INSTITUTO TECNOLÓGICO
DE CIUDAD MADERO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Tesis:

Estrategias de diversificación para la solución memética del problema de ordenamiento lineal.

Para obtener el grado de:

Maestro en Ciencias en Ciencias de la Computación

Presenta:

I.S.C. Shulamith Samanthan Bastiani Medina

Director de Tesis:

Dr. Héctor Joaquín Fraire Huacuja.

Co-director:

MC. Guadalupe Castilla Valdez

Ciudad Madero, Tamaulipas, México Mayo 2011

"2011, Año del Turismo en México"

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Cd. Madero, Tamps; a 22 de Marzo de 2011

OFICIO No.: U5. 102/11
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

C. SHULAMITH SAMANTHA BASTIANI MEDINA
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestría en Ciencias en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ESTRATEGIAS DE DIVERSIFICACIÓN PARA LA SOLUCIÓN MEMÉTICA DEL PROBLEMA DE ORDENAMIENTO LINEAL"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"Por mi Patria y por mi Bien"

M. P. MARÍA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

S.E.P.
DIVISION DE ESTUDIOS
DE POSGRADO E
INVESTIGACION
I T C M

c.c.p.: Archivo

MYCHC 'NLCO 'ayqc"

Ave. 10. De Mayo y Sor Juana I. De la Cruz, Col. Los Mangos, C.P. 89440 Cd. Madero, Tam.
Tels. (833) 3 57 48 20, Fax: (833) 3 57 48 20, Ext. 1002, email: itcm@itcm.edu.mx
www.itcm.edu.mx

Resumen

Interpretar los cambios en la economía de un país ha representado siempre un reto para los economistas debido a los cambios acelerados y a las múltiples interacciones entre los diferentes sectores económicos. Vasili Leontief desarrolló en 1936 un modelo de insumos-productos con el fin de apoyar a los economistas en la tarea de analizar la economía de un país. Dentro de este modelo existe el problema de la triangulación es de gran interés en el área computación por ser equivalente al Problema de Ordenamiento Lineal.

LOP además de tener una aplicación dentro de la economía, existen otras aplicaciones importantes en diversas áreas como lo son: en la electrónica, se utiliza para el diseño de circuitos integrados, en el campo de la automatización para el dibujado automático de grafos, en el área de redes inalámbricas se aplica tanto para la indexación como la ubicación de los datos y en el área de las ciencias sociales se utiliza en el problema de la agregación de preferencias individuales. Como anteriormente se comentó el desarrollo de la economía ha sido muy acelerado, tan solo en los años 80's se reportaron países cuyas economías contenían entre 400 y 500 sectores [Leotief, 1986].

La alta dimensión de las tablas generadas hacía imposible abordar el problema correspondiente de triangulación mediante los métodos clásicos de la programación lineal. En estos casos, las metaheurísticas representan una opción viable, ya que permiten obtener soluciones de buena calidad con una inversión de tiempo adecuada a las necesidades de las aplicaciones económicas y de toma de decisión.

Contenido

Índice de Figuras.

Índice de Tablas

Capitulo 1

Introducción

En este capítulo, se presentan las principales motivaciones para realizar esta investigación, así como también la descripción del problema a tratar, la definición formal, el contexto de la investigación, los objetivos, las delimitaciones y por último una descripción de la organización de la tesis.

Interpretar los cambios en la economía de un país ha representado siempre un reto para los economistas debido a los cambios acelerados y a las múltiples interacciones entre los diferentes sectores económicos. Vasili Leontief desarrolló en 1936 un modelo de insumos-productos con el fin de apoyar a los economistas en la tarea de analizar la economía de un país. Dentro de este modelo el problema de la triangulación es de gran interés en el área computación por ser equivalente al Problema de Ordenamiento Lineal. El modelo de Leontief ha sido de gran importancia ya que facilitó la interpretación de las complejas interacciones que hay entre los diferentes sectores de un país. En este modelo se divide la economía de un país en sectores y se construye una tabla en la cual las filas y columnas son los sectores identificados y el valor correspondiente a cada par de sectores representa el total de insumos que el sector correspondiente a la fila le entrega al sector que corresponde a la columna. Por otro lado el total producido por un sector se obtiene por la sumatoria de los valores en la fila correspondiente. Como resultado de este modelo, se obtiene una tabla que resume la interdependencia entre los sectores económicos de un país. [Leontief, 1986]

En el contexto de la economía, el problema de la triangulación que surge en el modelo de insumos-productos ha sido estudiado por Chiarini [Charini, 2004].

En este problema la economía de una región, generalmente una ciudad, es dividida en sectores y se construye una tabla cuadrada E de insumos-productos cuyo tamaño corresponde al número de sectores encontrados. En esta tabla, la entrada e_{ij} denota la cantidad de insumos (en valor monetario) que el sector i le proporciona al sector j en un período dado. Por tanto el problema de la triangulación consiste en la permutación

simultanea de las columnas y las filas de la Tabla *E*, tal que la suma de las entradas arriba de la diagonal principal sea lo más grande posible. La *linealidad* es un indicador que permite encontrar la jerarquía que presentan los sectores de una economía, de manera que los que son preferentemente productores aparecen primero y los consumidores aparecen al final de dicha jerarquía. Por lo tanto, valores cercanos a uno de este indicador representan economías con una fuerte dependencia hacia un sector, y será más susceptible los cambios en la demanda, mientras que las economías saludables tienden a valores medios en este indicador, ya que no existe una dependencia marcada hacía un determinado sector. Chiarini proporciona una definición formal de este concepto, y dice:

"Sea *n* el numero de sectores y $E = \{e_{ij}\}$ es la matriz cuadrada de tamaño *n* x *n* que representa la tabla de *insumos-productos*. Entonces, la linealidad λ de una economía está dada por:

$$\lambda = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} e_{ij}}{\sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} e_{ij}} \qquad (1.1)$$

La *linealidad* es la relación entre la suma de todos los elementos sobre la diagonal principal de la matriz y la suma del total de elementos de la matriz (excepto los elementos de la diagonal)." [Chiarini, 2004]

Por otro lado LOP tiene otras aplicaciones importantes en diversas áreas como lo son: en la electrónica se utiliza para el diseño de circuitos integrados, en el campo de la automatización para el dibujado automático de grafos, en el área de redes inalámbricas se aplica tanto para la indexación como la ubicación de los datos y en el área de las ciencias sociales se utiliza en el problema de la agregación de preferencias individuales. Como anteriormente se comentó el desarrollo de la economía ha sido muy acelerado, tan solo en los años 80's se reportaron países cuyas economías contenían entre 400 y 500 sectores [Leotief, 1986]. La alta dimensión de las tablas generadas hacía imposible abordar el problema correspondiente de triangulación mediante los métodos clásicos de la programación lineal. En estos casos, las metaheurísticas representan una opción viable, ya que permiten obtener soluciones de buena calidad con una inversión de tiempo adecuada a las necesidades de las aplicaciones económicas y de toma de decisión. Los métodos de

solución que se desarrollan en este proyecto constituyen el núcleo de un software de aplicación para el cálculo de la linealidad.

1.1 Problema de Investigación

El problema de ordenamiento lineal es un problema NP-duro [Karp, 1972], [Garey, 1979], [Laguna, 1998],[Festa, 1999], [García, 2001], [Chaovalitwongse, 2009], [García, 2005], [Campos, 2005]. Los problemas de ordenamiento lineal pueden modelarse como problemas de optimización combinatoria, y en ellos se pretende minimizar ó maximizar una función encontrando una permutación adecuada de los vértices de un grafo.

1.2 Definición Formal

La definición del problema LOP de acuerdo a Laguna [Laguna, 1998] dice:

"Dada una matriz de pesos $E = \{e_{ij}\}$, de tamaño m x m, el problema de ordenamiento lineal consiste en encontrar una permutación π de las columnas (y filas) de tal forma que se maximice la suma de los pesos en el triangulo superior de la matriz. Hablando matemáticamente, se debe maximizar:

$$f(\pi) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} e_{\pi_i \pi_j} \qquad (1.2)$$

Donde π_i es el índice de las columnas (y filas) en la posición i en la permutación."

1.2.1 Complejidad de LOP

La complejidad de LOP ha sido analizada por Garey y Karp, quienes desarrollan la prueba de complejidad NP-duro para el problema del conjunto de arcos de retroalimentación de peso mínimo, y por lo tanto para el problema LOP, dado que éste es equivalente al problema de la demostración[Karp, 1972], [Garey, 1975]; Mihalis Yannakakis presenta la prueba para el problema de borrado de arcos y nodos que puede ser transformado en LOP [Yannakakis, 1978]; posteriormente Festa establece que LOP es NP-duro debido a su

equivalencia con el problema del conjunto de arcos de retroalimentación [Festa, 1999]; finalmente, Chiarini [Chiarini, 2004], GrÄotschel [GrÄotschel, 1984] y Desagupta [Desagupta, 2006] establecen la misma equivalencia.

1.3 Justificación

Este trabajo ha sido ampliamente estudiado en el ámbito de la computación, y numerosas soluciones heurísticas y metaheurísticas han sido desarrolladas. Entre las más destacadas se encuentran la solución tabú [Laguna, 1998], el algoritmo memético implementado por Schiavinotto [Schiavinotto, 2003], la búsqueda dispersa propuesta por Campos[Campos, 1999], entre otras. Aunque las mejores soluciones logran resolver eficientemente la mayoría de los conjuntos de instancias OPT-I, y UB-I, sin embargo, quedan aún algunos conjuntos de instancias retadores dentro de las instancias UB-I (se tienen solo las mejores soluciones conocidas, no los óptimos). Por otro lado, como resultado del análisis crítico después de la revisión y estudio del estado del arte se encontraron diferentes áreas de oportunidad en el ámbito del diseño eficiente de las metaheurísticas. Estas áreas se presentan como preguntas de investigación en la siguiente lista.

De este conjunto se eligen dos preguntas para abordarse en este trabajo de investigación considerando principalmente las limitaciones de tiempo.

- **¿Es posible diseñar nuevas estrategias de diversificación que se incorporen en metaheurísticos del estado del arte para mejorar su desempeño? [Laguna, 1998].**
- ¿Es posible incorporar otras variantes de re-encadenamiento a las soluciones metaheurísticas de LOP reportadas en el estado del arte? [Laguna, 1998].
- **¿Es posible mejorar el algoritmo memético desarrollado por Schiavinotto, que utiliza una estrategia de diversificación mediante la re-inicialización de la población? [Schiavinotto, 2003].**
- ¿Es posible implementar nuevas formas de generar vecindades mediante el método de programación dinámica? [Congram, 2000].

- ¿Es posible implementar algunas vecindades exponenciales explorables en tiempo polinomial como las que se estudian teóricamente en el trabajo de Congram? [Congram, 2000].

En este proyecto de tesis se propuso ampliar el conocimiento para encontrar la respuesta de las tres primeras preguntas planteadas, y se seleccionaron debido a que las tres se refieren a problemas relacionados con el proceso de diversificación.

1.4 Objetivos.

1.4.1 Objetivos Generales

Incorporar estrategias de diversificación en la solución memética del problema de ordenamiento lineal.

1.4.2 Objetivos Específicos

- Análisis del Algoritmo de Referencia.
- Analizar estrategias de diversificación basadas en estrategias de búsqueda tabú.
- Analizar diferentes estrategias de diversificación provenientes de algoritmos de trayectoria y población.
- Diseñar las estrategias de diversificación propuestas.
- Implementar y evaluar las estrategias de diversificación diseñadas.

1.5 Alcance y limitaciones

En este proyecto se trabajó únicamente con el problema LOP y se utilizó como algoritmo de referencia la solución memética de Shiavinotto[Shiavinotto, 2004]. Mientras que la evaluación se realizó siguiendo la experimentación desarrollada por el mismo autor, pero utilizando únicamente el conjunto de instancias XLOLIB, por ser las más retadoras.

1.6 Organización de la Tesis

El capítulo 2 contiene el marco teórico, en el cual se describen el problema de investigación, el cálculo de la función objetivo, las diferentes búsquedas locales, los tipos de vecindades y las instancias utilizadas.

El capítulo 3 incluye la revisión de los principales trabajos del estado del arte, se muestra además un resumen sobre las principales características y áreas de oportunidad de acuerdo al análisis de los trabajos estudiados y que tienen relevancia para el proyecto.

El capítulo 4 incluye un estudio preliminar que se basa en el algoritmo de búsqueda local iterada. En esta metaheurística se evaluaron experimentalmente un conjunto de estrategias de búsqueda local implementadas.

El capítulo 5 incluye todos los detalles del algoritmo memético implementado, en él se incorporan las estrategias de diversificación basadas en búsquedas locales que tienen diferente nivel de intensificación-diversificación.

El capítulo 6 contiene las especificaciones y resultados del estudio experimental realizado para el algoritmo propuesto.

El capítulo 7 resume las principales conclusiones y aportaciones que se obtuvieron de esta investigación.

Capítulo 2

Marco Teórico

En este capítulo se presentan algunos conceptos fundamentales para el desarrollo del trabajo de investigación tales como: optimización, heurística, metaheurística, intensificación, diversificación, y complejidad, así como la clasificación de metaheurísticas

2.1 Definiciones.

2.1.1 Optimización.

En los campos de la ciencia como la informática, la inteligencia artificial y la investigación operativa, la optimización como una disciplina es fundamental. En [Duarte, 2007], el concepto de optimización lo definen como un proceso dentro del cual se pretende encontrar una solución deseable a un problema de optimización en un lapso de tiempo mínimo.

Un problema de optimización existe si y sólo si se puede disponer de un conjunto de soluciones candidatas diferentes que pueden ser comparadas entre sí. A continuación se expresa de manera formal un problema de optimización combinatoria. Dado un problema *P*, el cual se formular como una 3-tupla *P = (f, SS, F)*, donde *f* es la función a optimizar (ya sea para maximizar o minimizar), *F* es el conjunto de posibles soluciones factibles y *SS* es el espacio de soluciones:

$$P = \begin{cases} opt: & f(x), & \textit{Función Objetivo} \\ s.a., & & \\ & x \in F \subset SS & \text{Res}\textit{tricciones} \end{cases} \tag{2.1}$$

Los problemas de optimización se pueden dividir en 2 sub-categorías, en la primera existen problemas donde la solución esta codificada mediante valores reales, y en la segunda existen problemas donde las soluciones están codificadas por valores enteros. En la segunda sub-categoría existe un problema de gran interés dentro de la comunidad científica, el denominado *problema de optimización combinatoria*. Este tipo de problemas

pretende encontrar un objeto dentro de un conjunto finito de posibilidades (o conjunto que sea contable), el cual por lo general puede ser un número natural (o un conjunto de naturales), una permutación o una estructura de grafo (o Sub-grafo) según sea el caso [Papadimitriou, 1998]. Los ejemplos más conocidos de problemas de optimización combinatoria son el problema del agente viajero (TSP), el problema de la asignación cuadrática (QAP) o los problemas de planificación (Scheduling Problems). Una de las características de los problemas combinatorios, es tienen un algoritmo exacto, el cual permite resolver este tipo de problemas y obtener una solución óptima. Estos algoritmos consisten en una exploración exhaustiva del conjunto de soluciones.

En algunos casos estos algoritmos pueden ser muy ineficientes, ya que el tiempo que utilizan para resolver los problemas y encontrarles una solución, crece de forma exponencial con el tamaño del problema. Los problemas de optimización combinatoria se caracterizan principalmente por tener espacios de soluciones muy extensos.

Se ha estudiado esta clase de problemas para encontrar la manera de formularlos matemáticamente, los resultados de estos estudios muestran que existe un subconjunto de problemas, los cuales tienen algoritmos de resolución que presentan una complejidad computacional de tipo polinomial, es decir, el tiempo en que son ejecutados estos algoritmos crece polinomialmente con el tamaño del problema. Estos problemas pertenecen a una clase llamada clase *P*, los cuales se consideran que pueden ser resueltos de una manera más eficiente. Por otro lado, para los problemas que son de mayor interés científico, hasta el momento no se conoce un algoritmo de complejidad polinomial que los resuelva de forma exacta, estos problemas pertenecen a una clase llamada *NP*. Del interés practico que existe por resolver muchos de los problemas que pertenecen a esta clase y debido a la dificultad para resolverlos, nace la opción de encontrar soluciones de alta calidad en tiempos razonables, siendo que estas soluciones pueden no ser óptimas. Para resolver los problemas de optimización combinatoria es necesario utilizar algoritmos aproximados.

2.1.2 Heurísticas.

La mayoría de los problemas prácticos de gran interés tienen un espacio de búsqueda muy extenso, por lo cual no se conoce un algoritmo exacto con complejidad polinomial que los

pueda resolver en un tiempo aceptable. Es por esto que se utilizan métodos aproximados o heurísticas, las cuales permiten obtener una solución de calidad en tiempos razonables para estos problemas.

Heurística proviene del vocablo griego *heuriskein*, que significa "Encontrar, descubrir o hallar". [Duarte, 2007].

El término heurística fue empleado por primera vez por G. Polya en su libro *How to solve it* [Polya, 1957], quien la plantea como reglas con las cuales los humanos gestionan el conocimiento común. A los largo del tiempo se han formado definiciones acerca de las heurísticas, pero una de las mas exactas es la que presenta Zanakis en [Zanakis, 1981], la cual define a una heurística como *"Procedimientos Simples, a menudo basados en el sentido común, que se supone que obtendrán una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido."*

Uno de los principales defectos que tienen los métodos heurísticos es que son incapaces de escapar de los óptimos locales, ya que estos métodos no cuentan con ningún mecanismo que les permita seguir con la búsqueda y conseguir su objetivo de obtener la solución optima. Para evitar este tipo de problemas, se empezaron a desarrollar otro tipo de algoritmos de búsqueda *inteligente*, los cuales denominaron *metaheurísticas*, estos algoritmos son procedimientos de alto nivel que guían a métodos heurísticos y evitan un estancamiento en los óptimos locales, lo que hace más factible llegar a la solución óptima.

2.1.3 Metaheurísticas.

El científico Glover acuño el término *Metaheurística* en el año de 1986, describiéndolo como *"un procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimilidad local"*[Glover, 1986]. Por otro lado una de las definiciones más completa que se han formulado es la descrita por J.P. Kelly en [Kelly, 1996], la cual nos dice que *"Las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando*

diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos".

Las metaheurísticas pueden incluir métodos populares pero no están restringidos a estos, algunos de los ejemplo de estos métodos son: optimización por colonia de hormigas (ACO), algoritmos evolutivos (EA, dentro de los cuales se encuentran también los algoritmos genéticos (GA) y los algoritmos meméticos (MA)), procedimientos de búsqueda miope (constructiva, voraz o ávida), aleatorizados y adaptativos (GRASP), búsqueda local iterativa (ILS), re-encadenamientos de trayectorias (PR), recocido simulado (SA), búsqueda dispersa (SS) y búsqueda tabú (TS).

2.1.4 Clasificación de las Metaheurísticas

A través del tiempo diferentes investigadores han desarrollado un marco general de las metaheurísticas, esto ha permitido clasificarlas mediante un análisis, dentro del cual se han encontrado semejanzas y diferencias, de dichas metaheurísticas, estas características han permitido encontrar una estructura, para así clasificarlas. Existen diferentes tipos de clasificaciones entre las que se encuentran: la clasificación clásica, tabular, jerárquica y basada en la relación intensificación-diversificación. En este caso se estudiara la clasificación clásica, ya que es de gran interés para el desarrollo de este trabajo de investigación, dentro de ella existe una sub-categoría donde se clasifican las metaheurísticas dependiendo el número de soluciones, dentro de esta sub-categoría existen dos tipos de metaheurísticas, las trayectoriales y poblaciones.

Se denominan metaheurísticas trayectoriales, ya que los procesos que desarrollan sobre la búsqueda se caracterizan por tener una trayectoria dentro del espacio de soluciones, es decir, estas metaheurísticas parten de una solución factible, y conforme exploran el espacio de soluciones generan un camino o trayectoria, mediante operaciones de movimiento. Este camino o trayectoria formada por el algoritmo proporciona información acerca del comportamiento y la eficiencia del algoritmo. Algunas de la estrategias que emplean este tipo de metaheurísticas son: búsqueda multi-arranque, aleatoria y determinista. Dentro de las principales metaheurísticas trayectoriales se encuentran la Búsqueda Tabú (TS), Recocido Simulado (SA), Búsqueda de Vecindad Variable (VNS),

Búsqueda Local Guiada (GLS), Aceptación de Umbral (TA), Métodos Ruidosos (NM), FANS (*Adaptive Neighborhood Search*) y Búsqueda Local Iterada (ILS).

Por otro lado se encuentran las metaheurísticas poblacionales, las cuales utilizan un conjunto de soluciones iniciales, está generación se realiza en cada iteración del algoritmo, a este conjunto se le denomina población. Algunas de las metaheurísticas más importantes dentro de esta sub-categoría se encuentran: los algoritmos evolutivos (Algoritmo Genético (GA), Algoritmo Memético (MA)), Búsqueda Dispersa, Re-encadenamientos de Trayectorias (PR), Algoritmos de Estimación de la Distribución, Algoritmos Culturales y por último se tiene el algoritmo de Inteligencia de Enjambre y Opt. Por Enjambre de Partículas. De los algoritmos antes mencionados se eligieron tres, ya que son de gran interés para la resolución del Problema de Ordenamiento Lineal, los algoritmos que se estudiaron fueron: la Búsqueda Tabú y el algoritmo de Búsqueda Local Iterada, estos dos algoritmo pertenecientes a las metaheurísticas trayectoriales y el ultimo el algoritmo Memético, perteneciente a las metaheurísticas poblacionales.

2.1.5 Intensificación y Diversificación.

La idea original de la intensificación y diversificación nace en el marco de la *Búsqueda Tabú*. En la actualidad no solo la Búsqueda Tabú contiene este tipo estrategias, ya que se han extendido al resto de las metaheurísticas como parte de una medida de su *"potencialidad"* [Duarte, 2007].

La intensificación se puede definir como el proceso de búsqueda exhaustiva, que lleva a cabo una metaheurística dentro de una vecindad dada. Por lo general una metaheurística no intensifica en cualquier entorno, ya que toma en cuenta algunos factores, uno de ellos es verificar la calidad de la solución encontrada dentro de la vecindad, es decir si no es encontrada alguna solución de calidad dentro de la vecindad no tiene caso alguno que se intensifique la búsqueda en dicha región, por tanto no realiza una búsqueda exhaustiva de todo el espacio de soluciones.

La diversificación se define como el proceso mediante el cual una metaheurística puede ser capaza de visitar diversas vecindades lejanas. Por lo general una metaheurística no diversifica constantemente sin aplicar algún criterio, ya que se convertiría en una búsqueda aleatoria dentro del espacio de soluciones. Para que una metaheurística realice el

proceso de diversificación debe tener en cuenta algunos factores, uno de ellos es el verificar que cierta región del espacio de búsqueda no haya sido visitada.

2.1.6 Complejidad: problemas P y NP

Garey y Jhonson desarrollaron la teoría de la complejidad en la cual los problemas son caracterizados de acuerdo al estudio de la dificultad de su resolución en un ordenador. De acuerdo a los estudios que se han realizado los problemas se ha definido en varios tipos de clases, las principales son *P, NP, NP-completo* y *NP-duro*. [Garey, 1975]

Un problema se puede clasificar como un problema de la clase *P* siempre y cuando se pueda encontrar un algoritmo que lo resuelva en tiempo polinomial. En el campo de la optimización combinatoria existe una gran cantidad de problemas útiles en la práctica, pero tiene una peculiaridad, la cual es que son difíciles de resolver. Para estos problemas, hasta el momento no se ha encontrado ningún algoritmo que puede obtener una solución óptima en tiempo polinomial, por lo tanto estos problemas no se pueden resolver en tiempos de ejecución aceptables. Dependiendo de la dificultad de resolución de estos problemas es como se categorizan, existen problemas donde no se ha encontrado un algoritmo polinomial que lo resuelva, pero eso no impide saber en tiempo polinomial si un determinado valor corresponde a la solución de dicho problema, ha este tipo de problemas se les denomina *NP*. Por otro lado siempre existe la posibilidad de comprobar que un determinado valor es una solución a un problema en tiempo polinomial, por tanto se considera que los problemas *P* también son problemas *NP*, en caso de no encontrarse una forma sencilla de hacerlo, existe la posibilidad de ejecutar el algoritmo polinomial que resuelve el problema y se compara el resultado con el valor obtenido, entonces se puede decir que el conjunto de problemas *P* es un subconjunto de los problemas *NP*.

