Benemérita Universidad Autónoma de Puebla
Facultad de Cienias de la Computación

Análisis y Diseño de Algoritmos Evolutivos basados en Diversidad para Problemas Combinatorios

TESIS

Que para obtener el grado de

Licenciado en Ciencias de la Computación

PRESENTA:

Nayeli Angel Pérez

Directores de Tesis:

Dra. Darnes Vilariño Ayala
Benemérita Universidad Autónoma de Puebla

Dr. Carlos Segura González
Centro de Investigación en Matemáticas A.C

Dedicado a:
Mis padres, Andrea y Genaro.
Mi hermana menor, Anahi Angel.
Y a mi pequeña sobrina, María Fernanda.

"A veces la gente piensa que debe ser inteligente para poder triunfar, sin embargo, hay cosas aún más importantes (. . .) Una de las claves más importantes es la curiosidad por resolver los problemas que existen en el mundo y tener pasión hasta lograr resolverlas"
Dr. Raúl Rojas González.

II

Agradecimientos

En primer lugar quiero agradecer a mis padres, los dos grandes pilares de mi vida, Andrea Pérez Ortega y Genaro Angel García. Quienes día a día me educaron y formaron como persona correcta y de valores. Además de apoyarme durante cada día, noche y madrugada durante estos 5 años de universidad, así como en mis locuras y aventuras académicas y de vida. Gracias por confiar en mis habilidades, por impulsar y apoyarme en cada decisión. Mi hermana, Anahi Ángel Pérez, quien me ayudó a aprender a ser más tolerante y paciente, además de su ayuda en oratoria. A mi sobrina María Fernanda, mi niña bonita, debo de decir que si nuestro lazo familiar no inicio bien del todo poco a poco te has ganado este corazón y por eso eres la pequeña más especial de este mundo.

A mis asesores y profesores: Dr. Carlos Segura, gracias por su tiempo y paciencia para explicarme y guiarme durante el desarrollo de esta tesis. Dra Darnes Vilariño, quien a pesar de sus interminables actividades se daba el tiempo para atenderme y aclarar todas mi dudas. A mis sinodales, gracias por aceptar evaluar este trabajo. Al M.C Pedro Bello, mi tutor académico y la pregunta curiosa que me hizo a poco menos de media carrera "¿Qué estás haciendo con tu vida?". A la M.C Meliza Contreras, por aceptarme y contemplarme en sus proyectos, gracias por permitirme experimentar en el mundo de la investigación. Dr Manuel Martin muchas gracias, debo de atribuir a algunos de sus comentarios durante mi etapa como olímpica el que tomara la decisión de quedarme en la carrera. Mauricio, gracias por enseñarme a programar, es algo por lo que estaré agradecida durante toda la vida. Dr Rubin, muchas gracias por el espacio que nos brindó en el laboratorio.

Amigos míos, personas que permanecieron ahí a pesar de mi mal humor, diálogos cortantes y serios, y con pena he de mencionar que en alguna ocasión groseros. Ustedes se ganaron a pulsos mis buenos momentos, momentos de risas, de veladas de programación y sin dejar de mencionar aquellas ocasiones en las que cantábamos de felicidad. Créanme que los valoro y admiro, a cada uno de ustedes, por sus talentos y habilidades, y aplaudo por cada uno de esos buenos equipos que formamos en clases y fuera de ellas, es un gusto tenerlos como colegas: Daniel Rugerio, Edgardo Sánchez, Marlene Espinoza , Elena C. López, Julio César Pastor, Mariam Martínez, Marisol Oliváres, Rodolfo Aguirre, Alex Serrano, Sara

Ramírez, Martin García, Brisa Isabel Mendoza, Juan Ricardo García, Pablo Peralta, Alejandro Robles, Rodrigo Cuevas, Irais Monge, Martha Quintana, Juan Jésus Luna, Eduardo Contreras, y disculpen si he olvidado algún nombre, ya saben que tengo una memoria horrible de teflon, excusa por la cual espero ser disculpada.

Finalmente quiero agradecer a los directivos de la Facultad de Ciencias de la Computación, a la M.C Yalú Galicia e Ing. Alma Hernández. Un agradecimiento especial al Centro de Investigación en Matemáticas A.C, por la beca otorgada durante mi estancia en Guanajuato y mediante la que pude desarrollar este trabajo.

A todos ustedes y en especial a mis padres les dedico este trabajo.

Resumen

El grupo de problemas combinatorios es bastante amplio y es debido al gran número de aplicaciones de los problemas pertenecientes a este grupo que han sido objeto de estudio en gran variedad de trabajos. En el presente trabajo se abordan dos problemas del grupo combinatorio, el Problema de Ordenamiento Lineal y Problema de Asignación Generalizado.

En ambos problemas se implementan y analizan los algoritmos básicos presentados y desarrollados en trabajos anteriores de otros autores, incluyendo diversos mecanismos de cruces, mutación y búsquedas locales. Partiendo de estos esquemas se realizaron estudios de diversidad y se realizaron integraciones con nuevos mecanismos de gestión de diversidad, que usan internamente ciertos conceptos que surgen en el campo de optimización multi-objetivo. En base a diferentes métricas se pudo demostrar los beneficios de estos nuevos métodos, siendo capaces de mejorar significativamente los resultados reportados por otros autores.

La presente tesis se estructura en 5 capítulos. En el primer capítulo se plantean de modo detallado los objetivos y aspectos generales del trabajo, en el segundo capítulo se presentan los aspectos teóricos y antecedentes históricos de los Algoritmos Evolutivos. Posteriormente en los capítulos tercero y cuarto se realiza un análisis específico para dos problemas combinatorios, Problema de Ordenamiento Lineal y Problema de Asignación Generalizado respectivamente, para ambos casos se analizan diferentes propuestas para los mecanismos de un Algoritmo Evolutivos así como la introducción a la Gestión de Diversidad. Finalmente, en el quinto capítulo se presenta las conclusiones del trabajo así como el planteamiento de los trabajos futuros.

Índice general

Agradecimientos **III**

Resumen **V**

Lista de figuras **XI**

Lista de tablas **XIII**

Lista de algoritmos **XV**

1. Introducción **1**
1.1. Antecedentes 1
1.2. Planteamiento del problema 3
1.3. Objetivos 3
1.4. Justificación 4

2. Fundamentos Teóricos **5**
2.1. Optimización combinatoria 5
2.1.1. Problemas ejemplo 5
2.1.2. Análisis de los ejemplos 6
2.2. Metaheurísticas 8
2.2.1. Clasificación de las metaheurísticas 10
2.2.2. Cómputo evolutivo – Bases históricas 11
2.3. Algoritmos evolutivos 12
2.3.1. Representación 14
2.3.2. Función de evaluación 16
2.3.3. Tamaño de población 16
2.3.4. Mecanismos de selección de padres 17
2.3.4.1. Selección proporcional al fitness 17
2.3.4.2. Ranking lineal 18
2.3.4.3. Selección por torneo 19

2.3.5. Mutación 20
2.3.5.1. Mutación para representación binaria 20
2.3.5.2. Reestablecimiento aleatorio 21
2.3.5.3. Mutación por intercambio 21
2.3.5.4. Mutación en base a mezcla 22
2.3.5.5. Mutación basadas en inversiones 22
2.3.6. Recombinación 22
2.3.6.1. Cruce de preservación de distancia 23
2.3.6.2. Cycle crossover 23
2.3.6.3. Cruce uniforme 25
2.3.7. Mecanismos de selección de los sobrevivientes 25
2.3.7.1. Selección elitista 25
2.3.7.2. Selección por truncamiento 26
2.3.8. Calidad de los algoritmos 26

3. Problema del Ordenamiento Lineal 29
3.1. Marco teórico 29
3.2. Definición del problema 31
3.2.1. Aplicaciones 32
3.2.1.1. Agregación de preferencias individuales 32
3.2.1.2. Óptima relación tabular de descendientes 33
3.3. Solución propuesta 34
3.3.1. Representación de los individuos 34
3.3.2. Función de evaluación 35
3.3.3. Método de cruce 36
3.3.4. Mutación 36
3.3.5. Búsqueda local 37
3.3.6. Mecanismos de gestión de diversidad 39
3.3.6.1. Funciones para el cálculo de DCN 42
3.4. Resultados 44
3.4.1. Resultados generales 46
3.4.2. Pruebas estadísticas 47
3.4.3. Comportamiento de los esquemas 49

4. Problema de Asignación Generalizado 61
4.1. Marco teórico 61
4.2. Definición del problema 62
4.2.1. Aplicaciones 64
4.2.1.1. Calendarización de máquinas paralelas con costo 64
4.2.1.2. Asignación de trabajo en proyectos de desarrollo de software 64

4.3. Solución propuesta 64
4.3.1. Representación de los individuos 65
4.3.2. Función de evaluación 65
4.3.3. Método de cruce 67
4.3.4. Mutación 67
4.3.5. Búsqueda Local 69
4.3.5.1. Cambio en un gen 69
4.3.5.2. Intercambio 69
4.3.6. Mecanismos de gestión de diversidad 71
4.3.7. Resultados 72
4.3.7.1. Resultados generales 73
4.3.7.2. Pruebas estadísticas 73
4.3.7.3. Comportamiento de los esquemas 74

5. Conclusiones **79**

Bibliografía **81**

Índice de figuras

2.1. Red de conexión a) $N = 24$ b) $N = 10$ 7
2.2. Genealogía de las Metaheurísticas. 9
2.3. Clasificación general de los algoritmos de Optimización aproximada. 10
2.4. Modelo general de los algoritmos evolutivos. 13
2.5. Procesos en Fenotipo-Genotipo. 14
2.6. Representación del individuo. 15
2.7. Ejemplo del comportamiento de la selección ruleta 18
2.8. Ejemplo del comportamiento de la selección por Rankeo 19
2.9. Ejemplo del comportamiento de la selección por Torneo 20
2.10. Ejemplo de una mutación binaria 21
2.11. Ejemplo de una mutación Random 21
2.12. Ejemplo de una mutación Swap 21
2.13. Ejemplo de una mutación Scramble 22
2.14. Ejemplo de una mutación Invertida 22
2.15. Ejemplo del cruce DPX 23
2.16. Ejemplo del cruce de Ciclos 24
2.17. Ejemplo del cruce Uniforme 25

3.1. Cuatro diferentes soluciones de una instancia con $n = 6$. 32
3.2. Instancia del problema 35
3.3. Ejemplo de la inserción de la columna con índice 1 38
3.4. Efecto de la penalización, con distancia D 41
3.5. Movimiento de filas y columnas 42
3.6. Ejemplo del mecanismo de conteo por parejas invertidas 43
3.7. Ejemplo del mecanismo de suma de diferencia de posición 44
3.8. Ejecución de instancia N-be75tot_150, población 50 49
3.9. Ejecución de instancia N- tiw56r66_150, población 50 50
3.10. Ejecución de instancia N-tiw56n54_250, población 50 50
3.11. Ejecución de instancia N-be75oi_250, población 50 51
3.12. Ejecución de instancia N-be75oi_300, población 50 51

3.13. Ejecución de instancia N-stabu75_300, población 50 52
3.14. Ejecución de instancia N-t59d11xx_500, población 50 52
3.15. Ejecución de instancia N-t65w11xx_500, población 50 53
3.16. Ejecución de instancia N-t70n11xx_750, población 50 53
3.17. Ejecución de instancia N-tiw56r54_750, población 50 54
3.18. Ejecución de instancia N-be75tot_150, población 100 55
3.19. Ejecución de instancia N- tiw56r66_150, población 100 55
3.20. Ejecución de instancia N-tiw56n54_250, población 100 56
3.21. Ejecución de instancia N-be75oi_250, población 100 56
3.22. Ejecución de instancia N-be75oi_300, población 100 57
3.23. Ejecución de instancia N-stabu75_300, población 100 57
3.24. Ejecución de instancia N-t59d11xx_500, población 100 58
3.25. Ejecución de instancia N-t65w11xx_500, población 100 58
3.26. Ejecución de instancia N-t70n11xx_750, población 100 59
3.27. Ejecución de instancia N-tiw56r54_750, población 100 59

4.1. Ejemplo con solución σ admisible y μ no admisible. 63
4.2. Estructura del individuo GAP. 65
4.3. Costos tras penalización 67
4.4. Ejemplo de la sustitución de un alelo 68
4.5. Ejemplo de cálculo de DCN para GAP 72
4.6. Comportamiento de la evaluación de la instancia b05200 75
4.7. Comportamiento de la evaluación de la instancia c10100 75
4.8. Comportamiento de la evaluación de la instancia d10200 76
4.9. Comportamiento de la evaluación de la instancia e05100 76
4.10. Comportamiento de la evaluación de la instancia b20200 77
4.11. Comportamiento de la evaluación de la instancia c20200 77
4.12. Comportamiento de la evaluación de la instancia d20100 78
4.13. Comportamiento de la evaluación de la instancia e15900 78

Índice de cuadros

2.1. Ejemplo de tabla comparativa. 28

3.1. Resultados generales, población 100. Esquemas sin Gestión de diversidad. 46
3.2. Resultados generales, población 100. Esquemas con Gestión de diversidad. 47
3.3. Resultados generales, población 50. Esquemas sin Gestión de diversidad. 47
3.4. Resultados generales, población 50. Esquemas con Gestión de diversidad. 48
3.5. Comparativa entre métodos, población 100. 48
3.6. Comparativa entre métodos, población 50. 48

4.1. Tabla de resultados generales, corridas a 16 horas. 74
4.2. Tabla comparativa entre métodos. 74

Índice de algoritmos

1. Esquema general de un Algoritmo Evolutivo 13
2. Algoritmo estadístico para la comparación de conjuntos de datos . 27
3. Función de coste LOP 35
4. Mecanismo de cruce CX 36
5. Mecanismo de mutación Swap 37
6. Búsqueda local usada en plugin LOP 40
7. Esquema de selección de sobreviviente MULTI_ADAPTATIVE . 41
8. Función DCN de parejas invertidas 43
9. Función DCN de distancia en posición de los índices 44
10. Mecanismo Uniform Crossover 68
11. Mecanismo de mutación Restablecimiento Aleatorio 69
12. Mecanismo Local Search - Pares 70
13. Mecanismo Local Search - intercambio de Tareas 71
14. Mecanismo DCN para GAP 72

Capítulo 1

Introducción

1.1. Antecedentes

Se denomina optimización al proceso de buscar entre las posibles soluciones de un determinado problema cuál de ellas es la más adecuada en base a una serie de criterios cuantificables. Actualmente existen diversos métodos algorítmicos para afrontar problemas de optimización, pudiendo distinguir en un primer nivel entre métodos exactos y métodos aproximados [46]. Los métodos exactos son aquellos que aseguran que la solución obtenida es la óptima. Por su propia definición, si para una determinada situación se puede usar un método exacto, éste suele ser el más adecuado para la resolución de dicho problema. Sin embargo, los métodos exactos tienen el limitante de que generalmente, no se pueden aplicar cuando los problemas son muy complejos o de gran tamaño. Para estos últimos casos, se debe recurrir a técnicas aproximadas, cuyo propósito es encontrar una solución de la mayor calidad posible, pero que no tiene por qué ser la óptima.

Las metaheurísticas [20] son una de las técnicas de optimización aproximada más populares y han sido utilizadas en multitud de aplicaciones [45], es decir, en optimización de procesos industriales, en el diseño de redes, en la creación de métodos de minería de datos o en la predicción de las estructuras que forman las proteínas para su aplicación en la generación de fármacos. Una de las técnicas metaheurísticas que ha sido utilizada en un gran rango de aplicaciones de forma exitosa son los denominados algoritmos evolutivos [12], siendo éstos un conjunto de técnicas cuyo diseño está inspirado en la evolución natural de las especies. De esta forma, a través de la combinación de ideas inspiradas en la naturaleza y de diversos formalismos matemáticos, se dispone de un conjunto de técnicas de optimización generales, que deben ser adaptadas para la resolución de problemas concretos.

Los algoritmos evolutivos tienen sus orígenes en los años 60, y desde entonces

se han extendido en múltiples formas, a través de la incorporación de diferentes componentes [15]. Al abordar nuevos problemas mediante la aplicación de algoritmos evolutivos, una de las claves para tener éxito es ser capaz de seleccionar un conjunto de componentes y parámetros que permitan inducir un equilibrio adecuado entre la exploración y la intensificación del espacio de búsqueda correspondiente [13]. Tradicionalmente, varios de los componentes de los algoritmos evolutivos han contenido parámetros que permiten modificar el balanceo entre exploración e intensificación. Por ejemplo, la fase de selección de padres inherente a los algoritmos evolutivos, puede contener un parámetro con el que modelar el grado de intensificación [5]. En estos casos, a través del uso de técnicas de afinación de parámetros, se puede adaptar ese parámetro al problema concreto a resolver.

Las investigaciones hechas durante los últimos años han mostrado que en muchos casos buscar los mejores parámetros de estos componentes o incluso usar diversos componentes no es suficiente para inducir un equilibrio adecuado [25]. Por ello, han surgido varias técnicas que permiten modificar este equilibrio mediante la incorporación de técnicas basadas en la gestión explícita de la diversidad. Entre las técnicas de gestión de diversidad, cabe destacar las técnicas de nicho [28], las cuales están basadas en la formación y mantenimiento de subpoblaciones que están enfocadas a la exploración de distintas partes del espacio de búsqueda. Entre las técnicas de nicho más populares cabe mencionar las técnicas basadas en agrupamientos o clustering [16], las técnicas de multitud o crowding [27], las técnicas de limpieza de población o clearing [38], y las técnicas multi-objetivo basadas en diversidad [36]. Los algoritmos con técnicas de nicho han mostrado ser muy útiles al ser aplicados a problemas dinámicos, así como en optimización multi-modal, en las que se quieren obtener varios óptimos de manera simultánea. Sin embargo, cuando se tratan problemas de optimización global, los resultados obtenidos no han sido tan prometedores [28]. Para estos casos, se suele recurrir a técnicas dependientes del problema, y por tanto, poco genéricas [18].

Durante los últimos años, las técnicas multi-objetivo basadas en diversidad han ganado cierta popularidad. En este tipo de esquemas se resuelven problemas mono-objetivo tratándolos como multi-objetivo. Para ello se utilizan objetivos adicionales que son una métrica de la contribución a la diversidad aportada por cada individuo de la población. Así, de forma simultánea se intenta optimizar la función matemática asociada al problema mono-objetivo, y se intenta maximizar la diversidad. Durante el año 2014 se ha desarrollado un nuevo esquema basado en estas ideas (MULTI_DYNAMIC), que a través de la modificación de la fase de reemplazamiento, incorpora un balanceo dinámico entre la exploración y la explotación del espacio de búsqueda. En este esquema, teniendo en cuenta el criterio de parada fijado así como la cantidad de recursos ya utilizados, se adapta la importancia dada a la exploración y a la explotación. Este esquema se aplicó en primer lugar a un problema mono-objetivo (Two Dimensional Packing Problem

- 2DPP) que se diseñó para un concurso de optimización en el congreso "Genetic and Evolutionary Computation Conference" [35], uno de los congresos más prestigiosos del área de computación evolutiva. La técnica ha permitido obtener los mejores resultados conocidos para la instancia propuesta durante la competición. Posteriormente, en esta misma conferencia en la edición 2015, es presentada la aplicación al problema del Traveling Salesman Problem, obteniendo también resultados prometedores.

1.2. Planteamiento del problema

Entre las técnicas de optimización aproximada que mejores resultados han dado se encuentran los algoritmos evolutivos. En este proyecto se aplican variantes nuevas de algoritmos evolutivos a varios problemas combinatorios con el fin de estudiar las repercusiones que tienen sobre los resultados obtenidos. Debido a la aplicabilidad general de los problemas de optimización, las problemáticas sociales y/o económicas en que pueden repercutir los desarrollos de este proyecto son amplias, y por tanto las soluciones de calidad son necesarias y urgentes. Es por esto, que el proyecto resulta innovador y de calidad para el desarrollo de las ciencias en el área de cómputo evolutivo y en las aplicaciones que podrían surgir en el futuro.

1.3. Objetivos

A pesar de que la técnica MULTI_DYNAMIC ha alcanzado los mejores resultados conocidos hasta la fecha para el problema 2DPP, aún se deben hacer muchos estudios y desarrollos adicionales, ya que por un lado aún hay que demostrar la generalidad de la técnica, y por otro lado, consideramos que modificando algunas otras fases del algoritmo evolutivo, como la fase de variación, se pueden conseguir beneficios adicionales.

- Estudiar la generalidad de MULTI_DYNAMIC, así como de las nuevas ideas desarrolladas en la tesis, a través de la aplicación de los mismos a varios problemas de optimización combinatoria. En concreto, se aplica al Problema de Ordenamiento Lineal (LOP) y Problema de Asignación Generalizado (GAP).

- Estudiar para los problemas afrontados las diferentes formas de medir la diversidad y analizar las repercusiones de los mismos sobre los resultados obtenidos.

- Mejorar el estado del arte en la aplicación de algoritmos evolutivos a LOP y GAP.

1.4. Justificación

En el Centro de Investigación en Matemática A.C. se ha creado la herramienta METCO[24], que integra diferentes variantes de algoritmos evolutivos para la solución de problemas de optimización, esquemas de variación, búsquedas locales, etc... Durante el desarrollo del trabajo y con el fin de incrementar el número de esquemas implementados en dicha herramienta, se crearán plugins basados en metaheurísticas de solución a dos problemas de optimización, estos son LOP y GAP.

Los problemas elegidos son del tipo optimización combinatoria y pertenecen al grupo de NP-Duros, la resolución de estos problemas y mejora de los algoritmos existentes representan grandes avances en el área del cómputo evolutivo y en consecuencia se obtendrán grandes aplicaciones en problemáticas sociales. Entre los campos en que se aplican se destacan la arqueología, economía, teoría de grafos, traducción automática [42]; y específicamente en trabajos sobre calendarización (Scheduling), análisis de preferencias personales y en el diseño de tecnología (tarjetas, hardware) [20].

Los resultados obtenidos de los plugins adaptados a la herramienta METCO no solo tendrán resultados a corto alcance, sino que también brindaran datos para posteriores trabajos e investigaciones, impulsando las ciencias y las investigaciones en el área de optimización.

En el año de 1999 fue creada la Annual Conference on Genetic Algorithms (ICGA) y posteriormente paso a convertirse en la Genetic And Evolutionary Computation Conference. Igualmente surgió otro congreso anual importante que es el "IEEE Congress on Evolutionary Computation". Con los resultados del trabajo se pretende el desarrollo de un artículo para enviar a uno de éstos dos congresos.

Del mismo modo si se obtuvieran resultados de una calidad que permitiera superar a todo lo desarrollado hasta la fecha, se considerará el envío de los resultados a una revista nacional y/o internacional.

Capítulo 2

Fundamentos Teóricos

2.1. Optimización combinatoria

El análisis combinatorio es el estudio matemático de la disposición, agrupación, orden o la selección de objetos discretos, por lo general limitados en números. Muy recientemente, una nueva línea de investigación en el área de optimización ha ganado cada vez más importancia, es el área de problemas combinatorios. En los problemas abordados por este tipo de algoritmos el resultado objetivo no es el número de soluciones, sino cual es la mejor solución, esto con respecto al criterio de una función, que puede ser de maximización o minimización. En un aspecto formal es posible definir un problema de optimización combinatoria mediante la tupla $I = (U, P, val, extr)$ en donde:

- U es el espacio de búsqueda (espacio en que val y S están definidos).
- P es el predicado a satisfacer.
- val representa la función objetivo: $U \rightarrow \mathbb{R}$.
- $extr$ el valor de calidad (minimización o maximización)

De este modo P es satisfecho por el conjunto de soluciones $S = \{X \in U : X \text{satisface a } P\}$. Finalmente el objetivo principal es encontrar la solución del conjunto S que cumple con el valor de calidad $extr$, esto es maximización o minimización de soluciones. [40]

2.1.1. Problemas ejemplo

A continuación para ejemplificar se mostrarán algunos de los problemas de optimización con mayor presencia en el área de Ciencias de la Computación.

- **Problema de Agente Viajero.**
  El Problema del Agente Viajero (Traveling Salesman Problem, TSP) se define del siguiente modo: Dadas n ciudades y el costo Cij que se tiene al viajar de una ciudad i a otra ciudad j, se debe encontrar la ruta de costo mínimo para visitarlas todas pasando sólo una vez por cada una de ellas, y regresando a la de partida. A cada ruta se le llama tour o ciclo hamiltoniano. Este problema tiene diversas aplicaciones, en donde podemos mencionar:

  - Reparto de productos. Necesidad de rutas mínimas para reducir los costos y gastos.
  - Manufactura. Permite resolver problemas de fabricación para minimizar el número de desplazamientos al realizar una serie de perforaciones en una plancha o en un circuito impreso.

- **Problema de la Mochila.**
  El problema de la mochila (Knapsack Problem, KP) se define del siguiente modo: Una persona que tiene una mochila con cierta capacidad debe elegir de una lista de elementos cuales colocar dentro de la mochila. Cada uno de los elementos tiene un peso y aporta un beneficio. El objetivo de la persona es elegir los elementos que le permitan maximizar el beneficio sin excederse de la capacidad permitida.

  - Una posible aplicación práctica se da en un contexto de depósitos o almacenes, donde las mercancías deben ser ubicadas en estanterías de acuerdo a sus dimensiones y prioridad.

2.1.2. Análisis de los ejemplos

Recordar que el TSP tiene como objetivo planificar un recorrido que haga uso de las n ciudades sin pasar por más de una vez en alguna de ellas, es decir, si existe n ciudades se hará un recorrido de N lugares, para el primer lugar a visitar se tendrá la opción de elegir entre n ciudades, para la segunda visita solo se contará con $N-1$ ciudades, para el tercer lugar se contará con $N-2$ ciudades y así sucesivamente hasta llegar a visitar el último lugar en el que se ubicará a la ciudad restante por visitar. Cuanto mayor sea el número de ciudades, mayor va a ser el número de rutas posibles, y por lo tanto mayor será el esfuerzo requerido para calcular todas ellas. Así, el número de rutas posibles entre N ciudades va a ser igual a $(N-1)!$, lo que hace que la resolución del TSP mediante la obtención de todas las rutas posibles y comparación entre ellas sea poco factible incluso para un número de ciudades no elevado.

Tomar como ejemplo una red de 4 lugares o nodos, véase la figura 2.1, en donde por simple que parezca sería necesario hacer un cálculo de 6 combinaciones, es decir $(4-1)!$, y elevando simplemente hasta 10 el número de vértices de la red, las posibles rutas se disparan hasta más de tres millones $(10-1! = 362,880)$. Sin mencionar lo que ocurre para un caso con 100 ciudades.

