0, fitnessb=0, uno, dos,
ptocruce, ptoarchivoa, ptoarchivob, ptomuta, ptomutb, mayor1, posicion1, mayor2, posicion2, equipos,
menor1, posic, ptomuta2, ptomuta3, ptomutb2, ptomutb3, ptoarchivoa2, ptoarchivoa3, ptoarchivob2,
ptoarchivob3;
float pc, pm, direc, it;

char *variable;
   variable = (char *) malloc (2000 * sizeof(char));
   int  *dato;
   dato = (int *) malloc (999999* sizeof(int));
   char *t1;
   idea3 = fopen("idea3.txt", "w+");
   archivo = fopen("rail516.txt","r");

if (archivo == NULL) {
            printf ("\nEl contenido del archivo de prueba esta vacio \n\n");
   }else{
     fscanf(archivo,"%d",&columnas);
     fscanf(archivo,"%d",&filas);
     fscanf(archivo,"%d",&legs);

fprintf(idea3, "****ALGORITMO GENETICO RESOLUCION PROBLEMA CREW SCHEDULING
FORMULADO COMO SET COVERING****\n\n");
fprintf(idea3, "filas %d \n\ncolumnas %d \n\nlegs %d \n\n", filas, columnas, legs);
```

```
//definicion vector cobertura del vector datos
    int * cob;
    cob = (int*) malloc (sizeof(int) * (legs+1));
    for(i=1;i<(legs+1);i++){
          cob[i]=0;  }

//definicion vector cobertura 1
    int * cob1;
    cob1 = (int*) malloc (sizeof(int) * (legs+1));
    for(i=1;i<=legs;i++){
    cob1[i]=0;  }

//definicion vector  no cubiertos 1
    int * nocob1;
    nocob1 = (int*) malloc (sizeof(int) * (legs-150));
    for(i=1;i<=(legs-150);i++){
      nocob1[i]=0;}

//definicion vector costo del pairing
    int * costo;
    costo = (int*) malloc (sizeof(int) * (filas+1));
    for(i=0;i<filas;i++){
       costo[i]=0;}

//definicion vector obligatorios
    int * obliga;
    obliga = (int*) malloc (sizeof(int) * (legs/4));
        for(i=0;i<(legs/4);i++){
       obliga[i]=0;}

//definicion vector oblialea con pairs obligatorios
int * oblialea;
oblialea = (int*) malloc (sizeof(int) * (legs/4));
for(j=0;j<(legs/4);j++){
oblialea[j]=0;  }

//definicion vector vecescubiertos
int * vecescob;
vecescob = (int*) malloc (sizeof(int) * (max+1));
for(i=1;i<=max;i++){
vecescob[i]=0;  }

//definicion vector reemplaza
    int * reemplaza;
  reemplaza = (int*) malloc (sizeof(int) * (legs/4));
    for(i=0;i<(legs/4);i++){
          reemplaza[i]=0;  }
//definicion vector simulacion
    int * simulacion;
    simulacion = (int*) malloc (sizeof(int) * (legs+1));
    for(i=1;i<=legs;i++){
   simulacion[i]=0;  }

    int * aleainda;
    int * aleaindb;
    int * aleaind2;
```

```
    int * aleaind3;
    int * aleaind4;
    int * aleaind5;
    int * aleaind6;
    int * aleaind7;
    int * aleaind8;

// dimensiones individuo
filasind = (legs/columnas)+200;
fprintf(idea3,"\nfilas individuo %d \ncolumnas individuo %d\n", filasind, columnas);

aleainda = (int*) malloc (sizeof(int) * filasind);
aleaindb = (int*) malloc (sizeof(int) * filasind);
aleaind2 = (int*) malloc (sizeof(int) * filasind);
aleaind3 = (int*) malloc (sizeof(int) * filasind);
aleaind4 = (int*) malloc (sizeof(int) * filasind);
aleaind5 = (int*) malloc (sizeof(int) * filasind);
aleaind6 = (int*) malloc (sizeof(int) * filasind);
aleaind7 = (int*) malloc (sizeof(int) * filasind);
aleaind8 = (int*) malloc (sizeof(int) * filasind);

    i=0;
          while(feof(archivo) == 0){
                fgets(variable,2000,archivo);
       //separar en tokens por cada espacio q encuentre
                for(t1=strtok(variable," "); t1!=NULL; t1=strtok(NULL, " ")){
                     if(t1!=NULL){
                          d=atoi(t1);
                          dato[i]=d;
           i++;      }   }    }

//vector de individuos
   int * largo, **M;
   largo = (int*) malloc (sizeof(int) * filas);
   int r=1;
       for(i=0;i<filas;i++){
       costo[i]=dato[r];
       largo[i]=dato[r+1];
       r=r+largo[i]+3;       }

//llenado de la matriz que contiene los datos del archivo
      r=3;
    do{
      }while(filas <= 0);

//definicion matriz M con los datos del archivo
      M = (int**) malloc(filas*sizeof(int));
      for(i=0;i<filas;i++){
         M[i] = (int*)malloc(columnas*sizeof(int));          }


//fprintf(salida, "matriz de datos\n\n");
//matriz de datos del archivo
      for(i=0;i<filas;i++){
         for(j=0;j<largo[i];j++){
            M[i][j]=dato[r+j];
```

```
          // fprintf(idea3, "%d|",M[i][j]);
           }
         r=r+3+largo[i];
       //  fprintf(idea3, "\n");
              }

// llenado cobertura de datos
//fprintf(idea3, "cobertura datos\n\n");
 for(i=0;i<filas;i++){
  for(j=0;j< largo[i];j++){
        e=M[i][j];
        cob[e]=cob[e]+ 1;              }}

//llenado obliga
d=0;
contar=1;
while(contar<max){
for(i=0;i<=legs;i++){
    if(cob[i]==contar){
    obliga[d]=i;
    d++;        } }
    contar++;    }

//llenado oblialea
g=0;
while(g!=d){
for(i=0;i<filas;i++){
for(j=0;j<largo[i];j++){
if(M[i][j]==obliga[g]){
oblialea[g]=i;
g++;}}}}

//dejar al individuo sin pairings repetidos
for(i=0;i<d; i++){
for(j=(i+1);j<d;j++){
if(oblialea[i] == oblialea[j]){
       oblialea[j]=-1;      }           }     }
int aba=0;
//fprintf(idea3, "\n\noblialea\n\n");
for(i=0;i<d;i++){
if(oblialea[i]!=-1){
aba++;               }
//fprintf(idea3,"%d|", oblialea[i]);
}
//fprintf(idea3,"\n\naba (cantidad de pairs sin repetirse y que son obligatorios): %d\n\n", aba);


//llenado veces cubierto
m=1;
while(m<max){
contado=0;
for(i=1;i<=legs;i++){
 if(cob[i]==m){
 contado++;     }}
 vecescob[m]=contado;
 m++;       }
```

```
//definicion aleatorio ind
    int * aleaind1;
    aleaind1 = (int*) malloc (sizeof(int) * (filasind +10));
     for(j=0;j<filasind;j++){
          aleaind1[j]=0; }

//relleno del individuo con los obligatorios segun el maximo dado
int fer=0;
while(fer<aba){
for(i=0;i<d;i++){
tu=oblialea[i];
if(tu!=-1){
aleaind1[fer]=tu;
   }fer++;         }}
//fprintf(idea3, "\n\nlargo obligatorios (fer): %d\n\n", fer);

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//HASTA ACA EL INDIVIDUO ESTA RELLENO CON LOS PAIRS QUE CONTIENEN A LOS LEGS QUE -//ESTAN SOLO UNA VEZ Y QUE SON OBLIGATORIOS
////////////////////////////////////////////////////////////////////////////////////////////////////////////
cantind=9;
for(vez=2;vez<cantind;vez++){

//vaciado del individuo1
for(i=fer;i<filasind;i++){
aleaind1[i]=-1;
   }

// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind1[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
           e=M[k][j];
           cob1[e]=cob1[e]+ 1;             }} }

//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;


//definicion vector nocubiertoscobertura
int * nocobcob1;
nocobcob1 = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
nocobcob1[i]=0;  }
```

```
//llenado no cubiertos cobertura
for(i=0;i<largonocob;i++){
gig=nocob1[i];
gigo=cob[gig];
nocobcob1[i]=gigo;     }

contadora0=0;
// lectura nocobcobertura1
//fprintf(idea3,"\n\nnocobcob1\n\n");
for(i=0;i<largonocob;i++){
contadora0=contadora0 + nocobcob1[i];
//fprintf(idea3,"%d|", nocobcob1[i]);
}

//fprintf(idea3, "\n\ncontadora0: %d\n\n", contadora0);

//definicion vector aleatoria
int * aleatoria;
aleatoria = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
aleatoria[i]=-1;  }

//llenado aleatoria
for(i=0;i<largonocob;i++){
al=nocobcob1[i];
m = ( rand () % al )+1;
aleatoria[i]=m;      }
//definicion vector pfalta
int * pfalta;
pfalta = (int*) malloc (sizeof(int) * (contadora0+1));
for(i=1;i<=contadora0;i++){
pfalta[i]=0;  }

//llenado del vector pairings que faltan
ye=1;
while(ye<=contadora0){
for(h=0;h<largonocob;h++){
k=nocob1[h];
for(i=0;i<filas;i++){
for(j=0;j<largo[i];j++){
if(M[i][j]==k){
pfalta[ye]=i;
ye++;     } }  }  }  }

//definicion vector aleapair que son los pairs que corresponden en el aleatoria y que son los pairs que deben estar obligatoriamente para que esten todos cubiertos, pero que han sido elegidos aleatriamente
int * aleapair;
aleapair = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
aleapair[i]=-1;  }

//relleno del vector aleapair que posee los pairs y se llena con los datos del archivo pfalta del individuo
ka=aleatoria[0];
aleapair[0]= pfalta[ka];
count=nocobcob1[0];
```

```
for(h=1; h<largonocob; h++){
    for(i=0;i<(h-1);i++){
    count=count + nocobcob1[i];}
    total= count + aleatoria[h];
    count=0;
    aleapair[h]=pfalta[total];                    }

//anulacion de los repetidos
for(i=0;i<largonocob; i++){
for(j=(i+1);j<largonocob;j++){
if(aleapair[i] == aleapair[j]){
aleapair[j]=-1;              }     }       }

//dejo los ceros al final en el aleapair
ceros=0;
for(u=0;u<largonocob;u++){
if(aleapair[u]==-1){
ceros++;}     }

for(j=0;j<largonocob;j++){
for(i=0;i<largonocob;i++){
if(aleapair[i]==-1){
       for(h=(i);h<(largonocob-1);h++){
       aleapair[h]=aleapair[h+1];}
     }}}
for(u=(largonocob-ceros);u<largonocob;u++){
aleapair[u]=-1;     }

largoaleapair=0;
//fprintf(idea3,"\n\naleapair\n\n");
for(i=0;i<largonocob;i++){
//fprintf(idea3,"%d|",aleapair[i]);
if(aleapair[i]!=-1){
largoaleapair++;}}
//fprintf(idea3, "\n\nlargo aleapair %d\n\n", largoaleapair ); //sin pairs repetidos

//llenado del individuo hasta el final con lo que pueda del aleapair
//fprintf(idea3, "\nlargonocob: %d\n", largonocob);

 for(i=fer;i<(fer + largonocob);i++){
 aleaind1[i]=aleapair[i-fer];
 }


//fprintf(idea3, "\nusados, utilizados para llenar el individuo despues del fer: %d\n" ,usados);
//restoaleapair=largoaleapair-usados;
//fprintf(idea3, "\n\nrestoaleapair: %d ( %d despues de llenar hasta el tope el individuo de largo %d)\n\n",
restoaleapair, largoaleapair, filasind);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//HASTA ACA EL INDIVIDUO LLENO, PERO NO CUBRE TODOS LOS LEGS, COMIENZA LA BUSQUEDA ///DE LOS POR REEMPLAZAR POR LOS PAIRS CON LOS LEGS QUE FALTAN ////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

while(largonocob!=0){
// llenado cobertura 1
```

```
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind1[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;              }} }

//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;
//printf("\n\nlargonocob1: %d\n\n",largonocob);

for(i=0;i<largonocob;i++){
nocobcob1[i]=0;
}

//llenado no cubiertos cobertura
for(i=0;i<largonocob;i++){
gig=nocob1[i];
gigo=cob[gig];
nocobcob1[i]=gigo;     }

contadora0=0;