Los problemas *NP*-completos son un subconjunto de los problemas *NP*. Un problema Π se puede clasificar como *NP-completo* si existe un algoritmo de tiempo polinomial que resuelva su problema de decisión, y si además se conoce un problema Π' que pertenece a la categoría *NP-completo* (con certeza) y para el cual es posible encontrar una transformación f de Π' a Π, demostrando finalmente que la transformación se puede

realizar mediante un algoritmo de tiempo polinomial, entonces se puede decir, que un problema Π es *NP-completo*.[Garey, 1975]

Por último tenemos otro tipo de problema, que de igual forma que los *NP-completos* son difíciles de resolver o tiene un grado más alto de dificultad, este tipo de problemas son denominados *NP-duros*, estos problemas no son un subconjunto de los problemas *NP*, ya que no existe un algoritmo polinomial que permita verificar una solución a este problema.

Un problema *NP-duro* es considera como tal si existe un problema *NP-completo* que sea reducible a ese problema, es decir para resolver un problema *NP-duro* debe existir un algoritmo polinomial que use una caja negra y resuelva el correspondiente problema *NP-completo*. [Duarte, 2007]

2.2 Calculo de la función objetivo para el problema LOP

La definición de LOP establece que: Dada una matriz $E\{e_{ij}\}$ de tamaño *n x n*, LOP busca una permutación π de los índices de las columnas y filas tal que el valor de la siguiente expresión sea maximizado.

$$f(\pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} e_{\pi(i)\pi(j)} \qquad (2.2)$$

Es decir, tiene como objetivo encontrar una permutación de las columnas y filas de la matriz *E* tal que la suma de los elementos sobre la diagonal principal de dicha matriz sea maximizada. Enseguida se muestra un ejemplo del cálculo de la función objetivo para LOP. Dada la permutación $\pi = \{1, 2, 3\}$, y su correspondiente matriz de costos,

$$E = \begin{array}{c} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 4 & 10 & 3 \\ 15 & 2 & 1 \\ 7 & 9 & 6 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

el valor de la función objetivo para la matriz original es:

$$E_{LOP}(\pi) = e_{12} + e_{13} + e_{23} = 10 + 3 + 1 = 14$$

Si se aplica la permutación $\pi = \{3, 2, 1\}$,

Matriz Original

$$E = \begin{array}{c} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 4 & 10 & 3 \\ 15 & 2 & 1 \\ 7 & 9 & 6 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

Intercambio de Columnas 3,1

$$E = \begin{array}{c} \begin{matrix} 3 & 2 & 1 \end{matrix} \\ \begin{bmatrix} 3 & 10 & 4 \\ 1 & 2 & 15 \\ 6 & 9 & 7 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

Intercambio de Filas 3,1

$$E = \begin{array}{c} \begin{matrix} 3 & 2 & 1 \end{matrix} \\ \begin{bmatrix} 6 & 9 & 7 \\ 1 & 2 & 15 \\ 3 & 10 & 4 \end{bmatrix} \end{array} \begin{matrix} 3 \\ 2 \\ 1 \end{matrix}$$

Matriz Final

$$E = \begin{array}{c} \begin{matrix} 3 & 2 & 1 \end{matrix} \\ \begin{bmatrix} 6 & 9 & 7 \\ 1 & 2 & 15 \\ 3 & 10 & 4 \end{bmatrix} \end{array} \begin{matrix} 3 \\ 2 \\ 1 \end{matrix}$$

el valor de la función objetivo para la matriz permutada es:

$$E_{LOP}(\pi) = e_{12} + e_{13} + e_{23} = 9 + 7 + 15 = 26$$

2.3 Estrategias de construcción de la solución inicial

En las metaheurísticas, partir de una solución inicial de calidad representa un factor importante para alcanzar soluciones de buena calidad. En este trabajo se evaluaron tres diferentes métodos de construcción de la solución inicial.

2.3.1 Solución inicial ordenada

Para esta solución la permutación se construye posicionando los elementos de acuerdo a un índice secuencial ordenado, es decir: $\pi_i = i$, $i = 1,\ldots$*tamaño de la matriz*. En la figura 1 se muestra el algoritmo de la solución inicial ordenada.

```
1. Inicia Construccion_Ordenada
2.         for(i=1 ; i ≤ n ; i++)
3.                 orden [i] = i;
4.         Endfor
5. Fin Construccion_Ordenada
```

Figura 1. Algoritmo de la Solución Inicial Ordenada.

2.3.2 Solución inicial aleatoria

La construcción aleatoria genera una permutación de las filas y columnas en un orden aleatorio. A continuación se muestra el algoritmo en la figura 2:

```
1.  Algoritmo Construcción_aleatoria
2.      for(i=1…dim)
3.          π[i]←0
4.      for(j=1… j ← dim) hacer
5.          a = getrandom(1..dim+1-j)
6.          pos←1
7.          for(i=1…a) hacer
8.              mientras(π [pos]≠0)
9.                  pos++
10.             pos++
11.         fin_for
12.         mientras(π [pos]≠0)
13.             pos++;
14.         π[pos]←j;
15.         π_inv[j]←pos;
16.     fin_for
17. Fin Construcción_aleatoria
```

Figura 2. Algoritmo de Construcción aleatoria.

2.3.3 Solución inicial mediante la heurística de Becker

Becker en 1967 propuso una heurística basada en el cálculo de cocientes para clasificar cada sector (usando la interpretación de la economía). En particular, para cada sector i=1,…, m se calcula el cociente q_i:

$$q_i = \frac{\sum_{k=1}^{m} e_{ik}}{\sum_{k=1}^{m} e_{ki}} \qquad (2.3)$$

El sector con el valor *q* más grande se toma. Entonces, la columna y fila correspondiente se eliminan de la matriz y el procedimiento se aplica al resto de los sectores. A continuación se muestra un pseudocódigo del algoritmo de Becker, en la figura 3.

```
1.  Entrada: E = {e_ij}_mxm
2.  Salida: p=(p_1, p_2,..., p_m)
3.  Inicia Becker ( )
4.          I= {1,…, m}
5.          j= 0
6.          while (j<m)
7.                  k = argmax {q_i: i ∈ I}
8.                  j = j +1
9.                  p_j = k
10.                 I = I – { k}
11.         Endwhile
12. Fin Becker
```

Figura 3. Algoritmo de *Becker*.

Este método es muy rápido y produce resultados razonables a pesar de que es un proceso sencillo. [Laguna, 1998]. Utilizando este método se generan diez permutaciones de Becker, y de éstas se selecciona la mejor.

2.4 Vecindades

Uno de los elementos estructurales básicos dentro del estudio de los algoritmos de búsqueda local es la estructura de vecindad, la cual define el conjunto de soluciones que conforman el espacio de búsqueda. La definición de Blum para este concepto se da enseguida:

"Una estructura de vecindad es una función $N: S \rightarrow 2^S$ que asigna a cada $s \in S$ un conjunto de vecinos $N(s) \subseteq S$. $N(s)$ es llamado la vecindad de s" [Blum, 2003].

Sobre el concepto de vecindad, Duarte presenta la siguiente descripción: "Dada una solución $x \in SS$, la vecindad $N(x)$ de esa solución es un subconjunto del espacio de soluciones que contiene soluciones que están "*próximas*" de la solución considerada" [Duarte, 2007].

Otra descripción de la vecindad es presentada por Cruz, y dice: "Una vecindad está definida como el conjunto de soluciones cercanas a una solución inicial dada" [Cruz, 2008].

El concepto de vecindad está fuertemente ligado con la representación del problema, de manera que existe una gran variedad de vecindades dependiendo de la representación del problema. Por ejemplo, se tienen vecindades para problemas representados mediante permutaciones, tales como, la vecindad de inserción, y la de intercambio, ó para problemas que tienen representación binaria, tales como *1 flip*, *2 flip*, *n-flip*, etc. A continuación se describen algunas vecindades clásicas.

2.4.1 Vecindad de intercambio

La vecindad de intercambio N_x, es definida por la operación *intercambio*, esta toma como $\text{intercambio}: \Pi \times \{1,...,n\}^2 \to \Pi$, donde π es el conjunto de todas las permutaciones y se tiene para $i \neq j$:

$$Intercambio(\pi, i, j) \triangleq (..., \pi_{i-1}, \pi_j, \pi_{i+1}, ..., \pi_{j-1}, \pi_i, \pi_{j+1}) \tag{2.4}$$

Esta vecindad tiene el tamaño de $|N_x| = n(n-1)/2$.

2.4.2 Vecindad de inserción N_I

N_I, es definida por la operación de inserción: un elemento es la posición *i* es insertado en otra posición *j*. formalmente, $insert: \Pi \times \{1,...,n\}^2 \to \Pi$ es definido por $i \neq j$:

$$insert(\pi, i, j) \triangleq \begin{cases} (..., \pi_{i-1}, \pi_{i+1}, ..., \pi_j, \pi_i, \pi_{j+1}, ...) & i < j; \\ (..., \pi_{j-1}, \pi_i, \pi_j, ..., \pi_{i-1}, \pi_{i+1}, ...) & i > j; \end{cases} \tag{2.5}$$

La vecindad basada en inserciones tiene un tamaño de $|N_1(\pi)| = (n-1)^2$.

El incremento del costo en la función objetivo para un movimiento dado involucra solo a los elementos que intervienen en él mismo, la expresión 2.6 muestra esta situación.

$$\Delta_{CI}(\pi, i, j) \triangleq f(insert(\pi, i, j)) - f(\pi) \tag{2.6}$$

La siguiente expresión muestra el cálculo del incremento en la función de costo.

$$\Delta_{CI}(\pi, i, j) \triangleq \begin{cases} \sum_{k=i+1}^{j} e_{\pi_k \pi_i} - e_{\pi_i \pi_k} & i < j \\ \sum_{k=j}^{i-1} e_{\pi_i \pi_k} - e_{\pi_k \pi_i} & i > j \end{cases} \quad (2.7)$$

Como se puede apreciar, el costo del incremento es un valor constante y por lo tanto el costo del movimiento de inserción será *O (n²).*

El siguiente ejemplo muestra el proceso de aplicar un movimiento de inserción mediante una serie de movimientos *swap* consecutivos. Un movimiento de inserción (con argumentos *i* y *j*) es siempre equivalente a un movimiento *swap* |*i*-*j*|. Por ejemplo, para $\pi = (1\ 2\ 3\ 4\ 5\ 6)$, i =2 y j=5, el movimiento de inserción insert $(\pi, 2, 5)$ mediante una serie de movimientos *swap* sobre elementos consecutivos se muestra enseguida:

insert $(\pi, 2, 5)$ = (1 3 4 5 2 6)
swap $(\pi,2,3)$ = (1 3 2 4 5 6) = π'
swap $(\pi',2,4)$ = (1 3 4 2 5 6)= π"
swap $(\pi",2,5)$ = (1 3 4 5 2 6)= insert $(\pi, 2, 5)$

Costo de Inserción ó Swaps consecutivos

Insert(π, i, j) donde $j = i+1$

Si $j = i+1$

$$\text{Insert}(\pi, i, j) = \text{Insert}(\pi, i, i+1) = \text{swap}(\pi, i, i+1)$$

Si $j = i$ -1

$$\text{Insert}(\pi, i, j) = \text{Insert}(\pi, i, i-1) = \text{swap}(\pi, i, i-1)$$

Por lo tanto:

Si $j = i+1$

$$\text{Costo}(\text{Insert}(\pi, i, i+1)) = \sum_{k=i+1}^{j=i+1} (e_{\pi_k \pi_i} - e_{\pi_i \pi_k})$$

$$= e_{\pi_{i+1} \pi_i} - e_{\pi_i \pi_{i+1}} = e_{\pi_j \pi_i} - e_{\pi_i \pi_j}$$

$$\text{Costo}(\text{Insert}(\pi, i, j)) = e_{\pi_j \pi_i} - e_{\pi_i \pi_j} \quad \text{Si } i < j$$

Si $j = i$ -1

$$\text{Costo(Insert }(\pi, i, i-1)) = \sum_{k=i+1}^{i-1} (e_{\pi_i\pi_k} - e_{\pi_k\pi_i})$$

$$= e_{\pi_i\pi_{i-1}} - e_{\pi_{i-1}\pi_i} = e_{\pi_i\pi_j} - e_{\pi_j\pi_i}$$

Por lo tanto.

$$\text{Costo(Insert }(\pi, i, j)) = e_{\pi_i\pi_j} - e_{\pi_j\pi_i} \text{ Si } i > j$$

Ejemplo:

$$\text{Costo(Insert }(\pi, 2, 5)) = \sum_{k=3}^{j=5} (e_{\pi_2\pi_3} - e_{\pi_3\pi_2})$$

$$= (e_{\pi_3\pi_2} - e_{\pi_2\pi_3}) + (e_{\pi_4\pi_2} - e_{\pi_2\pi_4}) + (e_{\pi_5\pi_2} - e_{\pi_2\pi_5})$$

$$= \text{Costo(Insert }(\pi, 2, 3)) + \text{Costo(Insert }(\pi, 2, 4)) + \text{Costo(Insert }(\pi, 2, 5))$$

$$= \text{Costo(swap }(\pi, 2, 3)) + \text{Costo(swap }(\pi, 2, 4)) + \text{Costo(swap }(\pi, 2, 5))$$

2.5 Definición de Búsqueda Local

La idea básica que se tiene acerca del mecanismo que utilizan las búsquedas locales es la siguiente: estos algoritmo inician con una solución, la cual es generada aleatoriamente o mediante algún otro algoritmo, posteriormente a dicha solución se le aplica una transformación de algún conjunto dado de transformaciones para mejorar la solución actual, si la solución actual es peor que la solución mejorada, entonces esta se convierte en la actual. El algoritmo repite el mismo mecanismo hasta que ya no exista ninguna transformación que mejore la solución actual. La solución que se obtenga de este proceso no necesariamente es óptima, ya que dependerá de varios aspectos, como por ejemplo, que transformaciones fueron aplicadas o el tiempo en que se llevo a cabo el proceso. Cuando no es posible la obtención de una mejor solución el algoritmo se detiene en un óptimo local. En otras palabras los métodos de búsqueda local parten de una solución inicial factible dada y a partir de ella tratan de mejorar la solución actual mediante movimientos dentro de la vecindad de la solución. En la figura 4 se muestra el algoritmo básico de búsqueda local, mejor conocido como algoritmo de mejora iterativa [Blum, 2003].

```
1.  Inicia Algoritmo_Mejora_Iterativa ( )
2.    s = Genera_SoluciónInicial ( )
3.    Repeat
4.      mejor_costo =  Costo (s);
5.      Repeat
6.          s' = Aplicar_movimiento (s)
7.          if(Aceptable (s', s))
8.              Then s = s'
9.      until (Mejor en N(s))
10.   until (criterio de parada)
11. Fin Algoritmo_Mejora_Iterativa ( )
```

Figura 4. Algoritmo de Mejora Iterativa.

2.5.1 Búsqueda Local *Best.*

Las búsquedas de tipo *Best* se pueden aplicar sobre distintos tipos de vecindarios pero se caracterizan por que aplican el criterio *Best*, el cual consiste en revisar todos los vecinos de una solución y seleccionar el mejor de ellos (*Best*). La búsqueda *Best* que se describe aquí es la que se implementa en este trabajo de investigación y es algo diferente al algoritmo *Best* estándar. En esta búsqueda se parte de una solución factible a la que mejora progresivamente, para lo cual se examinan en la vecindad todos los posibles movimientos, y de todos, selecciona el mejor (aquel que produzca el mayor incremento en el valor de la función objetivo o que produzca el menor decremento de la función objetivo). Dado que la representación de la solución del problema es mediante permutaciones, se exploran todos los sectores en el orden dado por la permutación, buscando en todas las posiciones consecutivas hacia adelante y hacia atrás de la posición actual, y finalmente se aplica el movimiento (en este caso se aplica un movimiento de inserción). Se muestra el algoritmo en la figura 5.

```
1.  Inicia Busqueda_Local_Best (π, i)
2.  Δmax = -10000
3.  π = Generar_Permutacion ( );
4.  mejor_coste =  Coste (π);
5.  do
6.    for (j= i-1 to 1)
7.        mejora = 0;
8.        for(j=i-1 to 1)
9.            if (ΔI (π, i, j) > Δmax) Then
10.               Δmax = ΔI (π, i, j);
11.               jmax = j;
12.           endif
13.       endfor
14.       for(j=i+1 to n)
15.           if (ΔI (π, i, j) > Δmax) Then
16.               Δmax = ΔI (π, i, j);
17.               jmax = j;
18.           endif
19.       endfor
20.       π' = Insert (π, i, jmax);
21.       coste = coste + Δmax;
22.       π = π';
23.   endfor
24.   if (coste > mejor_coste)
25.       mejora=1;
26.   while (mejora ==1)
27.   return (π)
28. Fin Busqueda_Local_Best
```

Figura 5. Algoritmo de Búsqueda Local *Best*.

2.5.2 Búsqueda Local *First.*

La búsqueda local *First* parte de una solución factible y la mejora progresivamente, para ello examina su vecindad y selecciona el primer movimiento que produzca una mejora en la solución actual *(first improvement)*. Examina las posiciones consecutivas hacia adelante, buscando una solución que sea mejor que la actual, en caso de no encontrar una mejor solución, se examinan las posiciones hacia atrás de la posición actual, buscando una mejor solución, la revisión de las posiciones termina cuando encuentra la primera solución que sea mejor que la actual. En la figura 6 se muestra el algoritmo para la búsqueda *first*, en caso de un problema de maximización:

```
1.  Inicia  Busqueda_Local_First  (π, i)
2.  Δmax = -10000;
3.  π = Generar_Permutacion ( );
4.  mejor_coste = Coste (π);
5.  for (j= i+1 to n)
6.      mejora = 0;
7.      do
8.        j=i-1
9.        if (ΔI (π, i, j) > Δmax) Then
10.           Δmax = ΔI (π, i, j);
11.           jmax = j;
12.           mejora=1;
13.       endif
14.       j = j-1;
15.     while (not mejora and j ≥1)
16.     j=i+1;
17.     while (not mejora and j ≤ n)
18.           if (ΔI (π, i, j) > Δmax) Then
19.               Δmax = ΔI (π, i, j);
20.               jmax = j;
21.               mejora=1;
22.           endif
23.           j= j+1;
24.     endwhile;
25.     if (mejora == 1)
26.         π' = insert (π, i, jmax);
27.         coste = coste + Δmax;
28.         π = π';
29.     endif
30. endfor
31. return (π)
32. Fin Busqueda_Local_First
```

Figura 6. Algoritmo de Búsqueda *First.*

2.5.3 Búsqueda Local LS_f.

Se aplica una búsqueda local LS_f, en la cual la regla de pivoteo utiliza un criterio entre *best* y *first* de la mejora. La exploración de los índices para encontrar el mejor movimiento en cada índice, se hace aplicando vecindad de inserción N_1, y la estrategia de exploración mediante *swaps* consecutivos. El ciclo externo revisa para todos los sectores en el orden dado por la permutación cual es el mejor vecino y realiza el intercambio. La búsqueda termina cuando se han realizado la búsqueda del mejor (o el menos peor) sobre todos los vecinos de cada sector. A continuación se muestra su algoritmo en la figura 7:

```
1.  Inicia Busqueda_Local_LS_f (π, i)
2.  Δ_max = -10000
3.  π = Generar_Permutacion ( );
4.  mejor_coste = Coste (π);
5.      for (j= i+1 to n)mejora = 0;
6.          for(j=i-1 to 1)
7.              if (Δ_I (π, i, j) > Δ_max) Then
8.                  Δ_max = Δ_I (π, i, j);
9.                  j_max = j;endif
10.         endfor
11.         for(j=i+1 to n)
12.             if (Δ_I (π, i, j) > Δ_max) Then
13.               Δ_max = Δ_I (π, i, j);
14.               j_max = j;
15.             endif
16.         endfor
17.         π' = Insert (π, i, j_max);
18.         coste = coste + Δ_max;
19.         π = π';
20.     endfor
21. return (π)
22. Fin Busqueda_Local_LS_f
```

Figura 7. Algoritmo de Búsqueda Local LS_f.

2.6 Búsqueda Tabú (TS, Tabú Search).

La búsqueda tabú, es un algoritmo metaheurístico de búsqueda local para explorar más allá del óptimo local dentro del espacio de soluciones. Gracias a los éxitos que ha logrado este algoritmo en aplicaciones prácticas de optimización, su uso ha tenido un crecimiento muy grande en los últimos años. Se han realizado híbridos de este algoritmo, es decir, aplicaciones donde se conjuga con otros algoritmos. La búsqueda tabú se basa en que para poder clasificar de manera inteligente la solución de un problema, se debe incorporar memoria adaptativa y exploración sensible. La finalidad de la búsqueda tabú es encontrar formas nuevas y más efectivas de sacar ventaja a los conceptos, este proceso de búsqueda de acuerdo a Díaz et al [Diaz, 1996] consiste en:

Un procedimiento metaheurístico que incorpora estrategias para escapar de los óptimos locales. Las estrategias están basadas en exploración sensible y memoria adaptativa. En la exploración sensible se integran los principios de búsquedas inteligentes mediante reglas adaptativas que promueven combinaciones de movimientos y/o características de soluciones históricamente buenas. Por otro lado la memoria adaptativa se

implementa mediante procedimientos que guían la elección de soluciones locales utilizando información que se colecta durante la búsqueda.

Entre las estrategias de memoria están las memorias de corto y largo plazo. La memoria de corto plazo lleva la cuenta de los atributos de la solución que han sido cambiados en el pasado reciente, a los cuales designa como "Tabú-activos". Se suele llamar memoria basada en recencia (en hechos recientes), a las soluciones que contienen elementos tabú-activos, o combinaciones particulares de estos atributos, se les asocia el estatus de tabú. Esto evita que algunas soluciones del pasado reciente pertenezcan a $N(x)$, obteniéndose una vecindad reducida $N^*(x)$. Otro elemento característico es la memoria de largo plazo, esta estrategia incorpora penalizaciones e incentivos los cuales son determinados por el rango relativo de tiempo durante el que los atributos han pertenecido a soluciones visitadas durante la búsqueda, permitiendo así la diferenciación por regiones. La frecuencia con que cambian los atributos es registrada dentro de la frecuencia de transición, mientras que las frecuencias de residencia mantienen el registro de las duraciones relativas de los atributos en las soluciones generadas. Otras estrategias que pueden ser integradas al algoritmo tabú son:

Estrategia de Intensificación. Las estrategias de intensificación están basadas en la modificación de reglas de elección de tal manera que se favorezcan combinaciones de movimiento y características de solución que históricamente hayan sido buenas, pueden iniciar un regreso hacia regiones atractivas para buscar en ellas más extensamente. Entre las estrategias de intensificación que pueden implementarse esta el Re-encadenamiento de trayectorias y la aplicación de búsqueda local adicional.

Estrategias de diversificación. La estrategia de diversificación, están diseñadas para conducir la búsqueda hacia nuevas regiones. Con frecuencia están basadas en modificar las reglas de elección para llevar a la solución atributos que no hayan sido usados frecuentemente. Alternativamente, pueden introducir dichos atributos al reiniciar parcial o completamente el proceso de solución. La oscilación estratégica permitiendo movimientos hacia soluciones no factibles y movimientos para transformar esas soluciones no factibles en nuevas soluciones factibles, puede incorporarse con fines de diversificación. Incorporar aleatorización dentro de alguno de un proceso de la Búsqueda Tabú incorpora

diversificación. Utilizar un registro de movimientos para incorporar atributos que no se ha utilizado frecuentemente.