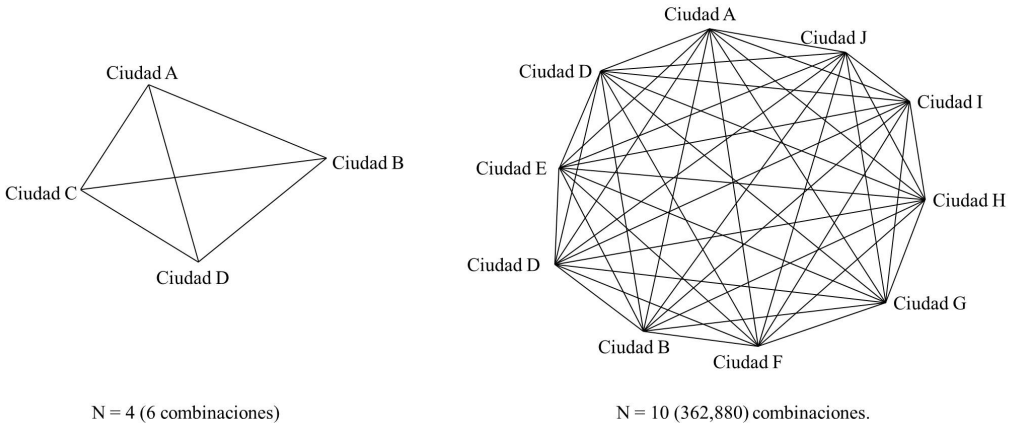


Figura 2.1: Red de conexión a) $N = 24$ b) $N = 10$

Ahora bien, para el caso del KP se observa una situación diferente en el crecimiento del número de soluciones a analizar. En este problema el objetivo es determinar los objetos a introducir en la mochila, maximizando el beneficio de la carga y no superando la capacidad de esta. Supongamos que M es el número de objetos a introducir en la mochila, para determinar una solución es necesario decidir por cada objeto si va o no a la mochila, para saber cuál es la mejor solución podríamos probar las 2^M posibilidades. El 2 se obtiene del hecho de que cada decisión es incluir o no al producto y M de la cantidad de productos.

En el caso del problema de la mochila, si contáramos con 4 productos, para saber cuál es la mejor solución podríamos probar las $2^4 = 16$ posibilidades. Sin embargo, si la cantidad de elementos por ejemplo ascendiera a 20, tendríamos que analizar $2^{20} = 1,048,576$ posibilidades.

Una vez analizado el número de soluciones candidatas en los ejemplos mostrados es importante destacar que estos problemas de optimización están catalogados como NP-Duros, lo que significa que el esfuerzo computacional que se debe llevar a cabo para encontrar una solución óptima crece de forma exponencial con la entrada del problema, para el caso del TSP y KP entradas que representan a ciudades y objetos respectivamente.

Debido a la dificultad que las técnicas convencionales presentan para resolver este tipo de problemas se han plateado otras posibilidades para afrontarlas, las cuales consisten en buscar una solución subóptima, pero en un tiempo razonable.

En algunos casos es posible encontrar incluso la solución óptima al problema. Este tipo de técnicas se dividen en dos grandes grupos: Heurísticas y Metaheurísticas, dichas técnicas serán descritas a continuación.

2.2. Metaheurísticas

Como se ha hecho notar en las secciones anteriores, obtener soluciones óptimas con métodos clásicos resulta extremadamente difícil y con un gasto importante de recursos de cómputo. En la mayoría de los casos resultan problemas totalmente intratables. De ahí que se opte y sea preferible obtener soluciones "aceptables", dichas soluciones obtenidas mediantes algoritmos heurísticos o metaheurísticos.

El concepto *Heurística* tiene su origen del antiguo griego de la palabra *heuriskein*, que significa *el arte de descubrir nuevas estrategias (reglas) para resolver problemas*, y hace referencia a un método de optimización aproximado, hecho a medida para un problema concreto.

Las heurísticas se dividen en dos categorías en base al modo en el que es generada la solución, que son las *heurísticas Constructivas* y *heurísticas de Mejora*. Las Heurísticas Constructivas construyen su solución en cada ciclo del algoritmo, es decir construyen una solución paso a paso. Son tres sus características: rápidas, fáciles de implementar y se pueden usar como punto de partida. Por otro lado las Heurísticas de Mejora, parten de una solución completa y definen transformaciones en cada una de las iteraciones del algoritmo. Este tipo de heurísticas suelen producir soluciones de calidad pero con la desventaja del tiempo que toman para ser generadas. En sus trabajos, Glover [19] clasifica las heurísticas en cuatro tipos, (1) controlled randomization, (2) learning strategies, (3) induced decomposition y (4) tabu search.

De igual modo el sufijo *meta* proviene del griego, que significa *Sobre el nivel de la metodología*. El término metaheurística fue introducido en 1986 por F. Glover en su trabajo publicado en [19], en donde se hace uso de este tipo de algoritmos para la mejora de técnicas de inteligencia artificial en problemas combinatorios. En este trabajo se define a las metaheurísticas como metodologías generales de nivel superior (plantillas) que se pueden utilizar como guía en el diseño de estrategias heurísticas subyacentes para resolver problemas de optimización específicas. Comúnmente estos métodos son más generales, lo que lo hace aplicables a una gran variedad de problemas.

A partir de la introducción del concepto en la resolución de problemas de optimización en 1945, día tras día el uso de este tipo de técnicas son más aceptadas, siendo su mayor crecimiento en los últimos 20 años, Véase figura 2.2. En 1947 G. Dantzig crea el algoritmo simplex, que es considerado como una búsqueda local

en los problemas de Programación Lineal. Entre las primeras referencias del uso de las metaheurísticas en la resolución de problemas de optimización combinatoria cabe destacar las de J. Edmonds en 1971, quien es el primero en presentar algoritmos Heurísticos Greedy.

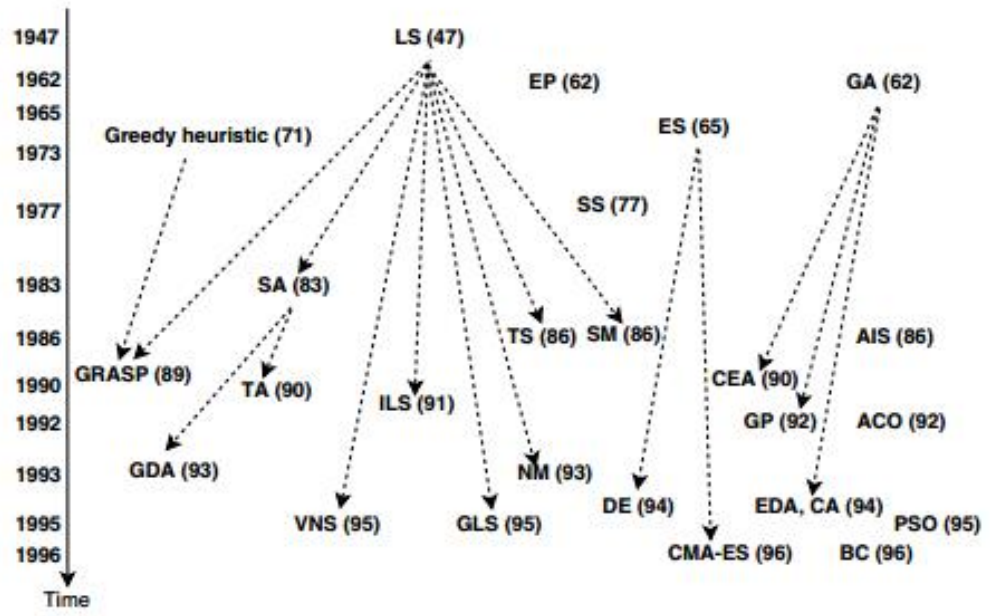


Figura 2.2: Genealogía de las Metaheurísticas.

Un aspecto importante sobre el diseño de metaheurísticas es que este tipo de algoritmos se enfrenta a dos criterios contradictorios, el primero, la exploración del espacio de búsqueda (*Diversificación*) y el segundo, la explotación de las mejores soluciones encontradas (*Intensificación*). En la intensificación las regiones prometedoras son exploradas más a fondo con la esperanza de encontrar mejores soluciones. Mientras que en la diversificación se visitan regiones no exploradas para tratar de identificar otras regiones prometedoras.

A modo de resumen, los algoritmos Heurísticos y Metaheurísticos pertenecen al grupo de Optimización aproximada; las Heurísticas son herramientas ad-hoc a problemas, mientras que las Metaheurísticas son técnicas generales adaptables a cualquier problema de optimización (Vease la figura 2.3).

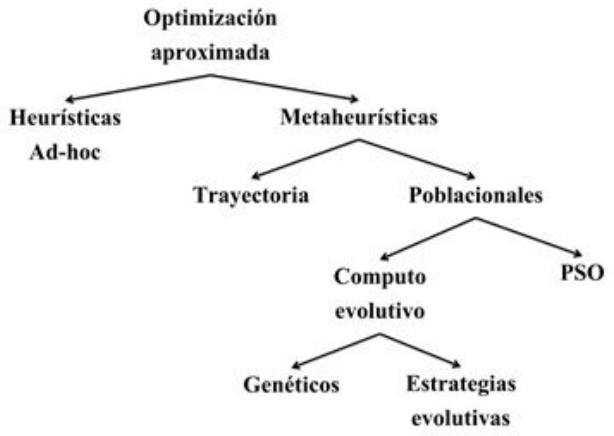


Figura 2.3: Clasificación general de los algoritmos de Optimización aproximada.

2.2.1. Clasificación de las metaheurísticas

Debido a las diversas características de este tipo de algoritmos las clasificaciones de estos son diversas.

- **Inspirados y no inspirados en la naturaleza.** Existen algoritmos cuya concepción y desarrollo fueron basados en procesos naturales como los algoritmos evolutivos y los sistemas inmunológicos artificiales de la biología; otros ejemplos incluyen la colonia de hormigas y abejas. Por otro lado algunos otros algoritmos como el Simulated Annealing son basados en procesos físicos.

- **Métodos con o sin memoria.** Algunos algoritmos como GRASP y Simulated Annealing carecen de memoria, es decir durante la búsqueda no almacenan información. A diferencia de algunos algoritmos como Tabu search, que retienen información a largo y corto plazo, y en base esa memoria modifican su comportamiento.

- **Determinista frente estocástico.** Algunos algoritmos como las búsquedas locales y búsqueda tabú son consideradas metaheurísticas deterministas debido a que resuelven un problema de optimización mediante la toma de decisiones deterministas. En metaheurísticas estocásticas, se aplican algunas reglas al azar durante la búsqueda (por ejemplo, recocido simulado, algoritmos evolutivos).

- **Búsqueda basada en población frente a búsqueda basada en una solución única o trayectoria.** Los algoritmos basados en trayectoria o solución única (por ejemplo, la búsqueda local, recocido simulado) manipulan

y transforma una única solución durante la búsqueda, mientras que en los algoritmos basados en población (por ejemplo, enjambre de partículas, algoritmos evolutivos) se mantiene toda la población o conjunto de soluciones.

2.2.2. Cómputo evolutivo – Bases históricas

Como su nombre lo indica el cómputo evolutivo basa sus teorías en el proceso natural de evolución y las teorías Darwinianas. Es en 1948 cuando comienzan los experimentos de optimización mediante "evolución y recombinación". Posteriormente, a inicios de los 60 el concepto comienza a tomar fuerza y son tres los principales trabajo que da base a esta teoría.

- Fogel, Owens y Wals y la introducción del concepto de programación evolutiva.
- Holland acuña el concepto de Algoritmo Genético.
- Rechenberg, Schwefel y Bienest desarrollan las estrategias evolutivas.

Es finalmente a inicios de los 80 y durante los 90 cuando esta área de investigación ve su máximo esplendor con la unificación de las diferentes corrientes y la creación de diversos foros científicos y congresos internacionales, tales como la International Conference on Genetic Algorithms (ICGA), Annual Conference on Genetic Programming que posteriormente se convierte en el congreso anual de mayor relevancia en el área Genetic and Evolutionary Computation Conference (GECCO).

En los algoritmos evolutivos las soluciones candidatas son denominadas *individuos*, un individuo es una única e irrepetible entidad de Fenotipo, dicha unidad es representada por el Genotipo mediante las unidades funcionales llamadas Genes. Existe una dependencia irrevocable entre el Fenotipo y Genotipo de un individuo, lo que ocasiona que cualquier modificación en el Genotipo produzca una modificación en el Fenotipo de un individuo. Dichas modificaciones serán producto de mutaciones o recombinaciones de los genes por reproducción sexual, estas definiciones son usadas en las teorías de la evolución de Darwin, en donde se explica la diversidad de las especies, y se señala que la selección natural favorece a los individuos con características más competitivas, debido a que en el ambiente natural los individuos deben competir por un recurso produciendo que el sobreviviente sea el más apto, es decir el que sea capaz de obtener el recurso. En pocas palabras hace referencia a la supervivencia de los individuos que mejor se adapten a las características del ambiente. Otro aspecto en las teorías de Darwin hace referencia a las variaciones del genotipo entre generaciones, desencadenando variaciones en el fenotipo del individuo entre los cambios generacionales. Lo

anterior representa de modo general la base de los algoritmos Bioinspirados en los que se señala que los individuos que sobreviven son los más aptos y fuertes, y en consecuencia son los que se reproducen y transmiten (heredan) su material genético a la siguiente generación, teniendo mayor probabilidad de generar cada vez individuos más aptos.

A finales del siglo 19, Mendel es el primero en estudiar y entender la herencia en los organismos, que es basada en el ADN – Acido Desoxirribonucleico – y los cambios en el código genético por cada transición generacional. Con sus estudios sentó las bases en la genética moderna. Estas ideas se usan para inspirar el diseño de algoritmos evolutivos.

Es debido al modo tan eficiente en el que la naturaleza hace su trabajo que estos han sido de inspiración para los investigadores de diversas áreas de ahí el desarrollo de los algoritmos evolutivos.

2.3. Algoritmos evolutivos

La idea principal de los algoritmos evolutivos es mostrada en algoritmo 1, y es posible resumirla mediante la siguiente idea: Dada una población de individuos, la presión del medio ambiente provoca la selección natural de los individuos, dicha presión causa un crecimiento en el *fitness* de los individuos.

El conjunto inicial de candidatos son obtenidos mediante mecanismos aleatorios, elementos que constituyen la función de dominio, para cada uno de estos elementos es posible obtener el valor *fitness* y posteriormente en base a dicho valor se selecciona a los mejores candidatos para semilla de la siguiente generación, esto con el fin de maximizar la función de calidad de determinado problema. Dichos elementos son llamados operadores padre a los cuales se les aplica una operación de recombinación y/o mutación.

- El operador de recombinación hace uso de dos de los individuos seleccionados para crear un individuo nuevo.

- La mutación hace uso de uno de los elementos seleccionados para crear un nuevo individuo.

Una vez aplicados los operadores mencionados se tiene una nueva generación de individuos, denominada (*offspring*). Finalmente, se ejecutan diversos mecanismos de selección, en donde los individuos compiten con los ya existentes con el fin de conformar la nueva generación de individuos. El proceso mencionado es iterativo hasta cumplir con cierto criterio de parada. En secciones posteriores se profundiza en cada uno de estos elementos.

Algorithm 1 Esquema general de un Algoritmo Evolutivo

Inicializar *Población* con soluciones candidatas aleatorias;
Evaluar cada candidato. .

1: **while** CONDICIÓN_TERMINACIÓN es insatisfecha **do**
2: SELECCIONAR padres
3: RECOMBINAR un par de padres
4: MUTAR el offspring resultante.
5: EVALUAR los elementos del offspring
6: SELECCIONAR individuos para la siguiente generación.
7: **end while**

El proceso de evolución hace que la población se adapte de modo creciente y gradual a las características del medio ambiente, lo que tiene como consecuencia que los individuos con mayor *fitness* (más aptos) sean los que sobrevivan. El esquema general de evolución es presentado en la figura 2.4.

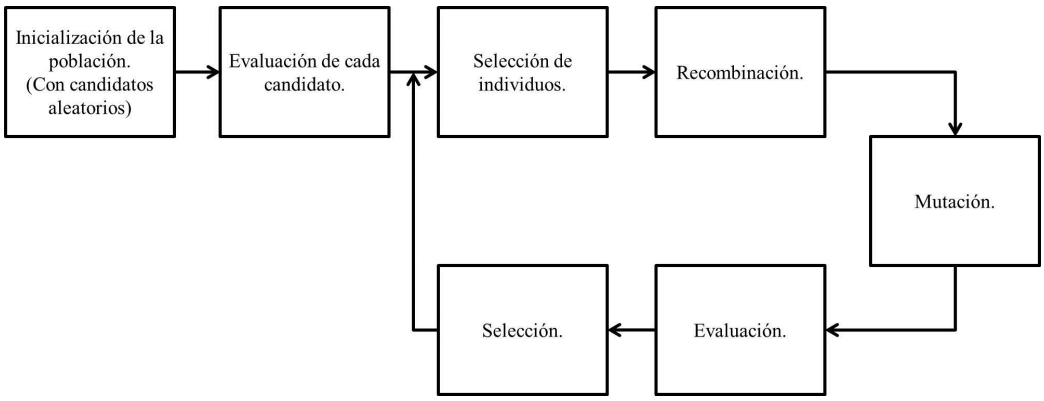


Figura 2.4: Modelo general de los algoritmos evolutivos.

Los algoritmos evolutivos no solo están determinados por el algoritmo general que presentan, es decir, existen conceptos importantes que definen el éxito o fracaso de estos, entre ellos se encuentran:

- Representación de las soluciones candidatas.
- Función de evaluación.
- Tamaño de la población.
- Mecanismos de selección de los padres.
- Operadores de variación (Recombinación y mutación).

- Mecanismos de selección de los sobrevivientes.

En las siguientes secciones de este trabajo se detalla cada uno de estos conceptos.

2.3.1. Representación

El primer paso en la definición de una Algoritmo evolutivo consiste en la creación de un vínculo entre *el mundo real* y *el mundo del Algoritmo evolutivo*, es decir crear un puente entre el contexto original del problema y el espacio de solución del problema, donde la evolución toma forma, este paso es comúnmente llamado *Representación*. Al igual que con los individuos reales, los individuos de los algoritmos evolutivos tiene dos tipos de características que los conforman, las primeras son los fenotipos, aquello que se manifiesta en el entorno y es evidente a primera vista, es decir los detalles y descripción del problema, el segundo tipo de características es la codificación del individuo, es decir el genotipo. Para crear este puente o vínculo existen dos procesos, el primero de ellos es el proceso del paso del fenotipo al genotipo, es decir la *codificación*, el segundo proceso es el inverso al anteriormente mencionado, y recibe el nombre de *decodificación*, véase la figura 2.5.

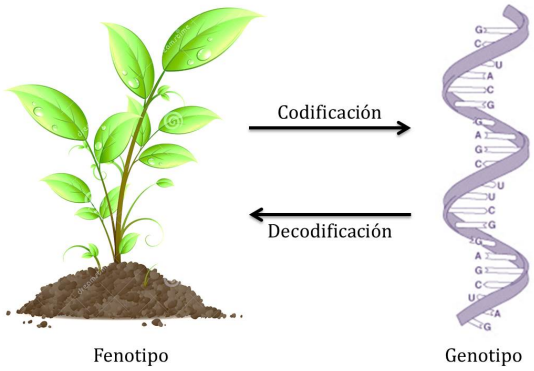


Figura 2.5: Procesos en Fenotipo-Genotipo.

La terminología común utiliza muchos sinónimos para nombrar los elementos de estos dos espacios (Problema original y el Algoritmo Evolutivo). En el lado del contexto del problema original, *solución candidato, fenotipo, e individuo* se utilizan para denotar puntos del espacio de posibles soluciones. Este espacio se llama comúnmente el *espacio fenotipo*. En el lado del algoritmo evolutivo, *genotipo, cromosoma, e individuo* puede ser utilizado para puntos en el espacio donde la búsqueda evolutiva realmente tiene lugar. Este espacio se denomina a menudo el *espacio genotipo*. También hay muchos términos sinónimos para los elementos de los individuos. Cada una de las posiciones a las que se le debe dar valor

para conformar un individuo comúnmente se llama *una variable, un lugar, una posición, o -en una terminología de biología - un gen*. Cada uno de los posibles valores que se pueden asignar a un gen puede ser llamado *valor o alelo*, véase figura 2.6.

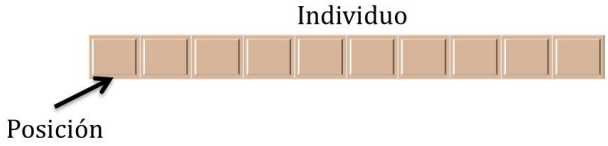


Figura 2.6: Representación del individuo.

Cuando se elige la representación del problema es importante seleccionar la representación correcta, generalmente lo conveniente es elegir la representación más natural y fácil de manejar durante la búsqueda de la solución del problema representado. A continuación se describen las representaciones más usuales en los algoritmos evolutivos.

- **Representación binaria.**
  Este tipo de representación es considerada una de las más simples y fáciles usadas para formar individuos, generalmente se hace uso en problemas que implican decisiones Booleanas, pero esto no es obligatorio.
  Ejemplo. Esta representación se puede observar en los números enteros, en donde en ocasiones resulta más sencillo operar con su representación binaria, aunque actualmente para estos últimos casos se suele usar la representación entera.

- **Representación entera.**
  Es preferible usar este tipo de representación en aquellos casos o problemas en los cuales es necesario usar un rango de valores enteros.
  Ejemplo. Supongamos que deseamos dibujar una ruta en una cuadricula, y cada uno de los movimientos son restringidos a {avanza, norte, sur, este, oeste}, valores que son mapeados a: {1,2,3,4,5}, como ejemplo en este caso, donde se obtiene una representación sencilla haciendo uso de números enteros.

- **Representación flotante.**
  Este tipo de representación es usada en problemas que requieren exactitud, o sus cadenas requieren de valores reales.

- **Representación mediante permutaciones.**
  La naturaleza de muchos problemas conlleva decidir una secuencia en que

una serie de eventos debe ocurrir, lo más conveniente para este tipo de problemas es optar por una representación dada por una permutación de enteros, en consecuencia los elementos dentro de los genes ocurren exactamente una vez. **Ejemplo.** Un ejemplo claro de esta representación se observa en el TSP, problema en que un agente debe recorrer todas las ciudades sin repetir alguna, si el conjunto de soluciones es dado por el conjunto C=1,2,3,4,5, y el orden de las ciudades en el individuo representa el orden en el que debe ir a una ciudad las soluciones estarán dadas por permutaciones similares a [3,4,1,5,2] y [2,4,1,3,5,].

2.3.2. Función de evaluación

El rol de la función de evaluación es representar los requerimientos de adaptación así como facilitar la selección de individuos. Para la perspectiva de solución de problemas, dicha función representa el valor que debe ser maximizado minimizado.

Técnicamente, esta función es usada para evaluar la calidad del fenotipo. De ahí que comúnmente sea llamada *función fitness*, función que se asociará con maximización de valores. En el caso de ser una función que se debe minimizar, comúnmente recibe el nombre de *función costo*. Aunque no resulta incorrecto referirse en ambos casos como *función fitness*.

Un ejemplo sencillo es observable con la función x^2 en enteros la cual se desea maximizar, el *fitness* del genotipo 10010 puede ser definido como el cuadrado del correspondiente fenotipo: $18^2 = 324$

2.3.3. Tamaño de población

La población es el conjunto que contiene a los elementos o individuos que representan a las posibles soluciones (genotipos). Además de representar la unidad de evolución. Es a partir de este conjunto de donde se obtienen los individuos que tomaran el rol de padres y por tanto serán usados en los métodos de cruce para la creación de nuevos individuos, que posteriormente sufrirán cambios mediante métodos de mutación, y finalmente tendrán la oportunidad de ser elegidos como individuos de la nueva población.

La *Diversidad* hace referencia a la similitud entre soluciones, individuos, fenotipos, genotipos y/o valores fitness incluidos en dicha población.

Generalmente, el tamaño de la población suele variar de problema a problema.

La convergencia prematura de un algoritmo evolutivo surge cuando los genes de los individuos de alta calidad dominan rápidamente la población, limitando a converger a un óptimo local. La convergencia prematura es generalmente debido

a la pérdida de la diversidad dentro de la población. Esta pérdida puede ser causada por la presión de selección, o la ejecución de los mecanismos de cruce, y un entorno de parámetros de evolución pobre. Este fenómeno se produce cuando la población de un algoritmo genético alcanza un estado subóptimo que los operadores genéticos ya no pueden producir descendencia con un mejor desempeño que sus padres. Para evitar la convergencia prematura, en un algoritmo genético es imprescindible preservar la diversidad de la población durante la evolución. [1]

2.3.4. Mecanismos de selección de padres

Los mecanismos de *selección de padres* tienen el objetivo de elegir de entre la población total a individuos, generalmente basados en su calidad, para ser usados en la creación de los nuevos individuos. Un individuo es *padre* si ha sido elegido bajo las normas de algunos algoritmos, que serán descritos más adelante; dichos algoritmos son generalmente probabilísticos.

Antes de introducir a cada uno de los métodos de selección es importante describir el concepto de *Presión de selección*, se entiende como presión de selección a la probabilidad de elegir a los mejores individuos, en comparación con el promedio de probabilidad de elección de toda la población. En general se refiere al "favoritismo" que tiene el método de selección de elegir los mejores individuos sobre individuos promedio. Sus características son las siguientes:

- Si la presión de selección es demasiado baja entonces la convergencia a la solución óptima es lenta.
- Si la presión de selección es muy alta entonces la diversidad en la población puede perderse de manera muy rápida, provocando un estancamiento en óptimos locales.

2.3.4.1. Selección proporcional al fitness

La idea principal de este método de selección se basa en que la probabilidad de selección sea proporcional a su base a su fitness, esto es: la probabilidad de que un individuo f_i sea seleccionado está dado por la ecuación 2.1, en pocas palabras lo que dice es que la probabilidad de selección depende completamente del valor de fitness del individuo comparado con el total de valores de fitness de la población.

$$\frac{f_i}{\sum_{j=1}^{\mu} f_j} \tag{2.1}$$

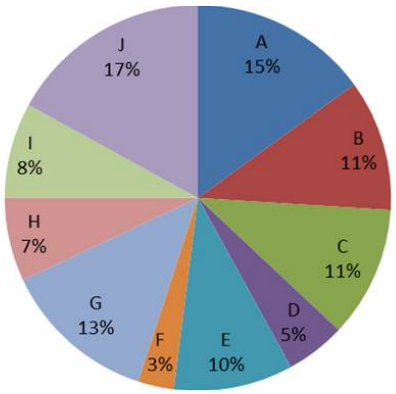
Este método fue introducido por J.H. Holland [21], a partir de la introducción del método, se han detectado ciertos problemas en este mecanismo de selección, entre ellos podemos encontrar:

- En algunos problemas los individuos que presentan un mejor fitness que el resto de la población se convierten rápidamente en la población total existente, este fenómeno es denominado: *Convergencia Prematura*
- Cuando los valores de fitness de los individuos son muy grandes con respecto a las diferencias entre ellos, casi no hay presión de selección, es decir, todos los individuos se seleccionan prácticamente con la misma probabilidad.

Para ver un ejemplo del comportamiento de este método de selección véase la imagen 2.7.

Individuo	Fitness	Probabilidad
A	84	0,15
B	58	0,11
C	61	0,11
D	25	0,05
E	57	0,1
F	16	0,03
G	71	0,13
H	37	0,07
I	44	0,08
J	96	0,17
Total	549	1

Tabla con individuos aleatorios y probabilidad de selección



Gráfica pastel con las probabilidades de selección de cada individuo

Figura 2.7: Ejemplo del comportamiento de la selección ruleta

2.3.4.2. Ranking lineal

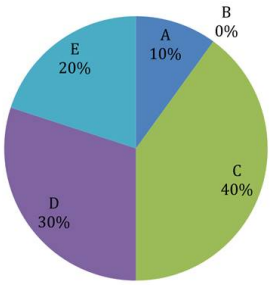
Este método es también basado en el fitness obtenido, el proceso en general es: Ordenar los μ individuos en base al fitness, de este modo es asignado un Rank id (asignando 0 al primer elemento y $\mu - 1$ al último elemento), posteriormente para calcular la probabilidad de elección de cada individuo se calcula mediante la ecuación 2.2. Se hace notar que el valor de s generalmente se encuentra en un rango de $1{,}0 < s \leq 2{,}0$

$$P(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)} \tag{2.2}$$

En la figura 2.8 se muestra un ejemplo de la aplicación en fitness obtenidos aleatoriamente.