// lectura nocobcobertura1
//fprintf(idea3,"\n\nnocobcob1\n\n");
for(i=0;i<largonocob;i++){
contadora0=contadora0 + nocobcob1[i];
//fprintf(idea3,"%d|", nocobcob1[i]);
}

for(i=0;i<largonocob;i++){
aleatoria[i]=0;
}

//llenado aleatoria
for(i=0;i<largonocob;i++){
al=nocobcob1[i];
m = ( rand () % al )+1;
aleatoria[i]=m;      }

for(i=0;i<contadora0;i++){
pfalta[i]=0;      }

 //llenado del vector pairings que faltan
ye=1;
while(ye<=contadora0){
for(h=0;h<largonocob;h++){
```

```
k=nocob1[h];
for(i=0;i<filas;i++){
for(j=0;j<largo[i];j++){
if(M[i][j]==k){
pfalta[ye]=i;
ye++;     } }  }  }  }

//relleno del vector aleapair que posee los pairs y se llena con los datos del archivo pfalta del individuo

for(i=0;i<largonocob;i++){
aleapair[i]=0;}

ka=aleatoria[0];
aleapair[0]= pfalta[ka];
count=nocobcob1[0];

for(h=1; h<largonocob; h++){
    for(i=0;i<(h-1);i++){
    count=count + nocobcob1[i];}
    total= count + aleatoria[h];
    count=0;
    aleapair[h]=pfalta[total];                    }

//printf("\n\n\n");
//for(i=0;i<largonocob;i++){
//printf("%d/", aleapair[i]);
  //       }
//printf("\n\n\n");

//anulacion de los repetidos
for(i=0;i<largonocob; i++){
for(j=(i+1);j<largonocob;j++){
if(aleapair[i] == aleapair[j]){
aleapair[j]=-1;               }     }        }

//dejo los ceros al final en el aleapair
ceros=0;
for(u=0;u<largonocob;u++){
if(aleapair[u]==-1){
ceros++;}     }

//for(j=0;j<largonocob;j++){
for(i=0;i<largonocob;i++){
if(aleapair[i]==-1){
       for(h=(i);h<(largonocob-1);h++){
       aleapair[h]=aleapair[h+1];}
     }}//}

for(u=(largonocob-ceros);u<largonocob;u++){
aleapair[u]=-1;     }

largoaleapair=0;
for(i=0;i<largonocob;i++){
if(aleapair[i]!=-1){
//printf("%d|", aleapair[i]);
largoaleapair++;}}
```

```
//printf("largoaleapair1: %d", largoaleapair);

//busqueda de los posibles reemplazantes y guardo en vector reemplazantes
//relleno vector simulacion
for(i=1;i<=legs;i++){
simulacion[i]=cob1[i];  }

au=0;
tr=vecescob[1];
for(i=tr;i<filasind;i++){
cont=0;
z=aleaind1[i];
if(z!=-1){
            for(j=0;j<largo[z];j++){
            xi= M[z][j];
            if(simulacion[xi]!=0 && simulacion[xi]!=1){
            cont++; }
                              }
if(cont==largo[z]){
reemplaza[au]=i;
au++;
for(h=0;h<largo[z];h++){
m=M[z][h];
simulacion[m]= simulacion[m]-1;
}}  }
}

for(i=0;i<filasind;i++){
if(aleaind1[i]==-1){
reemplaza[au]=i;
au++;
    }
     }
largoreemplaza=au;
//printf("\nlargoreemplaza %d\n", largoreemplaza);
//printf("\nlargoaleapair %d\n", largoaleapair);
//reemplazo
g=0;
while(g<largoreemplaza){
pape=aleapair[g];
ric=reemplaza[g];
aleaind1[ric]=pape;
g++;
}

// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind1[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }
```

```
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

//for(i=0;i<largonocob;i++){
//fprintf(idea3, "%d/", nocob1[i]);
  //        }

//printf("largonocob: %d", largonocob);

}//cierre while

//calculo del fitness
fitness1=0;
for(i=0;i<filasind;i++){
k=aleaind1[i];
if(k==-1){
cost=0;
    }else{
cost=costo[k];
         }
fitness1=fitness1+cost;                    }

//////////////////HASTA ACA EL INDIVIDUO CUBRE TODOS LOS LEGS//////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
for(i=1;i<=legs;i++){
simulacion[i]=cob1[i];  }

au=0;
tr=vecescob[1];

for(i=tr;i<filasind;i++){
cont=0;
z=aleaind1[i];
if(z!=-1){
            for(j=0;j<largo[z];j++){
            xi= M[z][j];
            if(simulacion[xi]!=0 && simulacion[xi]!=1){
            cont++; }
                        }
if(cont==largo[z]){
reemplaza[au]=i;
au++;
for(h=0;h<largo[z];h++){
m=M[z][h];
simulacion[m]= simulacion[m]-1;
}}  }}
largoreemplaza=au;
```

```
for(i=0;i<largoreemplaza;i++){
k=reemplaza[i];
aleaind1[k]=-1;
     }

//calculo del fitness
fitness1=0;
for(i=0;i<filasind;i++){
k=aleaind1[i];
if(k==-1){
cost=0;
    }else{
cost=costo[k];
          }
fitness1=fitness1+cost;                    }

//////////////////HASTA ACA EL INDIVIDUO CUBRE TODOS LOS LEGS Y ADEMAS TIENE UN FILTRO QUE ////ELIMIINA LOS PAIRS REPETIDOS Y LOS PAIRS REDUNDANTES/////////////////////////////////////////

if(vez==8){
for(i=0;i<filasind;i++){
aleaind8[i]=aleaind1[i];
}
fitness8=fitness1;
//fprintf(idea3, "\nfitness8: %d\n",fitness8);
//for(j=0;j<filasind;j++){
//printf("%d|", aleaind8[j]);
  //    }
}
if(vez==7){
for(i=0;i<filasind;i++){
aleaind7[i]=aleaind1[i];
}
fitness7=fitness1;
//fprintf(idea3,"\nfitness7: %d\n",fitness7);
//for(j=0;j<filasind;j++){
//printf("%d|", aleaind7[j]);
//}
}
if(vez==6){
for(i=0;i<filasind;i++){
aleaind6[i]=aleaind1[i];
}
fitness6=fitness1;
//fprintf(idea3,"\nfitness6: %d\n",fitness6);
//for(j=0;j<filasind;j++){
//printf("%d|", aleaind6[j]);
     //}

}
if(vez==5){
l=0;
for(i=0;i<filasind;i++){
aleaind5[i]=aleaind1[i];
}
```

```
fitness5=fitness1;
//fprintf(idea3, "\nfitness5: %d\n",fitness5);
//for(j=0;j<filasind;j++){
//printf("%d|", aleaind5[j]);
  //    }
}
if(vez==4){
for(i=0;i<filasind;i++){
aleaind4[i]=aleaind1[i];
}
fitness4=fitness1;
//fprintf(idea3, "\nfitness4: %d\n",fitness4);
//for(j=0;j<filasind;j++){
//printf("%d|", aleaind4[j]);
      //}
 }
if(vez==3){
for(i=0;i<filasind;i++){
aleaind3[i]=aleaind1[i];
}
fitness3=fitness1;
//fprintf(idea3, "\nfitness3: %d\n",fitness3);
//for(j=0;j<filasind;j++){
//printf("%d|", aleaind3[j]);
  //    }
}
if(vez==2){
for(i=0;i<filasind;i++){
aleaind2[i]=aleaind1[i];
}
fitness2=fitness1;
//fprintf(idea3, "\nfitness2: %d\n",fitness2);
//for(j=0;j<filasind;j++){
//printf("%d|", aleaind2[j]);
  //    }
}
}//cierre del for de los 8 individuos

//fprintf(idea3, "\nfitness1: %d\n",fitness1);
for(j=0;j<filasind;j++){
printf("%d|", aleaind1[j]);
      }
printf( "\n\n");


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//HASTA ACA TENGO LA POBLACION CON LOS 8 INDIVIDUOS QUE CUBREN TODOS LOS
///////LEGS////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
////////////////////////////////////aqui comienzan las iteraciones/////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

itera=0;
for(it=0;it<1000;it++){
printf("\nitera: %d\n", itera);
listo=0;
```

```
//elijo dos numeros de la poblacion
while(listo==0){
uno = (rand () % 8 ) + 1;
dos = (rand () % 8 ) + 1;
if(uno!=dos){
listo=1; }}

printf("\nuno %d \ndos: %d \n",uno, dos);

if(uno==1 && dos==2 ){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind1[i];
aleaindb[i]=aleaind2[i];}
fitnessa=fitness1;
fitnessb=fitness2;}
//
if(uno==1 && dos==3){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind1[i];
aleaindb[i]=aleaind3[i];}
fitnessa=fitness1;
fitnessb=fitness3;}
//
if(uno==1 && dos==4){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind1[i];
aleaindb[i]=aleaind4[i];}
fitnessa=fitness1;
fitnessb=fitness4;}
//
if(uno==1 && dos==5){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind1[i];
aleaindb[i]=aleaind5[i];}
fitnessa=fitness1;
fitnessb=fitness5;}
//
if(uno==1 && dos==6){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind1[i];
aleaindb[i]=aleaind6[i];}
fitnessa=fitness1;
fitnessb=fitness6;}
//
if(uno==1 && dos==7){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind1[i];
aleaindb[i]=aleaind7[i];}
fitnessa=fitness1;
fitnessb=fitness7;}
//
if(uno==1 && dos==8){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind1[i];
aleaindb[i]=aleaind8[i];}
fitnessa=fitness1;
```

```
fitnessb=fitness8;}
//
if(uno==2 && dos==1){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind2[i];
aleaindb[i]=aleaind1[i];}
fitnessa=fitness2;
fitnessb=fitness1;}
//
if(uno==2 && dos==3){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind2[i];
aleaindb[i]=aleaind3[i];}
fitnessa=fitness2;
fitnessb=fitness3;}
//
if(uno==2 && dos==4){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind2[i];
aleaindb[i]=aleaind4[i];}
fitnessa=fitness2;
fitnessb=fitness4;}
//
if(uno==2 && dos==5){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind2[i];
aleaindb[i]=aleaind5[i];}
fitnessa=fitness2;
fitnessb=fitness5;}
//
if(uno==2 && dos==6){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind2[i];
aleaindb[i]=aleaind6[i];}
fitnessa=fitness2;
fitnessb=fitness6;}
//
if(uno==2 && dos==7){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind2[i];
aleaindb[i]=aleaind7[i];}
fitnessa=fitness2;
fitnessb=fitness7;}
//
if(uno==2 && dos==8){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind2[i];
aleaindb[i]=aleaind8[i];}
fitnessa=fitness2;
fitnessb=fitness8;}
//
if(uno==3 && dos==1){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind3[i];
aleaindb[i]=aleaind1[i];}
fitnessa=fitness3;
```

```
fitnessb=fitness1;}
//
if(uno==3 && dos==2){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind3[i];
aleaindb[i]=aleaind2[i];}
fitnessa=fitness3;
fitnessb=fitness2;}
//
if(uno==3 && dos==4){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind3[i];
aleaindb[i]=aleaind4[i];}
fitnessa=fitness3;
fitnessb=fitness4;}
//
if(uno==3 && dos==5){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind3[i];
aleaindb[i]=aleaind5[i];}
fitnessa=fitness3;
fitnessb=fitness5;}
//
if(uno==3 && dos==6){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind3[i];
aleaindb[i]=aleaind6[i];}
fitnessa=fitness3;
fitnessb=fitness6;}
//
if(uno==3 && dos==7){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind3[i];
aleaindb[i]=aleaind7[i];}
fitnessa=fitness3;
fitnessb=fitness7;}
//
if(uno==3 && dos==8){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind3[i];
aleaindb[i]=aleaind8[i];}
fitnessa=fitness3;
fitnessb=fitness8;}
//
if(uno==4 && dos==1){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind4[i];
aleaindb[i]=aleaind1[i];}
fitnessa=fitness4;
fitnessb=fitness1;}
//
if(uno==4 && dos==2){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind4[i];
aleaindb[i]=aleaind2[i];}
fitnessa=fitness4;
```