Re-encadenamiento de Trayectorias. El re-encadenamiento de trayectorias proporciona una integración muy útil de las estrategias de intensificación y diversificación. Este enfoque se basa en explorar trayectorias que conectan soluciones élite para generar nuevas soluciones, empezando desde una de estas soluciones llamada *solución inicial*, y generando una trayectoria en el espacio de entornos que conduce hacia otras soluciones, llamadas soluciones guía. Esto se logra seleccionando movimientos que introducen atributos contenidos en las soluciones guías [Díaz, 1996]. En la figura 8 muestra el algoritmo de la búsqueda tabú.

```
1.  Inicia Busqueda_Tabu
2.  π = Generar (Solución_Inicial)
3.      while (Condición de Parada)
4.          V_I= Generar_Vecindad (π);
5.          V_R= Generar_Vecindad_Reducida (V_I);
6.          π'= Seleccionar_ Mejor (V_R);
7.          if (π' ≥ π)
8.            π = π';
9.          endif
10.         L_T = Actualizar_Lista_Tabú (π) //Insertar Activos;
11.         L_T' = Actualizar_Lista_Tabú (L_T)//Borrar elementos antiguos
12.     endwhile
13. Fin Busqueda_Tabu
```

Figura 8. Algoritmo de Búsqueda Tabú.

2.7 Algoritmo de Búsqueda Local Iterativa (ILS, Iterated Local Search).

A través de los años diversos investigadores han estudiado acerca de este algoritmo, el cual ha cambiado su nombre, de acuerdo a diferentes características distintivas. Algunas de las connotaciones que se le han asignado son: *descenso iterativo, cadenas de Markov de paso grande, algoritmo de Lin-Kernighan iterativo* y *optimización local encadenada*. Esta sencilla búsqueda, como cualquiera de las metaheurísticas basadas en búsqueda se define mediante, un espacio de soluciones *ES*, una vecindad *V* y una búsqueda local *BL*. Realiza un procedimiento de búsqueda local sobre el espacio de solución *ES* a través de movimientos dentro de la vecindad, y como resultado obtiene un espacio de soluciones óptimas *SO*. El algoritmo básico ILS está compuesto por cuatro elementos, el primero de

ellos es el proceso que genera una solución inicial, la cual puede ser obtenida de forma aleatoria, ordenada o mediante una heurística. El segundo de los elementos, es un proceso de búsqueda local sobre la solución inicial generada, la cual mejora mediante movimientos dentro de una vecindad. El tercero de los elementos que se diseña de acuerdo a las necesidades del problema a resolver, consiste en un proceso iterativo en el cual se aplica una perturbación; en la cual se modifican algunos de los elementos de las soluciones mejoradas, se añade ruido a la permutación, ó se optimiza de manera local una parte de la solución; seguido de un proceso de optimización local que se aplica a las soluciones que se obtienen de la perturbación. El último y cuarto de los elementos es el criterio de aceptación de la solución, mediante el cual se define que soluciones pasarán a la siguiente iteración. El equilibrio en la intensificación y diversificación del algoritmo depende fuertemente del criterio implementado, ya que si el algoritmo acepta solo soluciones con movimientos de mejora, se favorece la intensificación y si acepta cualquier movimiento dentro de la solución favorece a la diversificación. Es factible aplicar varios criterios de aceptación para el algoritmo [Duarte, 2007]. En la figura 9 se muestra el algoritmo ILS.

```
1. Inicia Búsqueda Local Iterada ( )
2. π = Generar (Solución_Inicial);
3. π'= Búsqueda_Local (π);
4.     Repetir
5.         π''= Perturbación (π');
6.         π^op = Búsqueda_Local (π'');
7.         π'= Criterio_Aceptación (π', π^op);
8.     Hasta (Criterio_Parada)
9. Fin Búsqueda Local Iterada
```

Figura 9. Algoritmo de la Búsqueda Local Iterada.

2.8 Algoritmo Memético (MA, Memetic Algorithm)

Los algoritmos meméticos surgen a través de las técnicas de optimización basadas en la combinación de ideas tomadas de diferentes algoritmos metaheurísticos y son basados en una población de agentes. Algunas de las técnicas básicas que implementa este algoritmo son la búsqueda basada en población, o una búsqueda local. Una las características distintivas del MA, es la incorporación del dominio del problema a resolver. En el ámbito de las ciencias naturales el término "meme" fue acuñado por R. Dawkins [Dawkins, 1976],

donde un "meme" es descrito como: una idea, melodía, frase o la moda de vestir, entre otros significados. Así como los genes se trasmiten mediante los cromosomas que el ser humano pasa a la siguiente generación, los memes se transmiten a través de los seres humanos de generación en generación mediante el conocimiento. Este tipo de algoritmo fuero desarrollado por P. Moscato [Moscato, 1989], quien propuso hibridar un algoritmo genético mediante los procedimientos del Recocido Simulado (SA).

El algoritmo MA reúne e implementa diferentes ideas y conceptos de técnicas de resolución, este tipo de algoritmos están basados en población, ya que generan una conjunto de soluciones candidatas para el problema a resolver. Estos algoritmos son semejantes a los algoritmos genéticos pero con la diferencia que típicamente incluyen un proceso de Búsqueda Local. El algoritmo memético básico cuenta con cinco procesos los cuales son: generación de la población inicial, búsqueda local, cruza, mutación y proceso de selección. El funcionamiento del algoritmo memético comienza por generar una población inicial (soluciones iniciales), en el contexto de los meméticos al conjunto de soluciones generadas se les denomina *agentes*, dichos agentes se relacionan entre sí, para realizar una competición y cooperación, los agentes mejoran su vida aplicando la búsqueda local. La competencia es una etapa de selección y actualización de elementos y la cooperación es la etapa donde crean nuevos agentes. Para crearlos existen dos operadores básicos de reproducción que son el cruzamiento y la mutación. Este algoritmo realiza el proceso de generación de agentes, competición y cooperación, hasta que alcance su criterio de parada. En la figura 10 se muestra el algoritmo memético. [Cotta, 2007] [Duarte, 2007] [Gendreau, 2010] [Merz, 1999] [Moscato, 2003].

```
1.  Inicia Memético
2.  t = 0;
3.  Inicializar_Poblacion (P(0));
4.  Búsqueda_Local (P(0));
5.      Repetir
6.          P' = Seleccion (P(t));
7.          Cruza (P');
8.          Mutación (P');
9.          Búsqueda_Local (P')
10.         P(t+1) = Criterio_Aceptación (P(t), P');
11.         t= t+1;
12.     until (criterio_parada)
13. Fin Memético
```

Figura 10. Algoritmo Memético.

2.9 Descripción de las Instancias.

Las instancias que fueron utilizadas en el estudio experimental fueron las denominadas UB-I, de las cuales las más retadoras son XLOLIB y Rand AII. A continuación se describen cada uno de los conjuntos de instancias que forman al conjunto UB-I.

Instancias Random de tipo AI: Los problemas de tipo I (llamados RandomAI), son generados de una distribución uniforme [0,100]. Este tipo de problemas fue propuesto en [Reinelt, 1985]. El problema fue originalmente generado de una distribución uniforme [0,25000]. Existen 3 conjuntos de 25 instancias, cuyos tamaños son: n=100, 150 y 200. Existen adicionalmente 25 instancias de 500 elementos.

XLOLIB: Estas instancias fueron generadas con base en la distribución que tienen las tablas LOLIB, las cuales estas contienen valores de las tablas de entrada-salida provenientes de la economía europea, belga, alemana y estadounidense. El conjunto XLOLIB contiene instancias de tamaños n=150 y n= 250.

Random AII: Los problemas que abarcan estas instancias son generados por el cálculo del número de veces que un sector aparece en una posición más alta que otra en una serie de permutaciones generadas al azar. Para resolver un problema de tamaño n, n / 2 permutaciones deben ser generadas. Dentro de estas instancias existen 25 casos con tamaños de 100, 150 y 200, respectivamente

Random B: en este tipo de problemas aleatorios, se trato de influir en la dificultad del problema para la experimentación computacional. Para llegar a este fin se generaron matrices donde las entradas de los valores arriba de la diagonal son realizadas de manera uniforme en el intervalo [0, U_1] y los valores debajo de la diagonal son realizados en el intervalo [0, U_2], donde $U_1 \geq U_2$. La diferencia $U_1 - U_2$ afecta la dificultad. Subsecuentemente, las instancias son transformadas a una forma normal y una permutación aleatoria es aplicada.

Spec: dentro de este conjunto de instancias se encuentran 4 grupos: Ex, econ las cuales son generadas a partir de la tablas de entrada-salida de la economía estadounidense, apt estas instancias fueron generadas a partir de los resultados del tour de tenis de los años 1993/1994 y por último se tienen las paley graphs.

Capítulo 3

Estado del Arte

Algunos de los trabajos más importantes que se identificaron en el estudio del estado del arte del problema LOP son: la búsqueda tabú con estrategias de intensificación y diversificación desarrollada por Laguna en [Laguna, 1998], el trabajo sobre vecindades exponenciales y búsqueda local de Congram [Congram, 2000], el trabajo que muestra un algoritmo memético de sólido rendimiento desarrollado por Schiavinotto [Schiavinotto, 2003], además de los trabajos de Campos [Campos, 2001], Chiarini [Chiarini, 2004], Martí [Martí, 2010] y Martí [Martí, 2009]. De los tres primeros trabajos se presenta una descripción más detallada, debido a su relevancia para el desarrollo de esta tesis.

3.1 Principales Trabajos Relacionados

3.1.1 Intensification and Diversification with Elite Tabú Search Solutions for the Linear Ordering Problem, [Laguna, 1998]

Se identificaron los aspectos de mayor importancia en este trabajo sobre LOP, entre ellos están: las Metaheurísticas, las vecindades, la solución inicial y la búsqueda local utilizadas.

En este trabajo se desarrolla una metaheurística Búsqueda Tabú, que incorpora con fines de intensificación una estrategia de Re-encadenamiento de trayectorias basada en soluciones elite, para la intensificación básica utiliza memoria basada en criterio tabú, y dentro de la diversificación se utilizo memoria de largo plazo mediante frecuencias, adicionalmente implementa una estrategia de diversificación basada en la operación de inversión de la permutación propuesta por Chanas y Kobylanski [Chanas, 1996].

Tres elementos críticos en el procedimiento de búsqueda tabú son: Los movimientos de inserción, las vecindades, y la medida de influencia.

Los movimientos de inserción son utilizados como el mecanismo primario para moverse de una solución a otra en TS_LOP. Se define INSERT_MOVE*(p_j,i)* como un movimiento que consiste en borrar p_j de la posición actual *j* para ser insertada en la posición *i* (i.e., entre el sector actual p_{i-1} y p_i). Esto da lugar a la operación en la permutación *p*', como se muestra a continuación:

$$p' = \begin{cases} (p_1,..., p_{i-1}, p_j, p_i,..., p_{j-1}, p_{j+1},..., p_m) \text{ para } i < j \\ (p_1,..., p_{j-1}, p_{j+1},..., p_i, p_j, p_{i+1},..., p_m) \text{ para } i > j \end{cases} \quad (3.1)$$

Entonces el valor de la función objetivo que corresponde a *p*' se obtiene de la siguiente forma:

$$C_E(p') = \begin{cases} C_E(p) + \sum_{k=i}^{j-1} \left(e_{p_j p_k} - e_{p_k p_j} \right) & \text{para } i < j \\ C_E(p) + \sum_{k=j+1}^{i} \left(e_{p_k p_j} - e_{p_j p_k} \right) & \text{para } i > j \end{cases} \quad (3.2)$$

La complejidad para la evaluación de *p*' en ambos casos es *O (m)*. El siguiente ejemplo ilustra el mecanismo de inserción. Suponga que la matriz correspondiente a la permutación p= {1, 2, 3, 4, 5, 6, 7} tiene la forma como se muestra a continuación, y que el método INSET_MOVE (p_6, 2) deberá ser evaluado.

$$E(p) = \begin{bmatrix} 0 & 12 & 5 & 3 & 1 & 8 & 3 \\ 6 & 0 & 3 & 6 & 4 & 4 & 2 \\ 8 & 5 & 0 & 5 & 7 & 0 & 3 \\ -2 & 7 & 2 & 0 & -3 & 6 & 0 \\ 8 & 0 & 3 & -1 & 0 & 4 & 1 \\ 9 & 1 & 6 & 2 & 13 & 0 & 4 \\ 2 & 9 & 4 & -5 & 8 & 1 & 0 \end{bmatrix}$$

Los elementos de la matriz que se encuentran dentro de los rectángulos son aquellos necesarios para evaluar la función objetivo correspondiente a la nueva permutación p' = {1, 6, 2, 3, 4, 5, 7). El valor de la función objetivo para *p* es 78, el de *p'* se convierte en:

$$C_E(p') = 78 + (1-4) + (6-0) + (2-6) + (13-4) = 78 + 8 = 86$$

Puede también ser verificado que el mejor movimiento que involucra a p_6 es INSERT_MOVE (p_6, 3), cuyo valor es 11. El valor del movimiento se obtiene de la diferencia entre el valor de la función objetivo antes y después del movimiento. En términos matemáticos:

$$MoveValue = C_E(p') - C_E(p) \qquad (3.3)$$

En este trabajo se estudiaron dos vecindades: N_1, es una vecindad de inserción mediante movimientos de sectores contiguos, y N_2, también es una vecindad de inserción, pero en ella se aplican los movimientos de inserción clásicos. Las definiciones para ellas son:

$$N_1 = \{ p' : INSERT\_MOVE(p_J, i),\ para\ j = 1,..,m-1 \text{ e } i = j+1 \}$$
$$N_2 = \{ p' : INSERT\_MOVE(p_j, i),\ \text{para } j = 1,...,m \text{ e } i = 1,2,...,\ j\text{-}1,\ j+1,...,m \} \qquad (3.4)$$

En conjunto con estas vecindades se estudian dos criterios para la selección del vecino. La estrategia *Best* selecciona el movimiento con el valor del movimiento más grande de entre todos los movimientos en el vecindario. La estrategia *first*, por otra parte, revisa la lista de sectores (en el orden dado por la permutación actual) en busca del primer sector (p_f) cuyo valor sea estrictamente positivo (un movimiento tal que $C_E(p') > C_E(p)$). El movimiento seleccionado por la estrategia *First* es entonces $INSERT\_MOVE(p_f, i^*)$, donde i^* es la posición que maximiza $C_E(p')$. Note que para N_1, $i^* = f+1$, mientras que para N_2, $i*$ es elegido de $i = 1,2,...,\ f\text{-}1,\ f+1,...,m$. Por lo tanto, la estrategia *first* usada en combinación con N_1 es equivalente a la buscar el primer movimiento de mejora en la vecindad.

Estas dos estrategias de selección y los dos tipos de vecindad anteriores, se combinaron para formar los cuatro diferentes tipos de búsquedas locales que fueron evaluados: *first(N_1), best(N_1), first(N_2),*y *best(N_2).*

- $first(N_1)$: realiza una búsqueda entre todos los vecinos y encuentra el primero que mejore, esto lo realiza mediante intercambios consecutivos.

- $best(N_1)$: mediante intercambios consecutivos revisa todos los vecinos, toma el de mayor valor (aunque no sea mejor que la solución actual).
- $first(N_2)$: realiza una búsqueda entre todos los vecinos y encuentra el primero que mejore. Aplica movimientos de inserción clásica.
- $best(N_2)$: esta búsqueda local revisa todos los vecinos, toma el de mayor valor (aunque no sea mejor que la solución actual). Esto lo realiza mediante movimientos de inserción clásica.

La medida de influencia se utiliza para sesgar la selección de los sectores, dando mayor probabilidad de selección a los sectores con mayor valor de este indicador. Previo a la fase de intensificación se realiza el cálculo del peso de los diferentes sectores, los cuales son almacenados en una matriz (matriz de influencia). Para realizar el cálculo de este peso se utiliza la siguiente ecuación:

$$w_j = \sum_{i \neq j} (e_{ij} + e_{ji}) \tag{3.5}$$

En la metaheurística se alterna entre la de intensificación y la de diversificación, para lo cual se inicia eligiendo aleatoriamente un sector, la probabilidad de que se elija el sector *j* es proporcional a el peso de w_j que representa su medida de influencia. Para el sector seleccionado, se revisan todos los posibles movimientos a realizar en su vecindario aplicando el criterio correspondiente y, el vecino con el mayor valor (o con el menor decremento de la función objetivo) se selecciona. Y se ejecuta incluso cuando el valor del movimiento no es positivo, dando lugar a un empeoramiento en el valor de la función objetivo actual. El sector que fue movido ahora se convierte en un *Tabú-Activo* dentro de la tabla *Tabú*, y este por lo tanto no podrá ser seleccionado en la siguientes iteraciones para realizar inserción.

El número de veces que el sector *j* ha sido elegido para un movimiento es acumulado en la tabla de frecuencias. Esta información es utilizada para la propuesta de la diversificación. La fase de intensificación termina después de un máximo de iteraciones consecutivas sin mejora alguna. Antes de que el algoritmo abandone esta fase, el procedimiento *first* (N_2) es aplicado a la mejor solución encontrada. Al momento de aplicar este procedimiento voraz, se garantiza como resultado un óptimo local.

La fase de diversificación es realizada hasta un máximo de iteraciones, en cada una de éstas un sector es seleccionado aleatoriamente, donde la probabilidad de selección del sector *j* es inversamente proporcional a la cantidad de la frecuencia. El sector elegido es colocado en la mejor posición, el cual es determinado por el valor del movimiento asociado con el movimiento de inserción en N_2^j.

La intensificación adicional se realiza mediante el proceso de Re-encadenamiento de trayectorias, para lo cual se utiliza la solución encontrada al final de la fase de intensificación, el proceso consiste en aplicar a la solución actual los movimientos transformación hacia un conjunto de soluciones élite (estas son las mejores soluciones encontradas a lo largo del proceso completo y sirven de soluciones guía).

Por otro lado, se incluye adicionalmente un proceso diversificador basado en memoria de largo plazo que complementa la fase de diversificación básica. Este proceso está inspirado en el algoritmo de Chanas y Kobylanski y aprovecha la característica de simetría muy particular del problema de ordenamiento lineal. En LOP, cuando se maximiza la suma de los valores sobre la diagonal principal, por consiguiente se minimiza la suma de los elementos que están bajo la diagonal principal.

3.1.2 Search Space Analysis of the Linear Ordering Problem, Darmstadt University of Technology. [Schiavinotto, 2003]

Las metaheurísticas utilizadas por [Schiavinotto, 2003] son: Búsqueda Local Iterativa y Algoritmos Genéticos, a continuación se describe cada uno de ellos.

Los resultados del análisis del espacio de búsqueda de LOP sugieren que los métodos que sean capaces de aprovechar el buen desempeño de la búsqueda local y que tengan correlación positiva de distancias de aptitud son prometedores para este problema. Los resultados de investigaciones previas sugieren que la Búsqueda Local Iterativa y el Algoritmo Memético son metaheurísticas que tienen esas características. [Boese, 1996] [Merz, 1999] [Merz, 2000].

El algoritmo de la Búsqueda Local Iterada es muy simple, pero a la vez muy poderoso como lo muestran las diversas aplicaciones con resultados de éxito. Éste algoritmo itera sobre la búsqueda local aplicando tres pasos importantes:

1. Perturbar una solución óptima a nivel local.
2. Optimizar esta solución con la búsqueda local elegida.
3. Elegir, basado en algún criterio aceptable, la solución que pasa a la siguiente fase de perturbación.

A continuación se presenta el algoritmo de la Búsqueda local iterativa conocida como ILS en la figura 11:

```
1. Inicia Algoritmo_BusquedaLocalIterada
2. π ← GeneraciónSoluciónInicial
3. π ← Búsqueda_Local (π);
4.     Repetir
5.         π' ← Perturbación (π);
6.         π' ← Búsqueda_Local (π');
7.         π ← Criterio_Aceptación (π, π', Antecedentes);
8.     until (Condición_Parada);
9. Fin Algoritmo_BúsquedaLocalIterada
```

Figura 11. Algoritmo de Búsqueda Local Iterada.

Las características finales del algoritmo ILS son:

- *GenerateInitialSolution:* el procedimiento de búsqueda local es el núcleo del algoritmo y el rendimiento global del ILS depende fuertemente de ello. Se uso la búsqueda local LS_f , porque obtiene los mejores resultados.
- Perturbación: como operador de perturbación se utilizó el movimiento de intercambio, debido a que el efecto de este movimiento no puede ser revertido por movimientos de inserción en el siguiente paso. El número de movimientos de intercambio para ser aplicado en una perturbación es un parámetro del algoritmo.
- *Criterio de Aceptación:* esto determina la siguiente solución a la cual se aplica la perturbación, probándose diferentes enfoques, de los cuales se eligió uno mediante un procedimiento automático de afinamiento:
  - Mejor aceptado: un nuevo optimo local es aceptado solamente si la función objetivo es ms grande que la mejor solución actual, es decir, es $f(\pi') > f(\pi)$;
  - Aceptación de un decremento pequeño: un número óptimo local es aceptado si la función objetivo $f(\pi')$ es más grande que $(1-\varepsilon) * f(\pi)$, donde ε es un parámetro que se afinó;

- Recocido simulado: se aplicó una prueba probabilística de aceptación/rechazo basada en el criterio estándar de Metrópolis. En este caso se afinaron tres parámetros (la temperatura inicial, la temperatura de enfriamiento, y el número de pasos entre reducciones consecutivas de temperatura).

Otra metaheurística implementada fue el algoritmo memético, éste es un algoritmo evolutivo que incorpora una búsqueda local. El algoritmo resultante es un algoritmo poblacional que realiza eficazmente búsquedas en el espacio de soluciones óptimas locales. A continuación se muestra el seudocódigo del algoritmo de memético en la figura 12.

```
1.  Poblacion ← {};
2.  for i=1…m do { m es el número de individuos}
3.      π ← BúsquedaLocal(GeneraSoluciónAleatoria());
4.      Población ← Población ∪ {π};
5.  end for
6.      Repetir
7.        descendientes ← {};
8.        for i ← 1 … # Cruzamiento do
9.            tomar π_a, π_b de la población
10.           descendientes ← descendientes ∪ {BúsquedLlocal(Cruzamiento(π_a, π_b)};
11.       end for
12.       Población ← SelecciónMejor(Población ∪ descendientes, m);
13.       if es la misma calidad de la solución promedio por mucho tiempo Then {Diversificación}
14.           Población ← SeleccionMejor(Población,1);
15.           For i=1…m-1 do { m es el número de individuos}
16.               π ← BúsquedaLocal (GeneraSolucionesAleatorias());
17.               Población ← Poblacion ∪ {π};
18.           End for
19.       End if
20.   Hasta (criterio de parada);
```

Figura 12. Algoritmo Memético.

En el primer paso del algoritmo anterior se crea una población inicial de individuos en forma aleatoria, y a cada uno se le aplica la búsqueda local LS_f. Entonces, en cada iteración un número de nuevos individuos son creados aplicando el operador de cruzamiento. Los individuos para los cuales se aplicó el cruzamiento fueron elegidos aleatoriamente de acuerdo a una distribución uniforme como es usual en los algoritmos de alto rendimiento. El operador de cruzamiento toma dos individuos de la población actual y los combina para formar un individuo. A cada hijo generado se le aplica la búsqueda local

LS_f. Finalmente, los mejores *m* individuos de la población original y de los recién generados son seleccionados para formar la nueva población; se eliminan los duplicados.
Se usó una diversificación cuya ejecución es disparada cuando el promedio de la función objetivo de la población no cambia en un número de pasos. Esta diversificación consistió en genera nuevamente la población inicial (aleatoriamente), manteniendo solamente el mejor individuo de toda la población. Los operadores de cruzamiento que se evaluaron fueron: DPX, CX, OB y Rank. Otro aspecto importante en el artículo fue el análisis de las vecindades estudiadas en este trabajo, sobre las que se detallan sus características principales.

La primera vecindad, N_x, es definida por la operación intercambio, esta operación se define como: *intercambio:* $\Pi \times \{1,...,n\}^2 \to \Pi$, donde π es el conjunto de todas las permutaciones y se tiene para $i \neq j$:

$$intercambio\ (\pi, i, j) \triangleq (..., \pi_{i-1}, \pi_j, \pi_{i+1}, ..., \pi_{j-1}, \pi_i, \pi_{j+1}) \tag{3.5}$$

El tamaño de esta vecindad es: $|N_x| = n(n-1)/2$.