Individuo	Fitness	Rank id	P(i) (s=2)
A	13	1	0,1
B	3	0	0
C	25	4	0,4
D	18	3	0,3
E	16	2	0,2
Total	75		1

Tabla con individuos aleatorios y probabilidad de selección



Gráfica pastel con las probabilidades de selección de cada individuo

Figura 2.8: Ejemplo del comportamiento de la selección por Rankeo

2.3.4.3. Selección por torneo

Este método como su nombre lo dice es basado en el proceso de torneos, en donde se mide la habilidad (fitness) de los objetos a "enfrentar". De modo general el algoritmo de este método es el siguiente: se seleccionan k individuos aleatoriamente ($2 \leq k \leq Poblacin\_Total$), posteriormente es elegido el individuo con mejor fitness (depende totalmente del tipo de problema: Maximización o minimización), este individuo formará parte del grupo de padres. El proceso mencionado se repite λ veces hasta obtener un conjunto de λ individuos (padres). Finalmente las parejas de individuos serán seleccionadas generalmente emparejando de forma aleatoria a los miembros del grupo anterior.

En la figura 2.9 se muestra un ejemplo con 8 individuos (a), posteriormente, en la subfigura *(b)* se muestra una agrupación con $k = 3$ y $\lambda = 4$, en *(c)* se aprecian los individuos seleccionados mediante una función de maximización, para finalmente hacer la elección de los $padres_i$, $i = 0, .., \lambda - 1$, mediante las funciones 2.3 y 2.4.

$$Padre_1(i) = i * 2 \tag{2.3}$$

$$Padre_2(i) = (i * 2) + 1 \tag{2.4}$$

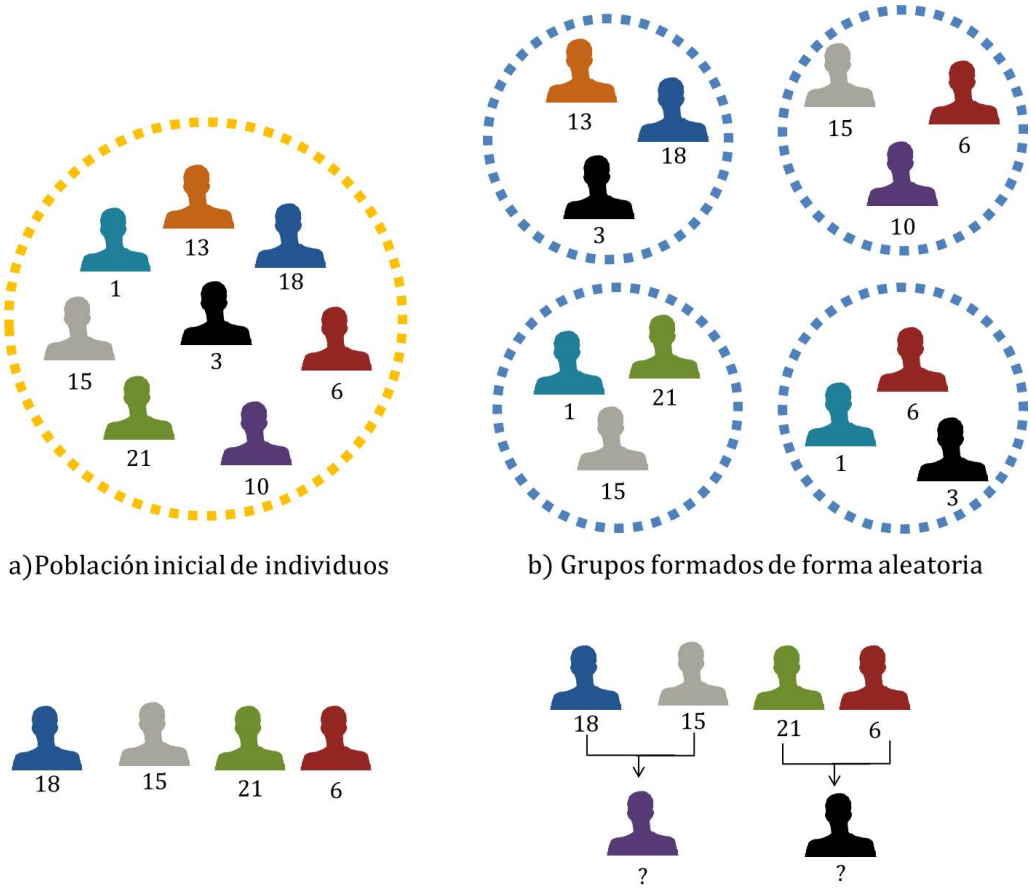


Figura 2.9: Ejemplo del comportamiento de la selección por Torneo

2.3.5. Mutación

La mutación es el operador de variación unario, debido a que para ser usado solo es necesario contar con un individuo. Este método es estocástico ya que depende de elecciones aleatorias. Existen diferentes métodos de mutación, que varían dependiendo del tipo de representación usada (binaria, flotante, entera) a continuación se describen algunas de las más usadas [12]

2.3.5.1. Mutación para representación binaria

Este tipo de representación se realiza, como su nombre lo dice, para cadenas binarias. En general el algoritmo de este método es el siguiente: Se eligen aleatoriamente x números ($1 \leq x \leq Longitud\_Cadena$), el nuevo valor para cada uno

de los elementos de la cadena se calcula mediante la ecuación 2.5, Un ejemplo de este método de mutación se puede observar en la figura 2.10, en donde los genes que se cambian son el 4º, 6º, 7º, 9º.

En cuanto a la forma de implementación el número de elementos mutados puede ser fijo o se puede establecer una probabilidad de mutación que se aplica a cada gen.

$$String_i(t) = \begin{cases} 0, & t = 1 \\ 1, & t = 0 \end{cases} \tag{2.5}$$

Figura 2.10: Ejemplo de una mutación binaria

2.3.5.2. Reestablecimiento aleatorio

Este mecanismo es considerado como la extensión del cambio de bit en la representación binaria. En base a una probabilidad P_i se elige para cada gen un elemento aleatorio del conjunto de valores admitidos (alelos). Este mecanismo es el más adecuado cuando el individuo está codificado mediante atributos enteros. Es importante mencionar que cada uno de los alelos tiene la misma probabilidad de ser elegido como el nuevo valor de un gen. Un ejemplo se observa en la figura 2.11, en donde con una probabilidad de 20 % se decide si el alelo contenido actualmente en el gen_i será cambiado por algún otro elemento del rango admisible, es decir $\{1, 2, \ldots, 10\}$. En este ejemplo se observan valores generados aleatoriamente, y aquellos genes en el que el porcentaje de mutación se cumple es observable el cambio, es decir los genes 4º y 8º.

Figura 2.11: Ejemplo de una mutación Random

2.3.5.3. Mutación por intercambio

Este operador trabaja bajo la selección de dos posiciones aleatoriamente (genes) en la cadena (genotipo), posteriormente los elementos que se encuentran en dichas posiciones son cambiados. En la figura 2.12 es posible observar un ejemplo de este método de mutación.

Figura 2.12: Ejemplo de una mutación Swap

2.3.5.4. Mutación en base a mezcla

Este es el primer método que trabaja con subconjuntos de la cadena, eligiendo dos elementos de la cadenas de modo aleatorio, que serán el inicio y el fin, respectivamente, del rango de elementos a intercambiar, posteriormente los elementos de este rango son intercambios aleatoriamente entre ellos. Véase la figura 2.13 en donde se presenta un ejemplo, cuyo rango de elementos se encuentra entre el índice 4 y 8, los elementos de dicho rango son ordenados de modo aleatorio.

Figura 2.13: Ejemplo de una mutación Scramble

2.3.5.5. Mutación basadas en inversiones

Al igual que el método anterior se hace uso de un subconjunto de elementos de la subcadena, posteriormente los elementos dentro de las posiciones del rango elegido son invertidas de acuerdo al orden en el que aparecen originalmente. Véase la figura 2.14 en la cual se observa un ejemplo de este método.

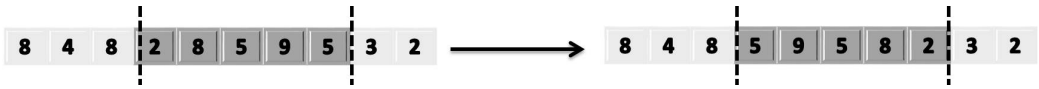


Figura 2.14: Ejemplo de una mutación Invertida

2.3.6. Recombinación

El proceso de recombinación es el proceso en donde uno o dos individuos son creados mediante la información de dos individuos ya existentes. Es un operador binario también llamado Crossover, es considerada por muchos uno de los conceptos más importantes para los algoritmos evolutivos [12]. Similarmente a la mutación la recombinación es un método estocástico que elige de modo aleatorio las partes de los padres que serán usadas y combinadas para crear el nuevo individuo. Este método se basa en la teoría de herencia en donde se menciona que los padres heredan ciertas características que se espera sean las mejores para obtener un individuo más hábil y por tanto con mayor probabilidad de sobrevivir.

En los siguientes apartados se describen los métodos más usuales en el desarrollo de las técnicas evolutivas [7][12].

2.3.6.1. Cruce de preservación de distancia

El mecanismo de *Cruce de Preservación de Distancia* (o Distance Preserving Crossover, DPX), es un método específico para representaciones basadas en permutaciones, como su nombre lo dice el objetivo de este mecanismo es preservar la distancia existente entre el espacio de soluciones, por este motivo el nuevo individuo es creado a una distancia igual de ambos padres.

En el primer paso de este mecanismo ubica los elementos repetidos en los padre en las mimas posiciones dentro del nuevo individuo (hijo), y posteriormente ubica el resto de los elementos en posiciones aleatorias mientras la posición esté disponible, siempre vigilando tres aspectos sumamente importantes.

- El elemento a insertar aún no existe en el individuo.
- En el resto de posiciones no aparece un valor que sea igual al de uno de los padres.

. De esta forma se crean individuos que se encuentran a una distancia similar de ambos padres. Un ejemplo se puede observar en la imagen 2.15, en donde se marcan los elementos repetidos en ambos padres y el resto den orden aleatorio, pero diferente al de los padres.

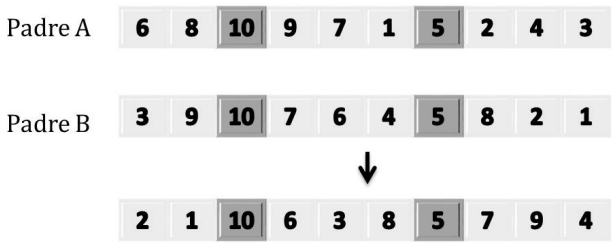


Figura 2.15: Ejemplo del cruce DPX

2.3.6.2. Cycle crossover

En [29] se hace referencia a este método como CX, y su objetivo principal es preservar la mayor cantidad de información de los padres como sea posible. De modo general, el algoritmo trabaja mediante la asignación de elementos repetidos en ambos padres y posteriormente la división de los elementos en ciclos. La construcción de ciclos es mediante las siguientes reglas:

- Inicia con una posición aleatoria libre dentro del nuevo individuo, toma el elemento de esa posición, de modo aleatorio de entre el Padre A y el Padre B, al padre que se elige se denomina $Padre_r$ y el padre no elegido $Padre_n$.

- Toma el elemento (alelo) del $Padre_n$.
- Va a la posición con el mismo alelo en el $Padre_r$.
- Se agrega este alelo al nuevo individuo.
- Se repite el proceso hasta llegar al primer alelo usado.

Véase 2.16 en donde se muestra un ejemplo con individuos generados aleatoriamente, y como se observa existen dos Alelos repetidos en los individuos (10 y 5) y dos ciclos de datos, el primero formado a partir del Alelo 2 del Padre A, y el segundo ciclo a partir del Alelo 8 del Padre B.

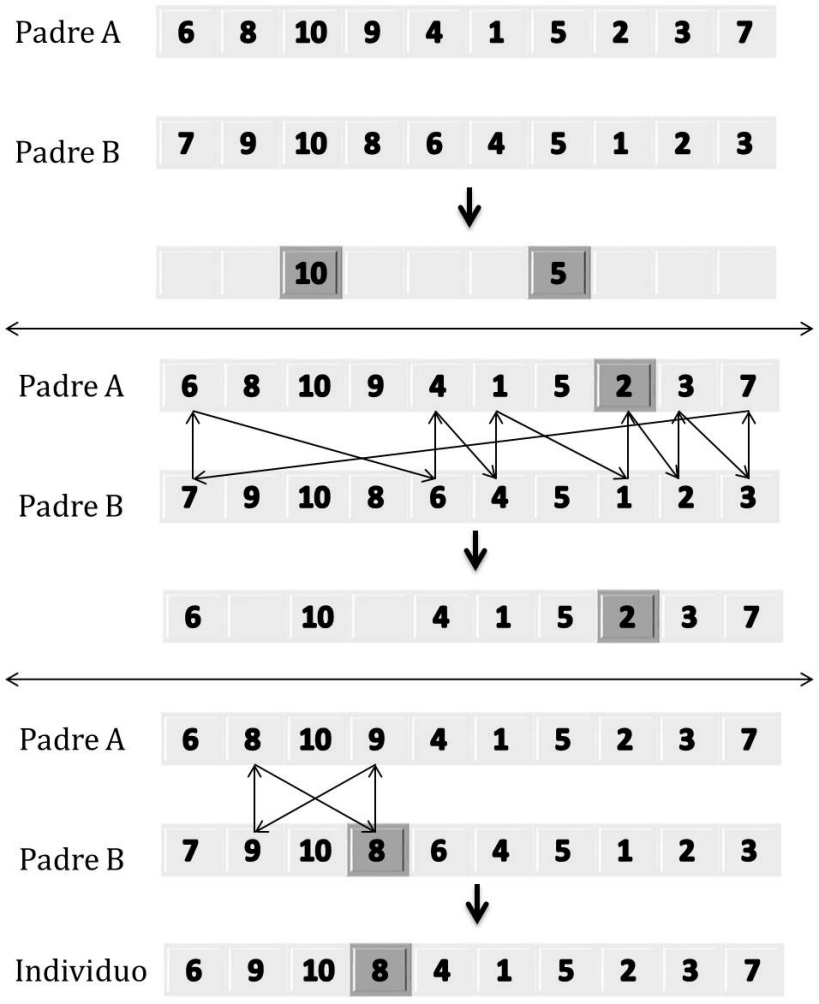


Figura 2.16: Ejemplo del cruce de Ciclos

2.3.6.3. Cruce uniforme

Este tipo de cruce tiene aplicaciones generalmente en individuos que no representan una Permutación. A diferencia de otro métodos, el Uniform Crossover genera dos individuos, El hijo A e hijo B conservan los alelos de los Padres A y B respectivamente, esto en base al valor de probabilidad P (Generalmente $P = 0{,}5$).

La herencia de elementos de los padres en los hijos depende totalmente de la Probabilidad P, véase el ejemplo 2.17 en donde en el caso a la probabilidad de cambio es de $20\,\%$, lo que representa la probabilidad de conservación de los individuos padres en un $80\,\%$, mientras que en el caso b con probabilidad de cambio de $50\,\%$ y $50\,\%$ de probabilidad de conservación de padres.

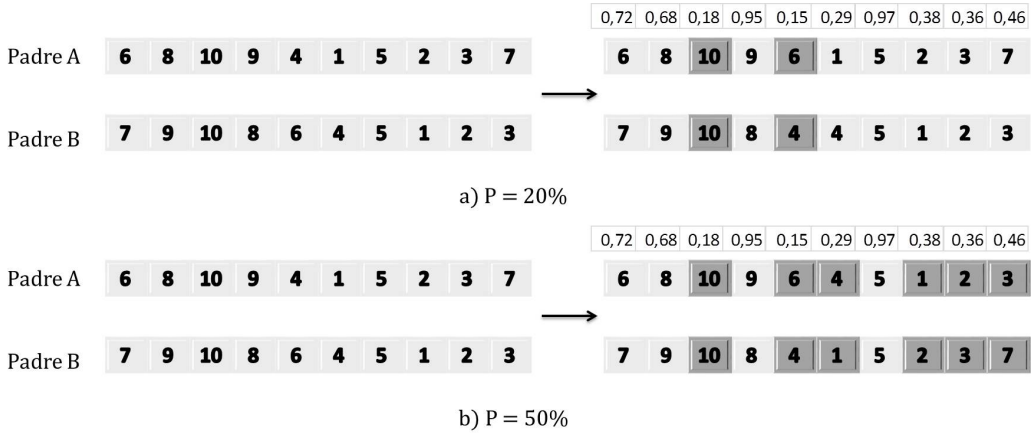


Figura 2.17: Ejemplo del cruce Uniforme

2.3.7. Mecanismos de selección de los sobrevivientes

El objetivo de los *Mecanismos de selección de sobrevivientes* es distinguir a los individuos en base a su calidad, proceso similar al proceso de selección de padres pero en una etapa diferente en el ciclo del Algoritmo evolutivo, en pocas palabras este mecanismo realiza la decisión de que individuos sobrevivirán en la siguiente generación. Este mecanismo es también llamado estrategia de reemplazamiento.

En resumen, podemos decir que el principal objetivo de las estrategías de reemplazamiento es la reducción de un grupo μ de individuos de la generación anterior y λ descendientes en un conjunto de μ individuos para la siguiente generación.

2.3.7.1. Selección elitista

La Real Academia Española define a la *Élite* como un grupo minoritario selecto. Generalmente, se habla de élite como sinónimo de un grupo elegido, escogido,

eminente o distinguido. En los algoritmos evolutivos esta definición no cambia, ya que un mecanismo de selección elitista hace referencia al método que conserva al mejor o los mejores individuos de la población anterior. Este método es de los más usados en los Algoritmos Evolutivos, además de ser bastante sencillo y su realización consiste en 2 etapas.

- Se selecciona una Elite de r individuos de la población anterior y se incorporan de modo directo en la nueva población. Generalmente los r elementos se seleccionan de modo directo (eligiendo los mejores individuos de la población) o estocástico.

- Los $n - r$ individuos faltantes son muestreados de entre la población existente y los nuevos individuos. En donde n es el tamaño de la población y comúnmente $1 \leq r \leq 2$.

2.3.7.2. Selección por truncamiento

En general este algoritmo ordena a los individuos de la población original y los nuevos individuos mediante las características de su fitness, para posteriormente conservar a los n mejores individuos, que se convertirán en los nuevos individuos de la población, en donde n es el tamaño de la población. Este mecanismo de selección tiene la característica principal de impulsar una convergencia muy rápida. El problema es que en muchas ocasiones se sufre de convergencia prematura y por tanto un estancamiento en óptimos locales, de ahí su ineficiencia, especialmente en problemas complejos o cuando se van a hacer ejecuciones a largo plazo.

2.3.8. Calidad de los algoritmos

Es importante medir la calidad de los algoritmos desarrollados, es por esto que es necesario realizar diversas pruebas estadísticas. Dicho conjunto de pruebas son realizadas sobre los datos obtenidos de las corridas de cada uno de los esquemas propuestos en este trabajo.

El conjunto de pruebas aplicadas en este trabajo son las descritas a continuación.

- **Test Shapiro-Wilk.** Prueba estadística para determinar la normalidad de un conjunto de datos y determinar si este sigue una distribución Gaussiana.

- **Test Levene.** Test usado para medir las varianzas poblacionales y determinar si son o no iguales (homogeneidad de varianzas entre un número determinado de muestras).

- **Test ANOVA.** Análisis de Varianza o comúnmente llamado ANOVA (por sus siglas en inglés). Prueba que realiza un análisis de las diferencias de entre las medias de dos conjunto de elementos.
- **Test Welch.** Mecanismo para evaluar la igualdad entre las medias de dos poblaciones, es usado para conjuntos de datos que poseen una distribución normal, pero sus varianzas son diferentes.
- **Test Kruskal-Wallis.** Mecanismo estadístico similar al test ANOVA, con la diferencia de que este mecanismo no sume normalidad en los datos.

El algoritmo 2 es el usado para el análisis estadístico de los conjuntos de datos resultado, dichos conjuntos de datos son obtenidos tras ejecutar corridas con los diferentes esquemas desarrollados. Que el algoritmo retorne el conjunto A tras la entrada de los conjuntos A y B significa que las diferencias entre ellos son estadísticamente significativas, y que la media y la mediana obtenida por A son más altos que la media y la mediana alcanzados por B.

Algorithm 2 Algoritmo estadístico para la comparación de conjuntos de datos

Entrada $Conjunto_A$ y $Conjunto_B$ con mismo número de elementos.
Salida C, Conjunto estadísticamente mejor.

```
1: if Shapiro-Wilk(Conjunto_A, Conjunto_B) = Distribución_Gaussiana) then
2:     if Levene(Conjunto_A, Conjunto_B) = Varianza_Homogénea then
3:         C ← ANOVA(Conjunto_A, Conjunto_B)
4:     else
5:         C ← Welch(Conjunto_A, Conjunto_B)
6:     end if
7: else
8:     C ← Kruskal-Wallis(Conjunto_A, Conjunto_B)
9: end if
10: return C
```

El conjunto de resultados obtenidos posterior a la evaluación con el algoritmo estadístico 2 son almacenados en una tabla del estilo de la tabla 2.1. En donde para cada algoritmo listado en las filas, la columna con la etiqueta $\uparrow$ muestra el total de casos donde el algoritmo es estadísticamente mejor, la etiqueta $\downarrow$ muestra la columna con el total de casos donde el algoritmo ha sido estadísticamente peor y la columna $\longleftrightarrow$ muestra el total de casos con resultados iguales entre los algoritmos comparados. Finalmente, la columna con la etiqueta $(-)$ muestra la diferencia entre las columnas $\uparrow$ y $\downarrow$, columna en base a la que serán ordenados de modo descendente los algoritmos.

	Clasificación de las instancias			
	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)
Algoritmo A				
Algoritmo B				

Tabla 2.1: Ejemplo de tabla comparativa.

Capítulo 3

Problema del Ordenamiento Lineal

El Problema del Ordenamiento Lineal (o Linear Ordering Problem, LOP) es uno de los problemas combinatorios más populares y antiguos del área de optimización, sus características son bien conocidas y por tal motivo ha sido fuertemente atacado por la comunidad científica. En este capítulo se presenta un método propuesto y basado en Algoritmos Evolutivos, con el cual se creó un plugin para la herramienta METCO. Este capítulo está dividido en cuatro secciones. En el primer apartado se encuentra un análisis breve de los diferentes métodos de solución propuestos a lo largo de los años. Posteriormente, en el segundo apartado se dan los detalles de la definición formal del problema, además de proveer un ejemplo, en el tercer apartado se dan los detalles de la solución propuesta así como las características y detalles del Plugin desarrollado para la incorporación de este problema en la herramienta METCO. En el cuarto apartado se muestran los resultados obtenidos durante la fase de pruebas de los plugins desarrollados.

3.1. Marco teórico

Dentro del grupo de problemas combinatorios clásicos podemos encontrar el Linear Ordering Problem (LOP), dicho problema cuenta con un número amplio de aplicaciones en diversas áreas, como ejemplo la economía, teoría de grafos, máquinas de traducción, psicología matemática, entre otras, de ahí el gran números de integrantes de la comunidad científica que han mostrado interés por este problema, destacando entre Chenery and Watanabe (1958), y posteriormente con los trabajos de Garey and Johnson (1979) quienes clasificaron este problema dentro del grupo de problemas NP-Hard [17]. Lo anterior da como resultado un número amplio de artículos en donde se describen diversas estrategias de solución que incluyen métodos exactos, heurísticas y metaheurísticas.

Dentro de los métodos exactos cabe destacar el método de Ramificación y

acotamiento desarrollado principalmente por Kaas en 1981 [22] y posteriormente en 2006 por Charon y Hudry [9]. También es de destacar el método de ramificación y corte planteado por Grötschel en 1984. Estos permiten obtener el óptimo en instancias con pocas filas y columnas, pero como es de esperarse el tiempo de cómputo crece exponencialmente, lo que provoca que para instancias mayores sus soluciones no sean obtenidas en tiempos razonables. Debido a estas observaciones, comienza el desarrollo de los primeros trabajos basados en Heurísticas, siendo pioneros Chenery and Watanabe en el año de 1958, cuya propuesta no incluye un algoritmo formal, pero si las ideas de tablas de sectores rankeados. Posteriormente Aujac y Becker proponen las primeras funciones de solución mediante triangulación, en 1960 y 1967 respectivamente.

En la última década se han aplicado muchas Metaheurísticas al LOP, y con esto aparecen las primeras propuestas de solución del LOP que introducen el concepto de estrategias de aprendizaje en las fases de exploración y explotación. En 1986 es Fred Glover quien en el mismo artículo en el que introduce el término metaheurística, habla sobre un algoritmo que auxiliado de una estructura de memoria guía un procedimiento heurístico de búsqueda local en la búsqueda de optimalidad global. Dicho algoritmo es definido como Tabu Search, posteriormente en 1999 Laguna et al., usó este algoritmo para proponer una solución al LOP [23].

En 1983 Kirkpatrick, S., Gelatt C.D y Vecchi, M.P en su trabajo Optimization by simulated annealing presentan una metodología basada en una aparente analogía entre el proceso físico conocido como recocido simulado y el proceso algorítmico de solución de un problema de optimización, dicha mitología es denominado simulated annealing. [33]

En 2001 en Campos et al. [6], se presenta una solución al LOP basada en una metodología propuesta en la década de los setentas. En esta metodología son introducidos nuevos conceptos y métodos de un proceso metaheurístico denominado Scatter search (SS), también conocido en castellano como Búsqueda Dispersa. Dicha metodología es basada en las estrategias para combinar reglas de decisión, así como en la combinación de restricciones [8].

Posteriormente aparece la metodología GRASP, que fue desarrolla a finales de la década de los ochentas por Feo y Resende, la idea de este algoritmo se basa en las iteraciones, en cada una de ellas se construye una solución de prueba y posteriormente se le aplica un procedimiento de mejoramiento en busca de obtener un óptimo local, si la solución obtenida resulta de mejor calidad será considerada como la solución final, en caso contrario es desechada.

Finalmente en Ceberio et al., (2013) se introduce el concepto de Restricted insert neighborhood [7], algoritmo de búsqueda local implementado en el algoritmo Memético original propuestos por Schiavinotto y Stützle (2014) [34], la búsqueda local hace las mejoras en los individuos mediante la obtención de soluciones o cambios que den como resultado un óptimo local, esto mediante descartar las

posiciones en los individuos que no generan un local óptimo. Sobre este concepto se habla a fondo en los siguientes apartados de este trabajo.

3.2. Definición del problema

El Linear Ordering Problem (LOP) es uno de los problemas clásicos de optimización combinatoria, Garey y Johnson [17] en 1979 clasificaron al LOP dentro del grupo de problemas NP-Hard. La importancia de este problema se centra en el amplio número de aplicaciones en diversas áreas como ejemplo la arqueología, la calendarización económica e incluso la psicología [12].

El LOP tiene dos diferentes definiciones, la primera se expresa mediante grafos y la segunda mediante matrices, dichas definiciones son equivalentes y no representan una diferencia amplia entre ellas, para este trabajo nos interesa la representación matricial o el llamado problema de Triangulación.