```
fitnessb=fitness2;}
//
if(uno==4 && dos==3){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind4[i];
aleaindb[i]=aleaind3[i];}
fitnessa=fitness4;
fitnessb=fitness3;}
//
if(uno==4 && dos==5){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind4[i];
aleaindb[i]=aleaind5[i];}
fitnessa=fitness4;
fitnessb=fitness5;}
//
if(uno==4 && dos==6){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind4[i];
aleaindb[i]=aleaind6[i];}
fitnessa=fitness4;
fitnessb=fitness6;}
//
if(uno==4 && dos==7){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind4[i];
aleaindb[i]=aleaind7[i];}
fitnessa=fitness4;
fitnessb=fitness7;}
//
if(uno==4 && dos==8){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind4[i];
aleaindb[i]=aleaind8[i];}
fitnessa=fitness4;
fitnessb=fitness8;}
//
if(uno==5 && dos==1){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind5[i];
aleaindb[i]=aleaind1[i];}
fitnessa=fitness5;
fitnessb=fitness1;}
//
if(uno==5 && dos==2){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind5[i];
aleaindb[i]=aleaind2[i];}
fitnessa=fitness5;
fitnessb=fitness2;}
//
if(uno==5 && dos==3){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind5[i];
aleaindb[i]=aleaind3[i];}
fitnessa=fitness5;
```

```
fitnessb=fitness3;}
//
if(uno==5 && dos==4){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind5[i];
aleaindb[i]=aleaind4[i];}
fitnessa=fitness5;
fitnessb=fitness4;}
//
if(uno==5 && dos==6){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind5[i];
aleaindb[i]=aleaind6[i];}
fitnessa=fitness5;
fitnessb=fitness6;}
//
if(uno==5 && dos==7){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind5[i];
aleaindb[i]=aleaind7[i];}
fitnessa=fitness5;
fitnessb=fitness7;}
//
if((uno==5 && dos==8)){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind5[i];
aleaindb[i]=aleaind8[i];}
fitnessa=fitness5;
fitnessb=fitness8;}
//
if(uno==6 && dos==1){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind6[i];
aleaindb[i]=aleaind1[i];}
fitnessa=fitness6;
fitnessb=fitness1;}
//
if(uno==6 && dos==2){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind6[i];
aleaindb[i]=aleaind2[i];}
fitnessa=fitness6;
fitnessb=fitness2;}
//
if(uno==6 && dos==3){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind6[i];
aleaindb[i]=aleaind3[i];}
fitnessa=fitness6;
fitnessb=fitness3;}
//
if(uno==6 && dos==4){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind6[i];
aleaindb[i]=aleaind4[i];}
fitnessa=fitness6;
```

```
fitnessb=fitness4;}
//
if(uno==6 && dos==5){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind6[i];
aleaindb[i]=aleaind5[i];}
fitnessa=fitness6;
fitnessb=fitness5;}

//
if(uno==6 && dos==7){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind6[i];
aleaindb[i]=aleaind7[i];}
fitnessa=fitness6;
fitnessb=fitness7;}
//
if(uno==6 && dos==8){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind6[i];
aleaindb[i]=aleaind8[i];}
fitnessa=fitness6;
fitnessb=fitness8;}
//
if(uno==7 && dos==1){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind7[i];
aleaindb[i]=aleaind1[i];}
fitnessa=fitness7;
fitnessb=fitness1;}
//
if(uno==7 && dos==2){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind7[i];
aleaindb[i]=aleaind2[i];}
fitnessa=fitness7;
fitnessb=fitness2;}
//
if(uno==7 && dos==3){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind7[i];
aleaindb[i]=aleaind3[i];}
fitnessa=fitness7;
fitnessb=fitness3;}
//
if(uno==7 && dos==4){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind7[i];
aleaindb[i]=aleaind4[i];}
fitnessa=fitness7;
fitnessb=fitness4;}
//
if(uno==7 && dos==5){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind7[i];
aleaindb[i]=aleaind5[i];}
```

```
fitnessa=fitness7;
fitnessb=fitness5;}
//
if(uno==7 && dos==6){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind7[i];
aleaindb[i]=aleaind6[i];}
fitnessa=fitness7;
fitnessb=fitness6;}
//
if(uno==7 && dos==8){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind7[i];
aleaindb[i]=aleaind8[i];}
fitnessa=fitness7;
fitnessb=fitness8;}
//
if(uno==8 && dos==1){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind8[i];
aleaindb[i]=aleaind1[i];}
fitnessa=fitness8;
fitnessb=fitness1;}
//
if(uno==8 && dos==2){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind8[i];
aleaindb[i]=aleaind2[i];}
fitnessa=fitness8;
fitnessb=fitness2;}
//
if(uno==8 && dos==3){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind8[i];
aleaindb[i]=aleaind3[i];}
fitnessa=fitness8;
fitnessb=fitness3;}
//
if(uno==8 && dos==4){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind8[i];
aleaindb[i]=aleaind4[i];}
fitnessa=fitness8;
fitnessb=fitness4;}
//
if(uno==8 && dos==5){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind8[i];
aleaindb[i]=aleaind5[i];}
fitnessa=fitness8;
fitnessb=fitness5;}
//
if(uno==8 && dos==6){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind8[i];
aleaindb[i]=aleaind6[i];}
```

```
fitnessa=fitness8;
fitnessb=fitness6;}
//
if(uno==8 && dos==7){
for(i=0;i<filasind;i++){
aleainda[i]=aleaind8[i];
aleaindb[i]=aleaind7[i];}
fitnessa=fitness8;
fitnessb=fitness7;}
//
//////////INDIVIDUO A Y B ASIGNADOS A LOS DOS INDIVIDUOS ELEGIDOS DE LA POBLACION PARA ///////SER TRATADOS//////////////////////////////////////////////////////////////////

//CRUZAMIENTO
pc= (rand() % 11)/10.0;
//fprintf(idea3,"\nPc: %.2f\n", pc);

if(pc<=0.8){
//cruzo
ptocruce= (rand()%(filasind - fer)+ fer);
direc = (rand() % 11)/10.0;
//fprintf(idea3, "\nptocruce: %d \n", ptocruce);
//fprintf(idea3,"direc: %.2f\n",direc);
if(direc<=0.5){
//superior
for(i=fer;i<ptocruce;i++){
aleainda[i]=aleaindb[i];
                    }     }

if(direc > 0.5){
//inferior
for(i=ptocruce;i<filasind;i++){
aleainda[i]=aleaindb[i];
     }      }}

//MUTACION
pm= (rand() % 11)/100.0;
//fprintf(idea3, "\nPm: %.2f\n", pm);

if(pm<=0.02){
//muto
ptomuta= rand() % filasind;
ptomuta2= rand()% filasind;
ptomuta3= rand() % filasind;
ptomutb= rand() % filasind;
ptomutb2= rand() % filasind;
ptomutb3= rand() % filasind;
ptoarchivoa= rand () %filas;
ptoarchivoa2= rand () %filas;
ptoarchivoa3= rand () %filas;
ptoarchivob= rand () %filas;
ptoarchivob2= rand () %filas;
ptoarchivob3= rand () %filas;
aleainda[ptomuta]=ptoarchivoa;
aleainda[ptomuta2]=ptoarchivoa2;
aleainda[ptomuta3]=ptoarchivoa3;
```

```
aleaindb[ptomutb]=ptoarchivob;
aleaindb[ptomutb2]=ptoarchivob2;
aleaindb[ptomutb3]=ptoarchivob3;


//fprintf(idea3, "\nptomuta:%d \nptomutb:%d \nptoarchivoa:%d \nptoarchivob:%d\n", ptomuta, ptomutb, ptoarchivoa, ptoarchivob);}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////HASTA ACA EL INDIVIDUO HA SUFRIDO LA MUTACION O EL ////////////////////////////////////////////////////////////////CRUZAMIENTO/////////////////////////////////////////////

// llenado cobertura a
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleainda[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
           e=M[k][j];
           cob1[e]=cob1[e]+ 1;             }} }

//relleno no cubiertos a
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

////////////////////////////WHILE PARA FILTRADO DEL INDIVIDUO A//////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
while(largonocob!=0){

//definicion vector nocubiertoscobertura
int * nocobcob1;
nocobcob1 = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
nocobcob1[i]=0;  }

//llenado no cubiertos cobertura
for(i=0;i<largonocob;i++){
gig=nocob1[i];
gigo=cob[gig];
nocobcob1[i]=gigo;     }

contadora0=0;
// lectura nocobcobertura1
//fprintf(idea3,"\n\nnocobcob1\n\n");
for(i=0;i<largonocob;i++){
contadora0=contadora0 + nocobcob1[i];
//fprintf(idea3,"%d|", nocobcob1[i]);}

//definicion vector aleatoria
```

```
int * aleatoria;
aleatoria = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
aleatoria[i]=-1;  }

//llenado aleatoria
for(i=0;i<largonocob;i++){
al=nocobcob1[i];
m = ( rand () % al )+1;
aleatoria[i]=m;      }

//definicion vector pfalta
int * pfalta;
pfalta = (int*) malloc (sizeof(int) * (contadora0+1));
for(i=1;i<=contadora0;i++){
pfalta[i]=0;  }

 //llenado del vector pairings que faltan
ye=1;
while(ye<=contadora0){
for(h=0;h<largonocob;h++){
k=nocob1[h];
for(i=0;i<filas;i++){
for(j=0;j<largo[i];j++){
if(M[i][j]==k){
pfalta[ye]=i;
ye++;     } }  }  }  }

//definicion vector aleapair
int * aleapair;
aleapair = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
aleapair[i]=-1;  }

//relleno del vector aleapair que posee los pairs y se llena con los datos del archivo pfalta del individuo
ka=aleatoria[0];
aleapair[0]= pfalta[ka];
count=nocobcob1[0];

for(h=1; h<largonocob; h++){
    for(i=0;i<(h-1);i++){
    count=count + nocobcob1[i];}
    total= count + aleatoria[h];
    count=0;
    aleapair[h]=pfalta[total];
}

//anulacion de los repetidos
for(i=0;i<largonocob; i++){
for(j=(i+1);j<largonocob;j++){
if(aleapair[i] == aleapair[j]){
aleapair[j]=-1;               }     }        }

//lectura aleapair
cuentar=0;
//fprintf(idea3,"\n\naleapair\n\n");
```

```
for(i=0;i<largonocob;i++){
//fprintf(idea3,"%d|",aleapair[i]);
cuentar++;}
//fprintf(idea3, "\n\nlargoaleapair: %d\n\n", cuentar);

//dejo los ceros al final en el aleapair
ceros=0;
for(u=0;u<largonocob;u++){
if(aleapair[u]==-1){
ceros++;}     }

for(j=0;j<largonocob;j++){
for(i=0;i<largonocob;i++){
if(aleapair[i]==-1){
       for(h=(i);h<(largonocob-1);h++){
       aleapair[h]=aleapair[h+1];}
     }}}
for(u=(largonocob-ceros);u<largonocob;u++){
aleapair[u]=-1;     }

//busqueda de los posibles reemplazantes y guardo en vector reemplazantes
//relleno vector simulacion
for(i=1;i<=legs;i++){
simulacion[i]=cob1[i];  }

au=0;
tr=vecescob[1];

for(i=tr;i<filasind;i++){
cont=0;
z=aleainda[i];
if(z!=-1){
            for(j=0;j<largo[z];j++){
            xi= M[z][j];
            if(simulacion[xi]!=0 && simulacion[xi]!=1){
            cont++; }
                              }
if(cont==largo[z]){
reemplaza[au]=i;
au++;
for(h=0;h<largo[z];h++){
m=M[z][h];
simulacion[m]= simulacion[m]-1;
}}  }}

largoreemplaza=au;

//reemplazo
for(i=0;i<largonocob;i++){
pape=aleapair[i];
if(pape!=-1){
ric=reemplaza[i];
aleainda[ric]=pape;       } }

// llenado cobertura 1
for(i=1;i<=legs;i++){
```

```
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleainda[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

//free(aleapair);

}////////////////cierre del while que limpia al individuo a
// individuoa


for(i=1;i<=legs;i++){
simulacion[i]=cob1[i];  }

au=0;
tr=vecescob[1];

for(i=tr;i<filasind;i++){
cont=0;
z=aleainda[i];
if(z!=-1){
            for(j=0;j<largo[z];j++){
            xi= M[z][j];
            if(simulacion[xi]!=0 && simulacion[xi]!=1){
            cont++; }                                  }
if(cont==largo[z]){
reemplaza[au]=i;
au++;
for(h=0;h<largo[z];h++){
m=M[z][h];
simulacion[m]= simulacion[m]-1;
}}  }}

largoreemplaza=au;

for(i=0;i<largoreemplaza;i++){
k=reemplaza[i];
aleainda[k]=-1;      }

//calculo del fitness
fitnessa=0;
for(i=0;i<filasind;i++){
k=aleainda[i];
```

```
if(k==-1){
cost=0;
    }else{
cost=costo[k];
         }
fitnessa=fitnessa+cost;                    }
printf("fitnessa: %d",fitnessa);


// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleainda[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;
//printf("largonocoba: %d", largonocob);



//////////////////////////////comienza individuo b
// llenado cobertura b
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaindb[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

//relleno no cubiertos b
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;


//////////WHILE PARA FILTRADO DEL INDIVIDUO B
while(largonocob!=0){
```

```
//definicion vector nocubiertoscobertura
int * nocobcob1;
nocobcob1 = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
nocobcob1[i]=0;  }

for(i=0;i<largonocob;i++){
nocobcob1[i]=0; }

//llenado no cubiertos cobertura
for(i=0;i<largonocob;i++){
gig=nocob1[i];
gigo=cob[gig];
nocobcob1[i]=gigo;     }

contadora0=0;
// lectura nocobcobertura1
//fprintf(idea3,"\n\nnocobcob1\n\n");
for(i=0;i<largonocob;i++){
contadora0=contadora0 + nocobcob1[i];
//fprintf(idea3,"%d|", nocobcob1[i]);
}

//definicion vector aleatoria
int * aleatoria;
aleatoria = (int*) malloc (sizeof(int) * largonocob);
for(i=0;i<largonocob;i++){
aleatoria[i]=-1;  }

//llenado aleatoria
for(i=0;i<largonocob;i++){
al=nocobcob1[i];
m = ( rand () % al )+1;
aleatoria[i]=m;      }

//definicion vector pfalta
int * pfalta;
pfalta = (int*) malloc (sizeof(int) * (contadora0+1));
for(i=1;i<=contadora0;i++){
pfalta[i]=0;  }

 //llenado del vector pairings que faltan
ye=1;
while(ye<=contadora0){
for(h=0;h<largonocob;h++){
k=nocob1[h];
for(i=0;i<filas;i++){
for(j=0;j<largo[i];j++){
if(M[i][j]==k){
pfalta[ye]=i;
ye++;     } }  }  }  }

//definicion vector aleapair
int * aleapair;
aleapair = (int*) malloc (sizeof(int) * largonocob);
```

```
for(i=0;i<largonocob;i++){
aleapair[i]=-1;  }

//relleno del vector aleapair que posee los pairs y se llena con los datos del archivo pfalta del individuo
ka=aleatoria[0];
aleapair[0]= pfalta[ka];
count=nocobcob1[0];

for(h=1; h<largonocob; h++){
    for(i=0;i<(h-1);i++){
    count=count + nocobcob1[i];}
    total= count + aleatoria[h];
    count=0;
    aleapair[h]=pfalta[total];                    }

//anulacion de los repetidos
for(i=0;i<largonocob; i++){
for(j=(i+1);j<largonocob;j++){
if(aleapair[i] == aleapair[j]){
aleapair[j]=-1;              }     }        }


//lectura aleapair
cuentar=0;
//fprintf(idea3,"\n\naleapair\n\n");
for(i=0;i<largonocob;i++){
//fprintf(idea3,"%d|",aleapair[i]);
cuentar++;}
//fprintf(idea3, "\n\nlargoaleapair: %d\n\n", cuentar);

//dejo los ceros al final en el aleapair
ceros=0;
for(u=0;u<largonocob;u++){
if(aleapair[u]==-1){
ceros++;}     }

for(j=0;j<largonocob;j++){
for(i=0;i<largonocob;i++){
if(aleapair[i]==-1){
       for(h=(i);h<(largonocob-1);h++){
       aleapair[h]=aleapair[h+1];}
      }}}
for(u=(largonocob-ceros);u<largonocob;u++){
aleapair[u]=-1;     }

//busqueda de los posibles reemplazantes y guardo en vector reemplazantes
//relleno vector simulacion
for(i=1;i<=legs;i++){
simulacion[i]=cob1[i];  }

au=0;
tr=vecescob[1];

for(i=tr;i<filasind;i++){
cont=0;
z=aleaindb[i];
```

```
if(z!=-1){
            for(j=0;j<largo[z];j++){
            xi= M[z][j];
            if(simulacion[xi]!=0 && simulacion[xi]!=1){
            cont++; }                                   }

if(cont==largo[z]){
reemplaza[au]=i;
au++;
for(h=0;h<largo[z];h++){
m=M[z][h];
simulacion[m]= simulacion[m]-1;
}}  }}

largoreemplaza=au;


//reemplazo
for(i=0;i<largonocob;i++){
pape=aleapair[i];
if(pape!=-1){
ric=reemplaza[i];
aleaindb[ric]=pape;       } }

// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaindb[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

//free(aleapair);

}/////////////////cierre del while que limpia al individuo b
// individuob

// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaindb[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
```

```
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;
//printf("\nlargonocobb: %d\n", largonocob);

for(i=1;i<=legs;i++){
simulacion[i]=cob1[i];  }

au=0;
tr=vecescob[1];

for(i=tr;i<filasind;i++){
cont=0;
z=aleaindb[i];
if(z!=-1){
            for(j=0;j<largo[z];j++){
            xi= M[z][j];
            if(simulacion[xi]!=0 && simulacion[xi]!=1){
            cont++; }
                        }
if(cont==largo[z]){
reemplaza[au]=i;
au++;
for(h=0;h<largo[z];h++){
m=M[z][h];
simulacion[m]= simulacion[m]-1;
}}  }
}

largoreemplaza=au;

for(i=0;i<largoreemplaza;i++){
k=reemplaza[i];
aleaindb[k]=-1;
     }

//////////////////////CALCULO DEL FITNESS//////////////////////////////////
fitnessb=0;
for(i=0;i<filasind;i++){
k=aleaindb[i];
if(k==-1){
cost=0;
    }else{
cost=costo[k];
          }
fitnessb=fitnessb+cost;                    }
printf("fitnessb: %d",fitnessb);
```

```
/*
// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaindb[i];
if(k!=-1){
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;              }} }

//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;
//printf("\nlargonocobb: %d\n", largonocob);*/

///definicion vector fit/
    int * fit;
   fit = (int*) malloc (sizeof(int) * 100);
    for(i=1;i<=10;i++){
          fit[i]=0;  }
fit[1]=fitness1;
fit[2]=fitness2;
fit[3]=fitness3;
fit[4]=fitness4;
fit[5]=fitness5;
fit[6]=fitness6;
fit[7]=fitness7;
fit[8]=fitness8;

printf("\n\n");
for(i=1;i<=8;i++){
printf("%d|", fit[i]);
     }
printf("\n\n");

mayor1=fit[1];
posicion1=1;
for(i=2;i<=8;i++){
   if(fit[i]>mayor1){
   mayor1=fit[i];
   posicion1=i;
    }
     }
if(posicion1!=1){
mayor2=fit[1];
posicion2=1;
for(i=2;i<=8;i++){
```

```
if(fit[i]>mayor2 && i!=posicion1){
mayor2=fit[i];
posicion2=i;
    }     }
}else{
mayor2=fit[2];
posicion2=2;
for(i=3;i<=8;i++){
if(fit[i]>mayor2 && i!=posicion1){
mayor2=fit[i];
posicion2=i;
    }      }
}

printf("\nmayor1, posicion1:%d,%d \nmayor2, posicion2: %d, %d\n", mayor1, posicion1, mayor2, posicion2);

if(fitnessa<=fitnessb && fitnessa< mayor1){
/////reemplazo en a en mayor1
if(posicion1==1){
for(i=0;i<filasind;i++){
  aleaind1[i]=aleainda[i];    }
  fitness1=fitnessa;
      }
if(posicion1==2){
for(i=0;i<filasind;i++){
  aleaind2[i]=aleainda[i];    }
  fitness2=fitnessa;
        }
if(posicion1==3){
for(i=0;i<filasind;i++){
  aleaind3[i]=aleainda[i];    }
  fitness3=fitnessa;
      }
if(posicion1==4){
for(i=0;i<filasind;i++){
  aleaind4[i]=aleainda[i];    }
  fitness4=fitnessa;
      }
if(posicion1==5){
for(i=0;i<filasind;i++){
  aleaind5[i]=aleainda[i];    }
  fitness5=fitnessa;
      }
if(posicion1==6){
for(i=0;i<filasind;i++){
  aleaind6[i]=aleainda[i];    }
  fitness6=fitnessa;
      }
if(posicion1==7){
for(i=0;i<filasind;i++){
  aleaind7[i]=aleainda[i];    }
  fitness7=fitnessa;
      }
if(posicion1==8){
for(i=0;i<filasind;i++){
```

```
   aleaind8[i]=aleainda[i];    }
   fitness8=fitnessa;
      }
if(fitnessb< mayor2){
//reemplazo b en mayor2
if(posicion2==1){
for(i=0;i<filasind;i++){
   aleaind1[i]=aleaindb[i];    }
   fitness1=fitnessb;
    }
if(posicion2==2){
for(i=0;i<filasind;i++){
   aleaind2[i]=aleaindb[i];    }
   fitness2=fitnessb;
    }
if(posicion2==3){
for(i=0;i<filasind;i++){
   aleaind3[i]=aleaindb[i];    }
   fitness3=fitnessb;
    }
if(posicion2==4){
for(i=0;i<filasind;i++){
   aleaind4[i]=aleaindb[i];    }
   fitness4=fitnessb;
   }
if(posicion2==5){
for(i=0;i<filasind;i++){
   aleaind5[i]=aleaindb[i];    }
   fitness5=fitnessb;
    }
if(posicion2==6){
for(i=0;i<filasind;i++){
   aleaind6[i]=aleaindb[i];    }
   fitness6=fitnessb;
    }
if(posicion2==7){
for(i=0;i<filasind;i++){
   aleaind7[i]=aleaindb[i];    }
   fitness7=fitnessb;
    }
if(posicion2==8){
for(i=0;i<filasind;i++){
   aleaind8[i]=aleaindb[i];    }
   fitness8=fitnessb;
    }
    }     }//fin if

if(fitnessb<fitnessa && fitnessb < mayor1){
//reemplazoo b en mayor1
if(posicion1==1){
    for(i=0;i<filasind;i++){
    aleaind1[i]=aleaindb[i];    }
    fitness1=fitnessb;
    }
if(posicion1==2){
    for(i=0;i<filasind;i++){
```

```
    aleaind2[i]=aleaindb[i];    }
    fitness2=fitnessb;
     }
if(posicion1==3){
    for(i=0;i<filasind;i++){
    aleaind3[i]=aleaindb[i];    }
    fitness3=fitnessb;
     }
if(posicion1==4){
    for(i=0;i<filasind;i++){
    aleaind4[i]=aleaindb[i];    }
    fitness4=fitnessb;
     }
if(posicion1==5){
    for(i=0;i<filasind;i++){
    aleaind5[i]=aleaindb[i];    }
    fitness5=fitnessb;
     }
if(posicion1==6){
    for(i=0;i<filasind;i++){
    aleaind6[i]=aleaindb[i];    }
    fitness6=fitnessb;
     }
if(posicion1==7){
    for(i=0;i<filasind;i++){
    aleaind7[i]=aleaindb[i];    }
    fitness7=fitnessb;
     }
if(posicion1==8){
    for(i=0;i<filasind;i++){
    aleaind8[i]=aleaindb[i];    }
    fitness8=fitnessb;
     }
if(fitnessa<mayor2){
//reemplazo a en mayor2
if(posicion2==1){
for(i=0;i<filasind;i++){
   aleaind1[i]=aleainda[i];    }
   fitness1=fitnessa;
     }
if(posicion2==2){
for(i=0;i<filasind;i++){
   aleaind2[i]=aleainda[i];    }
   fitness2=fitnessa;
     }
if(posicion2==3){
for(i=0;i<filasind;i++){
   aleaind3[i]=aleainda[i];    }
   fitness3=fitnessa;
     }
if(posicion2==4){
for(i=0;i<filasind;i++){
   aleaind4[i]=aleainda[i];    }
   fitness4=fitnessa;
     }
if(posicion2==5){
```