Una segunda vecindad, N_I, es definida por la operación de inserción, en la cual un elemento en la posición *i* es insertado en otra posición *j*. formalmente, $insert : \Pi \times \{1,...,n\}^2 \to \Pi$ es definido para $i \neq j$ mediante la expresión 3.6.

$$insert(\pi, i, j) \triangleq \begin{cases} (..., \pi_{i-1}, \pi_{i+1}, ..., \pi_j, \pi_i, \pi_{j+1}, ...) & i < j; \\ (..., \pi_{j-1}, \pi_i, \pi_j, ..., \pi_{i-1}, \pi_{i+1}, ...) & i > j; \end{cases} \tag{3.6}$$

La vecindad basada en inserciones tiene un tamaño $|N_1(\pi)| = (n-1)^2$.

La ecuación 3.7 presenta la función de incremento para el costo de un movimiento de inserción.

$$\Delta_I(\pi, i, j) \triangleq f(insert(\pi, i, j)) - f(\pi) \tag{3.7}$$

La operación asociada con esta operación está definida en 3.8.

$$\Delta_I(\pi, i, j) \triangleq \begin{cases} \sum_{k=i+1}^{j} c_{\pi_k \pi_i} - c_{\pi_i \pi_k} & i < j \\ \sum_{k=j}^{i-1} c_{\pi_i \pi_k} - c_{\pi_k \pi_i} & i > j \end{cases} \tag{3.8}$$

El costo para la evaluación de esta función es $O(|i-j|)$ y, por lo tanto, $O(n)$. Si se implementa una búsqueda local sencilla basada en esta vecindad, dicha búsqueda partiría tomando un índice *i*, tentativamente tratando de realizar todos los movimiento posibles de $insert(\pi, i, j)$ tomando en cuenta que *j* tendría un valor del rango de 0 a *n-1*. Dado que para cada movimiento de la evaluación Δ-función es lineal, el costo de la visita a una vecindad, que se realiza de manera exhaustiva es $O(n^3)$. Sin embargo, este procedimiento puede llegar a ser significativamente acelerado, si la vecindad es visitada de una manera más sistemática. Un caso particular de un movimiento de inserción es realizado si $i=j\pm1$; a este movimiento se le denomina *swap*, su función Δ es:

$$\Delta_S(\pi, i, j) \triangleq \begin{cases} c_{\pi_i\pi_j} - c_{\pi_j\pi_i} & i = j+1 \\ c_{\pi_j\pi_i} - c_{\pi_i\pi_j} & i = j-1 \end{cases} \tag{3.9}$$

Por lo tanto, el costo de la evaluación de Δ_S es constante. Además, un movimiento de inserción (con argumento i y j) es siempre equivalente al movimiento swap |i-j|. Por ejemplo, para n=7 y i =2, j=5, se tiene:

$$\pi = \begin{array}{l} 1\,2\,3\,4\,5\,6 \quad\quad\quad\quad\quad\quad\quad\quad \rightarrow \\ 1\,3\,2\,4\,5\,6 \;= \text{insert}(\pi,2,3) \;\rightarrow \\ 1\,3\,4\,2\,5\,6 \;= \text{insert}(\pi,2,4) \;\rightarrow \\ 1\,3\,4\,5\,2\,6 \;= \text{insert}(\pi,2,5). \end{array}$$

En el ejemplo, se puede ver que todas las permutaciones visitadas, cuando se transforma π por la aplicación de *insert(π,2,5)* se encuentra dentro de la vecindad-*insert* de π y es siempre obtenida aplicando un movimiento *swap* en un paso previo. Por lo tanto, la idea es usar solo movimiento *swaps* para visitar en su totalidad la vecindad N_I. Para cada índice *i* se aplicara todos los posibles movimientos *insert* (π,*i*,*j*), lo cual se realizara en 2 etapas. La primera empieza, con los índices *j* que van desde *i*-1 a 0 y entonces para índices *j* que varían de *i*+1 a *n*-1. En cada etapa una solución puede ser obtenida mediante la aplicación de un movimiento *swap*. Por lo tanto, cada solución en el vecindario puede ser obtenido en tiempo constante y por lo tanto el costo computacional total para evaluar la vecindad insertada se convierte en $O(n^2)$. Esta técnica fue inspirada por el método que Congram aplico para el algoritmo de Dynasearch.

Una de las búsquedas locales a la cual se le pone una atención especial dentro de este artículo es la llamada LS_f, dentro de la cual usa una regla de pivoteo, con una mezcla de las búsquedas locales *best* y el *first*, en la figura 13 se obseva :

```
1. Function visit_NI (π)
2.     for i= 0…n – 1 do
3.         r̄ ← argmax_{r.r≠i} f (insert(π, i, r)
4.         π' = insert (π, i, r)
5.         if f(π') > f(π) then
6.             return (π')
7.         end if
8.     end for
9. return (π)
```

Figura 13. Algoritmo de la búsqueda local LS_f.

La exploración de los índices para encontrar el mejor movimiento para cada índice es hecho explorando la evaluación de la función delta en tiempo constante. Se indica con LS_f la búsqueda local en N_I basada en la visita que se acaba de presentar; LS_f es una búsqueda local en N_I usando una primera estrategia de mejora aleatoria, donde la vecindad es explorada en un orden aleatorio; esta ultima vecindad examina un régimen requerido la Δ-función para ser calculado en tiempo lineal.

El área de oportunidad que se detecto para poder realizar una contribución al presente trabajo de investigación fue: el algoritmo memético tiene un sólido rendimiento, sin embargo es factible de mejorar debido a que utiliza una estrategia de diversificación basada en la generación aleatoria de la población cada vez que detecta algún estancamiento.

3.1.3 Polynomially Searchable Exponential Neighborhoods for Sequencing Problems in Combinatorial Optimisation [Congram, 2000]

El metaheurístico analizado dentro de este artículo es denominado dynasearch iterativo el cual es un algoritmo PSEN(polynomially searchable exponential neighbourhoods: vecindarios exponenciales de búsqueda polinómial), el cual utiliza la programación dinámica para combinar conjuntos independientes de movimientos individuales dentro de una vecindad tradicional. El conjunto de movimientos independientes para un movimiento

dynasearch individual puede ser realizado como una sola iteración. Las diferentes propuestas de vecindades que se realizan dentro del artículo son las que se mencionan a continuación.

Vecindad de inserción disjunta

Consiste de una combinación de movimientos disjuntos y puede ser formada utilizando exclusivamente las operaciones de bloques disjuntos. En la figura 14 se muestra un ejemplo:

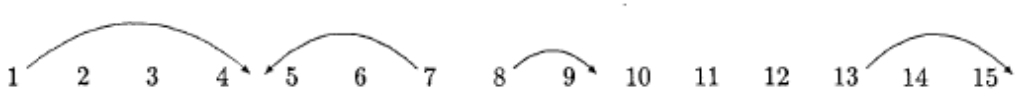

Figura 14. Vecindad de inserción disjunta.

Un bloque disjunto puede ser formado por la combinación de dos bloques disjuntos con el operador del bloque disjunto.

Vecindad de inserción anidada

Consiste de una combinación de movimientos anidados uno dentro del otro, y puede ser formado utilizando exclusivamente las operaciones de bloques anidados.

Figura 15. Vecindad de inserción anidada.

Como se muestra en la figura 15 un bloque anidado puede ser formado mediante la realización de una operación de bloques anidados.

Vecindades de inserción anidada dentro de inserción disjunta

La idea básica de esta vecindad, es que aunque un movimiento de la vecindad disjunta contenga el conjunto de movimientos disjuntos los cuales producen el aumento máximo en

el valor de la función objetivo, un aumento adicional puede ser posible mediante la realización de movimientos que son anidados dentro de los movimientos disjuntos. La elección de movimientos disjuntos puede cambiar en función de la asignación de los movimientos adicionales anidados. Sólo movimientos anidados son permitidos en los movimientos disjuntos, permitiendo así que el mejor movimiento de la vecindad anidada que se encuentra para cada secuencia parcial de un paso de pre-procesamiento. En consecuencia, hay una reducción en la complejidad computacional, en comparación con las vecindades combinadas la disjuntas y anidadas. En la figura 16 se muestra un ejemplo.

Vecindades de inserción disjunta dentro de inserciones anidadas.

Es una extensión de la vecindad anidada exactamente del mismo modo que la anidada dentro de la disjunta es una extensión de vecindades disjuntas. En la figura 17 se muestra un ejemplo:

Figura 17. Vecindades de inserción disjunta dentro de inserción anidadas.

En este trabajo se estudia otra forma de generar nuevas vecindades mediante el método de programación dinámica, el cual puede ser utilizado para generar nuevas vecindades que se incorporen en otras metaheurísticas. Por otro lado las vecindades que el autor propone y de las cuales presenta únicamente un estudio teórico pueden ser implementadas y evaluadas en un trabajo de investigación.

De los trabajos anteriormente descritos el más importante para el desarrollo de este proyecto fue [Shiavinotto, 2004], ya que los algoritmos que se tomaron como referencia fueron desarrollados a partir de la teoría descrita en este articulo. Atendiendo los resultados presentados por Shiavinotto en su estudio del análisis del espacio de búsqueda para LOP; los cuales muestran que las metaheurísticas más prometedoras son aquellas capaces de explotar el rendimiento de la búsqueda local y que presenten una correlación de distancias de aptitud positivas; se decidió abordar en esta tesis la solución del problema propuesto mediante un algoritmo de Búsqueda Local Iterada y un algoritmo Memético en los cuales

se incorporaron estrategias tendientes a mejorar su desempeño incorporando diversificación. Para evaluar los algoritmo propuestos se planteó utilizar las instancias mas retadoras que son las XLOLIB y Random AII.

Capitulo 4

Algoritmo de Búsqueda Local Iterativa

LOP es un problema ampliamente estudiado por la comunidad de investigadores sobre problemas de optimización y las metaheurísticas para resolverlos. Una de las metaheurísticas de solución que se aborda en este trabajo es el algoritmo de búsqueda local iterativa (ILS, Iterated Local Search). Dicha metaheurística se utiliza para realizar una evaluación preliminar de estrategias que aportan diferentes grados de diversificación dentro del espacio de soluciones. Esta diversificación se incorpora mediante estrategias de construcción inicial y procesos de búsqueda local. En este capítulo se describe la metaheurística ILS; en la cual se incorporan las búsquedas locales y los algoritmos de construcción inicial para su evaluación; y los parámetros de configuración utilizados en el algoritmo.

4.1 Búsqueda Local Iterativa

Duarte, en su trabajo [Duarte, 2007], define el procedimiento de búsqueda local iterativa en los siguientes términos:

"Dado un espacio de soluciones SS, una vecindad N y un procedimiento de búsqueda local LS, se establece una correspondencia entre SS y el espacio de soluciones de óptimos locales SS. ILS se basa en un procedimiento de búsqueda local sobre el espacio de soluciones de óptimos locales SS*".*

Esta definición describe de manera general el algoritmo ILS, sin embargo el algoritmo que se tomó como referencia para desarrollar el algoritmo ILS dentro este proyecto, fue el presentado por Schiavinotto [Schiavinotto, 2003].

El algoritmo de la Búsqueda Local Iterada tiene una estructura bastante simple, iterando en un ciclo en el cual intervienen tres elementos importantes:

1. Perturbar una solución óptima a nivel local.
2. Optimizar esta solución con la búsqueda local elegida.
3. Elegir, basado en algún criterio de aceptación, la solución que pasa a la siguiente fase de perturbación.

La estructura básica del método de búsqueda local iterativa en la figura 18.

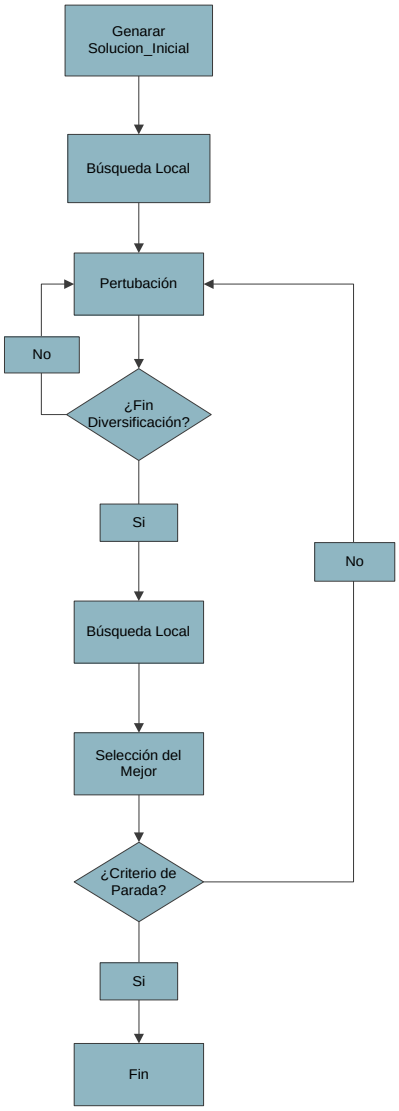

Figura 18. Estructura del algoritmo de búsqueda local iterativa.

El algoritmo de búsqueda local iterada comienza su funcionamiento, creando una solución inicial, en este caso se probaron tres tipos de construcciones las cuales son: aleatoria, ordenada y heurística de Becker, de las cuales la que tuvo un mejor desempeño fue la generación aleatoria, por lo que fue elegida para incorporarse en algoritmo ILS final. El siguiente proceso es la perturbación, en este caso se evaluó dos procesos de perturbación: movimiento de intercambio y movimiento de inserción aleatoria, la solución resultante de la perturbación es optimizada localmente, el proceso de perturbación se realiza mientras se cumpla la condición de diversificación. Posteriormente se aplica una búsqueda local más intensificadora; para lo cual se evaluaron la búsqueda local *Best*, *First* y LS_f, de las cuales se eligió la búsqueda local *Best*, por que provee un mayor rendimiento. El último proceso que se evaluó fue el criterio de selección, mediante el cual se decide qué soluciones pasarán nuevamente al proceso de perturbación. La condición que controla la ejecución de este algoritmo define la terminación de este. En este caso el algoritmo se ejecuta hasta que se cumplan 10 segundos de tiempo. A continuación se muestra el algoritmo de búsqueda local iterada en la figura 19.

```
1.  Algoritmo Búsqueda Local Iterada
2.  x ← Generar_SoluciónInicial( );
3.  x ← BusquedaLocalBest( x);
4.      Repetir
5.        {Perturbación}
6.            Repetir
a.              x' ← Insercion_Aleatoria(x);
b.              x' ← BusquedaLocalBest (x');
7.            Hasta (Fin_Diversificacion);
8.        {Termina Perturbación}
9.        x'←BusquedaLocalBest(x');
10.       x←CriteriodeAceptación(x,x',historia);
11.     Hasta (criterio de parada)
```

Figura 19. Algoritmo de búsqueda local iterativa.

4.2 Parámetros de entrada

Los principales parámetros que utiliza la configuración final de este algoritmo son:

- *dim.* Es el tamaño ó dimensión de cada instancia.

- *srand(1471).* Es la semilla que se utiliza en los procesos aleatorios, en este caso se mantiene constante y es igual a 1471.
- *maxtime.* Es el criterio de parada del algoritmo, en este caso se establece un tiempo de ejecución de 10 segundos para el algoritmo.
- *global.* El número máximo de iteraciones sin mejora.
- *fin_diver*. Su valor es la dimensión de la matriz entre dos, este parámetro es el criterio de parada para el proceso de la perturbación.
- *Criterio de aceptación.* Define la calidad mínima que debe tener una solución para pasar nuevamente al proceso de perturbación. En este caso una solución pasará nuevamente a la perturbación si no empeora más de un 8.7% a la mejor solución.
- *Criterio de parada*. Determina cuando el algoritmo ILS termina su ejecución; en este caso es una condición combinada y establece que el algoritmo seguirá ejecutándose mientras que el número de iteraciones sin mejora sea menor que 100 y que el tiempo no supere el valor de maxtime (10 segundos).
- *Fin_Diversificación*. Este criterio establece el número de iteración que comprende el proceso de perturbación; en este caso el valor que se le asigna es dim/2.

Capítulo 5

Algoritmo Memético Propuesto

Para la solución de LOP se ha mostrado que los algoritmos poblacionales tales como el algoritmo memético de Schiavinotto [Schiavinotto, 2003] y la búsqueda dispersa de García [García, 2005] presentan un desempeño sobresaliente a diferencia de otras metaheurísticas como se muestra en el trabajo de Martí et. al [Martí, 2009]. El algoritmo memético es el último de los algoritmos que se aborda dentro de este proyecto, en el cual se incorporan las estrategias de diversificación elegidas en el capítulo cuatro, como lo son las búsquedas locales y las construcciones iniciales. Además de estas estrategias se evaluó un ajuste del tamaño de la población. En este trabajo se toma como base el algoritmo memético desarrollado por Schiavinotto. Las estrategias que se evalúan son: búsquedas locales, algoritmos de construcción inicial y el ajuste del tamaño de la población; así como los parámetros de configuración utilizados.

5.1 Algoritmo Memético

Los algoritmos meméticos tienen su comienzo a finales de los años ochenta, aunque en décadas anteriores algunos trabajos tienen características similares. Estos algoritmos surgieron de la idea de combinar conceptos y estrategias de diferentes metaheurísticas con la idea de incorporar ventajas en el algoritmo. Estos algoritmos se nombraron "meméticos" a partir del surgimiento del término "meme", acuñado por R. Dawkins [Dawkins, 1976] en el contexto de la evolución cultural, y que puede verse como un sinónimo del término "gen" en el contexto de la evolución natural.

La característica distintiva de un algoritmo memético es que además de transmitirse información genética (implícita en el cromosoma) entre generaciones, se transmite "conocimiento", donde el conocimiento es información que se percibe del medio ambiente. Esta información sobre el medio ambiente puede transmitirse mediante estrategias

heurísticas, algoritmos aproximados y búsquedas locales, resultando en un algoritmo híbrido. El algoritmo memético mantiene una estructura que es bastante similar a la que tiene el algoritmo genético, con un elemento distintivo que es la búsqueda local. El algoritmo implementado incluye una etapa de diversificación adicional, e itera en un ciclo en el cual intervienen cuatro elementos importantes:

1. Operador de Cruza, mediante el cual se crean los nuevos agentes.
2. Búsqueda local, permite optimizar la población obtenida en la etapa de cruzamiento.
3. Criterio de aceptación, define los agentes de la población que pasan a la siguiente fase de diversificación o cruzamiento.
4. Diversificación, reinicializa la población en caso de estancamiento, aplicando enseguida optimización a nivel local.

El diagrama de la figura 20 muestra la estructura del algoritmo memético implementado.

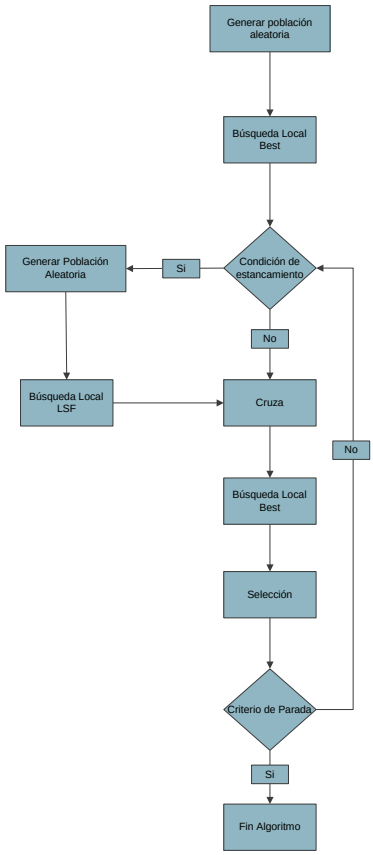

Figura 20. Estructura del Algoritmo Memético.

El algoritmo memético comienza creando una población inicial; en este caso se utilizó una construcción inicial aleatoria porque mostró tener un desempeño superior al de las demás estrategias evaluadas; y se aplica una búsqueda local *Best* a la población resultante.

El operador de cruza OB que se aplica sobre la población tiene por objetivo generar un número de hijos igual al tamaño de la población, para lo cual se seleccionan dos padres y se aplica el operador. El primer padre se elige seleccionando uno de los elementos de la población mediante selección uniforme siempre y cuando cumpla la condición de no ser un hijo reciente, el padre dos se selecciona aplicando una ruleta que da preferencia a los mejores elementos de la población y al igual que el padre uno no debe ser un hijo reciente; un hijo reciente es aquel que fue generado en la misma iteración global.

El operador OB incluye dos fases, en la primera, el primer padre es copiado en el hijo y en la segunda fase se seleccionan k posiciones ($0< k < n$), los elementos en las k posiciones se ordenan de acuerdo al orden en el segundo padre y se copian en el hijo. Una vez aplicado el operador de cruzamiento se aplica la búsqueda local *Best* para mejorar localmente la calidad de los elementos a partir de los cuales se formará la nueva población. Un ejemplo de este operador se muestra en la figura 21. El criterio de selección para construir la nueva población consiste en elegir los mejores 25 elementos de la población actual y de los hijos generados, eliminando los duplicados. Se implementa un proceso de diversificación adicional, que consiste en reinicializar aleatoriamente la población, con el fin de evitar convergencia prematura. Esta reinicialización de la población se aplica cuando el valor promedio de la función objetivo de la población no cambia durante cinco generaciones consecutivas. A esta nueva población se le aplica una búsqueda local de tipo LS_f. . El algoritmo memético implementado se muestra en la figura 22.

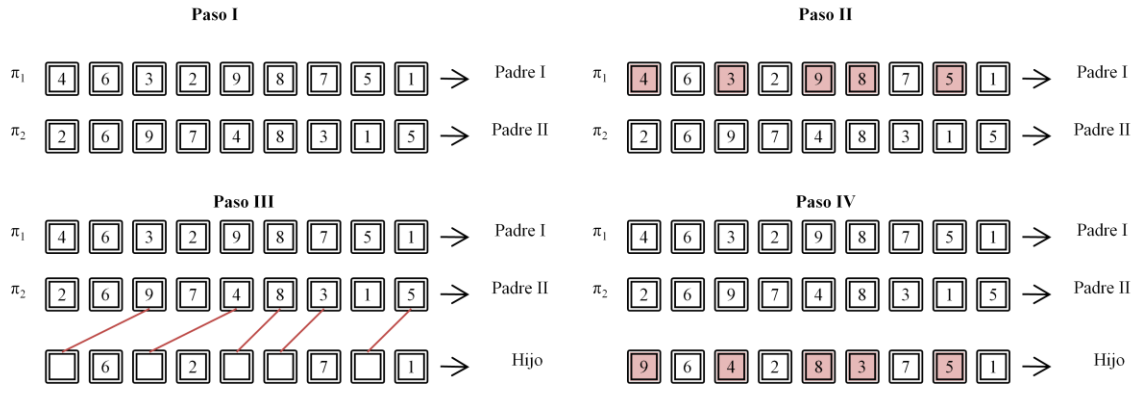

Figura 21. Ejemplo del cruzamiento OB.

```
1.  Algoritmo Memético
2.  Pob ← Generacion_Población_Aleatoria ( );
3.  Pob_mej ← BúsquedaLocalBest (Pob);
4.      Mientras (Not Condición_Parada)
5.          {Inicia_Diversificación}
6.              if (Mismo_Promedio = = Maximo_iteraciones)
7.                  Eliminar_Excepto_Mejor (Pob);
8.                  Pob ← Generacion_Población_Aleatoria ( );
9.                  Pob_mej ←BusquedaLocalLS_f (Pob);
10.             endif
11.         {Fin_Diversificación}
12.         for i ← 1 to tam_Pob
13.             Seleccionar_π_a_π_b_de Pob ( );
14.             Hijo ← Cruzamiento_OB (π_a, π_b);
15.         endfor
16.         Pob_mej ← BúsquedaLocalBest (Pob);
17.         Pob_ord ← Ordenar_Población (Pob);
18.         nueva_Pob ← Seleccionar_Mejor (Pob_ord)
19.     Fin Mientras
20. Fin Algoritmo Memético.
```

Figura 22. Algoritmo Memético Propuesto.