Sea una matriz $B_{n*n} = (B_{ij})$ de entradas numéricas, el Linear Ordering Problem consiste en determinar una permutación simultanea alfa de las fila y columnas de B, tal que la suma de las entradas sobre la diagonal principal sea maximizada. Se hace uso de la ecuación 3.1 como formalización del LOP:

$$f(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{\sigma_i \sigma_j} \tag{3.1}$$

En donde σ_i y σ_j denotan el índice de la fila (y columna) colocada en la posición i,j de la matriz transformada. A continuación se muestra un ejemplo de una instancia del LOP en donde $n = 6$. En la figura 3.1, se muestran cuatro soluciones diferentes, γ, β, α y σ. La matriz inicial es representada por la permutación $\gamma = (1, 2, 3, 4, 5, 6)$, y su fitness, $f(\gamma) = 185$, en el caso b se muestra un orden diferente de los elementos representados por la solución $\beta = (6, 5, 4, 1, 2, 3)$, con un fitness, $f(\beta) = 190$, que resulta de mejor calidad que la solución inicial. En una tercera combinación (caso c) representada por la solución $\alpha = (1, 6, 5, 4, 3, 2)$, se puede observar una mejora en el fitness, $f(\alpha) = 206$, pero dicha solución aún no alcanza el valor optimo máximo. Finalmente en el caso d se puede observar la solución óptima representada por $\sigma = (3, 5, 6, 4, 2, 1)$, con un fitness $f(\sigma) = 230$.

	1	2	3	4	5	6
1	0	1	13	16	22	3
2	20	0	1	13	7	13
3	11	11	0	13	29	2
4	20	21	13	0	1	22
5	6	16	20	9	0	29
6	9	17	4	17	23	0

a) $\gamma = (1, 2, 3, 4, 5, 6)$
$f(\gamma) = 185$

	6	5	4	1	2	3
6	0	23	17	9	17	4
5	29	0	9	6	16	20
4	22	1	0	20	21	13
1	3	22	16	0	1	13
2	13	7	13	20	0	1
3	2	29	13	11	11	0

b) $\beta = (6, 5, 4, 1, 2, 3)$
$f(\beta) = 190$

	1	6	5	4	3	2
1	0	3	22	16	13	1
6	9	0	23	17	4	17
5	6	29	0	9	20	16
4	20	22	1	0	13	21
3	11	2	29	13	0	11
2	20	13	7	13	1	0

c) $\alpha = (1, 6, 5, 4, 3, 2)$
$f(\alpha) = 206$

	3	5	6	4	2	1
3	0	29	2	13	11	11
5	20	0	29	9	16	6
6	4	23	0	17	17	9
4	13	1	22	0	21	20
2	1	7	13	13	0	20
1	13	22	3	16	1	0

d) $\sigma = (3, 5, 6, 4, 2, 1)$
$f(\sigma) = 230$

Figura 3.1: Cuatro diferentes soluciones de una instancia con $n = 6$.

3.2.1. Aplicaciones

Como se ha mencionado anteriormente, las aplicaciones del Linear Ordering Problem son diversas, a continuación se describen una de las aplicaciones más antiguas y que es presentada por Marti y Reinelt en [12].

3.2.1.1. Agregación de preferencias individuales

El linear Ordering Problem resulta muy recurrido cuando se trata de problemas de ordenación de objetos, entre ellos el agregado de preferencias individuales, también conocido como el problema de Kemeny, siendo esta el área de aplicación más antigua del LOP. Considere por ejemplo la siguiente situación. Un conjunto de n objetos $O_1, O_2, \ldots, O_n$, los cuales deben de ser rankeados por m personas de acuerdo a sus preferencias. Posteriormente es obtenida una tabla de objetos de

acuerdo a las preferencias de cada individuo por los objetos existentes. Una vez obtenida dicha tabla ¿Cómo rankear los objetos a modo que la mayor cantidad de individuos queden satisfechos? La solución intuitiva es la comparación de parejas de objetos, para un par O_iy O_j, $1 \leq i < j \leq n$ de objetos, cada persona debe decidir qué objeto prefiere entre O_iy O_j, el total es de $m\binom{a}{b}$ comparaciones en una matriz $H_{n*n} = (H_{ij})$ donde H_{ij} =número de personas que prefieren el objeto O_i sobre el objeto O_j.

En 1961 Slater propone otra forma de plantear este problema [39], mediante el siguiente cuestionamiento: ¿Cuál es el mínimo número de arcos que hay que invertir para convertir un tour T dado en un tour acíclico? En el contexto de preferencias, la entrada es una colección de rankings para todos los pares i y j de objetos indicando si i debe ser preferido antes de j o viceversa. Posteriormente, el problema se reduce a hallar el número máximo de rankings apareados sin contradicciones. El problema puede ser resuelto entonces definiendo una matriz en la cual la posición ij es 1 si el par de vértices (i, j) pertenecen al tour T, y es 0 en caso de que (i, j) no pertenezcan a T. Esto es igual a lo que se muestra en la ecuación 3.2

$$c_{ij} = \begin{cases} 1, if(i,j) \in T, \\ 0, otherwise \end{cases} \tag{3.2}$$

luego la solución del LOP para esta matriz entrega el mejor ordenamiento lineal para los n objetos.

Un ejemplo de aplicación social es observable en el contexto de las votaciones, en específico en un método desarrollado por Marie Jean Antoine Nicolas Caritat, el Marqués de Condorcet, en 1299. En dicho método se elige a un ganador virtual mediante la preferencia de los votantes, quienes asignan un valor numérico (acorde a su preferencia) y finalmente ordenan de mayor a menor a los candidatos.

3.2.1.2. Óptima relación tabular de descendientes

En el artículo publicado en 1947, resultado de la colaboración de Glover , F., K Lastorin , T., and Klingman , D. [28] se presenta una aplicación de LOP, una nueva clase de problemas cuyo origen es un estudio antropológico en el que se desea especificar un orden cronológico global de datos en cementerios antiguos. De modo general, se trata de un método que requiere la determinación de un orden parcial consistente para conjuntos de objetos, eventos, preferencias y similares, este tipo de problemas se caracterizan por tener r̈uidos(̈o contradicciones). La formulación matemática de esta clase de problemas se traduce en un modelo de retroalimentación mínimo de peso (inverso de la definición de matriz superior)de LOP. [28]

3.3. Solución propuesta

En esta parte del trabajo se propone el uso de Algoritmos Evolutivos combinados con Estrategias de Diversidad. Los algoritmos base usados en este apartado del trabajo están basados en los trabajos de Schiavinotto T. and Stützle T [34], así como en Ceberio et al [7].

Finalmente es importante mencionar que las implementaciones de dichas estrategias fueron desarrolladas en forma de módulos en lenguaje C++ y añadidas a la herramienta METCO. A continuación se describen los distintos procesos que componen los plugins desarrollados.

3.3.1. Representación de los individuos

Un aspecto importante en los algoritmos evolutivos es la representación, debido a que si esta etapa se hace de modo incorrecto el rendimiento se puede ver afectado severamente. Como se ha mencionado en el capítulo 2, la representación se hace mediante el mapeo del Genotipo (representación común) a cadenas genéticas o fenotipos.

Para poder hacer el mapeo del Fenotipo al Genotipo es importante hacer un análisis del problema y de los datos con los que se cuenta para dar solución al problema.

En base a la definición dada en la sección 3.2 se sabe que las instancias están dadas por matrices H_{n*n}, con filas y columnas numeradas por i, en dónde $i = 1, 2, \ldots, n$, tal que $i < i + 1$, el número que le sea asignado será el id de la fila o columna. Véase la figura 3.2 en donde se muestra el *orden o representación natural* de las filas y columnas.

Recordar que el objetivo de solución de este problema se resume en: Hallar el orden de las filas y columnas, de modo que la suma de los elementos de la matriz triangular superior sea la máxima posible.

Ahora bien, haciendo una breve inspección y análisis de la definición y objetivo del problema resulta evidente que el genotipo del individuo representa la posición de los identificadores de cada fila y columna, es decir el individuo está compuesto de n genes (posiciones) y con la posibilidad de colocar n alelos de modo irrepetible en cada uno de ellos (el id de la columna y fila), cabe señalar que los id de las filas y columnas son los asignados en la representación natural. Véase la figura 3.2

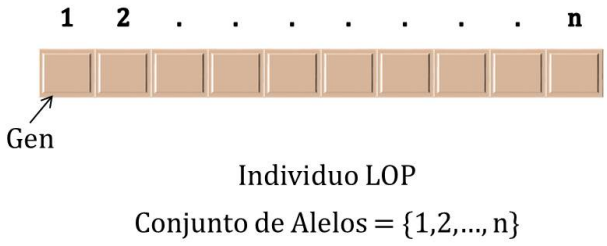


Figura 3.2: Instancia del problema

3.3.2. Función de evaluación

Una vez definida la representación del problema es posible obtener el segundo aspecto de un algoritmo evolutivo, la función de evaluación. Dicha función evalúa al individuo y regresa el fitness de éste. Con base a este fitness en los siguientes mecanismos se determina la calidad de cada individuo, en este caso el de mayor fitness por tratarse de un problema de maximización

Es posible desarrollar el algoritmo de evaluación con base a la función de evaluación del algoritmo 3, que implementa la función de coste 3.1.

Algorithm 3 Función de coste LOP

Entrada:
fenotipo, individuo a evaluar.
costos, matriz de datos a evaluar (instancia).
Salida:
Suma de los elementos de la matriz superior.

1: **procedure** EVALUATE($fenotipo : vector$, $costos : vector < vector >$)
2: $Suma = 0$
3: **for** $i \leftarrow 1 \,;\, i < n - 1 \,;\, i \leftarrow i + 1$ **do**
4: **for** $j \leftarrow i + 1 \,;\, j < n \,;\, j \leftarrow j + 1$ **do**
5: $Suma \leftarrow Suma + costos[fila][columna]$
6: **end for**
7: **end for**
8: **end procedure**

Finalmente, en el algoritmo 3 se observa que la evaluación no incluye un reordenamiento de las filas y columnas, es posible evitarlo mediante el acceso de los índices del genotipo (genes del individuo). Esto con el fin de disminuir la complejidad del algoritmo de evaluación, dado que para el caso de un reordenamiento de filas y columnas tendría que ejecutarse como mínimo $2n^2$ operaciones durante la evaluación de cada individuo, mientras que con el simple acceso a los alelos del

individuo, es posible hacer el cálculo con n^2 operaciones por cada individuo evaluado. Dicha análisis hace evidente la reducción de operaciones realizadas durante el cálculo del coste generado por determinado individuo.

3.3.3. Método de cruce

Como se ha mencionado en secciones anteriores el objetivo principal de este mecanismo es mantener un balance entre las fases de exploración y explotación, y de este modo generar un algoritmo con un rendimiento adecuado, de lo anterior, que el mecanismo de cruce resulte esencial en el proceso de creación de los individuos.

En el trabajo desarrollado y presentado en [34] se muestran una serie de pruebas con diferentes mecanismos de cruce, entre ellos los mencionados en la sección 2.3.6.2. En este trabajo se crean un conjunto de configuraciones candidatas para los algoritmos Memético e ILS. En total 78 y 144 configuraciones de algoritmos respectivamente. Cada una de estas configuraciones fue evaluada mediante un procedimiento basado en F-races [4], posterior al análisis, la configuración que obtuvo mejor rendimiento fue la conformada por el algoritmo de cruce CX. Posteriormente en Ceberio et al. [7] se menciona el uso del algoritmo original Memético e ILS con el método de cruce CX. De lo anterior que se decidiera implementar el algoritmo 4 basado en el mecanismo de cruce CX dentro del plugin de LOP.

Algorithm 4 Mecanismo de cruce CX

Entrada $Padre_A$, $Padre_B$ individuos existentes en la población
Salida $Individuo_{hijo}$.

1: **while** ALELOS_ DISPONIBLES **do**
2: Posición ← posicionLibreAleatoria($Individuo_{hijo}$)
3: **do**
4: $Padre_r$ ← padreAleatorio($Padre_A$, $Padre_B$)
5: $Padre_n$ ← padreLibre()
6: Alelo ← $Padre_n$[Posicion]
7: $Individuo_{hijo}$[Posicion] ← Alelo
8: Posicion ← posicion$Padre_r$(Alelo)
9: **while** $Individuo_{hijo}$[Posicion] = LIBRE
10: **end while**

3.3.4. Mutación

De modo similar al mecanismo de cruce, el mecanismo de mutación fue seleccionado en base a los resultados obtenidos por el análisis en el artículo [34], en

dichas pruebas, denominadas pruebas extremas, se omitió el mecanismo de cruce y/o mutación, obteniendo un mejor rendimiento en el algoritmo cuya configuración omitía el mecanismo de mutación.

Con el objetivo de crear un balance entre exploración y explotación se optó por hacer uso de un mecanismo de mutación simple y desechar la idea planteada en [34]. El mecanismo de mutación Swap no modifica de manera brusca las soluciones y por tanto podemos analizar individuos cercanos al individuo a mutar, de ahí que el algoritmo 5 sea el implementado en el Plugin LOP para METCO.

Algorithm 5 Mecanismo de mutación Swap

Entrada $Individuo$ individuo nuevo.

1: PosA ← aleatorioA_B(1,Individuo.size()-1)
2: PosB ← aleatorioA_B(PosA+1,Individuo.size())
3: Aux ← $Individuo$[PosA]
4: $Individuo$[PosA] ← $Individuo$[PosB]
5: $Individuo$[PosB] ← Aux

3.3.5. Búsqueda local

Una etapa de los algoritmos evolutivos enfocada a la fase de explotación es la búsqueda local, dichos mecanismos inician con una solución en determinada zona del espacio de soluciones y mediante modificaciones a ésta se busca llevar esta solución a ser un óptimo local de esa región del espacio de búsqueda. Para la implementación de este mecanismo se hace uso de la primera parte de la propuesta presentada en Ceberio et al.,[7]. Es importante mencionar que los algoritmos que hacen uso de mecanismo de búsqueda local son denominados Algoritmos miméticos.

En el trabajo anteriormente citado se presentan las siguientes propiedades. Sea la matriz B_{n*n} y un genotipo LOP σ.

- Cada índice $\sigma_i = k$, $i = 1$,...,n, es asociado a $2(n-1)$ entradas de B: $n-1$ para la fila k y $n-1$ para la columna k.
- El conjunto de entradas asociadas de cada índice $\sigma = k$, $i = 1, \ldots, n$ puede ser organizado en parejas, esto es, cada entrada en la fila k, $b_{k\sigma_j}$ (donde $j = 1, \ldots, n$), tiene un par en la columna k, $b_{\sigma_j k}$, colocada simétricamente con respecto a la diagonal principal.
- Todos los pares de entradas asociadas al índice $\sigma_i = k, \{b_{k1}, b_{1k}\}, \ldots, \{b_{kn}, b_{nk}\}$ son asociadas a este índice, independientemente de su posición y la posición del resto de los n-1 índices.

- Cada entrada $b_{\sigma_i\sigma_j}$ es asociada a dos indices, σ_i y σ_j.
- Para todo par de entradas $\{b_{\sigma_i\sigma_j}, b_{\sigma_j\sigma_i}\}$, una entrada es siempre asociada a una localidad sobre la diagonal principal y la otra sobre una localidad debajo de la diagonal principal, limitando de esta manera la mejor contribución de la aptitud de este par para $\max\{b_{\sigma_i\sigma_j}, b_{\sigma_j\sigma_i}\}$ en el mejor de los casos y para $\min\{b_{\sigma_i\sigma_j}, b_{\sigma_j\sigma_i}\}$ en el peor de los casos.

Debido a las propiedades de los indicies de las filas y columnas antes mencionadas es posible definir diferentes conceptos, de entre ellos el más importante y sobre el cual se basa la función de búsqueda local implementada en el plugin de LOP. Cuando un índice k donde $k = 1, \ldots, n$, es rankeado en una posición i en σ, la contribución del índice k en la función objetivo es dada por la suma de las entradas de la columna k en las filas $\{\sigma_1, \ldots, \sigma_{i-1}\}$ y la suma de las entradas de la fila k en las columnas $\{\sigma_{i+1}, \ldots, \sigma_n\}$. Esto es, los previos $i-1$ índices $\{\sigma_1, \ldots, \sigma_{i-1}\}$ y los posteriores $n-i$ índices $\{\sigma_{i+1}, \ldots, \sigma_n\}$ determinan la contribución del índice k a la función objetivo. Véase la formula 3.3 en donde se describe de modo formal esta definición.

$$c(\sigma, i) = \sum_{j=1}^{i-1} b_{\sigma_j\sigma_i} + \sum_{j=i+1}^{n} b_{\sigma_i\sigma_j} \tag{3.3}$$

Regresando al ejemplo presentado en la figura 3.1 es posible observar en la figura 3.3 dos permutaciones de los índices de la filas y las columnas, en ambos casos es posible observar los pares de entradas asociados al índice 1, así como la contribución de este posterior a su inserción, obsérvese el valor previo en el caso a) en donde se tiene un valor en los elementos superiores de 185 $(1{+}13{+}16{+}22{+}3)$, mientras que en el caso b) el valor de la suma es de 206 $(20{+}11{+}20{+}22{+}3)$, y por tanto obteniendo una contribución benéfica para la maximización de 21 unidades, obtenida mediante los pares $\{(1, 20), (13, 11), (16, 20)\}$.

	1	2	3	4	5	6
1	0	**1**	**13**	**16**	**22**	**3**
2	**20**	0	1	13	7	13
3	**11**	11	0	13	29	2
4	**20**	21	13	0	1	22
5	**6**	16	20	9	0	29
6	**9**	17	4	17	23	0

a) $\gamma = (1, 2, 3, 4, 5, 6)$
$f(\gamma) = 185$

	2	3	4	1	5	6
2	0	1	13	**20**	7	13
3	11	0	13	**11**	29	2
4	21	13	0	**20**	1	22
1	**1**	**13**	**16**	0	**22**	**3**
5	16	20	9	**6**	0	29
6	17	4	17	**9**	23	0

b) $\mu = (2, 3, 4, 1, 5, 6)$
$f(\mu) = 206$

Figura 3.3: Ejemplo de la inserción de la columna con índice 1

Una vez definido el proceso de inserción del índice k, es posible definir el algoritmo de inserción de un conjunto de índices, esto mediante la repetición del proceso antes mencionado para cada índice a insertar. Generalizando la idea anterior se obtiene el algoritmo 6, en donde se busca la posición ideal (si existe) para colocar una fila y columna, esto mediante la aplicación de la ecuación 3.3 hacia los extremos derecho izquierdo (superior e inferior respectivamente). Continuando con la iteración de dicho proceso para cada índice existente.

3.3.6. Mecanismos de gestión de diversidad

Uno de los principales objetivos de este trabajo es la implementación de *Mecanismos de Gestión de Diversidad* con el fin de obtener un mejor rendimiento en algoritmos evolutivos, y en específico los enfocados a la solución de problemas combinatorios. El esquema seleccionado se basa en las estrategias de reemplazamiento y es presentado en el artículo [37].

Antes de continuar es conveniente mencionar que el concepto multi-objetivo hace referencia a mecanismos que toman en cuenta un segundo valor además del Fitness como objetivo. Normalmente en este tipo de mecanismos se trata de la distancia entre individuos de la población, es decir tratan de optimizar una tupla *(Función Original, Función de diversidad)* de valores compuesta por el objetivo original y una medida de diversidad. Los mecanismos de Diversidad basados en estas técnicas han tomado popularidad en los últimos años y es en esta técnica en la que se basa el algoritmo usado en este trabajo. Dicha técnica es basada en el algoritmo MULTI [37]. En general el algoritmo MULTI se basa en la selección de los mejores individuos, en base a su fitness y en base a la métrica *Distancia al vecino cercano (Distance to the Closest Neighbor, DCN*), métrica que se refiere a la distancia existente entre los individuos ya seleccionados y los individuos no seleccionados.

El Algoritmo MULTI presenta un comportamiento adecuado para ejecuciones a tiempo corto, pero a medida que el número de generaciones incrementa o el tiempo de paro se hace mayor resulta ineficiente ya que pierde la diversidad. Por tal motivo se realizan mejoras basadas en incorporar el tiempo de ejecución en los temas de decisiones. En modo general, el principio detrás del esquema de remplazamiento usado se basa en la contribución a la diversidad de cada individuo. De este modo aquellos individuos que contribuyen poco a la diversidad (contribución medida sobre DCN) son omitidos de la nueva población, esto sin importar el valor de la función objetivo principal (fitness). Esto se logra mediante la penalización en la función objetivo, a modo de llevar su valor a uno muy bajo, 0 en nuestra implementación.

Obsérvese en la figura 3.4 el comportamiento del mecanismo, en donde D representa el mínimo DCN admitido y así como el efecto de la penalización de

Algorithm 6 Búsqueda local usada en plugin LOP

Entrada:

B, matriz de costos.

σ, combinación de índices.

Salida:

maxgain, ganancia máxima.

σ, nueva combinación de índices.

```
1:  procedure BÚSQUEDALOCAL(B : matriz, σ : vector)
2:      best_j ← 0
3:      maxgain ← 0
4:      gain ← 0
5:      for i ← 1; n ; i++ do
6:          gain ← 0
7:          maxgain ← 0;
8:          for j ← i-1; 1 ; j– do
9:              gain ← gain + (B[σ_i][σ_j] − B[σ_j][σ_i])
10:             if gain >maxgain then
11:                 maxgain ← gain
12:                 best_j ← j
13:             end if
14:         end for
15:         gain ← 0
16:         for j ← i+1; n ; j++ do
17:             gain ← gain + (B[σ_j][σ_i] − B[σ_i][σ_j])
18:             if gain >maxgain then
19:                 maxgain ← gain
20:                 best_j ← j
21:             end if
22:         end for
23:         if maxgain >0 then
24:             j ← best_j
25:             t ← σ_i
26:             if i <j then
27:                 for k ← i+1; j; k++ do
28:                     σ_{k−1} ← σ_k
29:                 end for
30:             else
31:                 for k ← i-1; j; k−− do
32:                     σ_{k+1} ← σ_k
33:                 end for
34:             end if
35:             σ_j ← σ_t
36:         end if
37:     end for
38: end procedure
```

los objetos que no cumplen con este valor. Como se mencionó anteriormente el valor de D sufre cambios en cada iteración en base al tiempo transcurrido, esto con el objetivo de perder la diversidad de manera paulatina durante el transcurso de la evolución de los individuos, por tanto el valor de D es calculado mediante $D = D_I - D_i * \frac{T_{Transcurrido}}{T_{Final}}$, en donde D_I es el valor de distancia máxima que puede alcanzar un individuo. Finalmente el algoritmo 7 el usado para la gestión de diversidad.

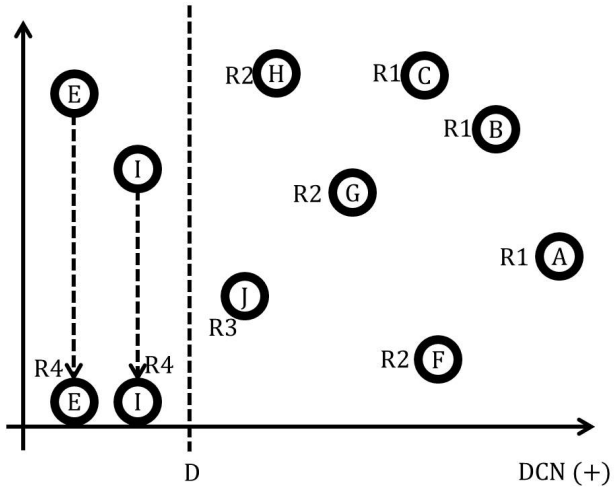


Figura 3.4: Efecto de la penalización, con distancia D

Algorithm 7 Esquema de selección de sobreviviente MULTI_ADAPTATIVE

MiembrosActuales = Población ∪ Offspring
Mejor = Individuo con mejor $f(x)$ en MiembrosActuales
NuevaPob = Mejor
MiembrosActuales = MiembrosActuales – Mejor
while | NuevaPob | <N **do**
    Calcular DCN de MiembrosActuales, considerando como referencia NuevaPob.
    $D = D_I - D_i * \frac{T_{Transcurrido}}{T_{Final}}$
    Penalizar(MiembrosActuales, D)
    ND = Selección de miembros no usados evitando la repetición.
    Selección = Individuos elegidos aleatoriamente de ND
    NuevaPob = NuevaPob ∪ Selección
    MiembrosActuales = MiembrosActuales – Selección
end while
Población = NuevaPob

3.3.6.1. Funciones para el cálculo de DCN

El cálculo del valor de DCN es fundamental para la ejecución de este algoritmo, debido a que sobre este valor recae el comportamiento adecuado del algoritmo y por tanto de la diversidad en la población. Se desarrollaron 2 mecanismos para el cálculo de DCN que se presentan a continuación, dichos mecanismos toman dos individuos y calculan la distancia existente entre ellos.

De modo general el valor de DCN representa las diferencias existentes entre individuos, el factor de importancia en LOP se encuentra en la posición de cada fila y columna, debido a que marca una diferencia significativa que una columna se encuentre en la parte superior o la parte inferior ya que el número de elementos que se ubicarán en la matriz superior varía. Véase la figura 3.6, en el caso a) la fila y columna indexada se ubica en la posición 5 mientras que en el caso b) se ubica en la posición 2, incrementando y reduciendo los valores que pertenecen a la matriz superior.

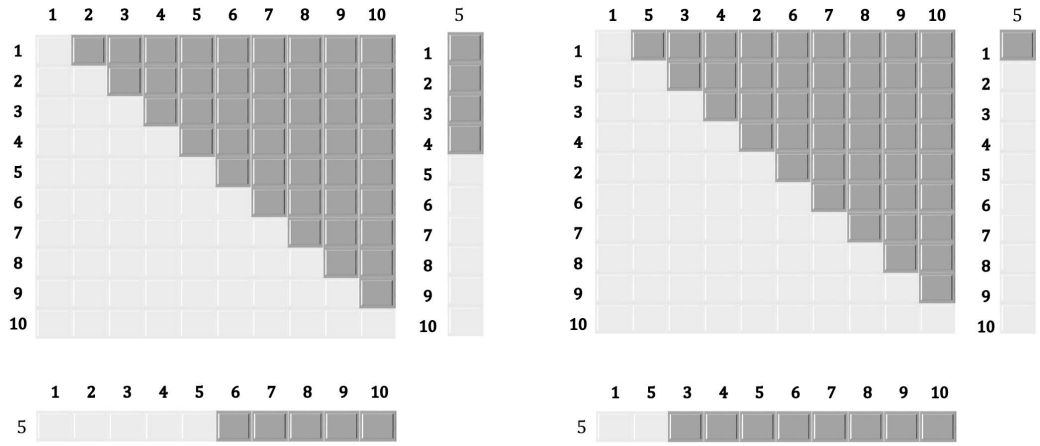


Figura 3.5: Movimiento de filas y columnas

3.3.6.1.1. DCN - Posición invertida de los elementos Como se mencionó anteriormente la distancia entre individuos se medirá mediante la posición de los índices, debido a esto la primera propuesta cuenta parejas de índices del individuo 1 que se encuentran en orden inverso en el individuo 2. Para ejemplificar véase la figura 3.6, en donde se muestran las parejas del individuo 1 $(10, 1)$ y $(9, 4)$, parejas que en el individuo 2 son $(1, 10)$ y $(9, 4)$.