```
for(i=0;i<filasind;i++){
   aleaind5[i]=aleainda[i];    }
   fitness5=fitnessa;
     }
if(posicion2==6){
for(i=0;i<filasind;i++){
   aleaind6[i]=aleainda[i];    }
   fitness6=fitnessa;
     }
if(posicion2==7){
for(i=0;i<filasind;i++){
   aleaind7[i]=aleainda[i];    }
   fitness7=fitnessa;
     }
if(posicion2==8){
for(i=0;i<filasind;i++){
   aleaind8[i]=aleainda[i];    }
   fitness8=fitnessa;
     }
     }    }// fin if

itera++;

}//cierre for iteraciones


fit[1]=fitness1;
fit[2]=fitness2;
fit[3]=fitness3;
fit[4]=fitness4;
fit[5]=fitness5;
fit[6]=fitness6;
fit[7]=fitness7;
fit[8]=fitness8;

printf("\n\n");
for(i=1;i<=8;i++){
printf("%d|", fit[i]);}

// FINALMENTE SE ELIGE AQUEL CON MENOR FITNESS Y SE INDICA LA SOLUCION Y EL FITNESS Y LA CANTIDAD DE EQUIPOS//////////

menor1=fit[1];
posic=1;
for(i=2;i<=8;i++){
    if(fit[i]<menor1){
    menor1=fit[i];
    posic=i;
     }
      }

final= posic;
fitnessfinal=menor1;
fprintf(idea3, "\n\nla solucion tiene como fitness: %d\n\n", fitnessfinal);
printf("\n\nla solucion tiene como fitness: %d\n\n", fitnessfinal);
```

```
if(final==1){
   fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind1[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
   // llenado cobertura 1
for(i=1;i<=legs;i++){
     cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind1[i];
if(k!=-1){
equipos++;
    for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
     nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
     nocob1[c]= i;
     c++;}}
largonocob=c;

printf("\nlargonocob1:%d\n", largonocob);
}


if(final==2){
  fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind2[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
// llenado cobertura 1
for(i=1;i<=legs;i++){
     cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind2[i];
if(k!=-1){
equipos++;
    for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
```

```
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

printf("\nlargonocob2:%d\n", largonocob);
}
if(final==3){
  fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind3[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind3[i];
if(k!=-1){
equipos++;
    for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;              }} }
printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

printf("\nlargonocob3:%d\n", largonocob);
}

if(final==4){
  fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind4[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind4[i];
if(k!=-1){
```

```
equipos++;
    for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }
printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

printf("\nlargonocob4:%d\n", largonocob);
}
if(final==5){
  fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind5[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind5[i];
if(k!=-1){
equipos++;
    for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }
printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

printf("\nlargonocob5:%d\n", largonocob);
}
if(final==6){
  fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind6[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
```

```
// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind6[i];
if(k!=-1){
equipos++;
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

printf("\nlargonocob6:%d\n", largonocob);
}
if(final==7){
  fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind7[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
// llenado cobertura 1
for(i=1;i<=legs;i++){
    cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind7[i];
if(k!=-1){
equipos++;
   for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }
printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
    nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
    nocob1[c]= i;
    c++;}}
largonocob=c;

printf("\nlargonocob7:%d\n", largonocob);
}
if(final==8){
```

```
  fprintf(idea3, "\nIndividuo Final\n");
   for(i=0;i<filasind;i++){
   fprintf(idea3, "%d|", aleaind8[i]);
}
fprintf(idea3, "\n\n");

equipos=0;
// llenado cobertura 1
for(i=1;i<=legs;i++){
     cob1[i]=0;  }
 for(i=0;i<filasind;i++){
k=aleaind8[i];
if(k!=-1){
equipos++;
    for(j=0;j< largo[k];j++){
            e=M[k][j];
            cob1[e]=cob1[e]+ 1;             }} }

printf("\nequipos: %d\n", equipos);
//relleno no cubiertos
for(i=0;i<(legs-150);i++){
     nocob1[i]=0;  }
c=0;
for(i=1;i<=legs;i++){
  if(cob1[i]==0){
     nocob1[c]= i;
     c++;}}
largonocob=c;

printf("\nlargonocob8:%d\n", largonocob);
}


//Se libera memoria
for(i=0;i<filas ;i++){
free(M[i]);
}
//Se Libera o borra el vector de punteros
free(M);
}


     final = time( NULL );
    //printf( "\nFinal: %u s\n", final );
   printf(  "\n\nNumero de segundos transcurridos desde el comienzo del programa: %f s\n", difftime(final,
comienzo) );