5.2 Parámetros de entrada

Los parámetros principales que se tomaron para el desarrollo de este algoritmo son los siguientes:

- *dim.* Tamaño ó dimensión de cada instancia.
- *srand(1471).* Semilla que se utiliza en los procesos aleatorios, en este caso se mantiene constante y es igual a 1471.
- *tamPoblacion.* Tamaño de la población inicial, en este caso es de 36 elementos.
- *elemConservDeLaPob*. Cantidad de hijos que se mantendrán después de la seleccion de los más aptos.
- *maxtime.* Tiempo límite de ejecución del algoritmo (10 segundos).
- *maxIterMismoProm*. Número máximo de itraciones con igual promedio de la función objetivo de la población.
- *iterMismoProm*. Esta variables representa el número de iteraciones con el mismo promedio de la función objetivo de la población.

- *factorElementosDeCruza*. Es el número de posiciones elegidas aleatoriamente del primer padre (0.4 **dim*).

5.3 Estrategias de diversificación propuestas

Con base en el análisis del estado del arte sobre el problema LOP, una de los aspectos que más influyen en el rendimiento de los algoritmos es lograr un balance adecuado entre la intensificación y diversificación dentro de la metaheurística utilizada, y en este trabajo de tesis se planteó alcanzar este balance mediante la incorporación de estrategias que promuevan una mejor diversificación del algoritmo. Se plantea un conjunto de estrategias que promueven la diversificación para mejorar el equilibrio en los procesos de diversificación-intensificación: incorporación de búsquedas locales con diferentes grados de intensificación, construcción inicial con diferentes niveles de intensificación, modificación del tamaño de la población y variación de la diversificación de la población mediante la variación del tiempo de ejecución. Las estrategias propuestas fueron probadas dentro del algoritmo memético implementado. En la figura 23 se muestra el algoritmo memético que se toma como base, para incorporar y evaluar las estrategias propuestas. Enseguida se describen las estrategias que obtuvieron el mejor desempeño dentro de los experimentos realizados.

```
1.  Poblacion ← {};
2.  for i=1…m do { m es el número de individuos}
3.        π ← BúsquedaLocal(GeneraSoluciónAleatoria());
4.        Población ← Población ∪ {π};
5.  end for
6.  Repetir
7.      descendientes ← {};
8.          for i ← 1 … # Cruzamiento do
9.                tomar π_a, π_b de la población
10.               descendientes ← descendientes ∪ {BúsquedLlocal(Cruzamiento(π_a, π_b)};
11.         end for
12.     Población ← SelecciónMejor(Población ∪ descendientes, m);
13.         if es la misma calidad de la solución promedio por mucho tiempo Then {Diversificación}
14.             Población ← SeleccionMejor(Población,1);
15.         For i=1…m-1 do { m es el número de individuos}
16.               π ← BúsquedaLocal (GeneraSolucionesAleatorias());
17.               Población ← Poblacion ∪ {π};
18.         End for
19.         End if
20. Hasta (criterio de parada);
```

Figura 23. Algoritmo Memético base.

5.3.1 Búsquedas locales con diferentes grados de intensificación.

El algoritmo memético, tuvo varias fases de evolución, en las cuales se realizaron nueve evaluaciones experimentales con dos tipos de búsquedas locales: búsqueda local *Best* y LS_f, las cuales fueron aplicadas los puntos 1, 2, y 3 de la metaheurística. La figura 5.6 muestra la estructura del algoritmo memético con los tres puntos de aplicación. Se producen nueve versiones distintas del algoritmo memético, con las dos búsquedas locales, y un caso en donde no se utiliza búsqueda local en dos puntos. El conjunto de todas las versiones evaluadas se muestra en la Tabla 1.

Tabla 1. Algoritmo memético con los tres puntos de aplicación.

Caso	Algoritmo	1	2	3
1	MDB	Na	*Best*	*Best*
2	MCB	*Best*	Na	*Best*
3	MCDB	*Best*	*Best*	*Best*
4	MDL	Na	LS_f	*Best*
5	MCL	LS_f	Na	*Best*
6	MCDL	LS_f	LS_f	*Best*
7	MCDBL	*Best*	LS_f	*Best*
8	MCDLB	LS_f	*Best*	*Best*
9	MCDBN	Na	Na	*Best*

De todas las versiones evaluadas los dos algoritmos que obtuvieron el mejor desempeño se muestran a continuaciones.

Algoritmo MCDBL (Algoritmo Memético con Búsqueda Local *Best* en la construcción inicial y Búsqueda Local LS_f en la fase de diversificación). El algoritmo MCDBL se puede observar en la figura 24, a continuación se muestra los principales elementos que contiene este algoritmo:

```
1.  Algoritmo Memético
2.  Pob ← Generacion_Población_Aleatoria ( );
3.  Pob_mej ← BúsquedaLocalBest (Pob); // 1
4.  Mientras (Not Condición_Parada)
5.          {Inicia_Diversificación}
6.              if (Mismo_Promedio = = Maximo_iteraciones)
7.                  Eliminar_Excepto_Mejor (Pob);
8.                  Pob ← Generacion_Población_Aleatoria ( );
9.                  Pob_mej ← BusquedaLocalLS_f (Pob);// 2
10.             endif
11.         {Fin_Diversificación}
12.         for i ← 1 to tam_Pob
13.             Seleccionar_π_a_π_b_de Pob ( );
14.             Hijo ← Cruzamiento_OB (π_a, π_b);
15.         endfor
16.         Pob_mej ← BúsquedaLocalBest (Pob); // 3
17.         Pob_ord ← Ordenar_Población (Pob);
18.         nueva_Pob ← Seleccionar_Mejor (Pob_ord)
19. Fin Mientras
20. Fin Algoritmo Memético.
```

Figura 24. Algoritmo Memético MCDBL.

Algoritmo MCDBN (Algoritmo Memético sin Búsqueda Local en la construcción inicial y en la fase de diversificación). El algoritmo MCDBN se puede observar en la figura 25, a continuación se muestra los principales elementos que contiene este algoritmo:

```
1.  Algoritmo Memético
2.  Pob ← Generacion_Población_Aleatoria ( );
3.  Sin_Búsqueda_Local;
4.  Mientras (Not Condición_Parada)
5.          {Inicia_Diversificación}
6.              if (Mismo_Promedio = = Maximo_iteraciones)
7.                  Eliminar_Excepto_Mejor (Pob);
8.                  Pob ← Generacion_Población_Aleatoria ( );
9.                  Sin_Búsqueda_Local;
10.             endif
11.         {Fin_Diversificación}
12.         for i ← 1 to tam_Pob
13.             Seleccionar_π_a_π_b_de Pob ( );
14.             Hijo ← Cruzamiento_OB (π_a, π_b);
15.         endfor
16.         Pob_mej ← BúsquedaLocalBest (Pob);
17.         Pob_ord ← Ordenar_Población (Pob);
18.         nueva_Pob ← Seleccionar_Mejor (Pob_ord)
19. Fin Mientras
20. Fin Algoritmo Memético.
```

Figura 25. Algoritmo Memético MCDLB.

5.3.2 Construcciones iniciales con diferentes niveles de intensificación.

Para la población inicial se probaron dos casos, uno donde se aplica la búsqueda local *Best* a todos los elementos de la población inicial (MB) y otro en cual no se aplica búsqueda local (MNB). Estos casos corresponden a los casos 2 y 9 de la Tabla 5.1.

5.3.3 Modificación del tamaño de la población.

Para lograr una variación en la diversificación del algoritmo memético se probaron diferentes tamaños de población, en este caso se utilizan tres tamaños de la población: 50 (M50), 100 (M100) y 36. El tamaño para el cual se obtuvo un mejor desempeño fue con 36 elementos.

5.3.4 Variación de la diversificación de la población mediante la variación del tiempo de ejecución.

En esta última estrategia se plantea ejecutar el algoritmo memético utilizando dos tiempos límites, 10 y 600 segundos de CPU (M10 y M600); tal como se muestra en el trabajo de Martí [Martí, 2009]. El Algoritmo que se utiliza para estos dos tiempos es el MCDBL (Algoritmo Memético con búsqueda local *Best* en la construcción inicial y con búsqueda local LS_f dentro de la fase de diversificación). Con esta estrategia se prueba que el algoritmo mantiene un buen balance en la diversificación-intensificación a los largo del tiempo, ya que en 600 segundos logra mejorar 16 de las mejores soluciones conocidas. Los resultados para estos dos tiempos se muestran en la sección 6.8.

Capitulo 6

Resultados Experimentales

En este trabajo se realizaron cuatro tipos de experimentos sobre el algoritmo memético que se desarrolló tomando como base el algoritmo de referencia descrito por Schiavinotto [Schiavinotto, 2003]. Esta experimentación fue equiparada con el trabajo reportado por Martí [Martí, 2009], para lo cual se tomaron las especificaciones descritas en dicho artículo, a continuación se describen dichas especificaciones.

6.1 Plataforma experimental.

Se realizaron cuatro tipos de experimentos para evaluar: estrategias de búsqueda local, estrategias de intensificación, el tamaño de la población como estrategia de diversificación y el desempeño del algoritmo memético propuesto. La experimentación se realizó utilizando una computadora Dell PowerEdge 2600, con procesador dual Intel XEON 3.06 GHZ, disco duro de 70 GB y 4GB de memoria ram. Se utilizó el sistema operativo Windows XP Profesional SP3 y el compilador Visual Studio 2005.

Se utilizaron las mismas instancias (UB-I) y condiciones de tiempo (10 seg. de CPU y 600 seg. de CPU), establecidas por Rafael Martí et al en [Martí, 2009].

6.2 Resultados de referencia

En esta investigación se utilizan las instancias UB-I descritas en la sección 2.10, ya que se considera que son las más retadoras, tomándose como referencia los resultados reportados en el trabajo desarrollado por Martí [Martí, 2009] sobre una biblioteca de referencia de heurísticas y metaheurísticas para LOP. Dado que los mejores valores conocidos actualmente para cada una de las instancias consideradas fueron obtenidos ejecutando

durante una hora el algoritmo memético (MS) de Schiavinotto [Schiavinotto, 2003], se realiza este mismo experimento con el algoritmo propuesto, superando para algunas instancias los resultados de Schiavinotto. Los resultados de las instancias para las cuales se superan los mejores valores conocidos se muestran en la tabla 6.10, se mejora el mejor valor conocido para una instancia en la prueba con 10 segundos y para 16 instancias en la prueba con 600 segundos. Las Tablas 2 y 3 muestran los resultados reportados para los algoritmos en el trabajo de la biblioteca de referencia para 10 y 600 segundos respectivamente, se puede observar que los algoritmos memético de Schiavinotto y búsqueda Tabú de Laguna tienen el mejor desempeño. Además de estos algoritmos se reporta el desempeño de los siguientes algoritmos metaheurísticos: Búsqueda de Vecindad Variable (VNS), Recocido Simulado (SA), Búsqueda Dispersa (SS), Grasp (Grasp), Búsqueda Local Iterada (ILS) y Algoritmo Genético.

Mientras que en las Tablas 4 y 5 se muestran estos mismos resultados para 10 y 600 segundos respectivamente, pero considerando los nuevos mejores valores alcanzados por el algoritmo implementado en este trabajo y reportados en la tabla 6.10. Para cada grupo de instancias y cada algoritmo, la tabla contiene el promedio del porcentaje de desviación con respecto al mejor valor conocido y el número de mejores conocidos encontrados.

Tabla 2. Resultados reportados en [Martí, 2009], tiempo de ejecución de 10 seg.

Inst	Indicador	BT	MA	VNS	SA	SS	Grasp	ILS	GA
Rand AI	%Dev	0.12	0.05	0.47	1.77	0.26	0.42	13.42	10.59
	#Mejores	5	33	0	0	1	0	0	0
Rand AII	%Dev.	0.01	0.00	0.01	0.07	0.02	0.04	23.42	35.97
	#Mejores	3	39	8	0	0	0	0	0
Rand B	%Dev.	0.00	0.00	0.00	0.31	0.04	0.00	13.44	0.91
	#Mejores	20	20	11	0	11	20	0	0
XLOLIB	%Dev.	0.62	0.12	0.42	0.53	0.68	1.14	22.37	23.99
	#Mejores	0	2	0	0	0	0	0	0
Special	%Dev.	0.43	0.03	0.50	2.05	0.32	0.65	15.97	9.27
	#Mejores	3	4	2	0	3	3	0	0

Tabla 3. Resultados reportados en [Martí, 2009], tiempo de ejecución de 600 seg.

Inst	Indicador	BT	MA	VNS	SA	SS	Grasp	ILS	GA
Rand AI	%Dev	0.10	0.00	0.41	0.40	0.18	0.28	11.84	1.14
	#Mejores	20	100	0	0	2	2	0	0
Rand AII	%Dev.	0.00	0.00	0.01	0.11	0.01	0.02	18.97	0.10
	#Mejores	21	50	16	0	2	0	0	0
Rand B	%Dev.	0.00	0.00	0.03	0.01	0.02	0.00	11.37	0.87
	#Mejores	20	20	12	12	13	20	0	0
XLOLIB	%Dev	0.39	0.00	0.34	0.61	0.82	0.57	19.47	2.15
	#Mejores	0	78	0	0	0	0	0	0
Special	%Dev	0.24	0.00	0.21	1.18	0.21	0.53	13.12	2.22
	#Mejores	4	7	3	2	3	3	0	0

Tabla 4. Resultados incorporando los nuevos mejores conocidos (t = 10 seg).

Inst	Indicador	**BT**	**MA**	VNS	SA	SS	Grasp	ILS	GA
Rand AI	%Dev	**0.12**	**0.04**	0.47	1.77	0.26	0.42	13.42	10.59
	#Mejores	**5**	**33**	0	0	1	0	0	0
Rand AII	%Dev.	**0.01**	**0.00**	0.01	0.07	0.02	0.04	23.42	35.97
	#Mejores	**3**	**39**	8	0	0	0	0	0
Rand B	%Dev.	**0.00**	**0.00**	0.00	0.31	0.04	0.00	13.44	0.91
	#Mejores	**20**	**20**	11	0	11	20	0	0
XLOLIB	%Dev	**0.62**	**0.12**	0.42	0.53	0.68	1.14	22.37	23.99
	#Mejores	**0**	**2**	0	0	0	0	0	0
Special	%Dev	**0.43**	**0.04**	0.50	2.50	0.32	0.65	15.97	9.27
	#Mejores	**3**	**3**	2	0	3	3	0	0

Tabla 5. Resultados incorporando los nuevos mejores conocidos (t = 600 seg.).

Inst	Indicador	**BT**	**MA**	VNS	SA	SS	Grasp	ILS	GA
Rand AI	%Dev	**0.10**	**0.00**	0.41	0.40	0.18	0.28	11.84	1.14
	#Mejores	**20**	**100**	0	0	2	2	0	0
Rand AII	%Dev.	**0.00**	**0.00**	0.01	0.11	0.01	0.02	18.97	0.10
	#Mejores	**21**	**50**	16	0	2	0	0	0
Rand B	%Dev.	**0.00**	**0.00**	0.03	0.01	0.02	0.00	11.37	0.87
	#Mejores	**20**	**20**	12	12	13	20	0	0
XLOLIB	%Dev	**0.39**	**0.00**	0.34	0.61	0.82	0.57	19.47	2.15
	#Mejores	**0**	**66**	0	0	0	0	0	0
Special	%Dev	**0.24**	**0.02**	0.21	1.18	0.21	0.53	13.12	2.22
	#Mejores	**4**	**5**	3	2	3	3	0	0

6.3 Prueba de Wilcoxon.

Como las metaheurísticas analizadas son aleatorias, para cada una se realiza una única corrida y se evalúa el desempeño comparativo aplicando la prueba de hipótesis no paramétrica de Wilcoxon. La prueba de Wilcoxon es utilizada para analizar muestras correlacionadas de tal forma que se pueda saber si ambas poseen una misma (Fig. 26_a) media o no (Fig. $26._b$).

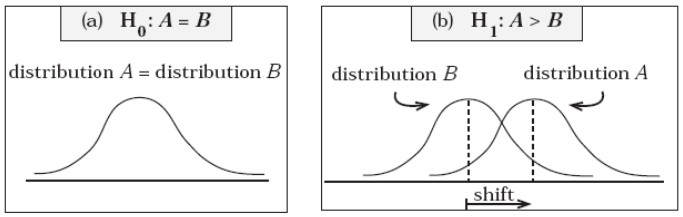

Figura 26. Distribuciones.

Para realizar esta prueba se requiere que: las muestras X_a y X_b tengan la misma escala y que los valores de X_a y X_b hayan sido tomados de manera aleatoria a partir de su población. Este tipo de prueba no requiere probar que los datos siguen una determinada distribución de probabilidad, ni se asume que siguen una distribución normal. La hipótesis nula considera que ambas muestras tienen la misma media y el propósito de la prueba de Wilcoxon, es determinar si la hipótesis nula se acepta o se rechaza. Para realizarla se aplica el siguiente procedimiento:

a) Primero se calcula la diferencia entre las dos muestras $X_a - X_b$
b) Se calcula el valor absoluto de el paso anterior $|X_a - X_b|$
c) Se ordenan los datos con relación al valor absoluto
d) Los valores en cero se eliminan y se determina el número de valores diferentes de cero (n).
e) Se ordenan de menor a mayor todos los valores.
f) Si se tienen k valores empatados en la posición p, se les asigna la posición $(k\text{-}1) + (1/k)$.
g) Se calculan $R+$ y $R-$, sumando las posiciones correspondientes a las diferencias positivas y negativas respectivamente.

h) En una tabla de Wilcoxon, se determina el intervalo $I(n,\alpha)= (r,R)$ considerando una confiabilidad de 1- α, donde *r* y *R* son los valores menor y mayor respectivamente, para el intervalo obtenido de la tabla de Wilcoxon.

i) Si $I = (R-, R+)$, entonces se establece la hipótesis nula si y solo si $I \subset I(n,\alpha)$

En el caso de un problema de maximización si X_a y X_b son dos muestras de la calidad (% error) de los algoritmos *a* y *b* respectivamente, entonces el algoritmo *a* tiene un mejor desempeño que el algoritmo *b* si $R+ > R-$. Además si $R+ \geq R$, entonces la diferencia en el desempeño es estadísticamente significativa (la hipótesis nula se rechaza), de lo contrario la diferencia no es estadísticamente significativa (la hipótesis nula se acepta). [García, 2008]

6.4 Experimentación Preliminar

El objetivo de este experimento es evaluar preliminarmente el desempeño del algoritmo ILS propuesto cuando utiliza las siguientes estrategias de construcción de soluciones iniciales y diversificación basadas en búsqueda local:

- ILSB: Algoritmo de Búsqueda Local Iterada con construcción inicial Becker.
- ILSA: Algoritmo de Búsqueda Local Iterada con construcción inicial aleatoria.
- ILSO: Algoritmo de Búsqueda Local Iterada con construcción inicial ordenada.
- ILSBB: Algoritmo de Búsqueda Local Iterada con Búsqueda Local *Best*.
- ILSBF: Algoritmo de Búsqueda Local Iterada con Búsqueda Local *First*.
- $ILSBLS_f$: Algoritmo de Búsqueda Local Iterada con Búsqueda Local LS_f.

La Tabla 6 y 7 muestra los resultados que se obtienen con las diferentes estrategias consideradas, cuando se resuelven las diferentes instancias consideradas. La tabla contiene para cada estrategia considerada y cada grupo de instancias, el porcentaje promedio de error con respecto al mejor valor conocido (Dev%), el número de mejores valores conocidos encontrados (#Mej); así como también se puede observa la evaluación de los mejores conocidos en la figura 28 y 30; y por último se muestra la suma de cada uno de estos indicadores (Total). Como se puede observar en la última columna y en la figura 27 y

29, los dos algoritmos con mejor desempeño en esta evaluación preliminar son ILSA, ILSO, ILSBB e ILSBLS$_f$.

Tabla 6. Resultados para las construcciones iniciales.

Algoritmo	Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)	Total
ILSB	Dev%	0.05149652	0.05502213	1.4818213	1.9126025	3.5009425
	#Mej.	4	0	0	0	4
ILSA	Dev%	0.00374547	0.01069641	0.9202137	1.2644701	2.1991257
	#Mej.	10	4	0	0	14
ILSO	Dev%	0.00710615	0.01367692	0.9449333	1.1752999	2.1410163
	#Mej.	6	1	0	0	7

Tabla 7. Resultados para la estrategia de Búsqueda Local.

Algoritmo	Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)	Total
ILSBB	Dev%	0.00385197	0.01124285	0.93285804	1.17053903	2.11834697
	#Mej.	8	4	0	0	12
ILSBF	Dev%	0.09508508	0.08272952	0.80436981	1.03163775	2.01382216
	#Mej.	0	0	0	0	0
ILSBLS$_f$	Dev%	0.01088905	0.01448707	0.84161563	1.22278827	2.08978002
	#Mej.	0	1	0	0	1

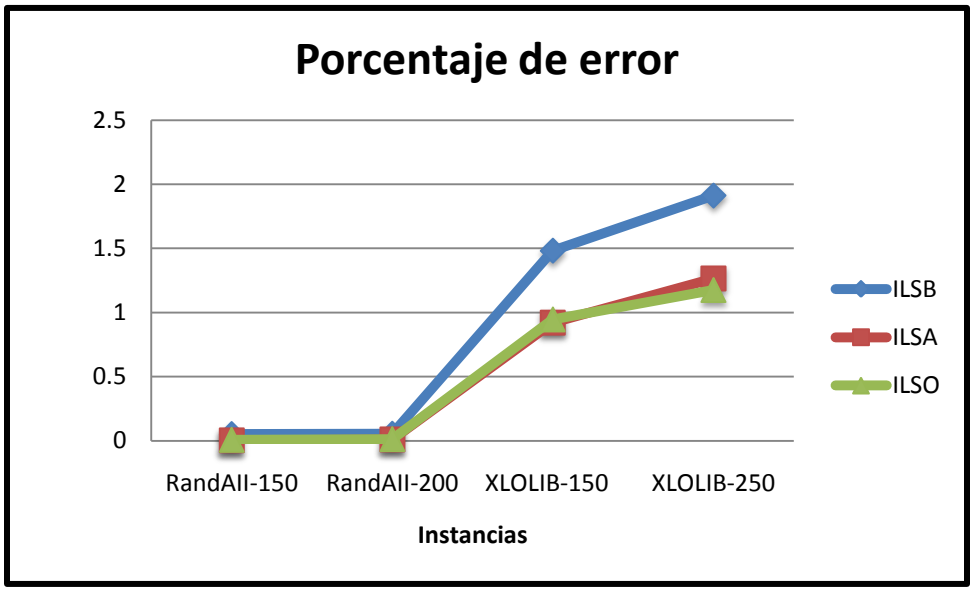

Figura 27. Evaluación de las construcciones iniciales en el algoritmo ILS.

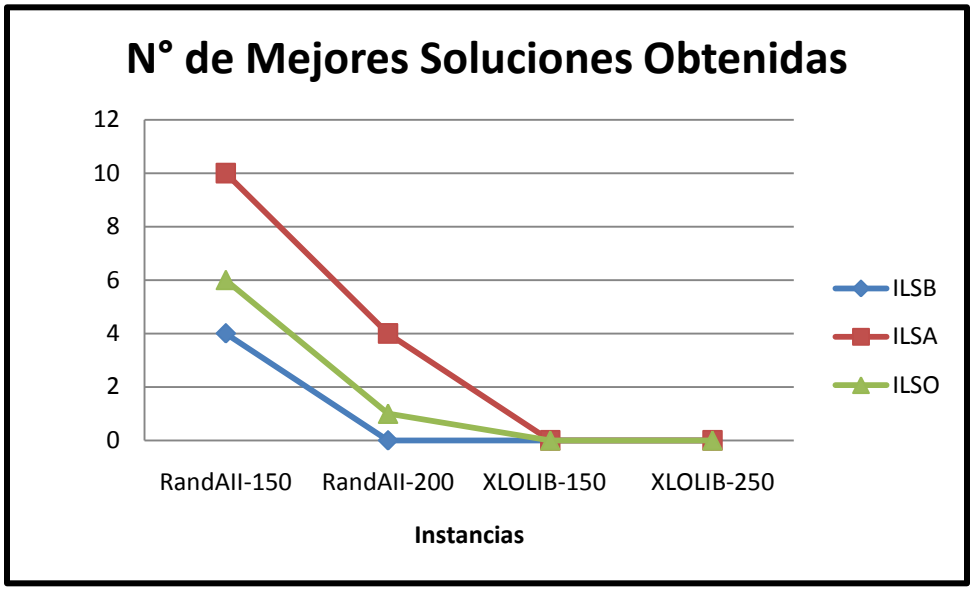

Figura 28. N° de Mejores soluciones obtenidas con las construcciones iniciales en el algoritmo ILS.