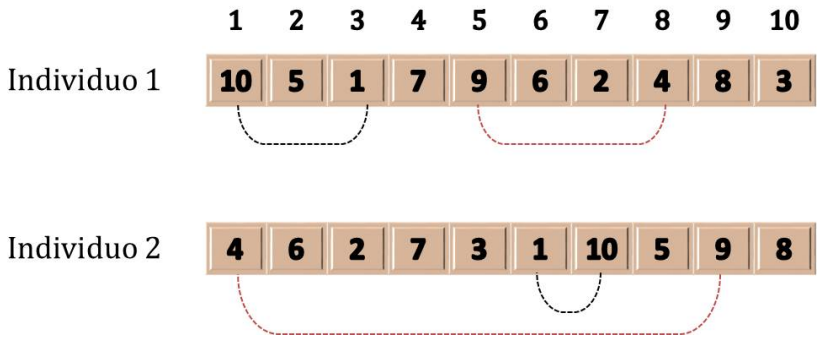


Figura 3.6: Ejemplo del mecanismo de conteo por parejas invertidas

El cálculo de la primera propuesta de DCN se hace mediante el algoritmo 8, de orden $O(n^2)$ debido a que revisa todas las parejas dentro del individuo, siendo n el tamaño del individuo. El valor máximo para DCN estará dado por $\frac{n*(n-1)}{2}$

Algorithm 8 Función DCN de parejas invertidas

ENTRADA:
IndividuoA, IndividuoB de dimensión n.
SALIDA:
Cont, total de parejas invertidas.

```
1: Cont ← 0
2: Posición ← Posición de los índices en el IndividuoB
3: for i ← 1 ; n-1 do
4:     for j ← i+1 ; n do
5:         if Posición[IndividuoA[i]] >Posición[IndividuoA[j]] then
6:             Cont ← Cont +1
7:         end if
8:     end for
9: end for
```

3.3.6.1.2. DCN – Diferencia entre posición La segunda propuesta de cálculo del DCN se realiza mediante la suma de valor absoluto de la distancia existente entre la posición del índice t en el individuo 1 y el individuo 2, donde $t = 1, 2, \ldots, n$. Véase la figura 3.7 en donde se muestra un ejemplo para algunos índices.

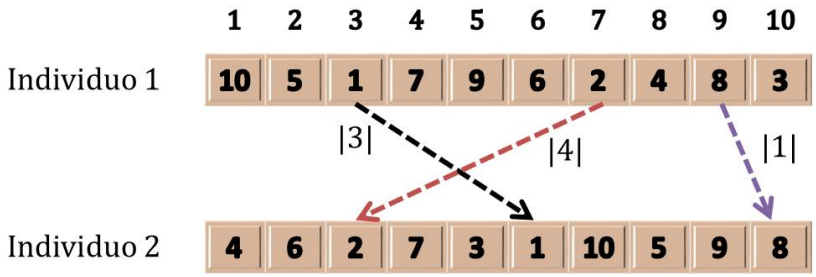


Figura 3.7: Ejemplo del mecanismo de suma de diferencia de posición

La segunda propuesta, el algoritmo 9, tiene complejidad lineal, debido a que únicamente es necesario revisar una vez cada uno de los genes del individuo.

Algorithm 9 Función DCN de distancia en posición de los índices

ENTRADA:
IndividuoA, IndividuoB de dimensión n.
SALIDA:
Sum, suma del valor absoluto de la distancia en los índices en los individuos.

```
1: Cont ← 0
2: Posición ← Posición de los índices en el IndividuoB
3: for i ← 1 ; n-1 do
4:     Sum ← Sum + |i-Posición[IndividuoA[i]]|
5: end for
```

3.4. Resultados

Para comprobar la eficiencia del método de optimización desarrollado se realizaron pruebas en el Clúster “El Insurgente”, cuyas características son:

- 112 nodos
- Intel(R) Xeon(R) CPU E55-2620 2.4 GHz
- Disco duro: SAT
- Red: Dos puertos: 10 GB

Se realizó una comparación con 3 diferentes configuraciones del Plugin y un algoritmo extra, que es el presentado en [7], los resultados de dicho trabajo son considerados los mejores para LOP hasta Enero del 2014. Las configuraciones usadas del Plugin LOP para METCO son:

- **GElit:**
  - Búsqueda local presentada en sección 3.3.5
  - Cruce CX, presentado en sección 3.3.3
  - Mutación presentada en sección 3.3.4
  - Reemplazamiento generacional con elitismo.
- **Div I:**
  - Búsqueda local presentada en sección 3.3.5
  - Cruce CX, presentado en sección 3.3.3
  - Mutación presentada en sección 3.3.4
  - Implementación de Mecanismo de diversidad con cálculo de DCN mediante *Posición invertida de los elementos*
- **Div II:**
  - Búsqueda local presentada en sección 3.3.5
  - Cruce CX, presentado en sección 3.3.3
  - Mutación presentada en sección 3.3.4
  - Implementación de Mecanismo de diversidad con cálculo de DCN mediante *Diferencia entre posición*

El conjunto de pruebas se realizó con 10 instancias de diferentes tamaños clasificadas del siguiente modo:

- Instancias Pequeñas.
  - N-be75tot_150
  - N-tiw56r66_150
  - N-tiw56n54_250
  - N-be75oi_250
- Instancias Medianas
  - N-be75oi_300
  - N-stabu75_300
  - N-t59d11xx_500
  - N-t65w11xx_500

- Instancias Grandes
  - N-t70n11xx_750
  - N-tiw56r54_750

Por cada instancia se realizaron 30 pruebas, con parámetros de probabilidad de cruce y mutación de 100

3.4.1. Resultados generales

Posteriormente se ejecutó un segundo conjunto de pruebas donde se realizaron ejecuciones de 4 horas, agregando un plus en variación de población, haciendo pruebas con Poblaciones de 50 y 100 individuos. En las tablas 3.1 y 3.2 se muestran los esquemas con población de 100 individuos, sin y con mecanismos de gestión de diversidad respectivamente. Mientras que en las tablas 3.3 y 3.4 se muestran los esquemas con población de 50 individuos, sin y con mecanismos de gestión de diversidad respectivamente.

En dichas tablas se observan los 4 esquemas comparados, 2 columnas por cada uno de ellos y para cada una de las 10 instancias, al hacer una breve inspección es fácil notar que para las ejecuciones con población de tamaño 100 el esquema Div II en comparación con el esquema Div I obtiene mejores resultados en el 90 % de las instancias, un 100 % en comparación con el esquema GElit y 90 % en comparación con el esquema Memético. Finalmente, comparando los resultados del esquema Div I con el esquema GElit se obtienen resultados de mayor calidad en 100 % de las instancias, y en comparación del esquema Memético en un 90 % de las instancias.

	Memético		GElit	
Instancia	Mejor	Promedio	Mejor	Promedio
N-be75tot_150	12288855	12288013,13	12288749	12269964,1
N-tiw56r66_150	1940755	1940755	1940755	1939353,267
N-tiw56n54_250	2099740	2099086,5	2099018	2097215,867
N-be75oi_250	5912446	5909541,833	5912138	5908498,333
N-be75oi_300	9401059	9393904,833	9402704	9392572,567
N-stabu75_300	15590312	15581817,93	15581398	15556832,3
N-t59d11xx_500	13282523	13276494,63	13279082	13259656,67
N-t65w11xx_500	19432779776	19420091665	19425827488	19402777462
N-t70n11xx_750	17021900	17009168,73	17022257	17004591,13
N-tiw56r54_750	17147336	17129345	17135214	17116774,03

Tabla 3.1: Resultados generales, población 100. Esquemas sin Gestión de diversidad.

Instancia	Div I		Div II	
	Mejor	Promedio	Mejor	Promedio
N-be75tot_150	12288855	12288851,47	12288855	12288855
N-tiw56r66_150	1940755	1940755	1940755	1940755
N-tiw56n54_250	2099909	2099594,633	2100200	2099669,2
N-be75oi_250	5912761	5911973,767	5912761	5912701,567
N-be75oi_300	9403164	9400835,533	9407004	9402988
N-stabu75_300	15592028	15588274,6	15594987	15590683,7
N-t59d11xx_500	13283489	13279796,8	13292605	13284981,1
N-t65w11xx_500	19431006038	19426723143	19443213906	19432440338
N-t70n11xx_750	17031875	17016508,87	17034418	17026610,23
N-tiw56r54_750	17148676	17139476,6	17157018	17146153,33

Tabla 3.2: Resultados generales, población 100. Esquemas con Gestión de diversidad.

Mientras que en las ejecuciones con población 50 el esquema *Div II* obtiene en el 80 % de las instancias los mejores resultados en comparación con el esquema *Div I*, 100 % en comparación con el esquema *GElit* y 90 % en comparación con el esquema *Memético*. Haciendo del esquema *Div II* el algoritmo superior en las ejecuciones de población 50. Por otro lado, comparando los resultados del esquema *Div I* con el esquema *GElit* se obtienen resultados óptimos en 100 % de las instancias, y en comparación del esquema *Memético* se obtiene un 90 %.

Instancia	Memético		GElit	
	Mejor	Promedio	Mejor	Promedio
N-be75tot_150	12288855	12287821,8	12288749	12269332,23
N-tiw56r66_150	1940755	1940755	1940755	1937501,6
N-tiw56n54_250	2099740	2099000,5	2099844	2096710,5
N-be75oi_250	5912761	5909554,633	5911335	5906447,7
N-be75oi_300	9402270	9395033,667	9402001	9390248,233
N-stabu75_300	15590606	15576481,87	15581668	15547848,7
N-t59d11xx_500	13284306	13273584,33	13273608	13254206,2
N-t65w11xx_500	19436292096	19416490121	19421691931	19394420565
N-t70n11xx_750	17020376	17005446,73	17015110	16992716,17
N-tiw56r54_750	17139552	17125297,8	17136309	17110543,33

Tabla 3.3: Resultados generales, población 50. Esquemas sin Gestión de diversidad.

3.4.2. Pruebas estadísticas

En general es evidente que el algoritmo que tiene implementado un esquema de diversidad proporciona mejores resultados que los esquemas que carecen de él.

Instancia	Div I		Div II	
	Mejor	Promedio	Mejor	Promedio
N-be75tot_150	12288855	12288855	12288855	12288855
N-tiw56r66_150	1940755	1940755	1940755	1940755
N-tiw56n54_250	2100200	2099562,3	2100200	2099603,8
N-be75oi_250	5912761	5911730,533	5912761	5912300,8
N-be75oi_300	9405393	9400975,067	9407004	9402013,067
N-stabu75_300	15592368	15586540,27	15594855	15587621,97
N-t59d11xx_500	13289318	13280651,17	13290520	13282219,77
N-t65w11xx_500	19437823937	19424446876	19442711013	19428451955
N-t70n11xx_750	17026788	17019439,3	17030799	17022191,43
N-tiw56r54_750	17153719	17138524,93	17153958	17144102,43

Tabla 3.4: Resultados generales, población 50. Esquemas con Gestión de diversidad.

Y además de que en las ejecuciones con población de 100 individuos el número de instancias con resultados de mejor calidad fue mayor.

Para dar aún más argumentos sobre la idea de la calidad del algoritmo con gestión de diversidad implementado se muestran las tablas 3.5 y 3.6, en donde tras un análisis probabilístico de comparación tipo *todos con todos*, se observa el número de instancias en las que cada esquema es mejor. Para cada esquema se realizó un total de 12 pruebas en ambos grupos de poblaciones, resultando ser DIV II el mejor esquema, obteniendo el mayor número de resultados de calidad, esto para ambos grupos de poblaciones.

	Pequeñas				Medianas				Grandes			
	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)
Div II	8	0	4	8	12	0	0	12	6	0	0	6
Div I	7	1	4	6	8	4	0	4	4	2	0	2
Memético	3	6	3	-3	3	8	1	-5	2	4	0	-2
GenElit	0	11	1	-11	0	11	1	-11	0	6	0	-6

Tabla 3.5: Comparativa entre métodos, población 100.

	Pequeñas				Medianas				Grandes			
	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)
Div II	8	0	4	8	10	0	2	10	5	0	1	5
Div I	7	1	4	6	8	2	2	6	4	1	1	3
Mémetico	4	6	2	-2	4	8	0	-4	2	4	0	-2
GenElit	0	12	0	-12	0	12	0	-12	0	0	0	0

Tabla 3.6: Comparativa entre métodos, población 50.

3.4.3. Comportamiento de los esquemas

En el grupo de gráficas que inicia con la gráfica 3.8 y termina con la gráfica 3.27 se muestra el comportamiento general obtenido tras el conjunto de pruebas realizadas, es posible argumentar que los esquemas con diversidad implementada son de mejor calidad que los que carecen de ella. A continuación se describe de modo detallado el comportamiento para los grupos de graficas con población 50 y población 100 de individuos.

En la gráfica 3.8 a la gráfica 3.17 se muestra el comportamiento de los 4 esquemas evaluados con cada una de las instancias y con una población de 50 individuos. Es fácil notar que a pesar de que los esquemas con diversidad implementada inicialmente tienen peores soluciones, tras el paso del tiempo de ejecución mejoran hasta llegar a un punto en el que coinciden con el resto de los esquemas y posteriormente alcanzar mejores resultados, es decir, presentan un comportamiento bastante estable y de calidad, mientras que los esquemas sin *diversidad* implementada resultan estancados desde el inicio de las ejecuciones, sin mostrar mejoras importantes durante las 4 horas de ejecución.

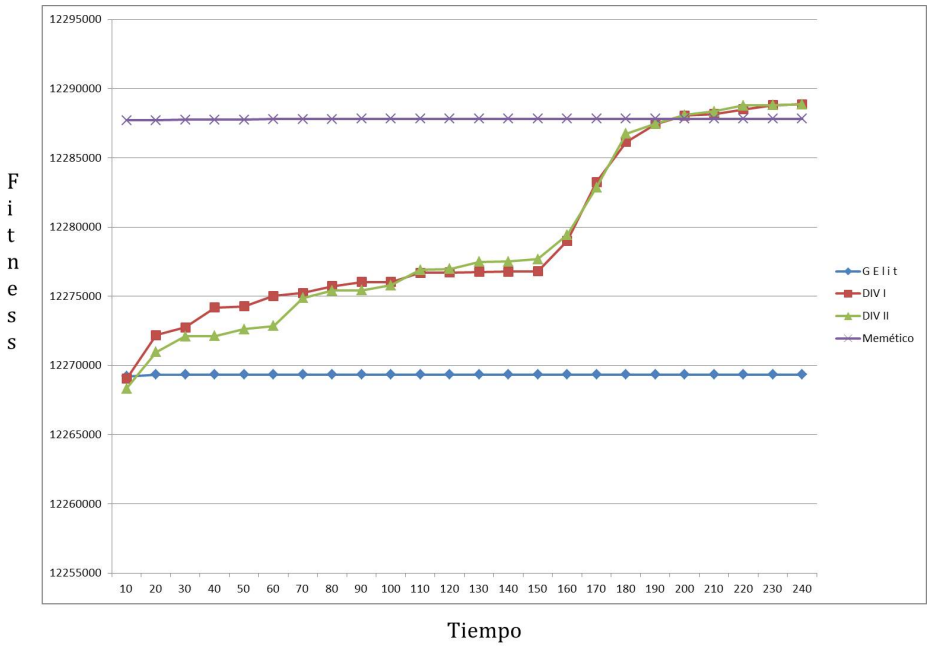


Figura 3.8: Comportamiento de la evaluación de la instancia N-be75tot_150 para una población de 50 individuos, tiempo de 4hr y resultados cada 10 minutos.

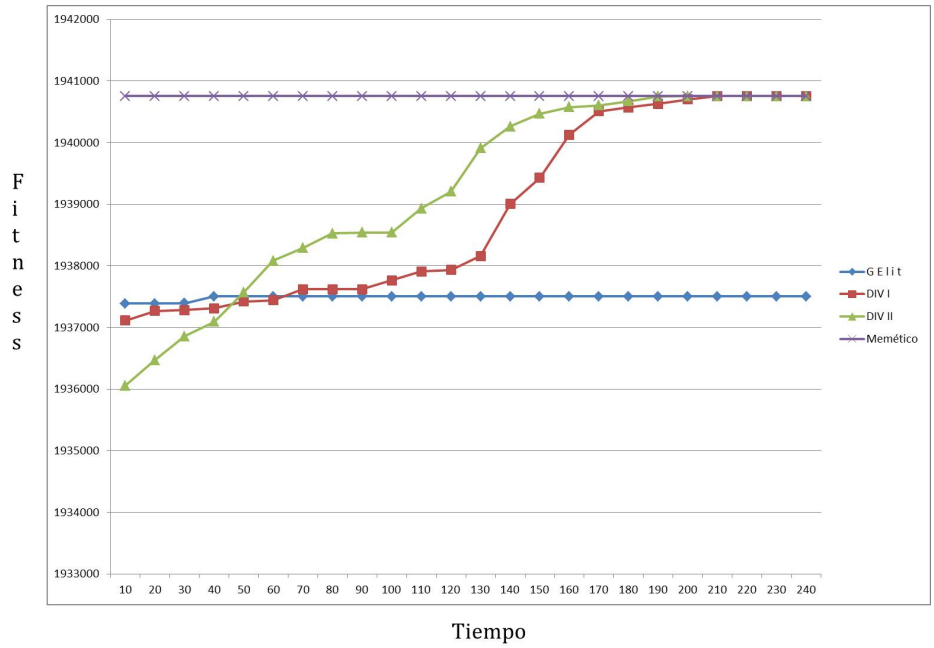


Figura 3.9: Comportamiento de la evaluación de la instancia N- tiw56r66_150 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

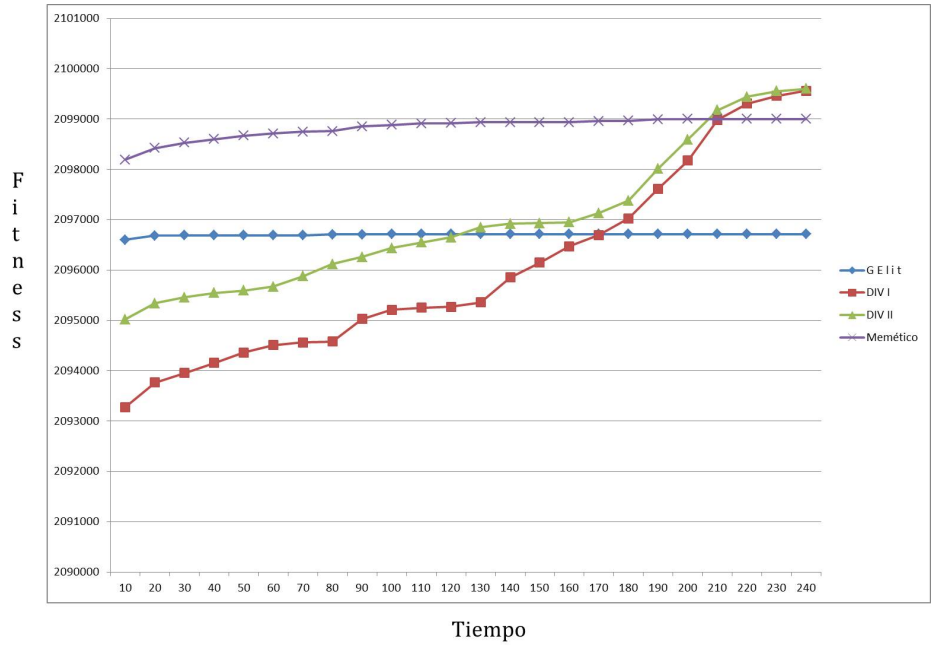


Figura 3.10: Comportamiento de la evaluación de la instancia N-tiw56n54_250 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

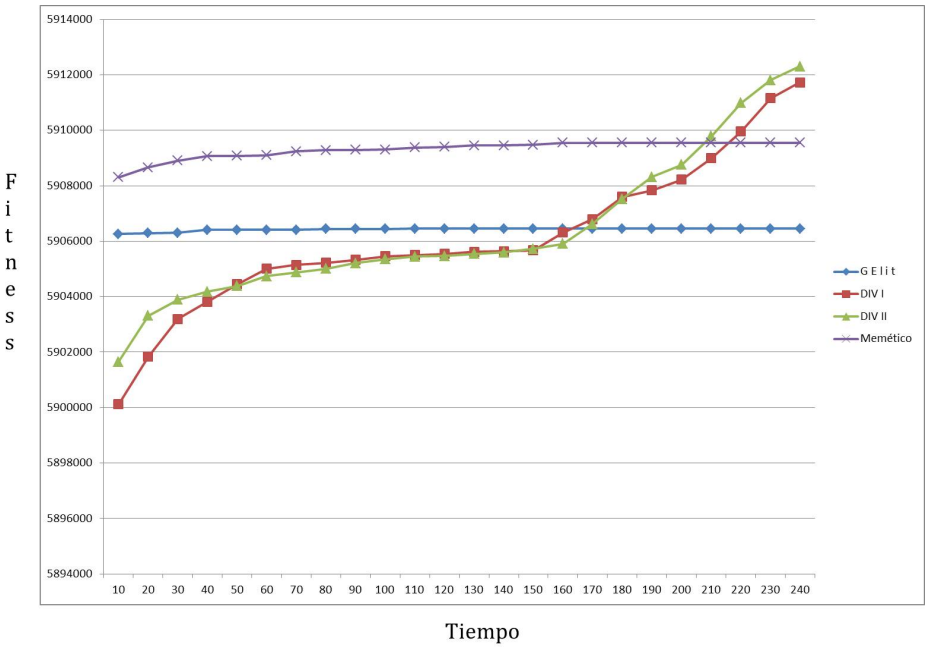


Figura 3.11: Comportamiento de la evaluación de la instancia N-be75oi_250 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

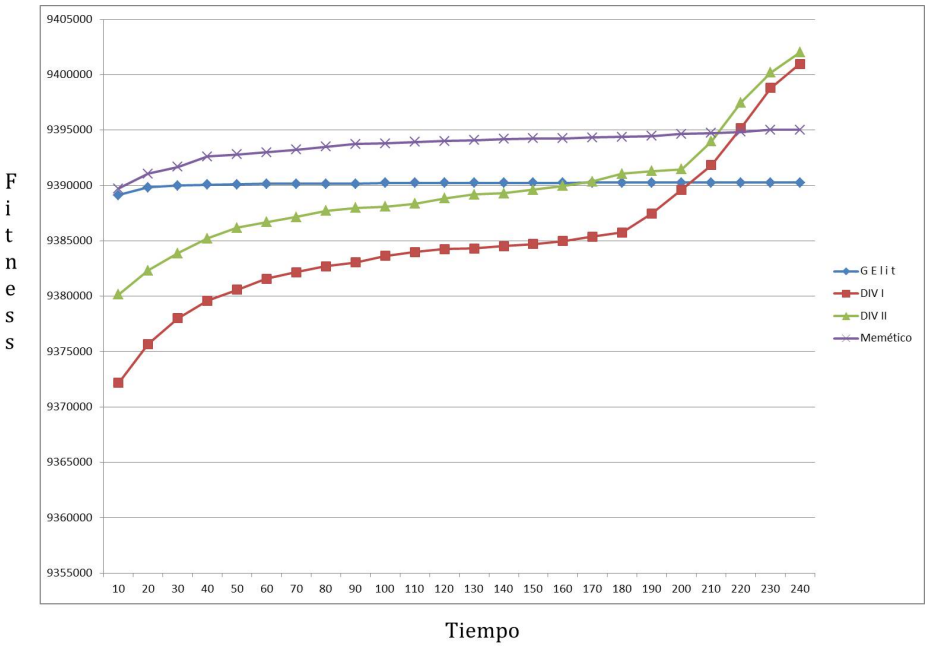


Figura 3.12: Comportamiento de la evaluación de la instancia N-be75oi_300 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

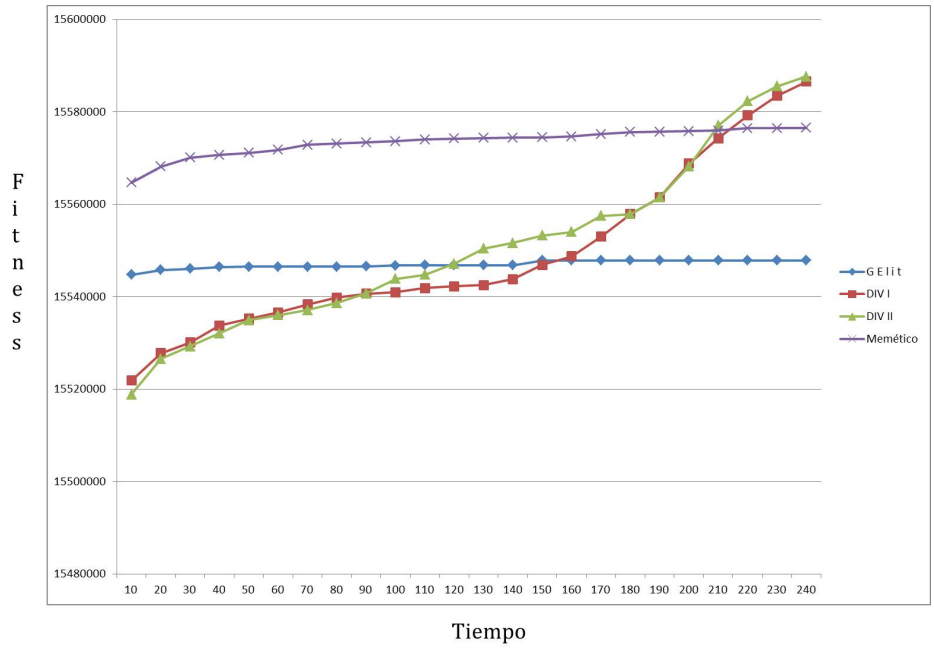


Figura 3.13: Comportamiento de la evaluación de la instancia N-stabu75_300 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

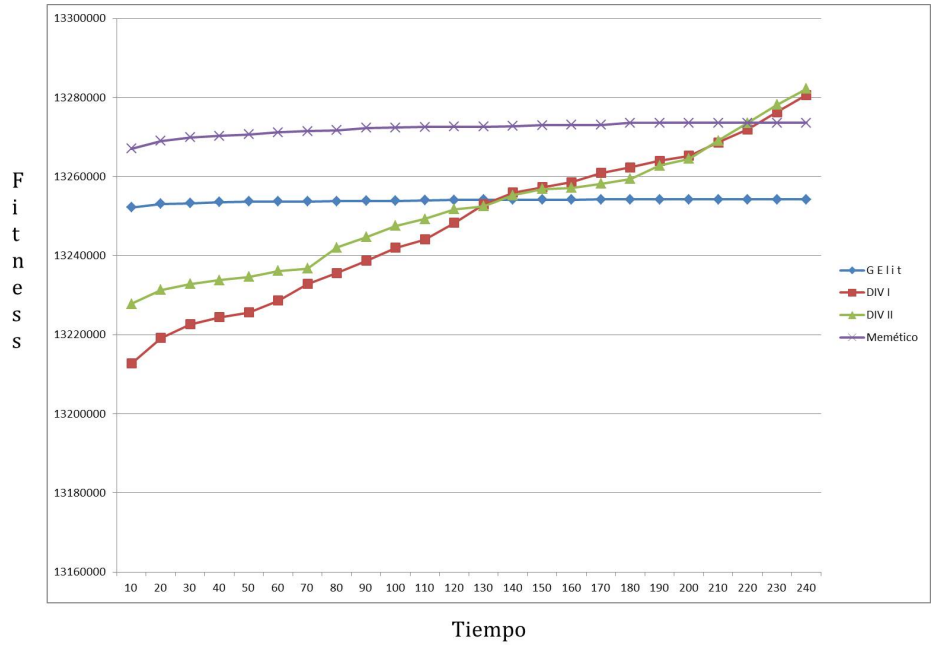


Figura 3.14: Comportamiento de la evaluación de la instancia N-t59d11xx_500 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