   tiempoComienzoPtr = gmtime( &comienzo );
   tiempoFinalPtr = gmtime( &final );
   printf( "Comienzo: %s\n", asctime(tiempoComienzoPtr) );
   printf( "Final: %s\n", asctime(tiempoFinalPtr) );
   //
 fclose(idea3);
 fclose(archivo);
 system("PAUSE");
}
```

APÉNDICE B – TABLAS

6 Individuos Mutación en 1 Punto

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	250	21	181	310	1	1500	250	71	145	246	1
500	250	35	177	301	2	1500	250	58	158	264	2
500	250	23	178	307	3	1500	250	51	159	267	3
500	250	33	172	298	4	1500	250	43	159	272	4
500	250	24	173	296	5	1500	250	35	151	251	5
500	250	19	178	305	6	1500	250	68	141	242	6
500	250	29	174	301	7	1500	250	77	160	278	7
500	250	21	169	289	8	1500	250	48	151	254	8
500	250	40	171	299	9	1500	250	61	152	259	9
500	250	18	175	302	10	1500	250	65	145	240	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	300	43	175	300	1	1500	300	51	148	233	1
500	300	28	170	296	2	1500	300	75	150	249	2
500	300	31	178	301	3	1500	300	56	153	252	3
500	300	33	171	290	4	1500	300	47	153	253	4
500	300	38	168	293	5	1500	300	72	147	238	5
500	300	37	165	286	6	1500	300	44	153	267	6
500	300	31	166	288	7	1500	300	66	155	264	7
500	300	47	171	291	8	1500	300	41	152	259	8
500	300	45	176	302	9	1500	300	58	149	237	9
500	300	29	175	300	10	1500	300	48	153	260	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	350	57	159	278	1	1500	350	66	148	245	1
500	350	45	175	299	2	1500	350	74	146	230	2
500	350	38	153	267	3	1500	350	69	150	228	3
500	350	36	165	287	4	1500	350	57	152	257	4
500	350	29	154	265	5	1500	350	52	159	268	5
500	350	38	173	291	6	1500	350	76	147	247	6
500	350	42	158	276	7	1500	350	49	146	240	7
500	350	18	170	290	8	1500	350	72	142	229	8
500	350	23	155	265	9	1500	350	34	145	232	9
500	350	32	176	301	10	1500	350	63	149	246	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	250	52	174	298	1	2000	250	66	144	231	1
1000	250	45	153	267	2	2000	250	73	148	238	2
1000	250	49	145	255	3	2000	250	56	148	245	3
1000	250	31	147	259	4	2000	250	42	145	236	4
1000	250	58	152	268	5	2000	250	49	142	227	5
1000	250	43	155	261	6	2000	250	62	147	243	6
1000	250	33	154	263	7	2000	250	37	144	233	7
1000	250	52	162	277	8	2000	250	53	143	229	8
1000	250	49	158	270	9	2000	250	77	147	248	9
1000	250	37	174	296	10	2000	250	61	142	226	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	300	53	154	256	1	2000	300	107	153	225	1
1000	300	60	153	273	2	2000	300	117	140	241	2
1000	300	63	157	276	3	2000	300	94	145	251	3
1000	300	55	173	289	4	2000	300	93	144	249	4
1000	300	43	171	283	5	2000	300	86	154	238	5
1000	300	39	155	256	6	2000	300	101	153	231	6
1000	300	28	172	285	7	2000	300	42	142	246	7
1000	300	41	154	271	8	2000	300	51	155	237	8
1000	300	47	175	292	9	2000	300	48	141	242	9
1000	300	32	153	272	10	2000	300	103	155	239	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	350	28	158	264	1	2000	350	133	155	225	1
1000	350	31	155	258	2	2000	350	102	147	233	2
1000	350	59	154	256	3	2000	350	99	155	237	3
1000	350	42	140	243	4	2000	350	76	142	214	4
1000	350	47	157	268	5	2000	350	85	155	238	5
1000	350	40	141	244	6	2000	350	28	155	235	6
1000	350	39	153	261	7	2000	350	35	143	213	7
1000	350	29	172	287	8	2000	350	93	140	241	8
1000	350	58	153	261	9	2000	350	115	154	222	9
1000	350	64	156	272	10	2000	350	87	152	220	10

Pruebas Rail516 algoritmo Versión A con 6 individuos

10 Individuos Mutación en 1 Punto

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	250	21	176	301	1
500	250	33	173	296	2
500	250	13	170	287	3
500	250	20	177	304	4
500	250	18	172	292	5
500	250	22	176	295	6
500	250	27	172	291	7
500	250	29	175	298	8
500	250	38	170	289	9
500	250	28	179	307	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	250	67	152	251	1
1500	250	55	136	199	2
1500	250	58	147	210	3
1500	250	45	149	240	4
1500	250	77	150	243	5
1500	250	43	154	258	6
1500	250	63	150	243	7
1500	250	52	140	199	8
1500	250	61	147	208	9
1500	250	78	148	248	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	300	34	175	299	1
500	300	23	156	270	2
500	300	27	156	271	3
500	300	46	172	283	4
500	300	25	176	302	5
500	300	32	175	305	6
500	300	38	173	293	7
500	300	32	174	295	8
500	300	27	171	283	9
500	300	39	176	301	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	300	37	147	247	1
1500	300	41	138	199	2
1500	300	68	154	229	3
1500	300	60	142	240	4
1500	300	38	148	247	5
1500	300	52	147	235	6
1500	300	81	161	253	7
1500	300	79	142	238	8
1500	300	44	160	252	9
1500	300	33	137	199	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	350	31	174	296	1
500	350	34	171	275	2
500	350	26	153	264	3
500	350	48	172	288	4
500	350	40	171	281	5
500	350	52	175	300	6
500	350	41	169	275	7
500	350	35	169	273	8
500	350	51	170	282	9
500	350	37	172	286	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	350	84	142	241	1
1500	350	79	146	237	2
1500	350	61	153	224	3
1500	350	68	139	199	4
1500	350	82	146	233	5
1500	350	59	148	242	6
1500	350	52	147	239	7
1500	350	67	144	249	8
1500	350	83	137	199	9
1500	350	66	148	248	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	250	55	169	276	1
1000	250	38	169	275	2
1000	250	49	175	290	3
1000	250	42	172	284	4
1000	250	55	154	266	5
1000	250	61	169	273	6
1000	250	53	169	276	7
1000	250	41	171	281	8
1000	250	39	154	268	9
1000	250	54	168	272	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	250	59	157	237	1
2000	250	47	149	223	2
2000	250	88	143	212	3
2000	250	52	139	199	4
2000	250	44	142	211	5
2000	250	40	147	217	6
2000	250	86	148	225	7
2000	250	51	156	238	8
2000	250	83	147	216	9
2000	250	55	150	227	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	300	61	166	272	1
1000	300	59	150	260	2
1000	300	48	169	278	3
1000	300	34	154	266	4
1000	300	66	152	264	5
1000	300	57	169	279	6
1000	300	37	151	261	7
1000	300	63	165	271	8
1000	300	46	154	266	9
1000	300	58	157	263	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	300	53	145	210	1
2000	300	55	144	205	2
2000	300	68	138	199	3
2000	300	70	154	233	4
2000	300	98	149	225	5
2000	300	71	148	214	6
2000	300	63	145	207	7
2000	300	55	150	229	8
2000	300	100	138	199	9
2000	300	62	144	208	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	350	29	152	262	1
1000	350	39	138	199	2
1000	350	45	161	271	3
1000	350	48	153	254	4
1000	350	69	162	274	5
1000	350	38	154	268	6
1000	350	47	152	255	7
1000	350	71	161	270	8
1000	350	52	163	276	9
1000	350	42	153	263	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	350	80	145	210	1
2000	350	102	148	216	2
2000	350	61	139	199	3
2000	350	40	143	202	4
2000	350	105	144	208	5
2000	350	73	147	218	6
2000	350	88	144	205	7
2000	350	51	139	199	8
2000	350	77	147	213	9
2000	350	89	150	228	10

Pruebas Rail516 algoritmo Versión A con 10 individuos

12 Individuos Mutación en 1 Punto

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	250	13	172	300	1
500	250	28	171	297	2
500	250	29	169	287	3
500	250	30	165	285	4
500	250	18	171	294	5
500	250	23	170	292	6
500	250	27	172	300	7
500	250	21	170	290	8
500	250	20	169	288	9
500	250	19	171	299	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	250	50	164	287	1
1500	250	47	143	239	2
1500	250	61	171	274	3
1500	250	46	163	260	4
1500	250	38	169	268	5
1500	250	35	159	254	6
1500	250	61	163	285	7
1500	250	43	144	246	8
1500	250	44	143	240	9
1500	250	50	143	239	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	300	33	163	284	1
500	300	29	162	280	2
500	300	37	162	279	3
500	300	35	160	273	4
500	300	35	170	290	5
500	300	30	170	291	6
500	300	25	168	288	7
500	300	21	165	284	8
500	300	28	163	280	9
500	300	37	162	281	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	300	42	140	233	1
1500	300	51	163	261	2
1500	300	66	158	250	3
1500	300	58	141	239	4
1500	300	50	142	240	5
1500	300	34	142	241	6
1500	300	43	141	236	7
1500	300	57	150	239	8
1500	300	66	143	245	9
1500	300	53	146	251	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	350	39	159	270	1
500	350	28	165	294	2
500	350	36	162	278	3
500	350	41	166	292	4
500	350	38	160	274	5
500	350	33	164	285	6
500	350	27	165	290	7
500	350	42	164	283	8
500	350	31	163	280	9
500	350	28	162	277	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	350	73	143	243	1
1500	350	69	142	240	2
1500	350	43	151	239	3
1500	350	71	150	229	4
1500	350	65	143	243	5
1500	350	49	141	237	6
1500	350	72	144	244	7
1500	350	68	142	239	8
1500	350	40	152	248	9
1500	350	73	144	247	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	250	40	164	286	1
1000	250	33	170	291	2
1000	250	37	162	282	3
1000	250	45	163	285	4
1000	250	31	160	276	5
1000	250	36	161	273	6
1000	250	29	163	293	7
1000	250	37	160	279	8
1000	250	30	162	281	9
1000	250	47	163	284	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	250	71	145	240	1
2000	250	69	146	229	2
2000	250	55	138	199	3
2000	250	63	149	229	4
2000	250	75	150	243	5
2000	250	54	149	235	6
2000	250	88	139	199	7
2000	250	57	140	233	8
2000	250	74	140	238	9
2000	250	75	146	240	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	300	47	160	278	1
1000	300	31	160	277	2
1000	300	38	161	280	3
1000	300	52	159	274	4
1000	300	47	161	280	5
1000	300	50	147	243	6
1000	300	44	147	245	7
1000	300	45	162	258	8
1000	300	33	143	238	9
1000	300	53	159	273	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	300	81	139	199	1
2000	300	102	141	210	2
2000	300	47	142	229	3
2000	300	39	142	231	4
2000	300	80	140	202	5
2000	300	63	143	238	6
2000	300	55	147	223	7
2000	300	106	141	233	8
2000	300	53	139	201	9
2000	300	88	141	227	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	350	48	146	243	1
1000	350	38	158	251	2
1000	350	53	159	255	3
1000	350	47	165	267	4
1000	350	34	164	263	5
1000	350	56	146	245	6
1000	350	48	158	250	7
1000	350	33	161	274	8
1000	350	45	160	258	9
1000	350	57	146	246	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	350	113	143	210	1
2000	350	79	141	207	2
2000	350	54	142	227	3
2000	350	107	142	213	4
2000	350	52	138	201	5
2000	350	89	137	203	6
2000	350	58	142	211	7
2000	350	47	140	205	8
2000	350	68	136	199	9
2000	350	112	137	200	10

Pruebas Rail516 algoritmo Versión A con 12 individuos

6 Individuos Mutación en 3 Puntos

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	250	37	173	305	1
500	250	34	172	294	2
500	250	24	175	299	3
500	250	28	176	299	4
500	250	41	170	286	5
500	250	34	173	292	6
500	250	18	178	307	7
500	250	17	172	289	8
500	250	23	172	295	9
500	250	28	175	301	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	250	76	155	255	1
1500	250	72	158	271	2
1500	250	75	154	256	3
1500	250	46	140	241	4
1500	250	57	159	265	5
1500	250	47	153	253	6
1500	250	80	148	233	7
1500	250	67	151	249	8
1500	250	54	145	234	9
1500	250	75	142	229	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	300	31	175	295	1
500	300	28	178	306	2
500	300	28	177	305	3
500	300	30	171	292	4
500	300	30	169	287	5
500	300	22	176	300	6
500	300	21	173	296	7
500	300	27	171	290	8
500	300	25	178	301	9
500	300	20	176	295	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	300	88	150	243	1
1500	300	72	155	252	2
1500	300	79	146	239	3
1500	300	73	140	220	4
1500	300	62	151	238	5
1500	300	66	143	221	6
1500	300	68	152	246	7
1500	300	74	155	257	8
1500	300	59	147	233	9
1500	300	46	146	219	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	350	45	176	302	1
500	350	71	173	294	2
500	350	27	173	293	3
500	350	37	173	297	4
500	350	33	171	291	5
500	350	34	171	295	6
500	350	37	171	289	7
500	350	44	171	291	8
500	350	39	176	294	9
500	350	31	178	301	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	350	76	159	254	1
1500	350	84	158	248	2
1500	350	77	154	226	3
1500	350	69	146	214	4
1500	350	76	155	239	5
1500	350	58	155	237	6
1500	350	58	156	246	7
1500	350	49	154	232	8
1500	350	56	140	207	9
1500	350	55	150	229	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	250	48	175	303	1
1000	250	40	171	294	2
1000	250	34	174	296	3
1000	250	25	171	291	4
1000	250	26	173	293	5
1000	250	37	174	288	6
1000	250	58	173	293	7
1000	250	43	167	288	8
1000	250	42	167	285	9
1000	250	54	171	289	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	250	62	151	228	1
2000	250	65	150	225	2
2000	250	71	147	219	3
2000	250	77	157	247	4