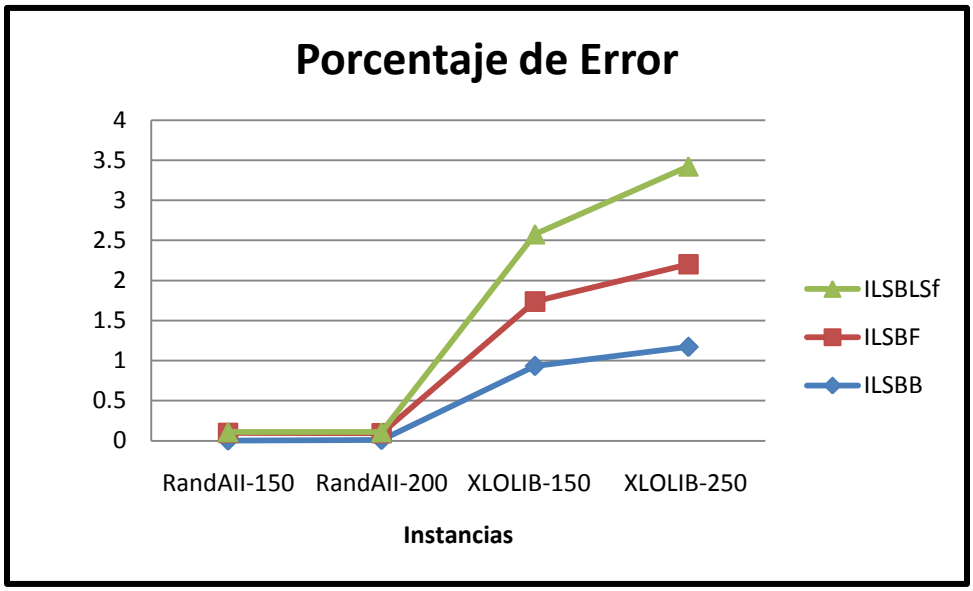

Figura 29. Evaluación de las búsquedas locales en el algoritmo ILS.

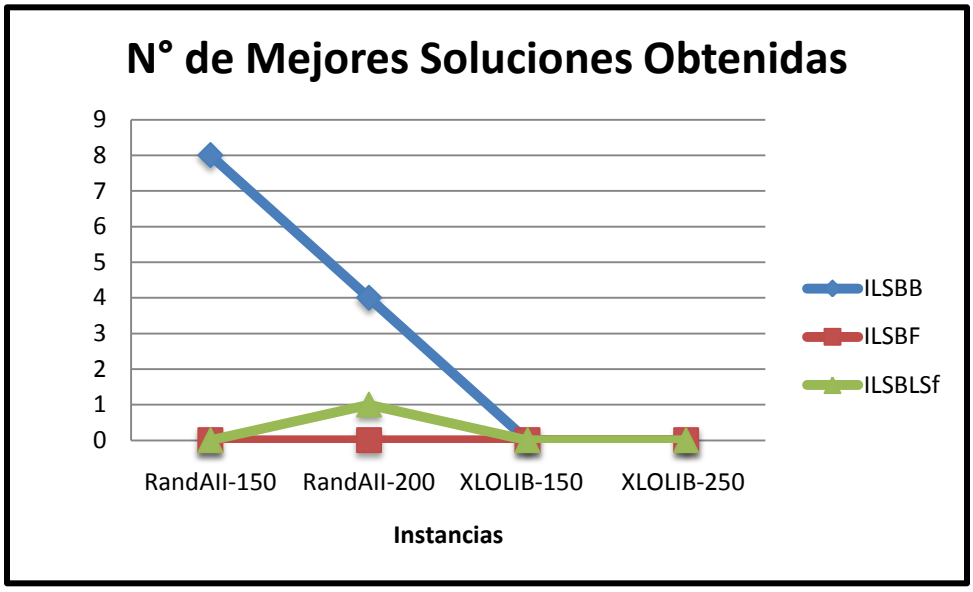

Figura 30. N° de Mejores soluciones obtenidas con las búsquedas locales en el algoritmo ILS.

Para determinar si la diferencia en el desempeño de ILSA, ILSO, ILSBB, ILSBF e $ILSBLS_f$, observada en la evaluación preliminar, es estadísticamente significativa se realiza la prueba de hipótesis de Wilcoxon. Cada algoritmo se ejecuta un tiempo límite de 10 segundos para cada instancia, manteniendo una semilla fija de 1471 con el fin de lograr reproducibilidad de los resultados. Las instancias se agrupan en cuatro conjuntos: RandAII de tamaño 100 (25 instancias), RandAII de tamaño 200 (25 instancias), XLOLIB de tamaño 150 (39 instancias), y XLOLIB de tamaño 150 (39 instancias). En la Tabla 8 se muestran los resultados del estudio de Wilcoxon, para la evaluación de los algoritmos ILSO e ILSA. Cuando *R*+ es mayor que *R*- el algoritmo ILSA es mejor que el algoritmo ILSO, ocurre lo contrario si *R*- es mayor que *R*+. Además, si el mayor de los valores es mayor o igual que el de referencia (*R*), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para el grupo de instancias XLOLIB (150) el algoritmo ILSA muestra un mejor desempeño que el algoritmo ILSO. Sin embargo, en las instancias Random AII existe empate y en las instancias XLOLIB (250) gana el algoritmo ILSO. Por lo tanto se considera que ambos tienen el mismo desempeño.

Tabla 8. Estudio de Wilcoxon para las construcciones iniciales (ILSO vs ILSA).

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	11	2	29	10
R^+	69	157	55	0
R^-	36	119	0	435
Ref	84	203	47	309
Mejor Alg.	Empate	Empate	ILSA	ILSO
Dif. Sig?	No	No	Si	Si

En la Tabla 9 se muestran los resultados del estudio de Wilcoxon, para la evaluación de los algoritmos ILSA e ILSB. Cuando *R*+ es mayor que *R*- el algoritmo ILSA es mejor que el algoritmo ILSB, ocurre lo contrario si *R*- es mayor que *R*+. Además, si el mayor de los valores es mayor o igual que el de referencia (*R*), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para todo el grupo de instancias el algoritmo ILSA muestra un mejor desempeño que el algoritmo ILSB. En este caso si existe una diferencia significativa en todos los casos de estudio. Por lo tanto se considera que el algoritmo ILSA tiene un mejor desempeño que el algoritmo ILSB.

Tabla 9. Estudio de Wilcoxon para las construcciones iniciales (ILSA vs ILSB).

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	1	0	0	0
R^+	291	325	779	780
R^-	6	0	1	0
Ref	219	236	531	531
Mejor Alg.	ILSA	ILSA	ILSA	ILSA
Dif. Sig?	Si	Si	Si	Si

En la Tabla 10 se muestran los resultados del estudio de Wilcoxon, para la evaluación de los algoritmos ILSO e ILSB. Cuando *R*+ es mayor que *R*- el algoritmo ILSO es mejor que el algoritmo ILSB, ocurre lo contrario si *R*- es mayor que *R*+. Además, si el mayor de los valores es mayor o igual que el de referencia (*R*), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para todo el grupo de instancias el algoritmo ILSO muestra un mejor desempeño que el algoritmo ILSB. En este caso si existe una diferencia significativa en todos los casos de estudio. Por lo tanto se considera que el algoritmo ILSO tiene un mejor desempeño que el algoritmo ILSB.

Tabla 10. Estudio de Wilcoxon para las construcciones iniciales (ILSO vs ILB).

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	0	0	0	0
R^+	310	323	779	780
R^-	15	2	1	0
Ref	236	236	531	531
Mejor Alg.	ILSO	ILSO	ILSO	ILSO
Dif. Sig?	Si	Si	Si	Si

En la Tabla 11 se muestran los resultados del estudio de Wilcoxon, para la evaluación de los algoritmos ILSBB e ILSBF. Cuando *R*+ es mayor que *R*- el algoritmo ILSBB es mejor que el algoritmo ILSBF, ocurre lo contrario si *R*- es mayor que *R*+. Además, si el mayor de los valores es mayor o igual que el de referencia (*R*), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para el grupo de instancias Random AII el algoritmo ILSBB muestra un mejor desempeño y para el grupo de instancias XLOLIB el algoritmo ILSBF muestra un mejor desempeño.

Tabla 11. Estudio de Wilcoxon para las búsquedas locales (ILSBB vs ILSBF).

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	0	0	0	0
R^+	325	325	245	210
R^-	0	0	535	570
Ref	236	236	531	531
Mejor Alg.	ILSBB	ILSBB	ILSBF	ILSBF
Dif. Sig?	Si	Si	Si	Si

En la Tabla 12 se muestran los resultados del estudio de Wilcoxon, para la evaluación de los algoritmos ILSBB e ILSBLS$_f$. Cuando *R*+ es mayor que *R*- el algoritmo ILSBB es mejor que el algoritmo ILSBLS$_f$, ocurre lo contrario si *R*- es mayor que *R*+. Además, si el mayor de los valores es mayor o igual que el de referencia (*R*), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para la mayoría del grupo de instancias los algoritmos muestra un empate. En este caso no existe una diferencia significativa en la mayoría de los casos de estudio. Por lo tanto se considera que los algoritmos ILSBB e ILSBLS$_f$ tienen un mismo desempeño.

Tabla 12. Estudio de Wilcoxon para las búsquedas locales (ILSBB vs ILSBLS$_f$).

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	0	1	0	0
R^+	290	211	302	506
R^-	35	89	478	274
Ref	236	219	531	531
Mejor Alg.	ILSBB	Empate	Empate	Empate
Dif. Sig?	Si	No	No	No

En la Tabla 13 se muestran los resultados del estudio de Wilcoxon, para la evaluación de los algoritmos ILSBF e ILSBLS$_f$. Cuando *R*+ es mayor que *R*- el algoritmo ILSBF es mejor que el algoritmo ILSBLS$_f$, ocurre lo contrario si *R*- es mayor que *R*+. Además, si el mayor de los valores es mayor o igual que el de referencia (*R*), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para la mayoría de los grupos de instancias el algoritmo ILSBLS$_f$ muestra un mejor desempeño que el algoritmo ILSBF. En este caso si existe una diferencia significativa en la mayoría de los casos de estudio. Por lo tanto se considera que el algoritmo ILSBLS$_f$ tiene un mejor desempeño que el algoritmo ILSBF.

Tabla 13. Estudio de Wilcoxon para las búsquedas locales (ILSBF vs ILSBLS$_f$).

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	0	0	0	0
R^+	0	0	446	628
R^-	325	325	334	152
Ref	236	236	531	531
Mejor Alg.	ILSBLS$_f$	ILSBLS$_f$	Empate	ILSBLS$_f$
Dif. Sig?	Si	Si	No	Si

Como resultado de esta evaluación preliminar, las estrategias que se eligieron para utilizar en el algoritmo memético fueron: la construcción de soluciones iniciales aleatoria (ILSA) y la diversificación basada en las búsquedas locales *Best* y *LS$_f$* (ILSBB e ILSBLSF$_f$).

6.5 Evaluación de estrategias de construcción de soluciones iniciales y de diversificación del algoritmo memético.

El objetivo de este experimento es evaluar preliminarmente el desempeño del algoritmo memético propuesto cuando utiliza las siguientes estrategias de construcción de soluciones iniciales y diversificación basadas en búsqueda local:

- Diversificación con Búsqueda Local Best (Div(Best)), MDB.
- Construcción Inicial con Búsqueda Local Best (Const(Best)), MCB.
- Construcción y diversificación con Búsqueda Local Best (Const(Best)-Div(Best)), MCDB.
- Diversificación con búsqueda local LSF(Div-LSF), MDL.
- Construcción Inicial con Búsqueda Local LSF (Const-LSF), MCL.
- Construcción y diversificación con Búsqueda Local Best (Const-Div LSF), MCDL.
- Construcción inicial con Búsqueda Local Best y Diversificación con Búsqueda Local LSF (Const-Div Best-LSF), MCDBL.

- Construcción inicial con Búsqueda local LSF y Diversificación con Búsqueda Local Best (Const-Div LSF-Best), MCDLB.
- Construcción y diversificación sin ningún tipo de búsqueda local (Const-Div Sin BL), MCDBN.

La Tabla 14 muestra los resultados que se obtienen con las diferentes estrategias consideradas, cuando se resuelven las diferentes instancias consideradas. La tabla contiene para cada estrategia considerada y cada grupo de instancias, el porcentaje promedio de error con respecto al mejor valor conocido (Dev%), el número de mejores valores conocidos encontrados (#Mej); así como también se puede observa la evaluación de los mejores conocidos en la figura 32; y por último se muestra la suma de cada uno de estos indicadores (Total). Como se puede observar en la última columna y en la figura 31, los dos algoritmos con mejor desempeño en esta evaluación preliminar son MCDBL y MCDBN.

Tabla 14. Resultados para la estrategia de Búsqueda Local.

Algoritmo	Búsqueda	Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)	Total
MDB	Div(Best)	Dev%	0.003	0.002	0.13	0.22	0.355
		#Mej.	21	12	0	0	33
MCB	Const(Best)	Dev%	0.003	0.01	0.13	0.23	0.373
		#Mej.	21	5	0	0	26
MCDB	Cons(Best)-Div(Best)	Dev%	0.009	0.01	0.14	0.23	0.389
		#Mej.	21	5	0	0	26
MDL	Div-LSF	Dev%	0.0003	0.002	0.134	0.237	0.3733
		#Mej.	22	8	0	0	30
MCL	Const-LSF	Dev%	0.0003	0.002	0.133	0.226	0.3613
		#Mej.	21	7	1	0	29
MCDL	Const-Div LSF	Dev%	0.0003	0.002	0.149	0.244	0.3953
		#Mej.	22	9	0	0	31
MCDBL	Const-Div Best-LSF	Dev%	0.0003	0.0022	0.1280	0.2080	0.3385
		#Mej.	20	7	1	0	28
MCDLB	Const-Div LSF-Best	Dev%	0.0004	0.0023	0.1449	0.2482	0.3958
		#Mej.	20	9	0	0	29
MCDBN	Const-Div Sin BL	Dev%	0.0001	0.0025	0.1530	0.1283	0.2839
		#Mej.	23	6	0	0	29

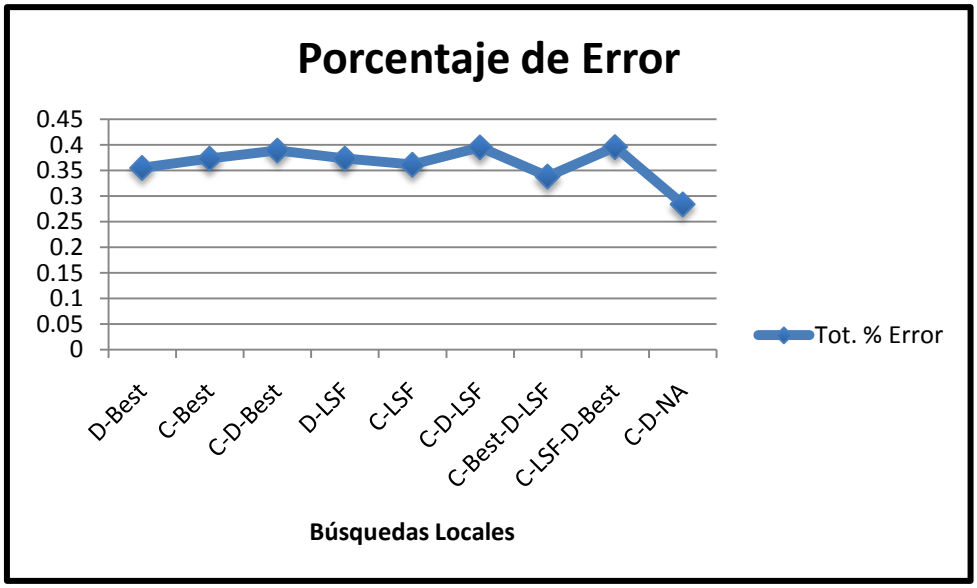

Figura 31 Porcentaje de error promedio para las instancias XLOLIB-Random AII.

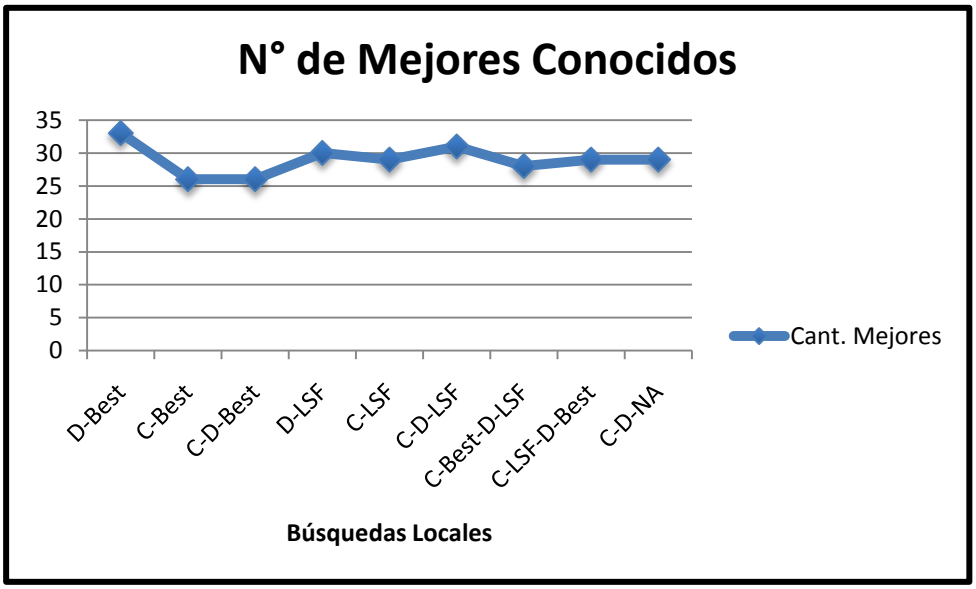

Figura 32. Comparativo del número de mejores conocidos para XLOLIB y RandomAII.

Para determinar si la diferencia en el desempeño de MCDBL y MCDBN, observada en la evaluación preliminar, es estadísticamente significativa se realiza la prueba de hipótesis de Wilcoxon. En la Tabla 15 se muestran los resultados del estudio de Wilcoxon. Cuando *R*+ es mayor que *R*- el algoritmo MCDBL es mejor que el algoritmo MCDBN, ocurre lo contrario si *R*- es mayor que *R*+. Además, si el mayor de los valores es mayor o igual que el

de referencia (*R*), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para todos los grupos de instancias el algoritmo MCDBL muestra un mejor desempeño que el algoritmo MCDBN. Sin embargo, esta diferencia no es estadísticamente significativa. Por lo tanto se considera que ambos tienen el mismo desempeño. Como resultado de esta evaluación, la estrategia que se utiliza en el algoritmo memético incluye la construcción de soluciones iniciales basada en búsqueda local *Best* y la diversificación basada en búsqueda local *LSF* (MCDBL),

Tabla 15. Estudio de Wilcoxon para las búsquedas locales.

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	3	16	39	39
R^+	0	83	411	416
R^-	6	53	369	364
Ref	NA	107	531	531
Mejor Alg.	Empate	MCDBL	MCDBL	MCDBL
Dif. Sig?	No	No	No	No

6.6 Evaluación de estrategias de generación de la población inicial

El objetivo de este experimento es realizar evaluar de forma preliminar el desempeño del algoritmo memético propuesto cuando se aplica la búsqueda local *Best* a todos los elementos de la población inicial (MB) y cuando no se aplica búsqueda local (MNB). La Tabla 16 muestra los resultados que se obtienen con las dos estrategias de generación de la población inicial, cuando se resuelven las diferentes instancias consideradas. La tabla contiene para cada estrategia de búsqueda local considerada y cada grupo de instancias, el porcentaje promedio de error con respecto al mejor valor conocido (Dev%), el número de mejores valores conocidos encontrados (#Mej); así como también se puede observa la

evaluación de los mejores conocidos en la figura 34; y por último se muestra la suma de cada uno de estos indicadores (Total). Como se puede observar en la última columna y el figura 33, cuando el algoritmo memético logra su mejor desempeño, menor porcentaje de error (0.2839) y el mayor número de mejores encontrados (30), cuando utiliza la búsqueda local.

Tabla 16 Evaluación de la intensificación del algoritmo.

Algoritmo	Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)	Total
MB	Dev%	0.0004	0.0023	0.1247	0.2300	0.2839
	#Mej.	21	8	1	0	30
MNB	Dev%	0.0010	0.0025	0.1263	0.2442	0.3740
	#Mej.	17	5	0	0	22

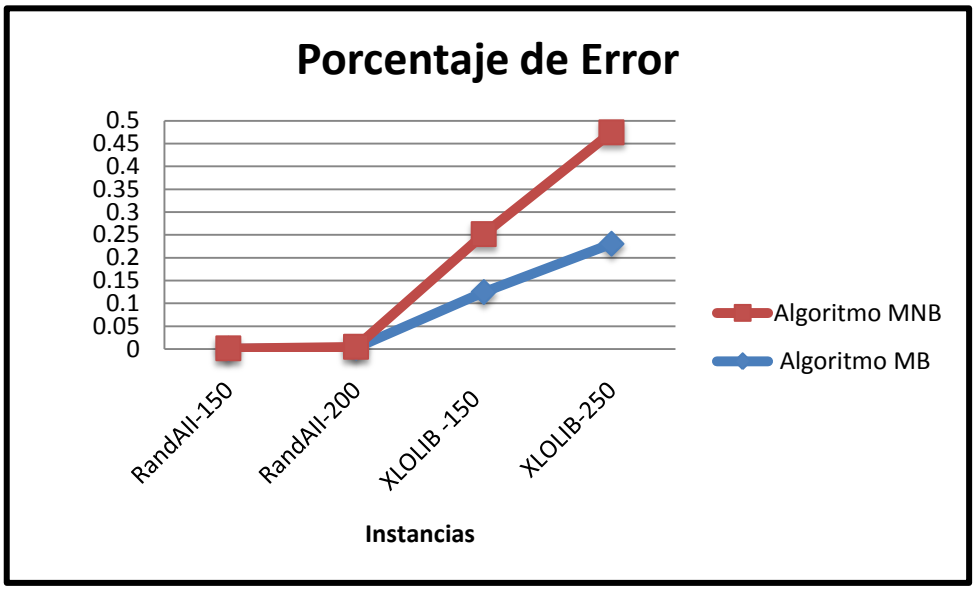

Figura 33. Evaluación de porcentaje de error entre los algoritmos MNB y MB.