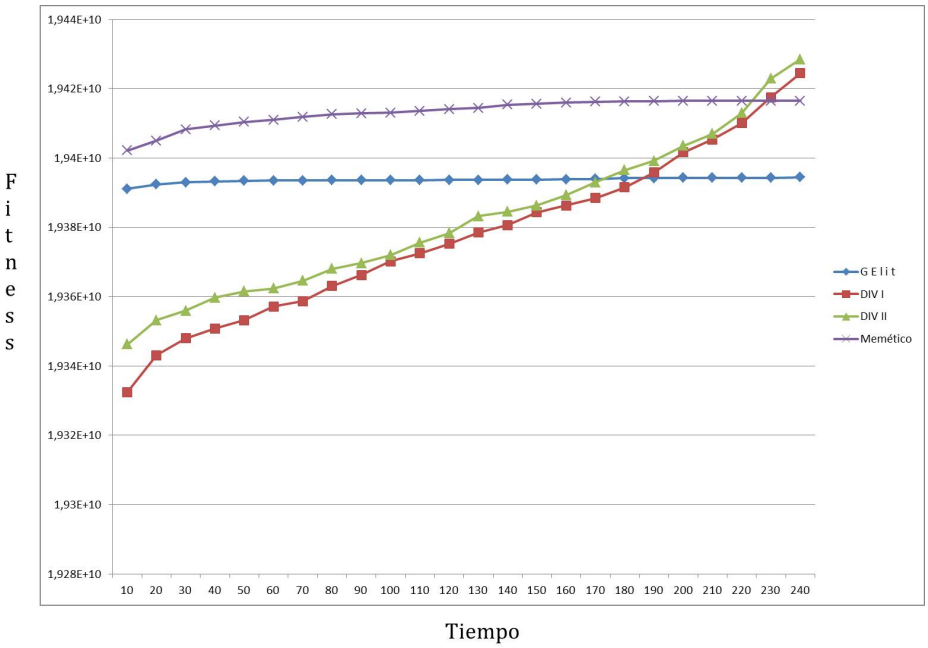


Figura 3.15: Comportamiento de la evaluación de la instancia N-t65w11xx_500 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

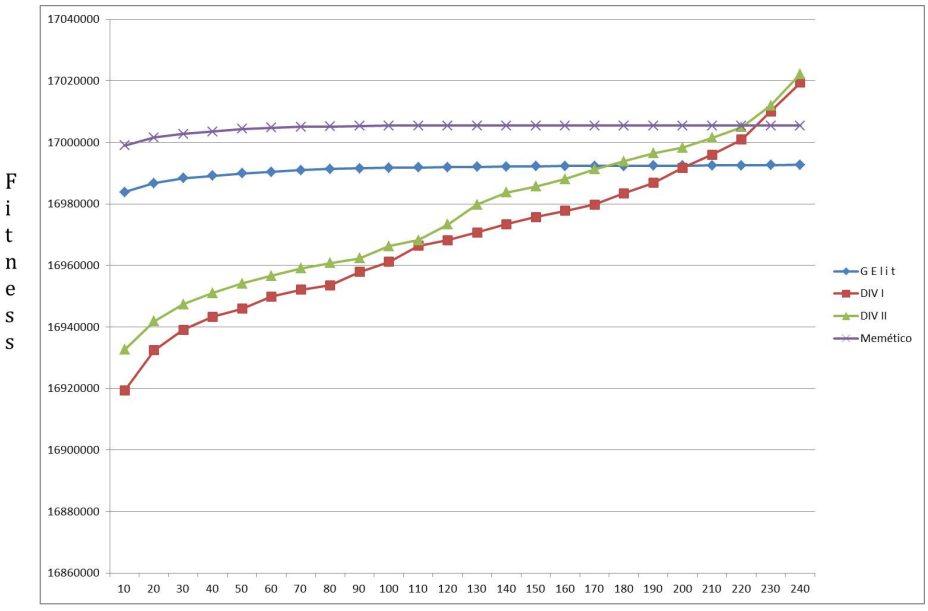


Figura 3.16: Comportamiento de la evaluación de la instancia N-t70n11xx_750 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 mi-

nutos.

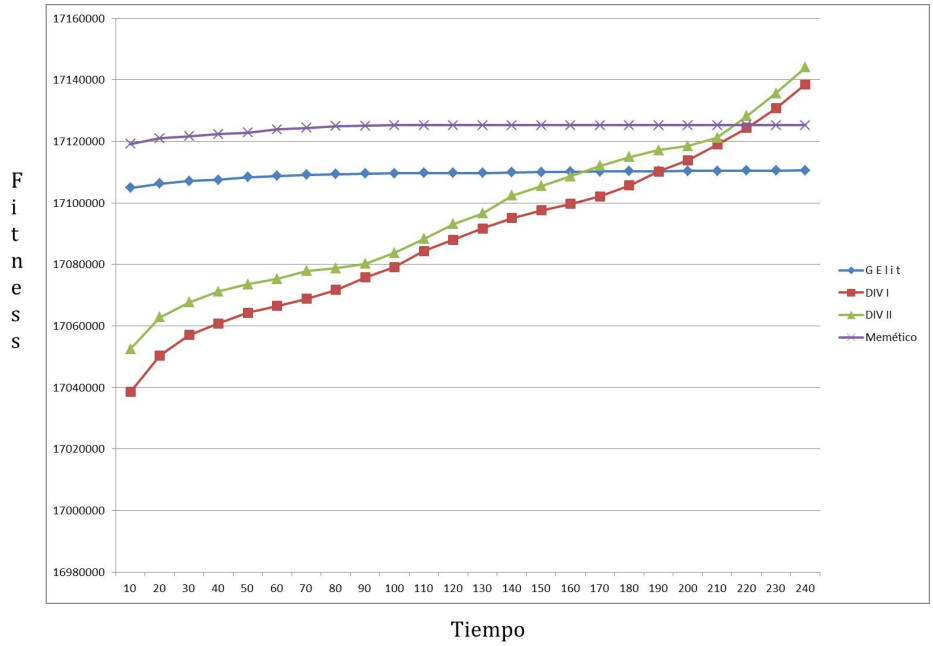


Figura 3.17: Comportamiento de la evaluación de la instancia N-tiw56r54_750 para una población de 50 individuos y tiempo de 4hr con resultados cada 10 minutos.

A continuación se muestran en la gráfica 3.18 a la gráfica 3.27 el comportamiento de los 4 esquemas evaluados con cada una de las instancias y con una población de 100 individuos. De modo similar que en las ejecuciones de individuos con población 50, los esquemas con *diversidad* presentan un comportamiento estable y creciente durante la ejecución de las corridas, esto a pesar de que los esquemas con diversidad implementada inicialmente tienen peores soluciones.

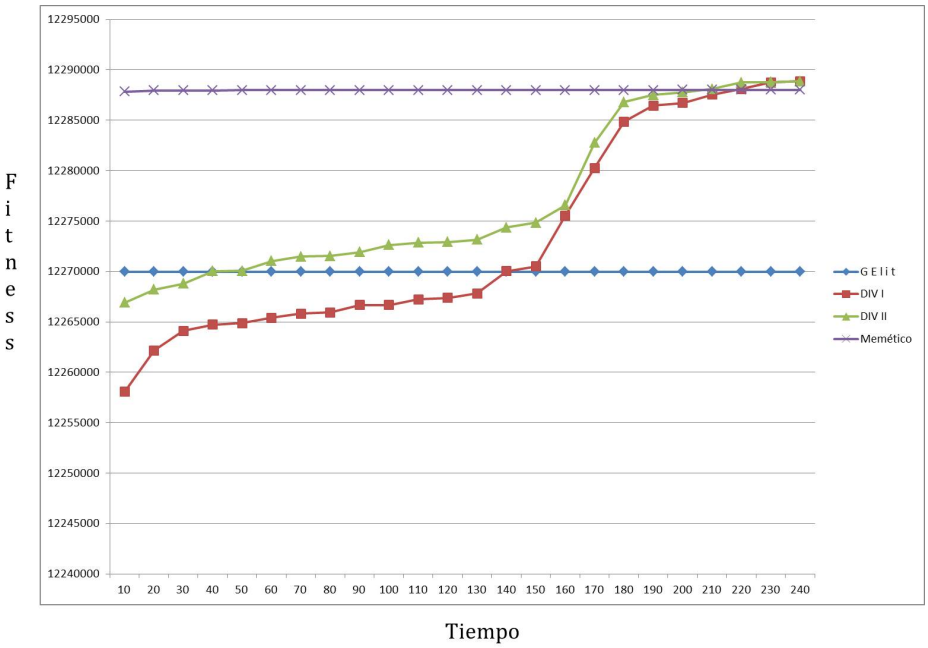


Figura 3.18: Comportamiento de la evaluación de la instancia N-be75tot_150 para una población de 100 individuos y tiempo de 4hr con resultados cada 10 minutos.

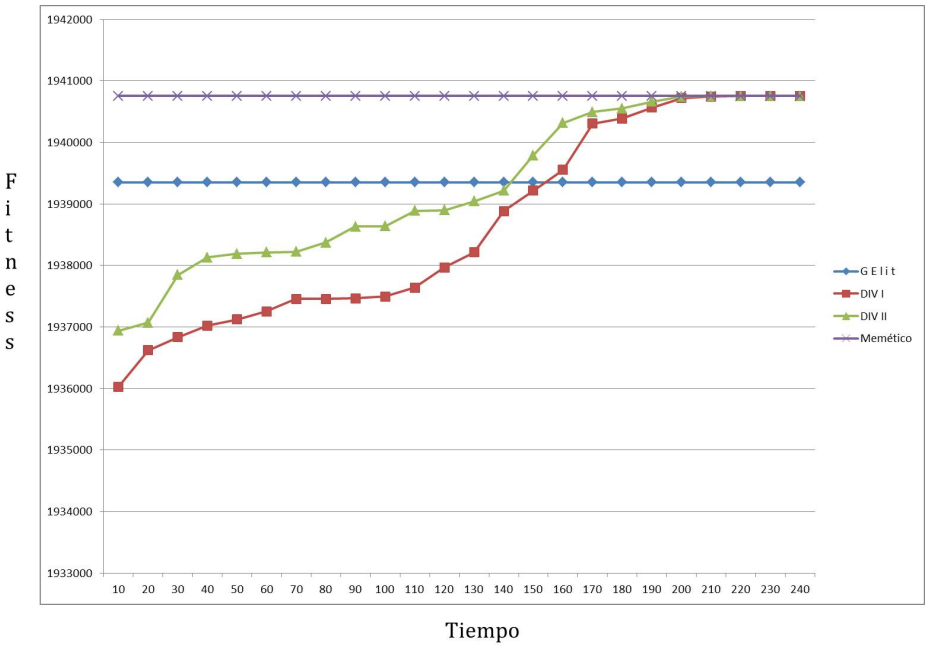


Figura 3.19: Comportamiento de la evaluación de la instancia N- tiw56r66_150 para una población de 100 individuos, tiempo de 4hr y resultados cada 10 minutos.

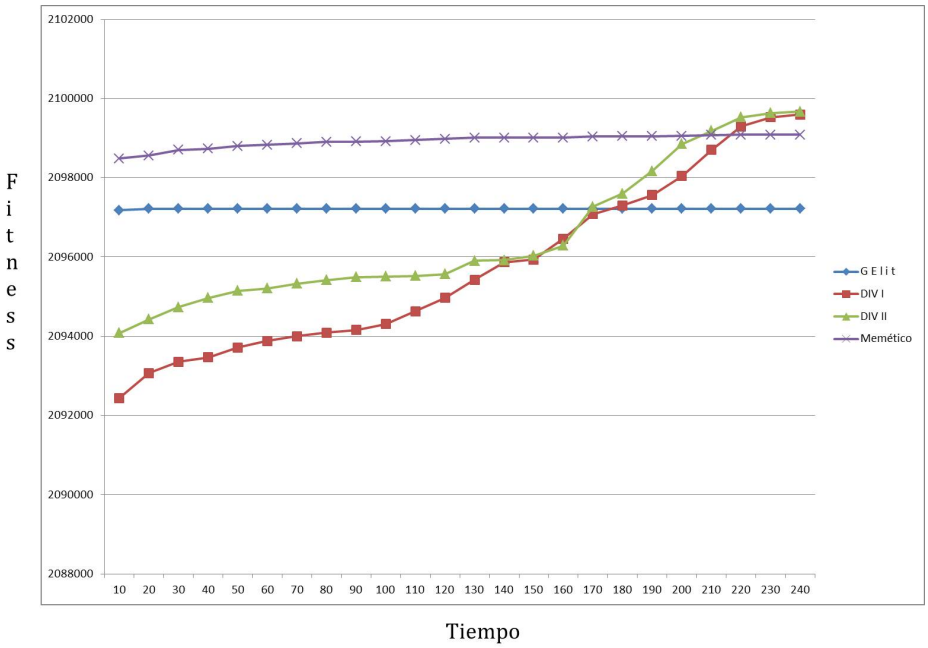


Figura 3.20: Comportamiento de la evaluación de la instancia N-tiw56n54_250 para una población de 100 individuos y tiempo de 4hr con resultados cada 10 minutos.

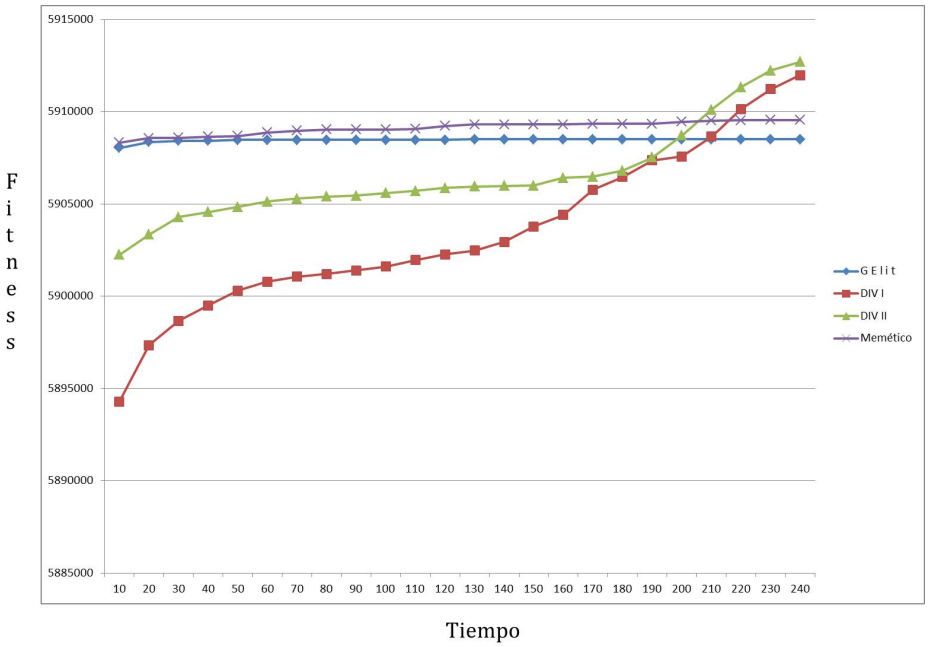


Figura 3.21: Comportamiento de la evaluación de la instancia N-be75oi_250 para una población de 100 individuos y tiempo de 4hr con resultados cada 10 minutos.

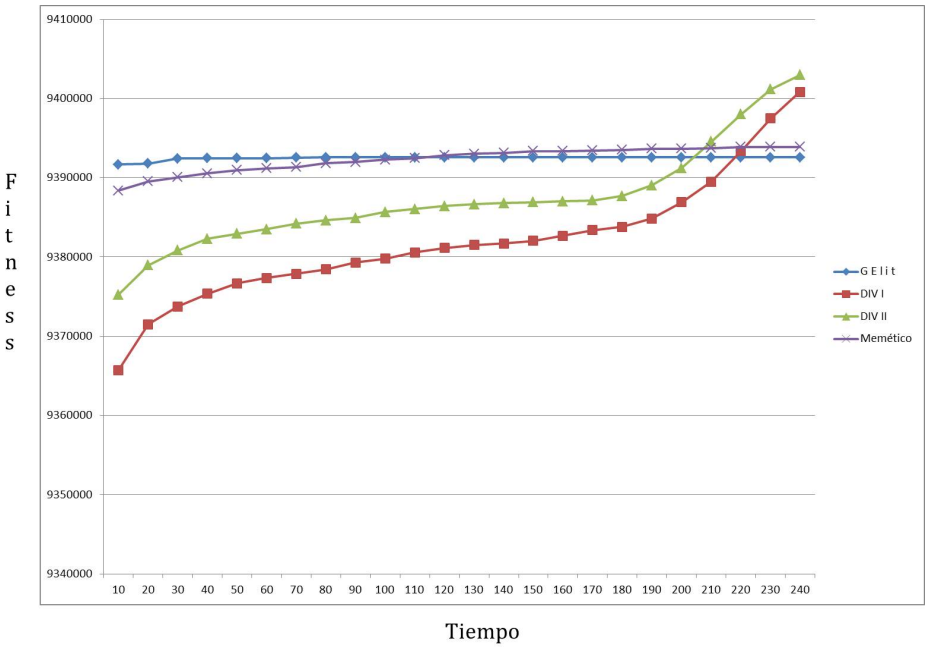


Figura 3.22: Comportamiento de la evaluación de la instancia N-be75oi_300 para una población de 100 individuos y tiempo de 4hr con resultados cada 10 minutos.

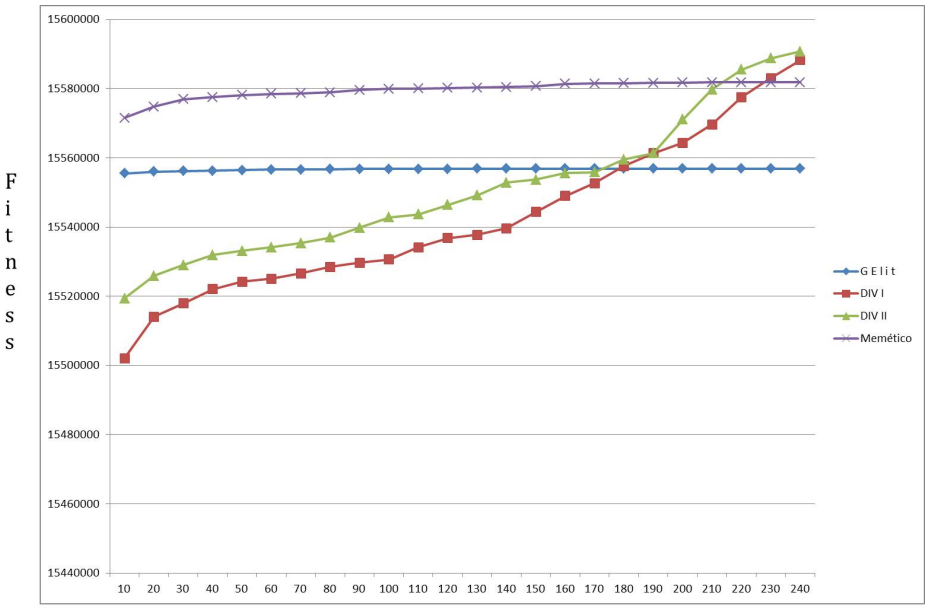


Figura 3.23: Comportamiento de la evaluación de la instancia N-stabu75_300 para una población de 100 individuos y tiempo de 4hr con resultados cada 10 minutos.

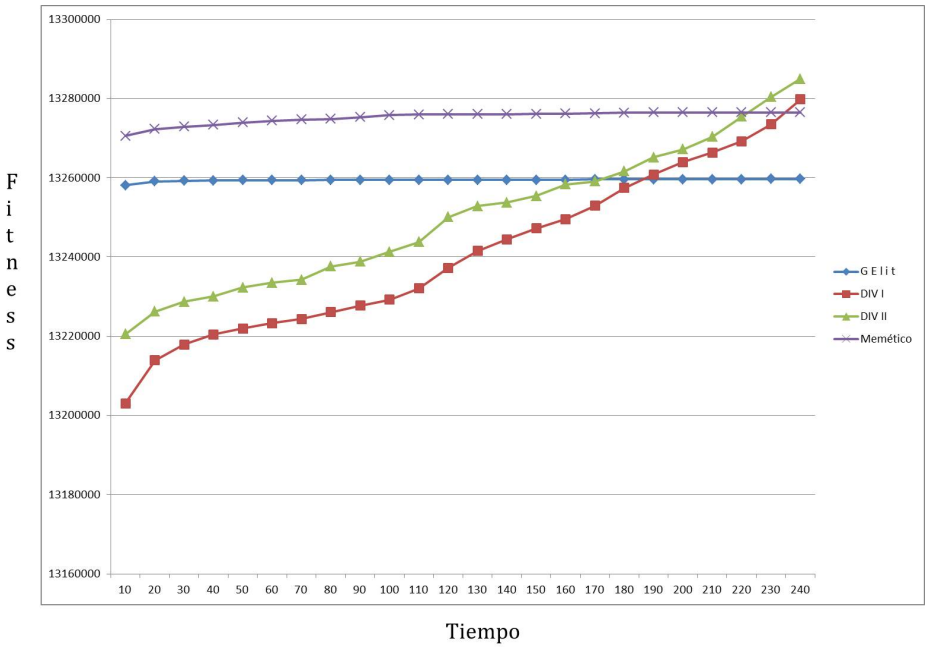


Figura 3.24: Comportamiento de la evaluación de la instancia N-t59d11xx_500 para una población de 100 individuos, tiempo de 4hr y resultados cada 10 minutos.

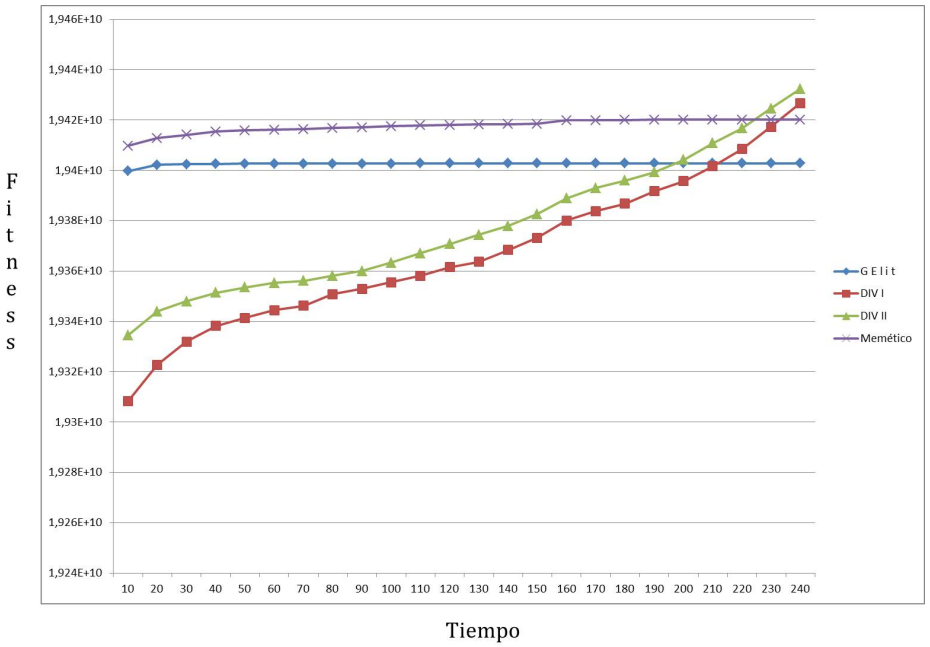


Figura 3.25: Comportamiento de la evaluación de la instancia N-t65w11xx_500 para una población de 100 individuos, tiempo de 4hr y resultados cada 10 minutos.

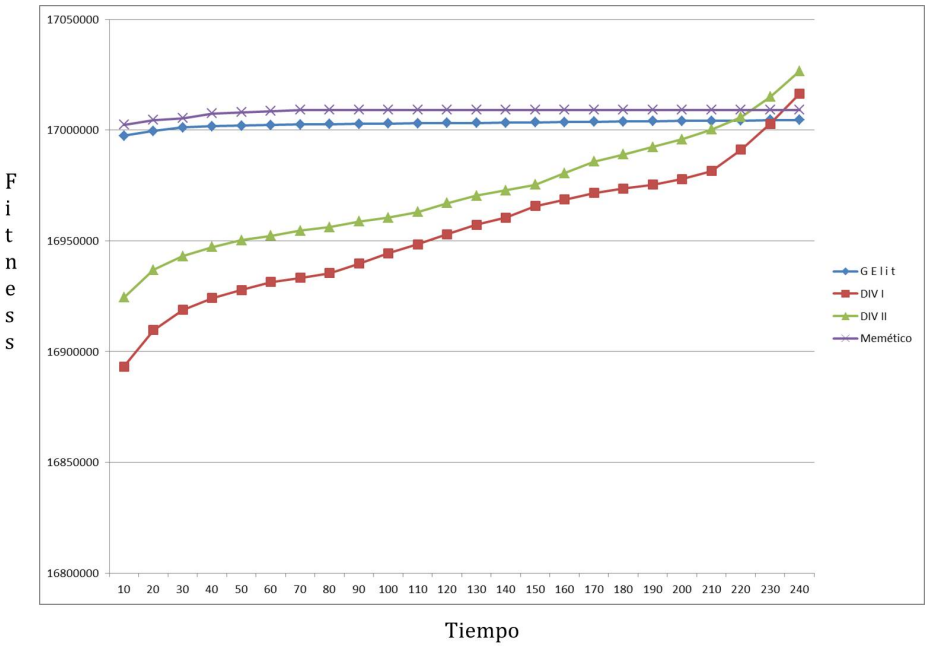


Figura 3.26: Comportamiento de la evaluación de la instancia N-t70n11xx_750 para una población de 100 individuos, tiempo de 4hr y resultados cada 10 minutos.

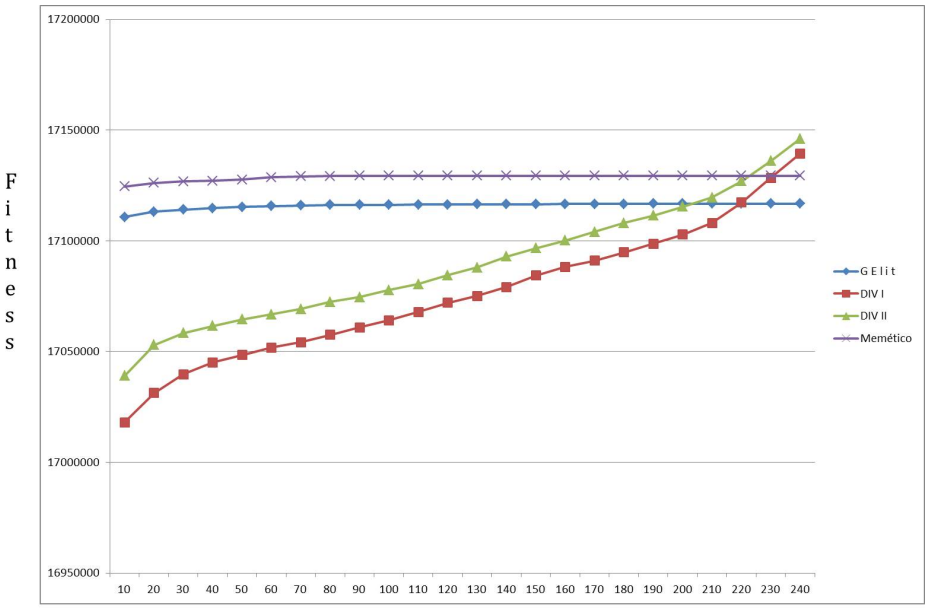


Figura 3.27: Comportamiento de la evaluación de la instancia N-tiw56r54_750 para una población de 100 individuos, tiempo de 4hr y resultados cada 10 minutos.