2000	250	58	156	236	5
2000	250	74	160	253	6
2000	250	82	147	218	7
2000	250	58	150	221	8
2000	250	74	152	224	9
2000	250	82	158	243	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	300	48	173	292	1
1000	300	45	169	288	2
1000	300	30	155	253	3
1000	300	46	173	289	4
1000	300	43	171	289	5
1000	300	39	175	298	6
1000	300	35	166	283	7
1000	300	52	171	292	8
1000	300	54	158	267	9
1000	300	49	157	264	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	300	87	148	201	1
2000	300	65	148	203	2
2000	300	74	154	229	3
2000	300	63	135	198	4
2000	300	55	147	210	5
2000	300	62	139	199	6
2000	300	67	143	205	7
2000	300	51	141	201	8
2000	300	87	137	200	9
2000	300	73	135	196	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	350	36	170	288	1
1000	350	44	169	289	2
1000	350	52	147	233	3
1000	350	47	158	278	4
1000	350	39	164	284	5
1000	350	52	155	263	6
1000	350	46	155	278	7
1000	350	61	152	271	8
1000	350	57	149	246	9
1000	350	54	149	249	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	350	69	147	204	1
2000	350	47	140	201	2
2000	350	62	137	199	3
2000	350	56	142	210	4
2000	350	57	138	199	5
2000	350	73	138	198	6
2000	350	92	137	200	7
2000	350	88	134	197	8
2000	350	84	143	204	9
2000	350	73	136	199	10

Pruebas Rail516 algoritmo Versión B con 6 individuos

10 Individuos Mutación en 3 Puntos

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	250	41	157	278	1
500	250	23	156	275	2
500	250	17	176	297	3
500	250	19	172	301	4
500	250	28	157	273	5
500	250	26	159	267	6
500	250	34	158	269	7
500	250	33	155	261	8
500	250	32	159	265	9
500	250	42	176	302	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	250	56	149	223	1
1500	250	43	148	220	2
1500	250	49	149	234	3
1500	250	51	134	194	4
1500	250	62	140	197	5
1500	250	69	151	239	6
1500	250	71	138	194	7
1500	250	62	137	195	8
1500	250	59	141	198	9
1500	250	52	147	219	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	300	43	150	260	1
500	300	29	158	265	2
500	300	28	146	250	3
500	300	31	145	257	4
500	300	33	173	302	5
500	300	32	171	284	6
500	300	30	160	277	7
500	300	42	158	264	8
500	300	41	159	269	9
500	300	50	160	264	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	300	62	148	218	1
1500	300	88	139	194	2
1500	300	89	148	220	3
1500	300	71	138	197	4
1500	300	62	151	237	5
1500	300	63	151	233	6
1500	300	74	137	195	7
1500	300	71	138	198	8
1500	300	87	151	223	9
1500	300	68	154	230	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	350	51	170	281	1
500	350	59	162	264	2
500	350	45	162	273	3
500	350	41	147	256	4
500	350	29	147	243	5
500	350	19	145	240	6
500	350	21	144	231	7
500	350	28	150	257	8
500	350	27	157	261	9
500	350	30	161	272	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1500	350	72	152	207	1
1500	350	77	156	218	2
1500	350	82	152	233	3
1500	350	85	141	202	4
1500	350	91	140	195	5
1500	350	92	153	243	6
1500	350	67	137	196	7
1500	350	69	154	236	8
1500	350	65	153	226	9
1500	350	54	134	194	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	250	46	148	256	1
1000	250	43	148	248	2
1000	250	48	145	249	3
1000	250	62	143	240	4
1000	250	68	144	245	5
1000	250	61	151	253	6
1000	250	59	150	251	7
1000	250	53	160	264	8
1000	250	54	159	261	9
1000	250	59	148	252	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	250	75	156	217	1
2000	250	62	156	216	2
2000	250	58	149	209	3
2000	250	70	138	196	4
2000	250	63	142	205	5
2000	250	52	141	197	6
2000	250	55	139	199	7
2000	250	60	156	217	8
2000	250	46	139	209	9
2000	250	67	140	211	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	300	62	144	240	1
1000	300	46	148	245	2
1000	300	44	142	229	3
1000	300	49	143	233	4
1000	300	51	144	238	5
1000	300	58	147	247	6
1000	300	52	139	198	7
1000	300	61	149	244	8
1000	300	52	140	199	9
1000	300	57	151	251	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	300	48	156	214	1
2000	300	57	155	212	2
2000	300	59	154	210	3
2000	300	61	144	199	4
2000	300	64	150	196	5
2000	300	82	147	195	6
2000	300	80	145	197	7
2000	300	49	145	198	8
2000	300	54	156	217	9
2000	300	52	133	194	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	350	48	146	240	1
1000	350	62	135	194	2
1000	350	56	138	198	3
1000	350	35	142	196	4
1000	350	59	150	241	5
1000	350	61	150	237	6
1000	350	67	150	230	7
1000	350	54	138	194	8
1000	350	38	152	240	9
1000	350	62	149	221	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
2000	350	83	139	206	1
2000	350	67	133	194	2
2000	350	75	138	201	3
2000	350	72	150	199	4
2000	350	91	139	196	5
2000	350	82	143	204	6
2000	350	88	137	198	7
2000	350	48	140	205	8
2000	350	63	139	194	9
2000	350	71	134	195	10

Pruebas Rail516 algoritmo Versión B con 10 individuos

12 Individuos Mutación en 3 Puntos

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	250	19	164	277	1	1500	250	91	142	196	1
500	250	37	163	283	2	1500	250	92	143	210	2
500	250	21	171	289	3	1500	250	88	148	209	3
500	250	29	170	291	4	1500	250	87	133	194	4
500	250	27	150	261	5	1500	250	75	157	233	5
500	250	41	156	279	6	1500	250	76	150	201	6
500	250	27	171	298	7	1500	250	71	160	240	7
500	250	29	153	262	8	1500	250	69	137	194	8
500	250	22	168	279	9	1500	250	72	138	196	9
500	250	29	162	269	10	1500	250	90	139	198	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	300	29	155	257	1	1500	300	90	139	194	1
500	300	32	171	288	2	1500	300	92	150	243	2
500	300	38	175	291	3	1500	300	89	155	242	3
500	300	28	170	272	4	1500	300	58	139	196	4
500	300	22	166	261	5	1500	300	86	157	233	5
500	300	27	150	243	6	1500	300	88	141	198	6
500	300	31	166	261	7	1500	300	84	147	241	7
500	300	30	170	281	8	1500	300	91	147	236	8
500	300	27	156	244	9	1500	300	72	150	234	9
500	300	39	167	269	10	1500	300	73	138	195	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	350	43	153	254	1	1500	350	82	150	229	1
500	350	33	140	220	2	1500	350	90	147	217	2
500	350	38	153	246	3	1500	350	89	139	195	3
500	350	36	149	233	4	1500	350	82	144	213	4
500	350	29	153	238	5	1500	350	59	146	219	5
500	350	41	151	231	6	1500	350	67	139	194	6
500	350	47	152	242	7	1500	350	65	146	230	7
500	350	49	160	261	8	1500	350	59	144	214	8
500	350	51	153	243	9	1500	350	89	145	221	9
500	350	48	152	241	10	1500	350	81	136	199	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	250	51	149	233	1	2000	250	87	139	196	1
1000	250	56	138	199	2	2000	250	82	134	194	2
1000	250	81	145	246	3	2000	250	91	145	210	3
1000	250	71	144	245	4	2000	250	99	144	203	4
1000	250	72	142	236	5	2000	250	73	135	194	5
1000	250	59	141	231	6	2000	250	82	144	207	6
1000	250	69	139	220	7	2000	250	98	146	211	7
1000	250	64	141	224	8	2000	250	59	139	195	8
1000	250	68	137	198	9	2000	250	85	140	199	9
1000	250	61	148	232	10	2000	250	87	139	197	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	300	69	140	219	1	2000	300	104	139	201	1
1000	300	73	140	217	2	2000	300	99	137	200	2
1000	300	72	141	223	3	2000	300	90	133	199	3
1000	300	78	145	231	4	2000	300	89	135	200	4
1000	300	62	139	194	5	2000	300	108	134	194	5
1000	300	66	143	217	6	2000	300	93	137	198	6
1000	300	60	141	201	7	2000	300	92	140	194	7
1000	300	59	143	205	8	2000	300	98	141	196	8
1000	300	58	142	207	9	2000	300	100	139	197	9
1000	300	48	139	196	10	2000	300	101	138	194	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	350	471	139	194	1	2000	350	99	135	195	1
1000	350	48	146	218	2	2000	350	107	133	194	2
1000	350	49	139	196	3	2000	350	97	136	197	3
1000	350	55	137	197	4	2000	350	106	141	200	4
1000	350	58	145	221	5	2000	350	220	135	194	5
1000	350	88	146	220	6	2000	350	119	141	201	6
1000	350	89	133	194	7	2000	350	113	140	198	7
1000	350	61	141	205	8	2000	350	102	140	199	8
1000	350	91	143	215	9	2000	350	200	138	196	9
1000	350	62	137	195	10	2000	350	101	133	195	10

Pruebas Rail516 algoritmo Versión B con 12 individuos

6 Individuos Mutación en 1 Punto

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	33	1	17	17	1
250	33	0	15	15	2
250	33	2	14	14	3
250	33	1	16	16	4
250	33	1	14	14	5
250	33	2	14	14	6
250	33	1	16	16	7
250	33	0	15	15	8
250	33	1	15	15	9
250	33	2	16	16	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	33	2	14	14	1
1000	33	2	13	13	2
1000	33	1	12	12	3
1000	33	0	12	12	4
1000	33	0	14	14	5
1000	33	1	13	13	6
1000	33	2	15	15	7
1000	33	2	13	13	8
1000	33	1	13	13	9
1000	33	3	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	53	2	15	15	1
250	53	2	14	14	2
250	53	1	15	15	3
250	53	0	16	16	4
250	53	2	14	14	5
250	53	1	13	13	6
250	53	2	15	15	7
250	53	1	16	16	8
250	53	1	13	13	9
250	53	1	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	53	2	14	14	1
1000	53	3	13	13	2
1000	53	0	12	12	3
1000	53	2	13	13	4
1000	53	1	14	14	5
1000	53	2	14	14	6
1000	53	1	12	12	7
1000	53	3	14	14	8
1000	53	1	14	14	9
1000	53	2	12	12	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	103	0	15	15	1
250	103	0	14	14	2
250	103	0	14	14	3
250	103	0	16	16	4
250	103	1	15	15	5
250	103	0	14	14	6
250	103	1	15	15	7
250	103	0	14	14	8
250	103	0	16	16	9
250	103	1	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	103	4	13	13	1
1000	103	1	13	13	2
1000	103	3	12	12	3
1000	103	2	13	13	4
1000	103	2	13	13	5
1000	103	1	15	15	6
1000	103	2	14	14	7
1000	103	1	14	14	8
1000	103	3	13	13	9
1000	103	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	33	2	13	13	1
500	33	1	15	15	2
500	33	2	13	13	3
500	33	1	15	15	4
500	33	1	14	14	5
500	33	2	14	14	6
500	33	1	15	15	7
500	33	2	13	13	8
500	33	1	15	15	9
500	33	1	12	12	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	33	9	14	14	1
4000	33	8	13	13	2
4000	33	8	14	14	3
4000	33	7	14	14	4
4000	33	10	14	14	5
4000	33	10	14	14	6
4000	33	8	14	14	7
4000	33	7	13	13	8
4000	33	8	14	14	9
4000	33	6	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	53	1	14	14	1
500	53	1	14	14	2
500	53	0	14	14	3
500	53	1	14	14	4
500	53	1	15	15	5
500	53	1	15	15	6
500	53	0	14	14	7
500	53	0	14	14	8
500	53	0	15	15	9
500	53	1	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	53	6	13	13	1
4000	53	9	12	12	2
4000	53	8	13	13	3
4000	53	9	13	13	4
4000	53	11	13	13	5
4000	53	8	12	12	6
4000	53	8	14	14	7
4000	53	6	12	12	8
4000	53	11	14	14	9
4000	53	9	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	103	1	14	14	1
500	103	2	14	14	2
500	103	1	14	14	3
500	103	2	14	14	4
500	103	0	13	13	5
500	103	1	14	14	6
500	103	1	14	14	7
500	103	2	15	15	8
500	103	1	14	14	9
500	103	1	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	103	10	13	13	1
4000	103	8	14	14	2
4000	103	10	13	13	3
4000	103	11	13	13	4
4000	103	11	12	12	5
4000	103	7	14	14	6
4000	103	11	13	13	7
4000	103	8	12	12	8
4000	103	10	13	13	9
4000	103	8	12	12	10

Pruebas R1 con 6 individuos en la Herramienta A

8 Individuos Mutación en 1 Punto

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	33	0	15	15	1
250	33	1	14	14	2
250	33	0	13	13	3
250	33	0	13	13	4
250	33	1	15	15	5
250	33	0	14	14	6
250	33	1	16	16	7
250	33	1	13	13	8
250	33	0	15	15	9
250	33	1	16	16	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	33	1	12	12	1
1000	33	1	13	13	2
1000	33	3	12	12	3
1000	33	3	12	12	4
1000	33	1	13	13	5
1000	33	3	12	12	6
1000	33	3	13	13	7
1000	33	3	13	13	8
1000	33	2	14	14	9
1000	33	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	53	1	14	14	1
250	53	1	15	15	2
250	53	1	14	14	3
250	53	1	15	15	4
250	53	1	13	13	5
250	53	1	14	14	6
250	53	0	13	13	7
250	53	1	13	13	8
250	53	0	16	16	9
250	53	1	15	15	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	53	1	14	14	1
1000	53	2	13	13	2
1000	53	3	13	13	3
1000	53	2	14	14	4
1000	53	2	12	12	5
1000	53	3	13	13	6
1000	53	1	14	14	7
1000	53	2	13	13	8
1000	53	3	13	13	9
1000	53	3	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	103	1	15	15	1
250	103	0	14	14	2
250	103	2	14	14	3
250	103	0	15	15	4
250	103	1	14	14	5
250	103	1	15	15	6
250	103	1	14	14	7
250	103	2	15	15	8
250	103	0	14	14	9
250	103	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	103	1	12	12	1
1000	103	2	13	13	2
1000	103	1	13	13	3
1000	103	3	14	14	4
1000	103	4	13	13	5
1000	103	2	12	12	6
1000	103	1	14	14	7
1000	103	4	12	12	8
1000	103	2	13	13	9
1000	103	3	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	33	2	14	14	1
500	33	1	12	12	2
500	33	1	13	13	3
500	33	1	13	13	4
500	33	2	12	12	5
500	33	1	13	13	6
500	33	1	13	13	7