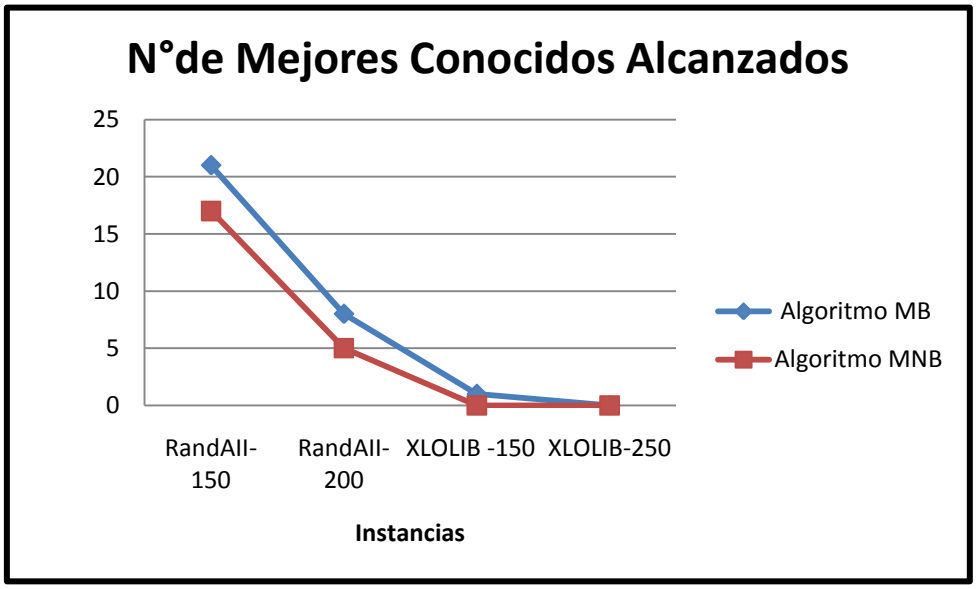

Figura 34. Evaluación de la Cantidad de numero de mejores obtenidos.

Para verificar los resultados de la evaluación preliminar, se realizó una prueba de Wilcoxon para determinar si las diferencias observadas, entre MB y MNB, son estadísticamente significativas. En la Tabla 17 se muestran los resultados del estudio de Wilcoxon, como se muestra en la tabla cuando R+ es mayor que R- el algoritmo MB es mejor que el algoritmo MNB, ocurre lo contrario si R- es mayor que R+. Además, si el mayor de los valores es mayor que el de referencia (R), entonces la diferencia en el desempeño es estadísticamente significativa, en caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para todos los grupos de instancias el algoritmo MNB muestra un mejor desempeño que el algoritmo MB. Sin embargo, esta diferencia no es estadísticamente significativa. Por lo tanto se considera que ambos tienen el mismo desempeño. Con base en estos resultados, se incorporó al algoritmo memético propuesto la estrategia de aplicar búsqueda local *Best* a cada uno de los individuos de la población inicial.

Tabla 17. Evaluación de la prueba de Wilcoxon, sobre la intensificación del algoritmo

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	6	18	39	39
R^+	2	64	367	365
R^-	19	107	413	415
Ref	21	131	531	531
Mejor Alg.	MNB	MNB	MNB	MNB
Dif. Sig.?	No	No	No	No

6.7 Evaluación de la modificación del tamaño de la población

El objetivo de este experimento es evaluar de forma preliminar el desempeño del algoritmo memético propuesta cuando utiliza diferentes tamaños de población, en este caso se utilizan dos tamaños de la población: 50 (M50) y 100 (M100) elementos. La Tabla 18 muestra los resultados que se obtienen con los dos tamaños de población considerados, cuando se resuelven las diferentes instancias. La tabla contiene para cada tamaño de población y cada grupo de instancias, el porcentaje promedio de error con respecto al mejor valor conocido (Dev%), el número de mejores valores conocidos encontrados (#Mej) y por último se muestra la suma de cada uno de estos indicadores (Total). Como se puede observar en la última columna y en la figura 35, cuando el algoritmo memético logra su mejor desempeño, menor porcentaje de error (0.3287) y mayor número de mejores encontrados (29), esto se puede ver en la figura 36, cuando utiliza un tamaño de población de 50 elementos.

Tabla 18. Porcentajes de error para la evaluación del tamaño de la población.

Algoritmo	Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)	Total
M50	Dev%	0.0002	0.0024	0.1300	0.1961	0.3287
	#Mej.	22	7	0	0	29
M100	Dev%	0.0011	0.0024	0.1568	0.1965	0.3568
	#Mej.	17	10	0	0	27

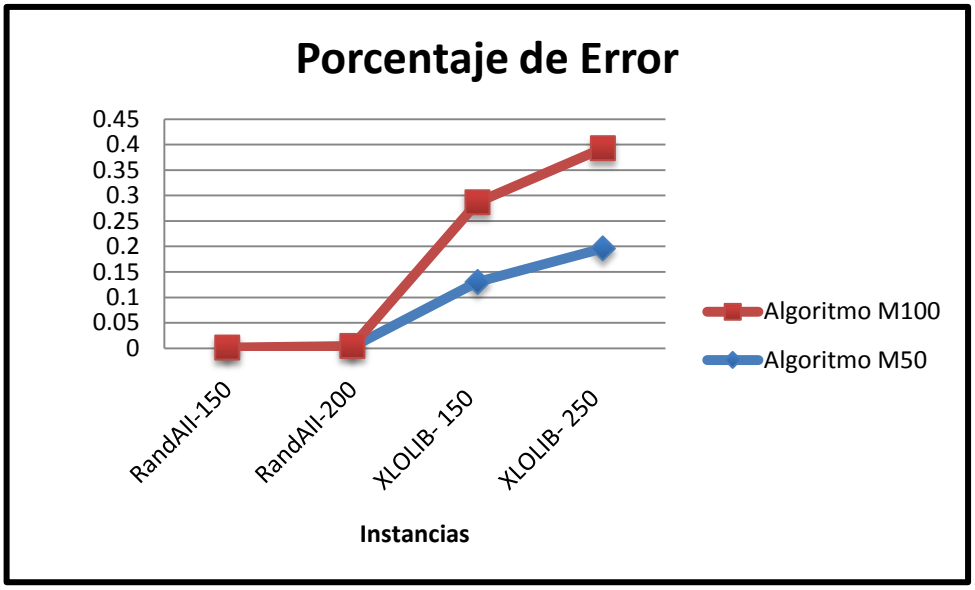

Figura 35. Evaluación del porcentaje de error.

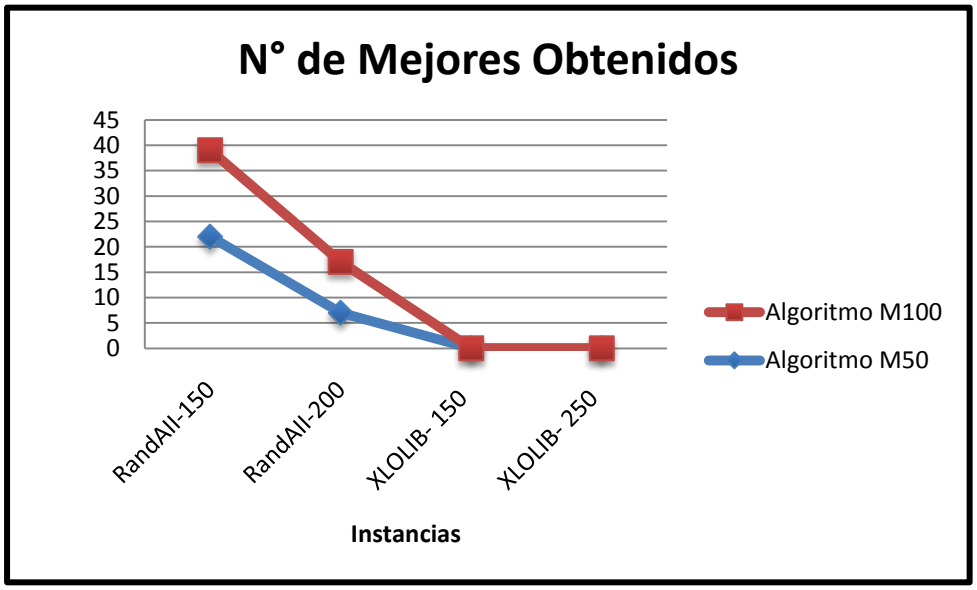

Figura 36. Numero de mejores Obtenidos.

La Tabla 19 muestra los resultados del estudio estadístico no-paramétrico de Wilcoxon. Como se muestra en la tabla cuando R+ es mayor que R- el algoritmo M50 es mejor que el algoritmo M100, ocurrirá lo contrario si R- es mayor que R+. Además, si el mayor de los valores es mayor que el de referencia (R), entonces la diferencia en el desempeño es estadísticamente significativa. En caso contrario ambos algoritmos tienen el mismo desempeño. En todos los casos la confiabilidad considerada es de 0.05. Al aplicar los criterios anteriores, se observa que para todos los grupos de instancias el algoritmo M50 muestra un mejor desempeño que el algoritmo M100. Sin embargo, esta diferencia no es estadísticamente significativa en la mayoría de los casos. Por lo tanto se considera que ambos tienen el mismo desempeño. Sin embargo, con base en estos resultados el tamaño de población inicial utilizado en el algoritmo memético propuesto es de 50 elementos.

Tabla 19. Prueba de Wilcoxon realizada a la evaluación de la población.(100 y 50).

Prom.	RandAII (150)	RandAII (200)	XLOLIB (150)	XLOLIB (250)
N	25	25	39	39
N	7	16	39	39
R^+	26	68	462	367
R^-	2	68	318	413
Ref	26	107	531	531
Mejor Alg.	M50	Empate	M50	M100
Dif. Sig.?	Si	No	No	No

6.8 Desempeño comparativo del algoritmo memético propuesto respecto al algoritmo memético de Schiavinotto [Schiavinotto, 2003] y la búsqueda Tabú de Laguna [Laguna, 1998].

El objetivo de este experimento es evaluar el desempeño del algoritmo memético propuesto cuando se ejecuta durante 10 y 600 segundos de CPU (M10 y M600). La Tabla 20 contiene para cada tiempo de ejecución considerado y cada grupo de instancias, el porcentaje promedio de error con respecto al mejor valor conocido (Dev%), el número de mejores valores conocidos encontrados (#Mej) y la última columna muestra la suma de cada uno de estos indicadores (Total), para una mejor visualización de estos resultados, se incluye la figura 37.

Tabla 20. Porcentajes de error para la evaluación del algoritmo Memético propuesto (M).

Alg. Memético	Prom.	RandAI (100)	RandAI (150)	RandAI (200)	RandAI (500)	RandAII (150)	RandAII (200)	RandB	XLOLIB (150)	XLOLIB (250)	Spec	Total
M10	Dev%	0.008	0.049	0.064	0.198	0.0003	0.001	0.274	0.132	0.205	0.073	1.0073
	#Mej.	18	1	1	0	21	12	0	1	0	4	58
M600	Dev%	0.000	0.003	0.010	0.010	0	0.0002	0.000	0.034	0.045	0.010	0.1022
	#Mej.	25	17	10	10	25	22	20	6	8	6	149

Un aspecto relevante del algoritmo memético propuesto es que logra mejorar uno de los mejores conocidos reportados en [Martí 2009] cuando se ejecuta 10 segundos. Por otro lado, cuando se ejecuta 600 segundos logra mejorar 16 mejores reportados en [Martí 2009], esto se puede ver en la figura 38. En la Tabla 21 se muestran los nuevos mejores valores alcanzados por el algoritmo memético propuesto para las instancias mejoradas, resaltando con negritas el mejor conocido mejorado en la prueba de 10 segundos.

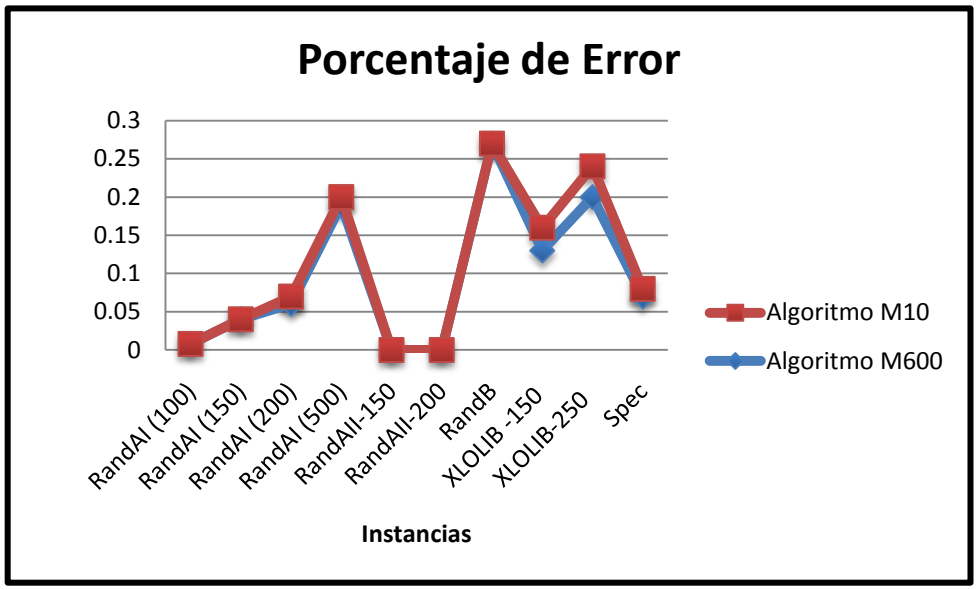

Figura 37. Evaluación del tiempo de ejecución, porcentaje de error.

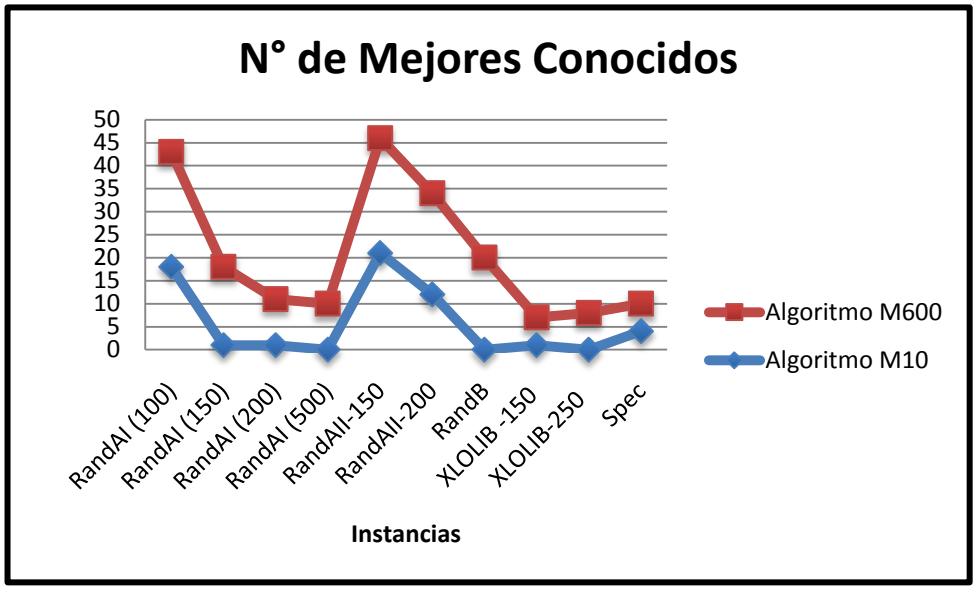

Figura 38. Número de mejores soluciones conocidas alcanzadas por el algoritmo memético propuesto con tiempos límite de 10 y 600 segundos.

Tabla 21. Nuevas mejores soluciones conocidas alcanzadas por el algoritmo memético propuesto.

Instancias	Mejores Conocidos Anteriores	Nuevos Mejores Conocidos
Rand AI		
N-t1d200.13	409234	409270
N-t1d500.15	2411718	2412400
N-t1d500.16	2416067	2416446
N-t1d500.17	2401800	2402438
N-t1d500.18	2421159	2421511
Spec		
N-atp134	1796	1797
N-atp163	2073	2075
XLOLIB		
N - t75d11xx_150.mat	**9642140**	**9643446**
N - tiw56r67_150.mat	2056347	2056446
N - stabu2_250.mat	11500448	11500845
N - stabu3_250.mat	11900315	11901939
N - t59d11xx_250.mat	3841167	3841376
N - t75d11xx_250.mat	25017059	25022475
N - t75n11xx_250.mat	4524942	4525197
N - tiw56n54_250.mat	2098726	2098877
N - tiw56n62_250.mat	4142745	4143351
N - tiw56n72_250.mat	11149706	11151094

La evaluación reportada en [Martí 2009], muestra que el algoritmo con mejor desempeño es el memético [Shiavinotto, 2004], seguido del algoritmo de búsqueda tabú [Laguna, 1998]. Para validar estadísticamente los resultados se aplica la prueba de Wilcoxon comparando el desempeño del algoritmo memético propuesto (M), el algoritmo memético de Schiavinotto (S) y la búsqueda Tabú de Laguna (T), para tiempos límite de 10 y 600 segundos. En la Tabla 22 se muestran los resultados comparativos para la pareja memético propuesto (M), memético de Schiavinotto (S), se puede ver que para todos los grupos de instancias el algoritmo S muestra un mejor desempeño que el algoritmo M. Esta diferencia es estadísticamente significativa en la mayoría de los casos. Es importante señalar que cuando los algoritmos se ejecutan 10 segundos, ambos algoritmos muestran el mismo desempeño sobre las instancias RandAI(100) y RandAII(150), esto se puede observar en la figura 39. Por otra parte, cuando se ejecutan 600 segundos, el algoritmo propuesto iguala el

desempeño del algoritmo memético de Schiavinotto para las instancias RandAI(100), RandAI(500), RandAII(150), RandB y XLOLIB(250), esto se puede observar en la figura 40. Como se observa, las estrategias de diversificación incorporadas en el algoritmo memético propuesto, generan un incremento en su desempeño sobre las instancias de mayor tamaño. Esto resulta más evidente si se analiza la capacidad del algoritmo propuesto para mejorar las mejores soluciones conocidas. Cuando se ejecuta 10 segundos solo mejora una de las mejores soluciones conocidas, pero cuando se ejecuta 600 segundos logra mejorar 16 de las mejores soluciones conocidas.

Tabla 22. Resultados de la prueba de Wilcoxon para el algoritmo memético propuesto y el algoritmo memético de Schavinotto.

Tiempo de ejecución	Prom.	RandAI (100)	RandAI (150)	RandAI (200)	RandAI (500)	RandAII (150)	RandAII (200)	RandB	XLOLIB (150)	XLOLIB (250)	Spec
10 seg.	N	25	25	25	25	25	25	20	39	39	7
	N	7	24	25	25	4	12	20	39	33	3
	R^+	23	282	246	325	10	55	210	749	558	4
	R^-	5	18	79	0	0	23	0	31	3	2
	R	26	219	236	236	NA	65	158	531	391	NA
	Mejor	S=M	S	S	S	NA	S=M	S	S	S	NA
	Dif. Sig.?	No	Si	Si	Si	NA	No	Si	Si	Si	Na
600 seg.	N	25	25	25	25	25	25	20	39	39	7
	N	0	8	16	15	0	3	0	39	1	3
	R^+	0	36	128	88	0	6	0	537	1	4
	R^-	0	0	8	32	0	0	0	243	0	2
	Ref	NA	33	107	95	NA	NA	NA	531	NA	NA
	Mejor	S=M	S	S	S=M	S=M	NA	S=M	S	S=M	NA
	Dif. Sig.?	NA	Si	Si	NA	NA	NA	NA	Si	NA	NA

La misma prueba de Wilcoxon se realiza para el algoritmo propuesto (M) y el algoritmo Tabú de Laguna (T) con el fin de comprobar que la mejora en la diversificación que se incorpora en el algoritmo propuesto produce un incremento en su desempeño sobre las instancias de mayor tamaño. La Tabla 23 muestra los resultados de esta evaluación, donde se observa que en la prueba con tiempo límite de 10 segundos el algoritmo memético propuesto supera significativamente al algoritmo de Búsqueda Tabú en la mayoría de las instancias. Solo para las instancias RandAI(500) y RandB ocurre lo contrario. Sin embargo, si los algoritmos se ejecutan 600 segundos el algoritmo memético propuesta incrementa su

desempeño, logrando superar o igualar el desempeño del algoritmo de Búsqueda Tabú sobre todas las instancias.

Con base en los resultados de las pruebas de Wilcoxon, se puede verificar que el algoritmo memético propuesto supera claramente al algoritmo de Búsqueda Tabú y tiene un desempeño ligeramente menor que el algoritmo memético de Schiavinotto, siendo superado significativamente para 6 de los 10 conjuntos evaluados en la prueba de 10 segundos, mientras que para 600 segundos únicamente es superado en 3 de los 10 grupos de instancias, esto se puede observar en las figuras 39 y 40, así como también se puede observar en las figuras 41 y 42 las evaluaciones del número de mejores soluciones alcanzadas.

Tabla 23. Resultados de la prueba de Wilcoxon para el algoritmo memético propuesto y el algoritmo de búsqueda tabú de Laguna.

Tiempo de ejecución	Prom.	RandAI (100)	RandAI (150)	RandAI (200)	RandAI (500)	RandAII (150)	RandAII (200)	RandB	XLOLIB (150)	XLOLIB (250)	Spec
10 seg.	N	25	25	25	25	25	25	20	39	39	7
	N	21	25	25	25	21	25	20	39	39	4
	R^+	47	52	13	325	0	0	210	0	0	0
	R^-	184	273	312	0	231	325	0	780	780	10
	R	173	236	236	236	173	236	158	531	531	NA
	Mejor	M	M	M	T	M	M	T	M	M	NA
	Dif Sig?	Si	Si	Si	Si	Si	Si	Si	Si	Si	NA
600 seg.	N	25	25	25	25	25	25	20	39	39	7
	N	7	24	25	25	9	20	0	39	39	3
	R^+	0	18	7	0	0	3	0	0	0	0
	R^-	28	282	318	325	45	207	0	780	780	6
	R	26	219	236	236	40	158	NA	531	531	NA
	Mejor	M	M	M	M	M	M	T-M	M	M	NA
	Dif Sig	Si	Si	Si	Si	Si	Si	NA	Si	Si	NA

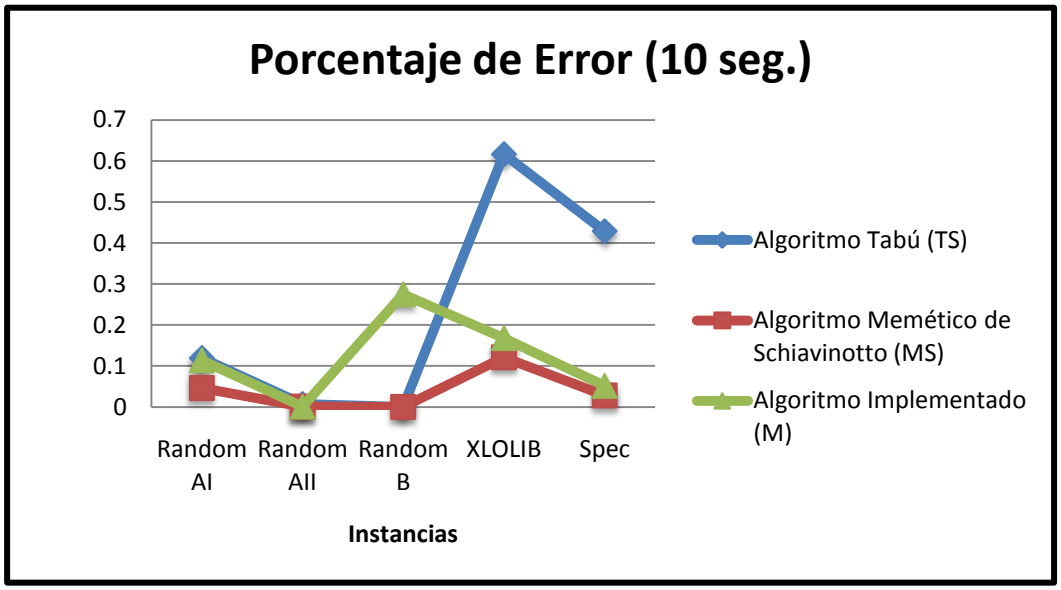

Figura 39. Porcentaje de error del algoritmo memético implementados, el algoritmo memético de Schiavinotto y el algoritmo de búsqueda Tabú de Laguna.