Capítulo 4

Problema de Asignación Generalizado

El Problema de asignación Generalizado (o Generalized Assignment Problem, GAP) es uno de los problemas combinatorios más antiguos del área de optimización, inicialmente planteado como un problema de transportación. En este capítulo se presenta el método propuesto que está basado en Algoritmos Evolutivos, y su implementación como plugin para la herramienta **METCO**. Este capítulo está dividido en cuatro secciones, en el primer apartado se encuentra un análisis breve de los diferentes métodos de solución propuestos a lo largo de los años para este problema. Posteriormente, en el segundo apartado se dan los detalles de la definición formal del problema, además de proveer un ejemplo, en el tercer apartado se dan los detalles de la solución propuesta así como las características y detalles del Plugin desarrollado dentro de la herramienta METCO. Finalmente, en el cuarto apartado se muestran los resultados de las pruebas realizadas.

4.1. Marco teórico

El *Generalized Assignment Problem* (GAP) es un problema de optimización combinatoria, en general el GAP es un problema de minimización de costos, cuyo objetivo es asignar m tareas a n agentes, tomando en cuenta la capacidad de cada agente. Este problema fue clasificado dentro de los problema NP-Hard en 1986 por Fisher, Jaikumar and Van Wassenhove.

Las aplicaciones del problema son variadas y en diferentes áreas, de modo general se encuentran la calendarización, en problemas de capacidad o asignación de trabajo en redes computacionales.

El GAP fue estudiado por primera vez por Srinivasan y Thompson para resolver un problema de transportación [41], son Ross y Soland quienes introducen

el concepto de Generalized Assignment Problem [32]. Dicho concepto se trata de la generalización de una propuesta anterior de DeMaio y Roveda [26] en donde la capacidad de absorción de una tarea es independiente a un agente, esto es $a_{ij} = a_j, \forall i$.

Con el paso de los años y el avance en los algoritmos se introducen algunas propuestas de solución basadas en metaheurísticas, entre las que se encuentran la técnica Variable Depth Search Heuristic (VDSH) introducida por Amini y Racer [2], mecanismo que es una generalización de búsqueda local. Posteriormente Yagiura et al., propone una técnica denominada Variable Depth Search (VDS), método que usa movimientos de intercambio para explorar la mayor parte del espacio de búsqueda [44]. Osman hace uso de otras metodologías usadas para dar solución a este problema, se trata de una técnica hibrida que combina técnicas Simulated Annealing y Tabu search [31]. La técnica Tabu Search resultó una técnica eficiente por lo que posteriormente Yaguira et al., retoma la idea haciendo la introducción de operaciones sobre vecindarios [43]. Se han introducido otros tipos de conceptos entre los que es posible encontrar algoritmos paralelizados como el desarrollado por Asahiro [3] y mecanismos basado en búsqueda local desarrollada anteriormente por Yaguira. Posteriormente Diaz y Fernandez retoman la idea de Tabu Search agregando modificaciones de adaptación haciendo de su técnica un mecanismo flexible [11]. Finalmente las soluciones basadas en la naturaleza son introducidas por Chu y Beasley, quienes desarrollan un esquema basado en Algoritmo Genéticos [10], mientras que Monfred y Etemadi usan una técnica basada en Redes Neuronales [30]. Para una revisión más exhaustiva revisar [14][20].

4.2. Definición del problema

El GAP consiste en la minimización del costo de la asignación de n tareas a m agentes sin repetición y no excediendo la capacidad de cada agente. De modo formal el problema se define del siguiente modo: Sea $I = \{1, 2, \ldots m\}$ un conjunto de agentes y $J = \{1, 2, \ldots, n\}$un conjunto de tareas. Para $i \in I$, $j \in J$, se define c_{ij} como el costo de asignar una tarea j a un agente i (o asignar a un agente i una tarea j), r_{ij} como el recurso requerido por un agente i para realizar la tarea j, y b_j es la capacidad del agente i. Y finalmente la variable x_{ij} que se define mediante la función 4.1.

$$x_{ij} = \begin{cases} 1 & Agente\ i\ realiza\ la\ tarea\ j \\ 0 & En\ otro\ caso \end{cases} \tag{4.1}$$

Con base en la definición, la expresión matemática de GAP está dada por la fórmula 4.2

$$Minimizar \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{4.2}$$

$$Sujeto\ a \sum_{i \in I} x_{ij} = 1, \forall j \in J \tag{4.3}$$

$$\sum_{j \in J} r_{ij} x_{ij} \leq b_i, \forall i \in I \tag{4.4}$$

$$x_{ij} \in \{0, 1\}, \forall i \in I, \forall j \in J \tag{4.5}$$

En la figura 4.1, se muestra un ejemplo en donde $I = \{1, 2, \ldots, 5\}$ y $J = \{1, 2, \ldots, 8\}$, $C_{5,8}$ matriz de costos que implica que el agente i realice la tarea j, B_{1*5} matriz de capacidad de los agentes, $R_{5,8}$ matriz de capacidad que absorbe la tarea j si es realizada por el agente i, en donde las filas hacen referencia a los agentes y las columnas a las tareas, así como 2 conjuntos solución formados por parejas del tipo (i, j), tal que $i \in I$ y $j \in J$ que representan el agente que realizará cierta tarea. En el caso a) se presenta la solución $\sigma = (1,1), (1,7), (2,3), (3,5), (4,4), (5,2), (5,6), (5,8)$ con un coste igual a 64, mientras que la solución b) $\mu = (1,1), (1,2), (1,7), (2,8), (3,4), (4,6), (5,3)$ con un costo de 44, costo que si bien es menor no es admisible debido a que la capacidad para que los agentes se encarguen de determinada tarea es superior a la permitida por cada agente en específico en los agentes 1 y 3.

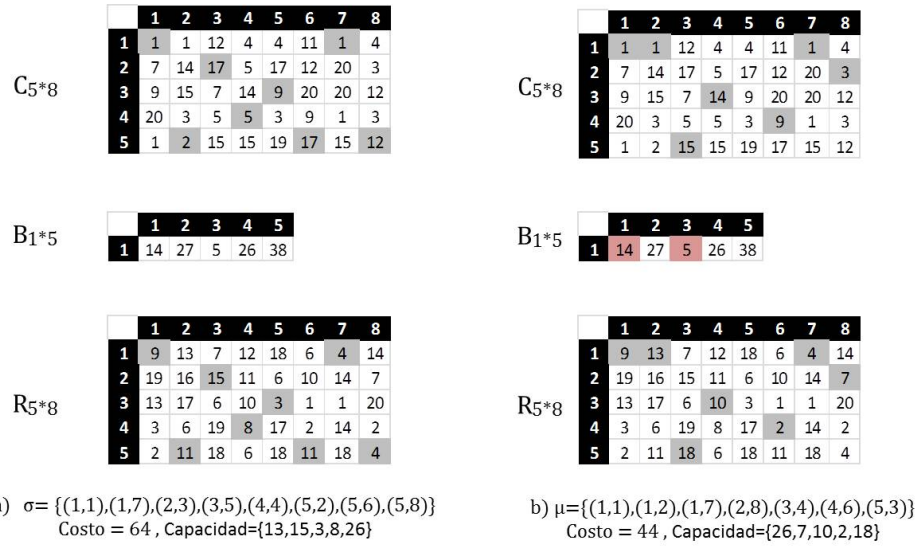


Figura 4.1: Ejemplo con solución σ admisible y μ no admisible.

Nótese que en GAP el espacio de búsqueda con n agentes y m tareas, es dado por m^n, en un ejemplo con $n = 3$ y $m = 10$, cuenta con $1,000$ posibles soluciones entre admisibles y no admisibles, mientras que en un ejemplo con $n = 10$ y $m = 100$ existen $100,000,000,000,000,000,000$ posibles soluciones.

4.2.1. Aplicaciones

Como se mencionó inicialmente el GAP tiene diversas aplicaciones entre las que podemos encontrar la asignación de problemas de calendarización y asignación de espacio o de trabajo.

4.2.1.1. Calendarización de máquinas paralelas con costo

El GAP se puede aplicar a la calendarización de máquinas en paralelo, máquinas que además requieren de un tiempo de ejecución, así como un costo por ejecutar. Cada trabajo es procesado por exactamente una máquina, es decir un trabajo j es procesado por una maquina i, máquina que requiere tiempo P_{ij} e inclusive un costo C_{ij}. Cada máquina es válida para ejecutarse por T_i unidades de tiempo, y el objetivo es minimizar el costo total necesario.

4.2.1.2. Asignación de trabajo en proyectos de desarrollo de software

Es posible ver la aplicación de GAP en metodologías de desarrollo de Software como SCRUM, técnica en la que se asignan tareas a individuos en base a sus características. Si bien GAP es visto de una manera más abstracta es posible realizar la adaptación con el objetivo de dar solución, supóngase que existen n desarrolladores y m actividades, en base a contrato cada desarrollador n trabaja un tiempo T_n, es decir el tiempo de labores puede ser menor, pero no mayor a este, además de que puede realizar tantas actividades como pueda en ese tiempo. Además cada desarrollador tiene definido un $Costo_{nm}$ para la actividad que le es asignada. Además de que existe un tiempo TT_{nm} que indica el tiempo que tardará cada empleado n en realizar una actividad m. El objetivo es minimizar el costo del proyecto asignando tantas tareas como sea posible a cada desarrollador, esto respetando el tiempo que cada empleo tiene en su contrato.

4.3. Solución propuesta

Al igual que la propuesta presentada en el capítulo 3, para el problema abordado en este capítulo se propone una solución basada en Algoritmos evolutivos que integra un esquema de Gestión de diversidad. De modo similar se desarrolló

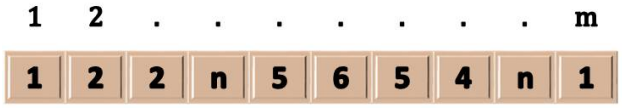
un plugin en C++ que fue incorporado a la herramienta METCO. A continuación se detallan los elementos de la propuesta presentada.

4.3.1. Representación de los individuos

A diferencia de LOP el GAP no representa las soluciones mediante permutación, otra característica de este problema es que los elementos dentro de los genes pueden repetirse. En este problema se cuenta con dos tamaños, n que es el número de agentes y m es el número de tareas a realizar. Es por esto que se tienen dos opciones para la dimensión del individuo, ambas con características diferentes y por tanto el manejo de los genes, dichas opciones se describen a continuación.

- **La dimensión del individuo es** $1xn$**.** Es decir cada gen del individuo representa a un agente, de este modo los alelos posibles serán el conjunto $J = \{1, 2, \ldots, m\}$, recordar que la definición de GAP menciona que cada agente puede realizar más de una tarea, por lo que si el individuo queda estructurado con una dimensión de $1xn$ resultaría tedioso almacenar cada una de las tareas asignadas a cada individuo, y en este caso se recurrirá a implementar otro tipo de estructura.
- **La dimensión del individuo es** $1xm$**.** Es decir cada gen del individuo representaría una tarea, con alelos en el conjunto $I = 1, 2, \ldots, n$, eventualmente existirían genes con alelos repetidos, debido a que un grupo de tareas pueden ser realizadas por un mismo agente, pero no una tarea por dos agentes.

Una vez analizadas las dos opciones así como las características que tendrían los individuos en ambos casos, resulta evidente que el individuo contendrá m genes, véase la figura 4.2 en donde se muestra la estructura del individuo usado.



Individuo GAP

Conjunto de Alelos = {1,2,..., n}

Figura 4.2: Estructura del individuo GAP.

4.3.2. Función de evaluación

Posterior a la definición del individuo y en base a la función de definición de GAP es posible definir la función de evaluación, dicha función evaluará con base

a 2 aspectos:

- El coste de las tareas realizadas por cada agente.
- La penalización por solución inadmisible, es decir, la suma de las capacidades de las tareas asociadas a un agente es superior a su capacidad.

Posterior a este análisis es posible definir la función de evaluación cubriendo los aspectos antes mencionados. El primer punto es cubierto completamente con el algoritmo de la función que define a GAP. En el segundo punto se encuentra descrito un aspecto importante a tomar en cuenta, cada individuo tiene un límite de capacidad para la realización de la tarea es decir aquellas soluciones que exceden la capacidad de cada individuo (aun cuando sus costos sean buenos) son consideras inadmisibles, es decir, a diferencia de LOP en GAP es posible existan individuos que quedan fuera de la zona de soluciones admisibles, de ahí que sea necesario desarrollar un mecanismo de penalización basado en la capacidad de cada individuo.

Como se ha mencionado el mecanismo de penalización está basado en la capacidad que tiene cada individuo para realizar tareas, esto con el fin de dirigir las soluciones a espacios de búsqueda admisibles. Es decir si un individuo i tiene una capacidad b_i y la suma de los costes de las tareas asignadas es $SumCap_i$, con $SumCap_i > b_i$ lo conveniente es penalizar bajo la diferencia existente entre esto dos valores, es decir $b_i - SumCap_i$, con el objetivo de incrementar aún más el valor y evitar sea elegido. Dicho valor es multiplicado por un $VALOR\_MAXIMO$ (VAL_MAX), finalmente la función de evaluación con penalización quedaría definida por la ecuación 4.6 y 4.7.

$$Fitness = Costo\_GAP + \sum_{i=1}^{m} Penalizacin \tag{4.6}$$

$$Penalizacin = \begin{cases} SumCap_i - b_i * VAL\_MAX & SumCap_i - b_i > 0 \\ 0 & SumCap_i - b_i \leq 0 \end{cases} \tag{4.7}$$

Por ejemplo, supongamos que se tiene un *Agente A* con capacidad de trabajo de 10 y un *Agente B* con capacidad de 20, y la suma de las capacidades de las tareas asignadas es 12 y 23 respectivamente, el valor del fitness con penalización sería la suma del coste retornado por la función de coste de $GAP + (12 - 10) * VAL\_MAX + (23 - 20) * VAL\_MAX$ de este modo se estaría penalizando soluciones inadmisibles y así incrementado el valor del coste lo que evitaría fueran elegidos debido a que GAP es un problema de minimización.

Retomando el caso *(b)* de la figura 4.1 y $VAL\_MAX = 1000$, el fitness penalizado sería $(44 + (26 - 14) * VAL\_MAX + (10 - 5) * VAL\_MAX)$, es decir 17044 ,véase la figura 4.3.

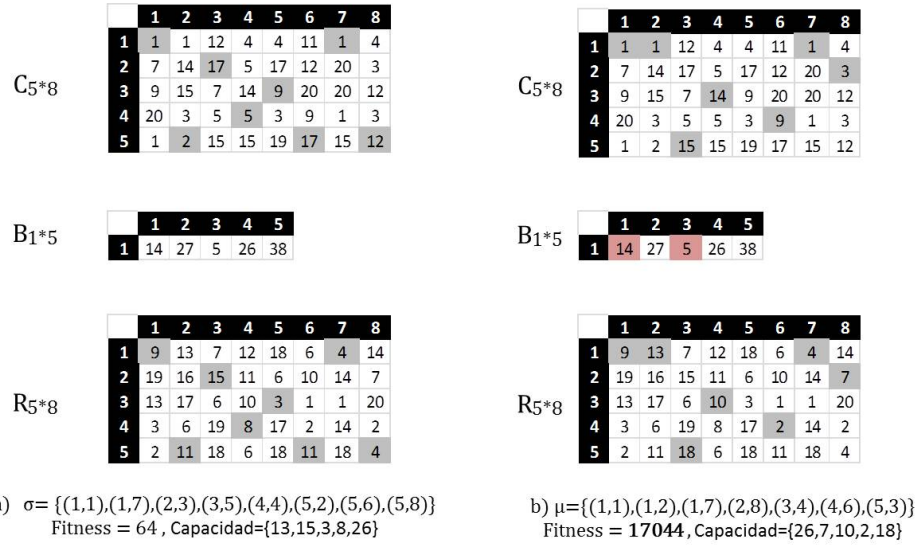


Figura 4.3: Costos tras penalización

4.3.3. Método de cruce

Como se ha mencionado en secciones anteriores, las soluciones de GAP a diferencia de LOP no son soluciones dadas por permutaciones, entre los mecanismos de mutación dirigido a este tipo de problemas se encuentran descrito en la sección 2.3.6.3. En base a lo anterior el algoritmo 10 fue desarrollado e incorporado a la herramienta METCO.

4.3.4. Mutación

En GAP no existe dependencia directa entre los elementos del individuo es decir a diferencia de LOP en donde no se puede el valor de un único gen, en GAP si se puede. Así, los genes de los individuos de GAP no tienen un vínculo. El valor del costo puede empeorar o mejorar si el alelo de ese gen es sustituido, esto debido a que el costo por asignar el nuevo alelo será menor o mayor, del mismo modo con la capacidad asignada en dicho alelo y como consecuencias de la penalización. Véase la figura 4.4 en donde se retoma el ejemplo de la figura 4.3, se muestran dos soluciones $\sigma = \{1, 2, 3, \mathbf{1}, 4, 2, 4, 5\}$ y $\mu = \{1, 2, 3, \mathbf{5}, 4, 2, 4, 5\}$, como se observa el gen 4 inicialmente contiene el alelo 1 generando un costo de 13054, mientras que cuando dicho gen contiene el alelo 5 el costo es de 6065, obteniendo una mejora en el individuo.

Algorithm 10 Mecanismo Uniform Crossover

Entrada

$Padre_A$, $Padre_B$ individuos existentes en la población.
P, porcentaje de cruce.

Salida

$Hijo_A$, $Hijo_B$ individuos nuevos.

```
1:  for t ← 1 ; m do
2:      r ← Aleatorio_0aN(100)
3:      if r ≤ P then
4:          Hijo_A[t] ← Padre_B[t]
5:          Hijo_B[t] ← Padre_A[t]
6:      else
7:          Hijo_A[t] ← Padre_A[t]
8:          Hijo_B[t] ← Padre_B[t]
9:      end if
10: end for
```

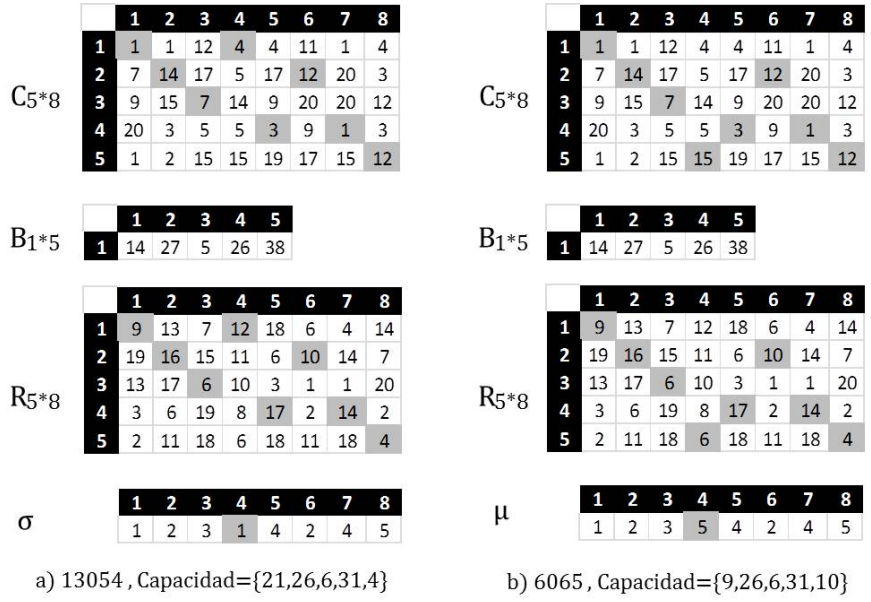


Figura 4.4: Ejemplo de la sustitución de un alelo

Debido a este análisis se realiza la elección del mecanismo de mutaciones *Restablecimiento Aleatorio*, presentado en la sección 2.3.5.2. El algoritmo 11 conserva o asigna un nuevo alelo en base a un porcentaje P_i, esto para cada uno de los genes.

Algorithm 11 Mecanismo de mutación Restablecimiento Aleatorio

Entrada $Individuo_k$, Individuo a mutar.
P, porcentaje de mutación.

1: **for** t ← 1 ; $Individuo_k.size()$ **do**
2: r ← Aleatorio_0aN(100)
3: **if** r ≤ P **then**
4: aux ← Aleatorio_0aM($Individuo_k.size()$)
5: $Individuo_k$[t] ← aux
6: **end if**
7: **end for**

4.3.5. Búsqueda Local

Se desarrollaron dos mecanismos de búsqueda local, mecanismos cuyo funcionamiento es estocástico, además de que ambos mecanismos son ejecutados mientras exista una mejora en el fitness.

4.3.5.1. Cambio en un gen

El objetivo principal de este mecanismo es conseguir una mejora dentro de las soluciones vecinas donde los vecinos se crean combinando el valor de un único gen. Esto es posible mediante el listado de cada una de los pares (i, j) en donde $i \in I$ y $j \in J$, siendo los agentes y tareas respectivamente. Posteriormente dicho conjunto de pares es ordenado de modo aleatorio (con el fin de realizar cambios en el individuo y estos sean aleatorios). Este proceso es repetido mientras exista un par (i, j) que contribuya a la mejora del individuo. El algoritmo 12 es el usado en el desarrollo del plugin GAP. Es importante mencionar que en este algoritmo no se realizan evaluaciones completas (las definidas en la sección 4.3.2), sino parciales, evaluaciones que resultan muy rápidas en comparación a una completa.

4.3.5.2. Intercambio

Este mecanismo está basado en el intercambio de tareas, es decir si el agente A tiene asignado una tarea W y el agente B la tarea F, el objetivo es intercambiar las tareas de ambos agentes en busca de una mejora, en caso de no obtenerse se re-asignan a su agente original.

El algoritmo 13, funciona mediante la generación de todas las posibles parejas $(Tarea_x, Tarea_y)$ y posteriormente una ordenación aleatoria de estas parejas, finalmente se realizan los cambios y se repite el proceso mientras exista un cambio positivo, es decir, se obtenga una mejora en el fitness del individuo. Al igual

Algorithm 12 Mecanismo Local Search - Pares

Entrada

Individuo, individuo existentes en la población.
$Costo$ valor del fitness del individuo.

Salida

Individuo, que es el óptimo local.

```
1:  Cambio ← TRUE
2:  while  Cambio = TRUE do
3:      Cambio ← FALSE
4:      Pares ← GeneraPares()
5:      for t←1 ; Pares.size() do
6:          aux ← Individuo[Pares[t].getTarea()]
7:          Individuo[Pares[t].getTarea()] ← Pares[t].getAgente()
8:          CostoAux ← evaluarIndividuo(Individuo)
9:          if ( then CostoAux - Costo >0 )
10:             Cambio ← TRUE
11:         else
12:             Individuo[Pares[t].getTarea()] ← aux
13:         end if
14:     end for
15: end whilereturn Individuo
```

que en el mecanismo anterior, las evaluaciones realizadas en esta propuesta son también evaluaciones parciales.

Algorithm 13 Mecanismo Local Search - intercambio de Tareas

Entrada

Individuo, individuo existentes en la población.
$Costo$ valor del fitness del individuo.

Salida

Cambio, TRUE si existe cambio o ganancia, FALSE en otro caso.

```
1: Cambio ← TRUE
2: while Cambio = TRUE do
3:     Cambio ← FALSE
4:     Pares ← GeneraParesTareas()
5:     for t←1 ; Pares.size() do
6:         aux ← Individuo[Pares[t].getTareaX()]
7:         Individuo[Pares[t].getTareaX()] ← Individuo[Pares[t].getTareaY()]
8:         Individuo[Pares[t].getTareaY()] ← aux
9:         CostoAux ← evaluarIndividuo(Individuo)
10:        if ( then CostoAux - Costo >0 )
11:            Cambio ← TRUE
12:        else
13:            aux ← Individuo[Pares[t].getTareaX()]
14:            Individuo[Pares[t].getTareaX()] ← Individuo[Pares[t].getTareaY()]
15:            Individuo[Pares[t].getTareaY()] ← aux
16:        end if
17:    end for
18: end while
```

4.3.6. Mecanismos de gestión de diversidad

Como se menciona en la sección 3.3.6 la pérdida de diversidad es frecuente durante el proceso de evolución de los individuos, debido a esto es conveniente introducir un esquema de Gestión de diversidad. Recordar que el mecanismo introducido es el MULTI_DYNAMIC, dicho mecanismo hace uso de la métrica *Distancia al vecino cercano (Distance to the Closest Neighbor, DCN*), dicha métrica calcula distancia existente entre individuos para posteriormente penalizar a los que contribuyen menos en diversidad y finalmente seleccionar a aquellos que además de tener un fitness adecuado contribuyan a la diversidad de la población de posibles soluciones.

El aspecto de mayor relevancia en este mecanismo es el cálculo del DCN, debido a que este valor determina la contribución de un individuo a la diversidad. Para el cálculo del DCN en GAP se hace uso de la idea anteriormente presentada, aquella que menciona la importancia del contenido de los genes sobre la posición de los alelos. Por este motivo el algoritmo 14 el mecanismo desarrollado para el cálculo de DCN es la distancia de Hamming, métrica que cuenta las diferencias en dos cadenas dadas, en este caso las diferencias entre los genes de dos individuos. Véase la imagen 4.5 en donde se muestran dos individuos y se señalan aquellas posiciones donde el contenido de los genes es diferente. En este caso la distancia es 7.

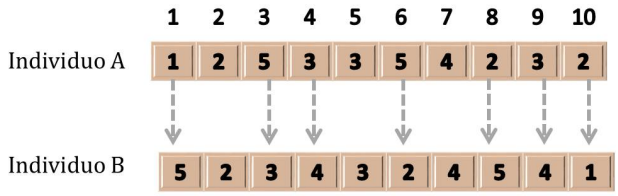


Figura 4.5: Ejemplo de cálculo de DCN para GAP

Algorithm 14 Mecanismo DCN para GAP

Entrada $Individuo_A$, $Individuo_B$, individuos existentes en la población.

Salida *Cont*, valor de DCN.

```
1: Cont ← 0
2: for t←1 ; m do
3:     if ( then Individuo_A[t] != Individuo_B[t] ) Cont ← Cont + 1
4:     end if
5: end for
```

4.3.7. Resultados

Para la realización de las pruebas se usaron dos algoritmos, el primero sin mecanismos de gestión de diversidad, nombrado **GElit** y el segundo con mecanismo de gestión de diversidad denominado **Div**, este segundo hace uso del mecanismo de cálculo de DCN mencionado en 4.3.6. Ambas estructuras hacen uso del mecanismo de cruce mencionado en la sección 4.3.3, la mutación mencionada en la sección 4.3.4 y los mecanismos de búsqueda local de la sección 4.3.5.

En la fase de pruebas se realizaron corridas con 8 instancias, las instancias son nombradas en base al número de agentes y a las tareas que contienen. Las instancias son clasificadas del siguiente modo.