500	33	1	15	15	8
500	33	3	14	14	9
500	33	3	12	12	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	33	7	14	14	1
4000	33	10	13	13	2
4000	33	8	14	14	3
4000	33	9	14	14	4
4000	33	7	14	14	5
4000	33	9	14	14	6
4000	33	9	13	13	7
4000	33	7	12	12	8
4000	33	8	13	13	9
4000	33	7	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	53	1	14	14	1
500	53	1	13	13	2
500	53	2	14	14	3
500	53	1	13	13	4
500	53	2	14	14	5
500	53	2	14	14	6
500	53	1	13	13	7
500	53	0	13	13	8
500	53	0	12	12	9
500	53	1	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	53	7	13	13	1
4000	53	9	13	13	2
4000	53	7	13	13	3
4000	53	8	14	14	4
4000	53	10	13	13	5
4000	53	8	13	13	6
4000	53	11	14	14	7
4000	53	9	12	12	8
4000	53	11	13	13	9
4000	53	11	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	103	1	14	14	1
500	103	1	12	12	2
500	103	1	14	14	3
500	103	2	12	12	4
500	103	1	13	13	5
500	103	1	13	13	6
500	103	2	14	14	7
500	103	1	13	13	8
500	103	1	14	14	9
500	103	2	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	103	10	12	12	1
4000	103	12	13	13	2
4000	103	7	13	13	3
4000	103	7	14	14	4
4000	103	11	13	13	5
4000	103	6	14	14	6
4000	103	8	12	12	7
4000	103	12	12	12	8
4000	103	8	13	13	9
4000	103	10	12	12	10

Pruebas R1 con 8 individuos en la Herramienta A

10 Individuos Mutación en 1 Punto

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	33	0	14	14	1
250	33	1	16	16	2
250	33	1	14	14	3
250	33	0	15	15	4
250	33	1	16	16	5
250	33	1	16	16	6
250	33	0	13	13	7
250	33	1	14	14	8
250	33	0	15	15	9
250	33	0	16	16	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	33	2	14	14	1
1000	33	2	12	12	2
1000	33	2	13	13	3
1000	33	1	13	13	4
1000	33	4	12	12	5
1000	33	3	13	13	6
1000	33	4	14	14	7
1000	33	6	14	14	8
1000	33	3	13	13	9
1000	33	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	53	1	13	13	1
250	53	0	14	14	2
250	53	1	15	15	3
250	53	1	16	16	4
250	53	1	15	15	5
250	53	0	13	13	6
250	53	1	13	13	7
250	53	1	16	16	8
250	53	1	15	15	9
250	53	0	16	16	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	53	4	12	12	1
1000	53	3	14	14	2
1000	53	3	14	14	3
1000	53	4	14	14	4
1000	53	3	13	13	5
1000	53	2	12	12	6
1000	53	4	13	13	7
1000	53	2	14	14	8
1000	53	1	12	12	9
1000	53	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	103	1	14	14	1
250	103	0	16	16	2
250	103	1	15	15	3
250	103	0	13	13	4
250	103	1	14	14	5
250	103	1	14	14	6
250	103	1	15	15	7
250	103	0	14	14	8
250	103	1	16	16	9
250	103	2	15	15	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	103	2	14	14	1
1000	103	4	12	12	2
1000	103	4	13	13	3
1000	103	3	13	13	4
1000	103	3	12	12	5
1000	103	2	13	13	6
1000	103	3	12	12	7
1000	103	4	12	12	8
1000	103	3	12	12	9
1000	103	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	33	2	14	14	1
500	33	2	13	13	2
500	33	1	15	15	3
500	33	2	14	14	4
500	33	1	14	14	5
500	33	2	15	15	6
500	33	1	15	15	7
500	33	1	14	14	8
500	33	1	14	14	9
500	33	1	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	33	8	12	12	1
4000	33	10	13	13	2
4000	33	8	12	12	3
4000	33	9	13	13	4
4000	33	6	14	14	5
4000	33	14	13	13	6
4000	33	9	13	13	7
4000	33	11	12	12	8
4000	33	8	13	13	9
4000	33	12	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	53	2	15	15	1
500	53	2	15	15	2
500	53	2	14	14	3
500	53	3	15	15	4
500	53	0	13	13	5
500	53	2	13	13	6
500	53	0	14	14	7
500	53	3	14	14	8
500	53	2	14	14	9
500	53	1	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	53	7	12	12	1
4000	53	13	13	13	2
4000	53	9	12	12	3
4000	53	10	13	13	4
4000	53	7	13	13	5
4000	53	10	12	12	6
4000	53	13	13	13	7
4000	53	11	13	13	8
4000	53	8	12	12	9
4000	53	7	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	103	1	14	14	1
500	103	1	14	14	2
500	103	2	13	13	3
500	103	1	14	14	4
500	103	1	13	13	5
500	103	2	14	14	6
500	103	2	13	13	7
500	103	2	13	13	8
500	103	1	14	14	9
500	103	1	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	103	13	13	13	1
4000	103	10	13	13	2
4000	103	13	12	12	3
4000	103	8	13	13	4
4000	103	13	12	12	5
4000	103	15	13	13	6
4000	103	7	12	12	7
4000	103	9	13	13	8
4000	103	12	12	12	9
4000	103	11	13	13	10

Pruebas R1 con 10 individuos en la Herramienta A

6 Individuos Mutación en 3 Puntos

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	33	0	14	14	1	1000	33	2	13	13	1
250	33	2	13	13	2	1000	33	2	12	12	2
250	33	2	13	13	3	1000	33	2	12	12	3
250	33	0	13	13	4	1000	33	2	12	12	4
250	33	1	14	14	5	1000	33	2	13	13	5
250	33	0	14	14	6	1000	33	2	13	13	6
250	33	1	14	14	7	1000	33	2	13	13	7
250	33	0	12	12	8	1000	33	2	13	13	8
250	33	1	15	15	9	1000	33	3	13	13	9
250	33	3	13	13	10	1000	33	3	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	53	1	13	13	1	1000	53	1	13	13	1
250	53	1	14	14	2	1000	53	2	14	14	2
250	53	1	13	13	3	1000	53	2	12	12	3
250	53	1	13	13	4	1000	53	2	13	13	4
250	53	1	14	14	5	1000	53	1	14	14	5
250	53	0	13	13	6	1000	53	2	12	12	6
250	53	1	13	13	7	1000	53	3	12	12	7
250	53	0	13	13	8	1000	53	2	14	14	8
250	53	1	13	13	9	1000	53	2	13	13	9
250	53	0	13	13	10	1000	53	2	12	12	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	103	1	14	14	1	1000	103	2	13	13	1
250	103	0	14	14	2	1000	103	2	14	14	2
250	103	1	14	14	3	1000	103	2	13	13	3
250	103	0	14	14	4	1000	103	2	13	13	4
250	103	1	14	14	5	1000	103	2	14	14	5
250	103	0	14	14	6	1000	103	1	15	15	6
250	103	0	14	14	7	1000	103	2	14	14	7
250	103	0	14	14	8	1000	103	2	14	14	8
250	103	0	14	14	9	1000	103	2	14	14	9
250	103	0	14	14	10	1000	103	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	33	1	13	13	1	4000	33	8	14	14	1
500	33	1	13	13	2	4000	33	7	13	13	2
500	33	2	13	13	3	4000	33	8	14	14	3
500	33	1	15	15	4	4000	33	9	14	14	4
500	33	1	14	14	5	4000	33	10	13	13	5
500	33	1	14	14	6	4000	33	11	14	14	6
500	33	1	13	13	7	4000	33	8	14	14	7
500	33	1	13	13	8	4000	33	8	13	13	8
500	33	1	13	13	9	4000	33	8	14	14	9
500	33	1	12	12	10	4000	33	6	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	53	0	13	13	1	4000	53	7	12	12	1
500	53	1	14	14	2	4000	53	9	13	13	2
500	53	1	14	14	3	4000	53	8	12	12	3
500	53	1	14	14	4	4000	53	9	13	13	4
500	53	1	14	14	5	4000	53	10	13	13	5
500	53	0	14	14	6	4000	53	8	13	13	6
500	53	0	13	13	7	4000	53	8	14	14	7
500	53	0	14	14	8	4000	53	7	13	13	8
500	53	1	14	14	9	4000	53	9	14	14	9
500	53	1	15	15	10	4000	53	9	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	103	1	14	14	1	4000	103	8	14	14	1
500	103	1	13	13	2	4000	103	7	14	14	2
500	103	1	14	14	3	4000	103	8	14	14	3
500	103	0	14	14	4	4000	103	9	13	13	4
500	103	1	14	14	5	4000	103	10	13	13	5
500	103	1	15	15	6	4000	103	7	14	14	6
500	103	1	14	14	7	4000	103	8	13	13	7
500	103	1	15	15	8	4000	103	8	14	14	8
500	103	1	14	14	9	4000	103	7	13	13	9
500	103	0	13	13	10	4000	103	9	13	13	10

Pruebas R1 con 6 individuos en la Herramienta B

8 individuos Mutación en 3 Puntos

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	33	1	14	14	1	1000	33	2	13	13	1
250	33	1	13	13	2	1000	33	3	13	13	2
250	33	0	13	13	3	1000	33	2	12	12	3
250	33	1	13	13	4	1000	33	3	13	13	4
250	33	1	13	13	5	1000	33	2	12	12	5
250	33	1	14	14	6	1000	33	1	12	12	6
250	33	1	14	14	7	1000	33	2	13	13	7
250	33	1	13	13	8	1000	33	3	13	13	8
250	33	1	14	14	9	1000	33	2	13	13	9
250	33	1	14	14	10	1000	33	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	53	0	13	13	1	1000	53	2	13	13	1
250	53	1	13	13	2	1000	53	2	13	13	2
250	53	1	14	14	3	1000	53	3	12	12	3
250	53	0	14	14	4	1000	53	2	13	13	4
250	53	1	13	13	5	1000	53	2	12	12	5
250	53	1	14	14	6	1000	53	2	13	13	6
250	53	0	13	13	7	1000	53	2	13	13	7
250	53	0	14	14	8	1000	53	2	13	13	8
250	53	0	14	14	9	1000	53	2	13	13	9
250	53	0	14	14	10	1000	53	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	103	0	14	14	1	1000	103	2	13	13	1
250	103	0	14	14	2	1000	103	2	13	13	2
250	103	0	14	14	3	1000	103	2	13	13	3
250	103	0	14	14	4	1000	103	2	13	13	4
250	103	1	14	14	5	1000	103	2	13	13	5
250	103	1	14	14	6	1000	103	2	12	12	6
250	103	1	14	14	7	1000	103	2	13	13	7
250	103	0	14	14	8	1000	103	2	12	12	8
250	103	0	14	14	9	1000	103	2	13	13	9
250	103	0	14	14	10	1000	103	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	33	1	12	12	1	4000	33	4	13	13	1
500	33	2	12	12	2	4000	33	9	13	13	2
500	33	2	13	13	3	4000	33	8	13	13	3
500	33	1	13	13	4	4000	33	9	12	12	4
500	33	1	13	13	5	4000	33	7	13	13	5
500	33	1	12	12	6	4000	33	9	13	13	6
500	33	1	13	13	7	4000	33	9	13	13	7
500	33	1	14	14	8	4000	33	9	11	11	8
500	33	2	12	12	9	4000	33	9	13	13	9
500	33	2	12	12	10	4000	33	9	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	53	2	12	12	1	4000	53	8	12	12	1
500	53	1	13	13	2	4000	53	9	13	13	2
500	53	1	14	14	3	4000	53	9	13	13	3
500	53	1	13	13	4	4000	53	8	13	13	4
500	53	0	14	14	5	4000	53	9	13	13	5
500	53	0	14	14	6	4000	53	8	13	13	6
500	53	1	13	13	7	4000	53	7	12	12	7
500	53	1	13	13	8	4000	53	9	13	13	8
500	53	1	13	13	9	4000	53	10	11	11	9
500	53	1	13	13	10	4000	53	7	12	12	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento	iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	103	0	13	14	1	4000	103	9	13	13	1
500	103	1	12	12	2	4000	103	9	13	13	2
500	103	1	13	13	3	4000	103	7	13	13	3
500	103	0	13	13	4	4000	103	7	14	14	4
500	103	1	13	13	5	4000	103	6	14	14	5
500	103	0	13	13	6	4000	103	6	14	14	6
500	103	0	13	13	7	4000	103	8	13	13	7
500	103	1	13	13	8	4000	103	9	13	13	8
500	103	1	12	12	9	4000	103	8	13	13	9
500	103	1	12	12	10	4000	103	9	13	13	10

Pruebas R1 con 8 individuos en la Herramienta B

10 individuos Mutación en 3 Puntos

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	33	1	13	13	1
250	33	1	14	14	2
250	33	1	13	13	3
250	33	1	13	13	4
250	33	1	14	14	5
250	33	1	14	14	6
250	33	1	13	13	7
250	33	1	13	13	8
250	33	1	13	13	9
250	33	1	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	33	4	12	12	1
1000	33	3	13	13	2
1000	33	2	13	13	3
1000	33	1	13	13	4
1000	33	3	12	12	5
1000	33	3	13	13	6
1000	33	3	12	12	7
1000	33	3	13	13	8
1000	33	3	13	13	9
1000	33	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	53	1	12	12	1
250	53	0	14	14	2
250	53	0	13	13	3
250	53	0	13	13	4
250	53	1	13	13	5
250	53	0	13	13	6
250	53	0	14	14	7
250	53	1	14	14	8
250	53	0	13	13	9
250	53	0	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	53	2	14	14	1
1000	53	3	13	13	2
1000	53	3	14	14	3
1000	53	2	12	12	4
1000	53	3	13	13	5
1000	53	2	13	13	6
1000	53	2	13	13	7
1000	53	2	13	13	8
1000	53	2	12	12	9
1000	53	2	12	12	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
250	103	0	14	14	1
250	103	0	14	14	2
250	103	0	15	15	3
250	103	0	13	13	4
250	103	0	13	13	5
250	103	0	14	14	6
250	103	0	13	13	7
250	103	0	14	14	8
250	103	1	13	13	9
250	103	0	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
1000	103	2	13	13	1
1000	103	2	13	13	2
1000	103	2	13	13	3
1000	103	3	13	13	4
1000	103	2	13	13	5
1000	103	2	13	13	6
1000	103	5	11	11	7
1000	103	2	12	12	8
1000	103	2	13	13	9
1000	103	2	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	33	1	13	13	1
500	33	1	12	12	2
500	33	1	13	13	3
500	33	1	14	14	4
500	33	1	12	12	5
500	33	2	13	13	6
500	33	1	12	12	7
500	33	2	13	13	8
500	33	2	13	13	9
500	33	1	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	33	10	12	12	1
4000	33	10	13	13	2
4000	33	10	12	12	3
4000	33	9	13	13	4
4000	33	10	11	11	5
4000	33	11	12	12	6
4000	33	10	13	13	7
4000	33	11	12	12	8
4000	33	10	13	13	9
4000	33	11	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	53	1	14	14	1
500	53	0	13	13	2
500	53	1	13	13	3
500	53	1	15	15	4
500	53	0	13	13	5
500	53	2	13	13	6
500	53	1	14	14	7
500	53	2	13	13	8
500	53	2	13	13	9
500	53	0	13	13	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	53	10	13	13	1
4000	53	12	13	13	2
4000	53	9	13	13	3
4000	53	10	11	11	4
4000	53	10	13	13	5
4000	53	10	13	13	6
4000	53	11	13	13	7
4000	53	10	13	13	8
4000	53	10	12	12	9
4000	53	10	11	11	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
500	103	1	13	13	1
500	103	0	14	14	2
500	103	1	13	13	3
500	103	1	14	14	4
500	103	1	13	13	5
500	103	1	14	14	6
500	103	1	13	13	7
500	103	1	14	14	8
500	103	1	14	14	9
500	103	1	14	14	10

iteraciones	tamaño individuo	tiempo	equipos	costo	intento
4000	103	9	12	12	1
4000	103	10	13	13	2
4000	103	10	13	13	3
4000	103	8	13	13	4
4000	103	10	11	11	5
4000	103	10	13	13	6
4000	103	10	13	13	7
4000	103	9	13	13	8
4000	103	12	12	12	9
4000	103	10	12	12	10

Pruebas R1 con 10 individuos en la Herramienta B