- Pequeñas
  - b05200: 5 agentes y 200 tareas.
  - c10100: 10 agentes y 100 tareas.
  - d10200: 10 agentes y 200 tareas.
  - e05100: 5 agentes y 100 tareas.
- Grandes
  - b20200: 20 agentes y 200 tareas.
  - c20200: 20 agentes y 200 tareas.
  - d20100: 20 agentes y 100 tareas.
  - e15900: 15 agentes y 900 tareas.

Las corridas fueron ejecutadas en diferentes tiempos, mostrando mejor comportamiento en los algoritmos y resultados en tiempo de 16 horas. Para dichas ejecuciones se usaron diferentes configuraciones de los algoritmos, mostrando resultados óptimos con la siguiente configuración.

- Porcentaje de cruce: 20 %.
- Porcentaje de mutación: 5 %.
- Tamaño de Población: 100 individuos.
- Local Search: Uso de ambos métodos propuestos.

4.3.7.1. Resultados generales

En la tabla 4.1 se muestran los resultados generales, en donde se observa que en el 100 % de las instancias el esquema con diversidad ofrece mejores resultados en el promedio obtenido.

4.3.7.2. Pruebas estadísticas

Posteriormente se realizaron un conjunto de pruebas estadísticas en donde se comparan los resultados obtenido por GElit y Div, esto bajo la clasificación de instancias grandes y pequeñas, es en el 100 % de las instancias que se obtiene un mejor resultado en esquemas con gestión de diversidad. En la tabla 4.2 se observan dichos resultados, mostrando nuevamente al algoritmo con mecanismo de diversidad como el de mejor rendimiento.

Instancia	GElit		Div	
	Mejor	Promedio	Mejor	Promedio
b05200	3559	3561,8	3558	3559,6
b20200	2341	2342,4	2341	2341,4
c10100	1404	1404,9	1403	1403,7
c20200	2409	2417	2407	2412,3
d10200	12699	12727,5	12663	12683,1
d20100	6353	6386,7	6324	6344,7
e05100	12745	12776,8	12719	12740,2
e15900	118880	120315,4	117414	118095,9

Tabla 4.1: Tabla de resultados generales, corridas a 16 horas.

	Pequeñas				Grandes			
	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)	$\uparrow$	$\downarrow$	$\longleftrightarrow$	(-)
Div	4	0	0	4	4	0	0	4
Gelit	0	4	0	-4	0	4	0	-4

Tabla 4.2: Tabla comparativa entre métodos.

4.3.7.3. Comportamiento de los esquemas

A continuación se muestra el comportamiento durante la ejecución de los esquemas con y sin diversidad implementada, desde la gráfica 4.6 a la gráfica 4.13. En primer lugar, analizando el esquema con gestión de diversidad resulta evidente su comportamiento constante, es decir, los valores de fitness decrecen de modo constante mientras que el esquema que carece de gestión de diversidad presenta decrementos bruscos o estancamientos en resultados por largos periodos de tiempo.

Otro punto observable, es la calidad de los individuos iniciales, en el esquema Div en 7 de las 8 instancias los individuos presentan cierta calidad desde el inicio, calidad que es mantenida durante toda la ejecución y en el caso que esto no ocurre el resultado superior es obtenido a lo largo de la ejecución.

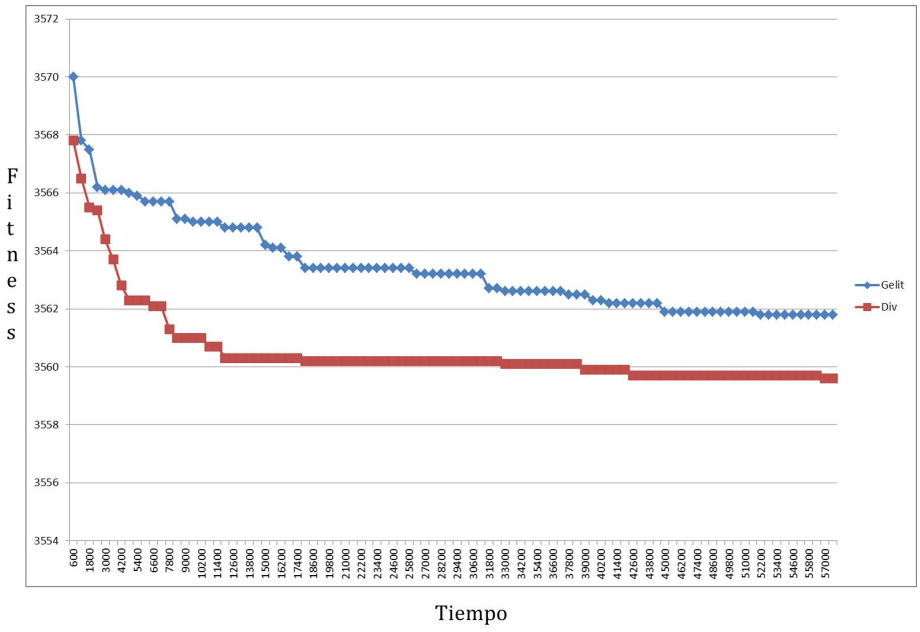


Figura 4.6: Comportamiento de la evaluación de la instancia b05200 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

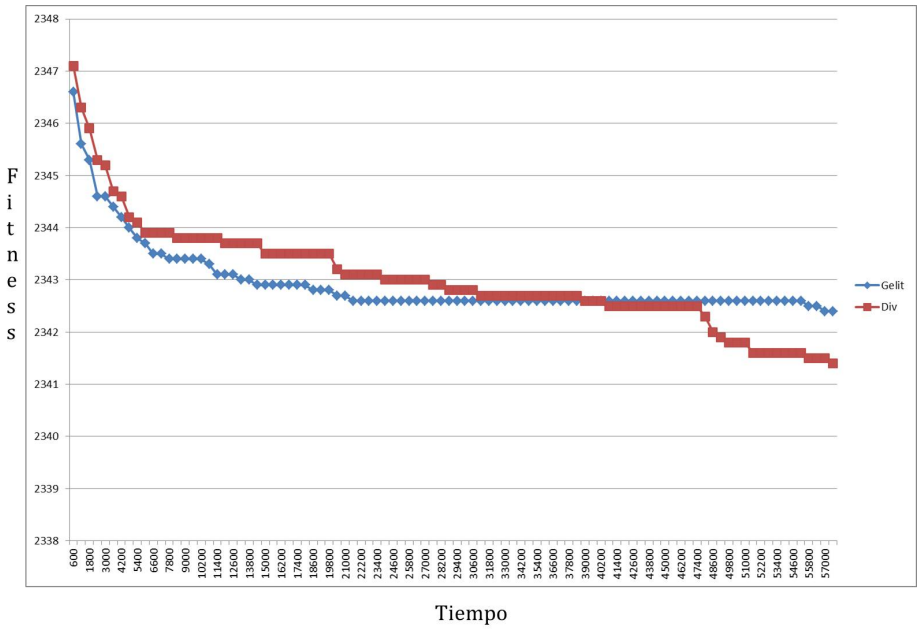


Figura 4.7: Comportamiento de la evaluación de la instancia c10100 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

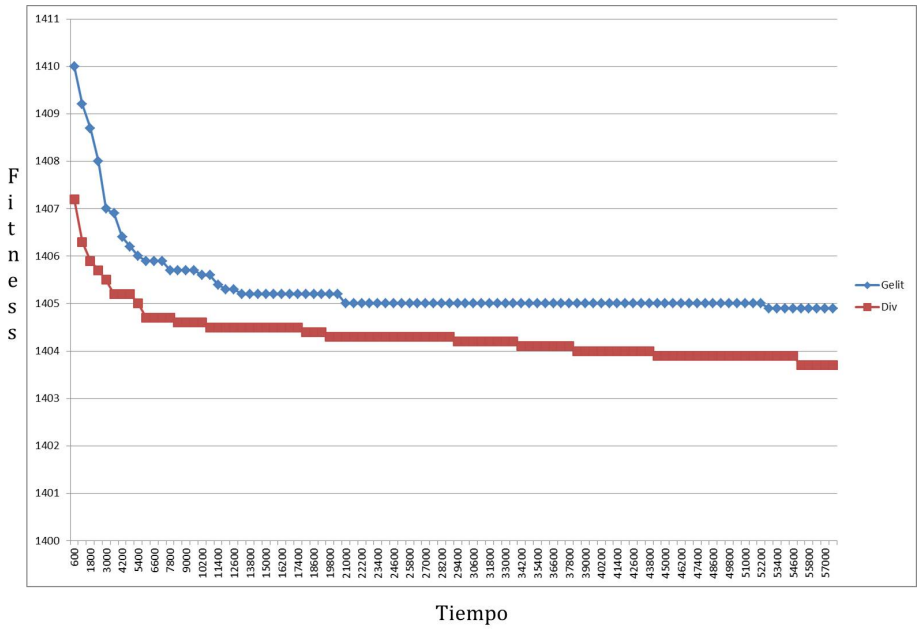


Figura 4.8: Comportamiento de la evaluación de la instancia d10200 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

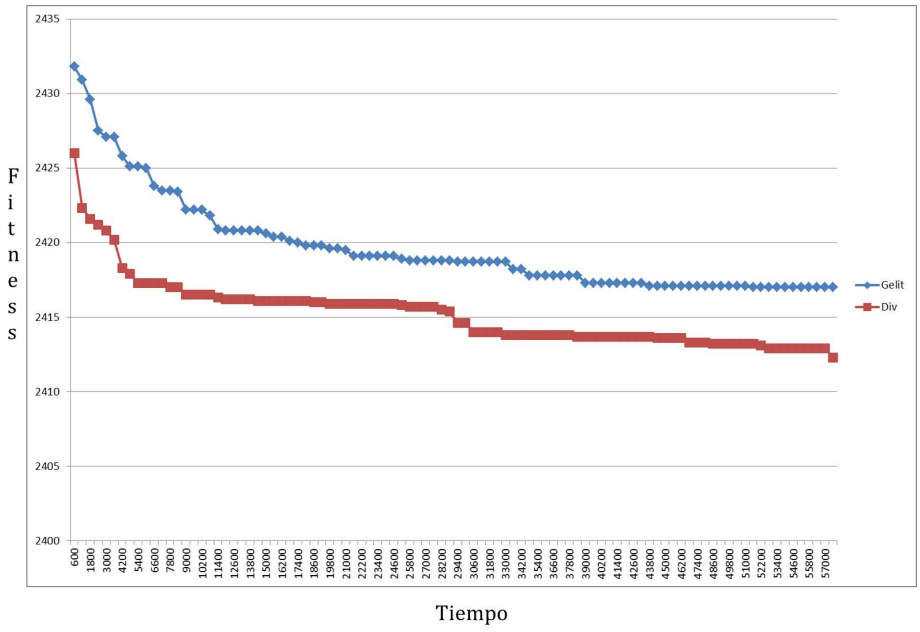


Figura 4.9: Comportamiento de la evaluación de la instancia e05100 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

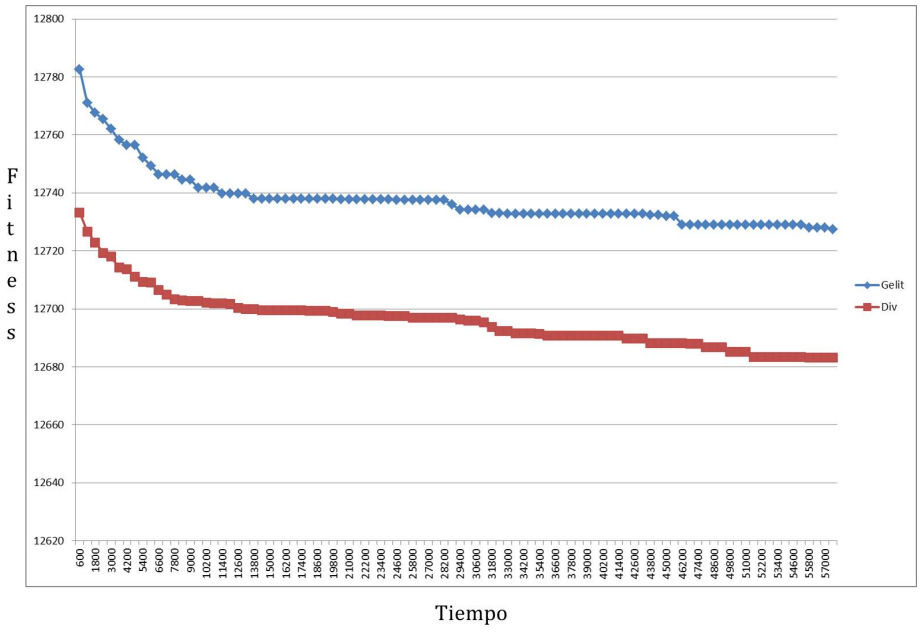


Figura 4.10: Comportamiento de la evaluación de la instancia b20200 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

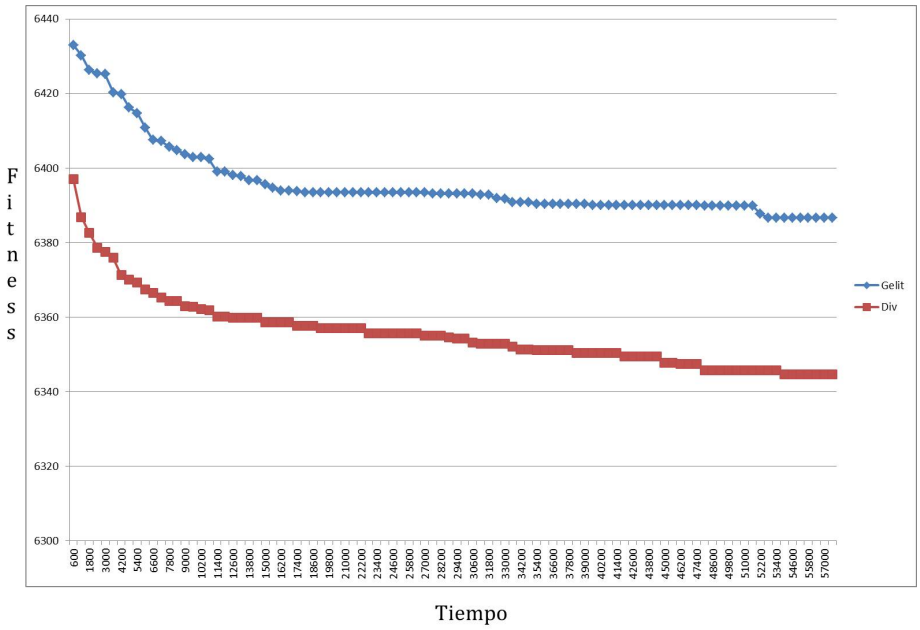


Figura 4.11: Comportamiento de la evaluación de la instancia c20200 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

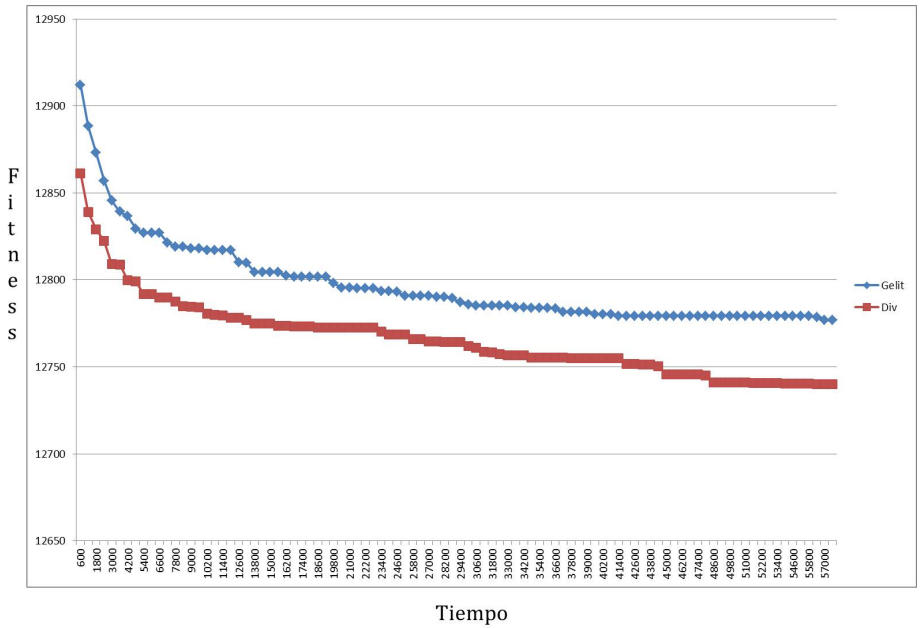


Figura 4.12: Comportamiento de la evaluación de la instancia d20100 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

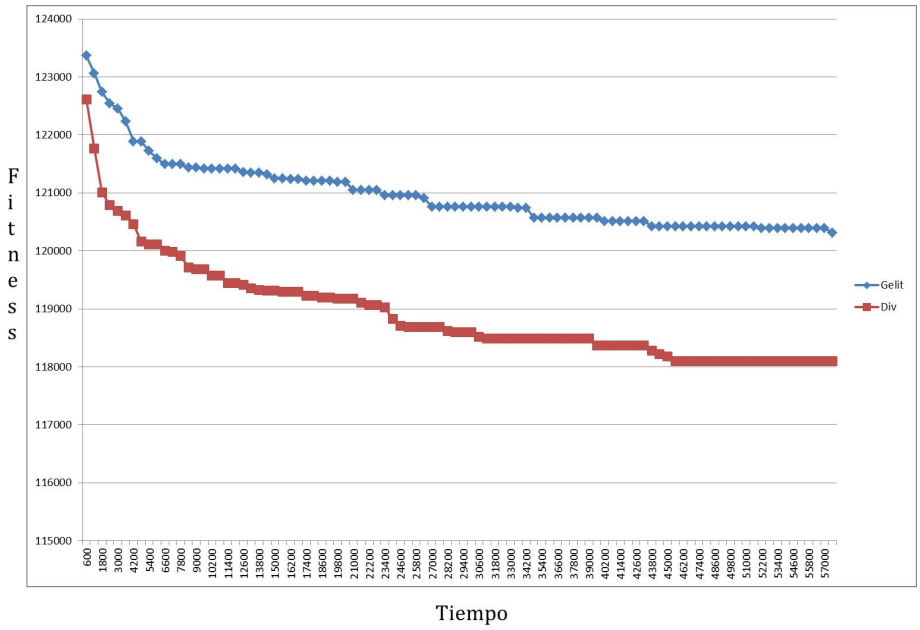


Figura 4.13: Comportamiento de la evaluación de la instancia e15900 para una población de 100 individuos y tiempo de 16hr con impresión de resultados cada 10 minutos.

Capítulo 5

Conclusiones

Durante el presente trabajo se desarrollaron y analizaron diferentes métricas para la gestión de diversidad en dos problemas del grupo combinatorio, GAP y LOP. Para ambos problemas se realizó una comparativa entre algoritmos con y sin gestión de diversidad implementada, además se realizó un análisis de diferentes algoritmos formados por distintas combinaciones de mecanismos de cruce, mutaciones, búsqueda local y gestión de diversidad.

Tras el análisis de los resultados obtenidos de las diversas pruebas realizadas, es observable que en ambos problemas los algoritmos en donde se implementó la propuesta del mecanismo de Gestión de Diversidad se obtuvieron resultados de mayor calidad sobre aquellos que carecían de este mecanismo. Para el caso de LOP, en donde se comparó la propuesta de este trabajo con el algoritmo presentado en [7], algoritmo cuyos resultados son los mejores hasta la fecha, se obtuvieron resultados de mayor calidad en la propuesta hecha a LOP en el presente trabajo.

Dadas las restricciones de tiempo asociadas al desarrollo de este trabajo, se identificaron ciertas líneas de investigación prometedoras en las que no se pudo profundizar. Algunas líneas que se podrían desarrollar en el futuro son:

- Ampliar MULTI_DYNAMIC diseñando un nuevo esquema que permita que en las fases finales de optimización se balancee aún más hacia la explotación del espacio de búsqueda.

- Combinar la utilización de MULTI_ DYNAMIC con un esquema adaptativo para la variación. Específicamente se probarán esquemas que en las fases iniciales realicen mutaciones más disruptivas que en las fases finales. Esto se aplicará modificando la probabilidad de mutación a lo largo de la ejecución o usando varios operadores.

Bibliografía

[1] Preventing premature convergence in genetic algorithm using DGCA and elitist technique. *International Journal of Advanced Research in Computer Science and Software Engineering 4*, 6 (Junio 2014), 410 – 418.

[2] AMINI, M. M., AND RACER, M. A hybrid heuristic for the generalized assignment problem. *European Journal of Operational Research 87*, 2 (1995), 343 – 348.

[3] ASAHIRO, Y., ISHIBASHI, M., AND YAMASHITA, M. Independentand cooperative parallel search methods for the generalized assignment problem. *Optimization Methods and Software 18*, 2 (2003), 129–141.

[4] BIRATTARI, M., STÜTZLE, T., PAQUETE, L., AND VARRENTRAPP, K. A racing algorithm for configuring metaheuristics. *GECCO 2* (2002), 11–18.

[5] BLICKLE, T., AND THIELE, L. A comparison of selection schemes used in evolutionary algorithms. *Evol. Comput. 4*, 4 (1996), 361–394.

[6] CAMPOS, V., GLOVER, F., LAGUNA, M., AND MARTÍ, R. An experimental evaluation of a scatter search for the linear ordering problem. *Journal of Global Optimization 21*, 4 (2001), 397–414.

[7] CEBERIO, J., MENDIBURU, A., AND LOZANO, J. A. The Linear Ordering Problem Revisited. *European Journal of Operational Research* (2014).

[8] CHAOVALITWONGSE, W. A., PARDALOS, P. M., RESENDE, M. G. C., DON, AND GRUNDEL, A. Revised grasp with path-relinking for the linear ordering problem, 2009.

[9] CHARON, I., AND HUDRY, O. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR 5*, 1 (2007), 5–60.

[10] CHU, P. C., AND BEASLEY, J. E. A genetic algorithm for the generalised assignment problem. *Comput. Oper. Res. 24*, 1 (Jan. 1997), 17–23.

[11] DÍAZ, J. A., AND FERNÁNDEZ, E. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research 132*, 1 (2001), 22 – 38.

[12] EIBEN, A. E., AND SMITH, J. E. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[13] F. G. LOBO, C. F. L., AND MICHALEWICZ, Z. *Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence*. Springer, 2007.

[14] FLOUDAS, C., AND PARDALOS, P. *Encyclopedia of Optimization*. No. v. 1 in Encyclopedia of Optimization. Springer, 2008.

[15] FOGEL, D. B. *Evolutionary Computation: The Fossil Record*. 1998, Wiley-IEEE Press.

[16] GAO, W., YEN, G., AND LIU, S. A cluster-based differential evolution with self-adaptive strategy for multimodal optimization. *Cybernetics, IEEE Transactions on 44*, 8 (Aug 2014), 1314–1327.

[17] GAREY, M., JOHNSON, D., AND STOCKMEYER, L. Some simplified np-complete graph problems. *Theoretical Computer Science 1*, 3 (1976), 237 – 267.

[18] GARZA-FABRE, M., TOSCANO-PULIDO, G., AND RODRIGUEZ-TELLO, E. Multi-objectivization, fitness landscape transformation and search performance: A case of study on the hp model for protein structure prediction. *European Journal of Operational Research 243*, 2 (2015), 405 – 422.

[19] GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research 13*, 5 (1986), 533 – 549. Applications of Integer Programming.

[20] GLOVER, F., AND KOCHENBERGER, G. *Handbook of Metaheuristics, International series in operations research and management science*. Kluwer Academic Publishers, 2003.

[21] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

[22] KAAS, R. A branch and bound algorithm for the acyclic subgraph problem. *European Journal of Operational Research 8*, 4 (1981), 355–362.

[23] LAGUNA, M., MARTI, R., AND CAMPOS, V. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & Operations Research 26*, 12 (1999), 1217 – 1230.

[24] LEON, C., MIRANDA, G., AND SEGURA, C. Metco: A parallel plugin-based framework for multi-objective optimization. *International Journal on Artificial Intelligence Tools 18*, 04 (2009), 569–588.

[25] M. CREPINSEK, S.-H. L., AND MERNIK, M. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv. 45*, 3 (2013), 35:1–35:33.

[26] MAIO, A. D., AND ROVEDA, C. An all zero-one algorithm for a certain class of transportation problems. *Operations Research 19*, 6 (1971), 1406–1418.

[27] MANNER, R., MAHFOUD, S., AND MAHFOUD, S. W. Crowding and preselection revisited. In *Parallel Problem Solving From Nature* (1992), North-Holland, pp. 27–36.

[28] MENGSHOEL, O. J., AND GOLDBERG, D. E. The crowding approach to niching in genetic algorithms. *Evol. Comput. 16*, 3 (Sept. 2008), 315–354.

[29] MERZ, P., AND FREISLEBEN, B. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem, 1999.

[30] MONFARED, M. A. S., AND ETEMADI, M. The impact of energy function structure on solving generalized assignment problem using hopfield neural network. *European Journal of Operational Research 168*, 2 (2006), 645–654.

[31] OSMAN, I. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations-Research-Spektrum 17*, 4 (1995), 211–225.

[32] ROSS, G., AND SOLAND, R. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming 8*, 1 (1975), 91–103.

[33] SAFRO, I., RON, D., AND BRANDT, A. Fast multilevel algorithms for linear ordering problems.

[34] SCHIAVINOTTO, T., AND STÜTZLE, T. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms 3*, 4 (2005), 367–402.

[35] SEGREDO, E., SEGURA, C., AND LEÓN, C. Memetic algorithms and hyperheuristics applied to a multiobjectivised two-dimensional packing problem. *J. of Global Optimization 58*, 4 (Apr. 2014), 769–794.

[36] SEGURA, C., COELLO, C. A. C., MIRANDA, G., AND LEÓN, C. Using multi-objective evolutionary algorithms for single-objective optimization. *4OR 11*, 3 (2013), 201–228.

[37] SEGURA, C., COELLO, C. A. C., SEGREDO, E., AND HERNANDEZ, A. A novel diversity-based replacement strategy for evolutionary algorithms.

[38] SINGH, G., AND DEB, DR., K. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (New York, NY, USA, 2006), GECCO '06, ACM, pp. 1305–1312.

[39] SLATER, P. Inconsistencies in a schedule of paired comparisons. *Biometrika 48*, 3-4 (1961), 303–312.

[40] SOUZA, A. *Combinatorial Algorithms, Lecture Notes*. Humboldt University Berlin, 2010.

[41] SRINIVASAN, V., AND THOMPSON, G. L. An algorithm for assigning uses to sources in a special class of transportation problems. *Operations Research 21*, 1 (1973), 284–295.

[42] WASSENHOVE, L. The linear ordering problem revisited. *European Journal of Operational Research 60 (1992)* (1990), 260–272.

[43] YAGIURA, M., IBARAKI, T., AND GLOVER, F. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing 16*, 2 (2004), 133–151.

[44] YAGIURA, M., YAMAGUCHI, T., AND IBARAKI, T. A variable depth search algorithm with branching search for the generalized assignment problem. *Optimization Methods and Software 10* (1998), 419–441.

[45] YANG, X. *Engineering Optimization: An Introduction with Metaheuristic Applications*. Wiley Online Library Books, 2010.

[46] ZELINKA, I., SNSEL, V., AND ABRAHAM, A. *Handbook of Optimization: From Classical to Modern Approach*. Springer Publishing Company, Incorporated, 2012.