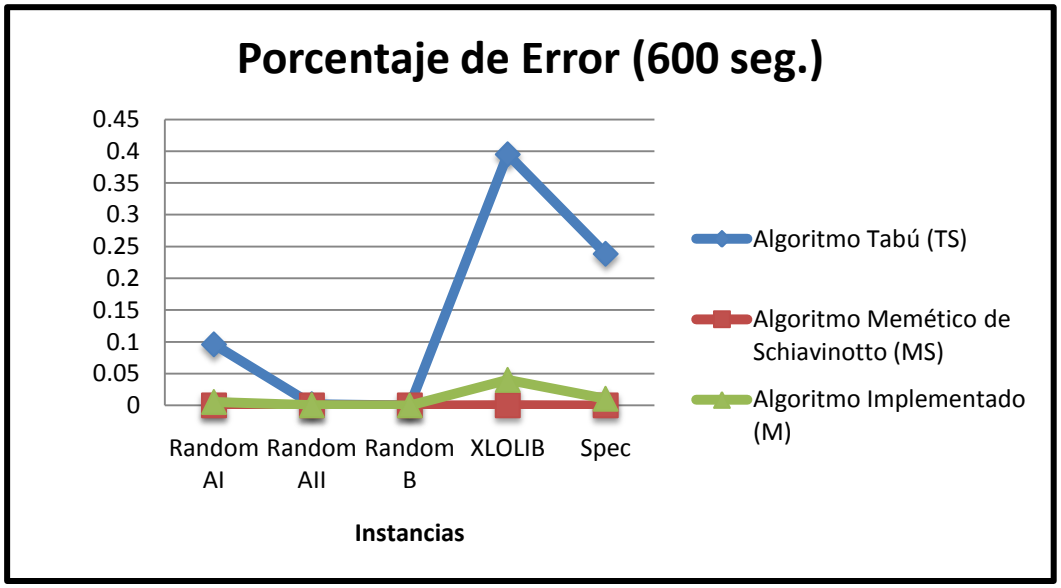

Figura 40. Porcentaje de error del algoritmo memético implementados, el algoritmo memético de Schiavinotto y el algoritmo de búsqueda Tabú de Laguna.

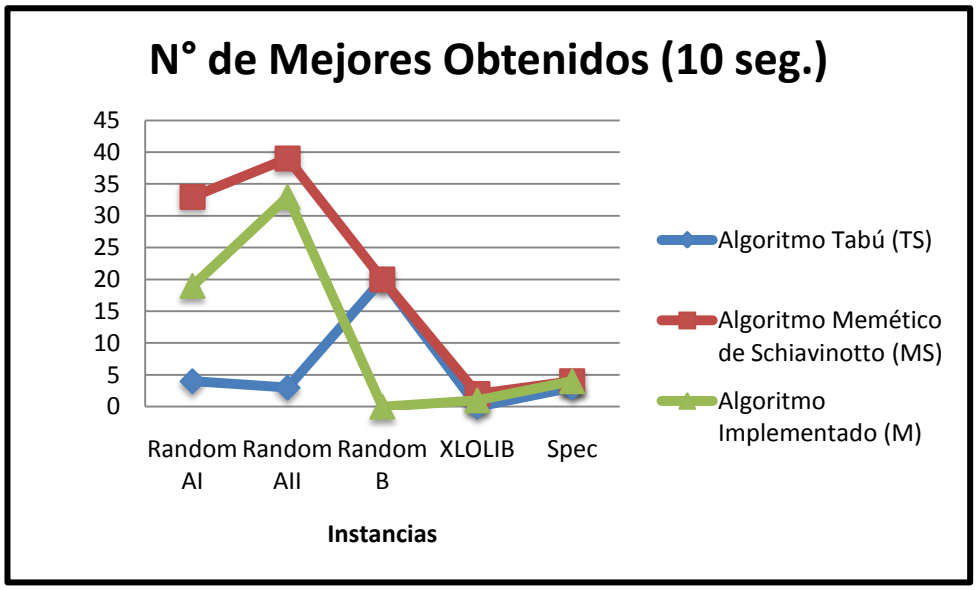

Figura 41. Número de mejores soluciones conocidas alcanzadas, con el algoritmo memético de Schiavinotto, el algoritmo de búsqueda Tabú de Laguna y el algoritmo propuesto, con tiempos límite de 10 segundos.

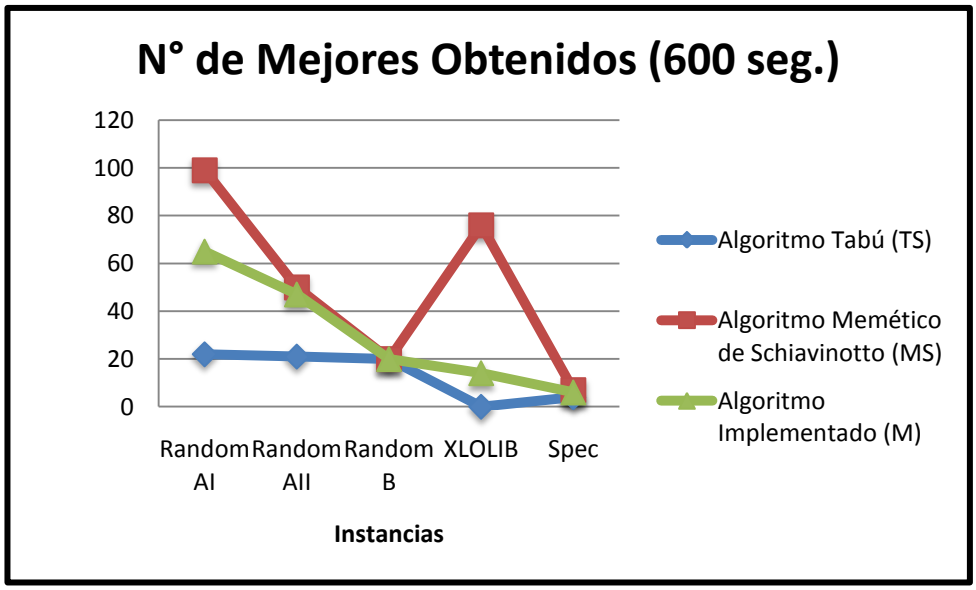

Figura 42. Número de mejores soluciones conocidas alcanzadas, con el algoritmo memético de Schiavinotto, el algoritmo de búsqueda Tabú de Laguna y el algoritmo propuesto, con tiempos límite de 600 segundos.

Capitulo 7

Conclusiones y Trabajos Futuros

En este capítulo se describen las principales aportaciones de este trabajo de investigación, así como las líneas abiertas que se identificaron durante el proceso de investigación. Las aportaciones más relevantes de este trabajo son las siguientes:

- Evaluación de un conjunto de estrategias de construcción de las soluciones iniciales para la metaheurística ILS (sección 6.4).
- Diseño y evaluación de un algoritmo de búsqueda local iterada que mejora el desempeño del algoritmo ILS del estado del arte [Martí, 2009] en un 99.95% en relación al porcentaje de error, e incrementa el número de mejores soluciones conocidas de 0 a 14, para las instancias Random AII. Mientras que para las instancias XLOLIB mejora el desempeño en un 95% en relación al porcentaje de error. (sección 6.4)
- El diseño y evaluación de una estrategia para mejorar la diversificación con base en la aplicación de búsquedas locales con diferentes niveles de intensificación, en los diferentes puntos de la metaheurística.
- Un estudio sobre el impacto del tamaño de la población en el balance de intensificación-diversificación y en la mejora en el desempeño del algoritmo memético.
- Un estudio del comportamiento del algoritmo memético, variando el tiempo límite de ejecución de un tiempo corto (10 segundos), a un tiempo prolongado (600 segundos), con el fin de verificar la estabilidad del algoritmo memético propuesto. Los resultados de este estudio muestran la estabilidad del algoritmo memético propuesto, el cual mejora su desempeño con el incremento del tiempo límite de ejecución en un 89%.

- Un estudio sobre la evaluación preliminar de las diferentes alternativas de estrategias de diversificación consideradas, con el propósito de seleccionar las más prometedoras. Para elegir entre estas se realizaron pruebas de hipótesis de Wilcoxon, determinando si las diferencias en el desempeño son significativas o no. Como resultado de este análisis se determinaron las estrategias que se usan en el algoritmo memético propuesto [Secciones 6.5, 6.6, 6.7 y 6.8].
- Un algoritmo memético que *se posiciona como el segundo mejor del estado del arte* [Martí, 2009]. El algoritmo propuesto logró superar 17 de las mejores soluciones conocidas hasta hoy para el conjunto de instancias de referencia del problema LOP. Estas nuevas mejores soluciones serán una referencia del estado del arte para el diseño de las metaheurísticas de alto desempeño.

Para difundir los resultados de esta investigación se elaboró el artículo "Multiple Local Searches to Balance Intensification and Diversification in a Memetic Algorithm for the Linear Ordering Problem", el cual fue sometido y aceptado en el "Hais 2011" (ver anexo A).

Una actividad que permitió enriquecer este proyecto consistió en realizar una estancia de investigación en la Universidad Rey Juan Carlos de Móstoles comunidad de Madrid, España, donde se tuvo como asesor externo al Dr. Abraham duarte Muñoz; quien es un investigador con amplia experiencia sobre el problema de estudio. Durante esta estancia se presentó un avance sobre el diseño e implementación del algoritmo memético desarrollado, con el fin de recibir retroalimentación y asesoría sobre posibles elementos que pudieran incorporarse para mejorar el desempeño del algoritmo. Particularmente, se recibieron observaciones respecto a la estructura de los experimentos y la implementación de nuevas técnicas. Las áreas de oportunidad identificadas en este trabajo de investigación se enfocan en el uso estrategias de diversificación basadas en búsquedas locales para el diseño e implementación de algoritmos metaheurísticos poblacionales del estado del arte tal como la Búsqueda Dispersa. Este constituye el trabajo futuro más importante que se tiene identificado.

Anexo A

Carta de Aceptación

HAIS11.- PAPER ID: HAIS11-SS01-1821. Decision on your paper

jueves, 3 de febrero de 2011, 2:05

De: "escorchado@ubu.es" <escorchado@ubu.es>

Para: gpe_cas@yahoo.com.mx

Dear Guadalupe Castilla Valdez,

You submitted a paper to be considered for publication in the HAIS11 Conference.

Details of the paper are as follows:
PAPER ID: HAIS11-SS01-1821
PAPER TITLE: Multiple Local Searches to Balance Exploitation and Exploration in Memetic Algorithm for the Linear Ordering Problem
We are pleased to inform you that your paper has been accepted, subject to certain specified amendments being satisfactorily completed.

Please log-in to VIC using the URL
http://www.gicap.ubu.es:8080/vic/conference/HAIS11

Username:gcastilla
Password:gcastilla

Once logged in to VIC you will be able to see the two independent reviews we received, and details of the decision by the Session Chair. You will also be able to obtain guidance on the required revisions you need to make.
Please modify the paper following the guidance on the required revisions. Include as a final page of the paper a commentary describing the changes you have made to the paper, and the way in which these changes satisfy the requirements of the reviewers. Then please re-submit the revised paper to VIC system by 1st February, 2011. You will then be informed shortly about the final acceptance of your paper (by the 1st February, 2011). At this point you will need to provide an editable soft copy of the paper for the type-setters to use: sources files, pdf file and electronically signed or scanned copyright.

NOTICE: the final camera ready version must comply with LNCS format. Any incorrect format may risk the paper being excluded in the HAIS11 LNCS Proceedings. At least one author for each accepted paper must register by .
HAIS 2011

Multiple Local Searches to Balance Intensification and Diversification in a Memetic Algorithm for the Linear Ordering Problem

Héctor Joaquín Fraire Huacuja[1], Guadalupe Castilla Valdez[1], Claudia G. Gómez Santillan[1], Juan Javier González Barbosa[1], Rodolfo A. Pazos R.[1], Shulamith S. Bastiani Medina[1], David Terán Villanueva[1]

[1]Instituto Tecnológico de Ciudad Madero, México.
1o. de Mayo y Sor Juana I. de la Cruz S/N C.P. 89440
Cd. Madero, Tamaulipas, México.

hfraire@prodigy.net.mx, gpe_cas@yahoo.com.mx, cggs71@hotmail.com, jjgonzalezbarbosa@yahoo.com.mx, r_pazos_r@yahoo.com.mx, b_shulamith@hotmail.com, david_teran01@yahoo.com.mx

Abstract. The Linear Ordering problem (LOP) is an NP-hard problem, which has been solved using different metaheuristic approaches. The best solution for this problem is a memetic algorithm, which uses the traditional approach of hybridizing a genetic algorithm with a single local search; on the contrary, in this paper we present a memetic solution hybridized with multiple local searches through all the memetic process. Experimental results show that using the best combination of local searches, instead of a single local search, the performance for XLOLIB instances is improved by 11.46% in terms of quality of the solution. For the UB-I instances, the proposed algorithm obtained a 0.12% average deviation from the best known solutions, achieving 17 new best known solutions. A Wilcoxon test was performed, ranking the proposed memetic algorithm as the second best solution of the state of the art for LOP. The results show that the multiple local searches approach can be more effective to get a better control in balancing intensification/diversification than the single local search approach.

Keywords: Metaheuristics, Memetic algorithm, Linear Ordering Problem, Local Search, Intensification/diversification balance.

1. Introduction

Given a C_{ij} matrix of weights of size $n \times n$, the linear ordering problem (LOP) consists in finding a permutation p of columns (and rows) such that the sum of the weights in the upper triangle is maximized. Its mathematical expression is:

$$\max\left(C_{LOP}(p) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C_{p_{(i)}\ p_{(j)}} \right)$$

Here p_i is the index of column (and row) i in the permutation. In LOP, the permutation p provides the ordering of rows and columns, as described by Laguna [1].

The Linear Ordering Problem has been described as an NP-hard problem by Karp and Thatcher [2] and Garey and Johnson [3].

Important applications of LOP exist in scheduling, social sciences, electronics, archeology, and particularly in economics, where it is applied to solve the problem of triangulating the input-output tables in the economic model by Leontief [4].

2. Related work

The memetic algorithm by Schiavinotto and Stutzle [5] currently offers the best solution of the state of the art for LOP. In this work a genetic algorithm was hybridized with a single local search on an insertion

neighborhood. The single local search included in the memetic algorithm uses a criterion between first and best. The final configuration of memetic includes a population size of 25 individuals, an offspring formed by 11 children generated by OB crossover, a diversification by reinitializing of population (after 30 iterations have occurred without changes in fitness average). The final memetic algorithm didn´t include a mutation operator.

One of the best solutions for LOP is the Tabu search by Laguna et al. [1]. This solution includes an intensification phase using short-term memory based on a tabu criterion, a diversification process through a long-term memory that uses a frequency register, and an additional intensification process that applies a path relinking strategy based on elite solutions. The Tabu search algorithm uses two different local searches (first and best) in its different processes. These local searches explore an insertion neighborhood through consecutive swap movements.

A benchmark library and a comprehensive study of heuristic methods were developed by Martí [6]. In this work, under the same experimental conditions and on the same standard sets of instances (OPT-I and UB_I), 10 of the top performing heuristic algorithms are assesed. The highest performing algorithms were the memetic algorithm and Tabu search.

Memetic algorithms are commonly implemented as evolutionary algorithms endowed with a single local search component [7]. The use of multiple local searches in the memetic algorithm is associated to learning approaches [8], but this strategy has a high computational cost. On the contrary, we propose a less expensive new approach by incorporating in a memetic algorithm multiple local searches with different levels of intensification, according to the requirements of the processes included in the algorithm.

3. Local Searches

The insertion neighborhood used by the local searches implemented, is defined by the insert operation: $\pi \times \{1, \ldots, n\}^2 \rightarrow \pi$.

$$insert(\pi, i, j) \triangleq \begin{cases} (\ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_j, \pi_i, \pi_{j+1}, \ldots) & i < j \\ (\ldots, \pi_{j-1}, \pi_i, \pi_j, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots) & i > j \end{cases}$$

This neighborhood was implemented applying the exploration strategy inspired on the Dynasearch method proposed by Congram [9]. The insertion is made by a series of swap movements between adjacent elements, where $i = j \pm 1$. The change in the objective function value is given by:

$$\Delta_s(\pi, i, j) \triangleq \begin{cases} c_{\pi_i \pi_j} - c_{\pi_j \pi_i} & i = j+1; \\ c_{\pi_j \pi_i} - c_{\pi_i \pi_j} & i = j-1. \end{cases}$$

Since the change evaluation for each movement is constant, the cost to evaluate the neighborhood is $O(n^2)$. For optimizing the execution time, the costs of all the swap moves are pre-calculated like in the Tabu search by Laguna [1]. The expression for this calculation is:

$$d_{ij} = c_{ij} - c_{ji} \quad \forall i,j = 0..\ n\text{-}1.$$

In this work two local search algorithms with different levels of intensification/diversification were implemented. The algorithm in Figure 3 corresponds to LS_f local search [5], which has an external cycle to examine all the sectors in the permutation order. For each sector, an internal cycle evaluates all the neighbors searching for the best (if none is better it returns the less worse neighbor solution).

```
LS_f Algorithm(π)
   for (i = 0.. n-1) do
      r⁻ ←arg max r. r = i f (insert(π,i,r⁻ ))
      π'= insert ((π,i,r⁻))
      if f (π') > f (π) then
         return (π')
      end_if
   end_for
  return (π)
end_LS_f,Algorithm (π)
```

Figure 3. LS_f algorithm

On the other hand the *Best* local search [1] is much more intensive: it starts from a feasible solution which is improved over the time. All sectors are scanned in the permutation order; all the consecutive positions forward and backward from the current position are examined, choosing the position which produces the highest increment (or the lowest decrease) in the objective function value for carrying out the movement. The process is repeated as long as the current solution is improved, Figure 4 shows the algorithm for the *Best* local search. It is worth mentioning that the *Best* local search is more intensive and its cost is greater than the cost of the LS_f algorithm. As we can see, both searches have a variable cost.

```
Best Algoritm (π)
        do
                for (i = 1.. nsectors)  hacer
                        r⁻ ←arg max r. r ≠ i f (insert(π,i,r⁻ ))
                        π'= insert ((π,i,r⁻))
                end_for
        while ((f(π') > f(π))
    return (π')
end Best Algoritm (π)
```

Figure 4. *Best* algorithm

4. Proposed memetic algorithm (MLSM)

The proposed memetic algorithm that we denote by the acronym MLSM (multiple local search memetic) is hybridized with different local searches. Figure 1 shows the pseudo-code for MLSM.

```
MLSM Memetic Algorithm
    Population←RandomGeneratepopulation( );
      ImprovePopulation ←BestLocal Search(Population);
    while (not stop condition)
        if stagnation then
                Population←Selectbestindividual(Population);
            Population← RandomGeneratepopulation( );//diversification
                ImprovePopulation← LSFLocalSearch(Population);
            end_if
            for i←1..#crossovers do
                select π a, π b from Population( );
                offsprings←OB_crossover( );
        end_for
            ImprovePopulation← BestLocalSearch(Population);
            Population←SortPopulation(Population);
            best_individual←Select_best(SortedPopulation)
    end_while ;
end_Memetic Algorithm
```

Figure 1. Proposed memetic algorithm MLSM

In the proposed memetic algorithm, the initial population is obtained by generating randomly 36 individuals, to which the *Best* local search is applied.

For subsequent generations, two parents are randomly selected for applying crossover. From the first parent a set of positions are randomly selected (0.4 * instance size), then the values in the non-selected positions are copied directly to the corresponding positions of the offspring. In the next step, the values in the selected positions are ranked according to the order in the second parent and copied to the vacant positions of child. Figure 2 shows this crossover process. In the selection process, the first father is randomly chosen according to a uniform distribution and the second parent is chosen giving preference to the best elements of the population. A general condition for the selection of parents is that they haven't recently generated offspring. The *Best* local search is applied to the new population.

To build the new population, the 25 best individuals are chosen from the current population and from the generated offspring, duplicates are eliminated.

An extra diversification by randomly regenerating the individuals in the current population is applied, remaining only the best individual. This process is triggered to avoid premature convergence when the average of the population objective function does not change during five consecutively generations. In this case an LS_f Local search is applied to all the individuals in the new population.

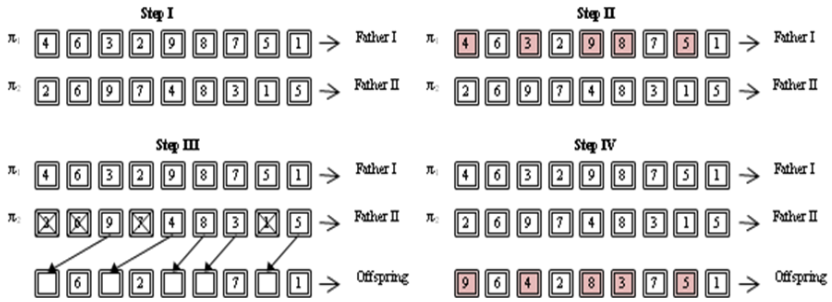

Figure 2. An OB crossover example

5. Experimental results

Experimental work was conducted on a Dell Power Edge 2600 with a XEON processor at 3.06 GHZ and 4GB in RAM. The algorithm was coded in C++, and compiled in Visual Studio 2005. Each algorithm was executed

using two time limits (10 and 600 seconds) for each instance and a seed of 1471. The instances were grouped into four sets: RandAII, Spec, RandB, XLOLIB, and RandAI.

A first experiment to assess different combinations of the local search strategies (*Best* (B) and LS_f (LSF)) on the memetic algorithm was performed. They are applied to the initial population, to the population after the crossover and to the population after the diversification process. The experiment was carried out using the XLOLIB instances. Table 1 shows the results for the best combinations. The first column shows the combination of local searches. The combination (B, B, B) corresponds to a single local search memetic algorithm (SLSM, traditional approach), that is used as base case. The third column includes the average deviation from the best known solution (%), and the fourth column contains the difference in performance for the corresponding combination with respect to the performance of the base combination. The best combination (B, LSF, B), highlighted in the table, achieves a performance improvement of 11.4% with respect to the base case. In the table, the acronym NA indicates that no local search was applied.

A second experiment was carried out to assess the performance of the proposed memetic algorithm, which incorporates the best configuration of local searches. The MLSM algorithm was compared average wise with the two best metaheuristics: tabu search by Laguna (TS) [1] and memetic algorithm by Schiavinotto (SM) [5]. The results for these two metaheuristics were obtained from the benchmark library developed by Marti et al. [6] and are shown on Table 2.

Table 1. Average deviation from best known solutions and percentage improvement with respect to the base SLSM algorithm

Combination of local searches	Memetic algorithm	Average deviation from best known (%)	Percentage improvement versus base SLSM
B, B, B	Base SLSM	0.18974923	0
LSF, LSF, B	MLSM	0.19692336	-3.78084802
B, LSF, B	MLSM	0.16805435	11.43344824
LSF, B, B	MLSM	0.19661945	-3.62068399
NA, NA, B	SLSM	0.17349502	8.566153338
LSF, LSF, LSF	SLSM	0.42988623	-126.554927
B, LSF, LSF	MLSM	0.40227557	-112.003796

Table 2. Experimental results for UB-I instances (10 sec.)

Instances	Performance	TS by Laguna [3]	SM by Schiavinotto [5]	MLSM
RandA1	% Error (Avg)	0.12	0.05	0.11
	# Best	5	33	19
RandA2	% Error (Avg)	0.01	0	0.001
	# Best	3	39	33
RandB	% Error (Avg)	0	0	0.27
	# Best	20	20	0
XLOLIB	% Error (Avg)	0.61	0.12	0.169
	# Best	0	2	1
Spec	% Error (Avg)	0.45	0.049	0.073
	# Best	3	3	4
% Average Error		0.23	0.04	0.12
#Total Best		31	97	57

As can be seen, the proposed MLSM algorithm achieves a comparable performance to the memetic algorithm by Schiavinotto (SM). Afterwards non-parametric Wilcoxon tests were performed to compare MLSM with respect to the tabu and memetic algorithms, applying time limits of 10 and 600 seconds. MLSM significantly outperforms the Tabu solution for the two different time limits (the results are not shown by space restrictions). The results of the experiment with SM algorithm are shown in the Table 3. In this table, when R^+ is larger than R^-, the SM algorithm is better than MLSM, the opposite is true if R^- is greater than R^+. If the greatest of R^+ and R^- is greater than the reference value (R), then the performance difference is statistically significant, otherwise both algorithms have the same performance. In all cases, considered reliability is 0.05. As we can see, the memetic algorithm by Schiavinotto[5] outperforms significantly the

performance of MLSM in 6 of 10 cases for 10 seconds. In the test with 600 seconds MLSM was outperformed by SM only in 3 of 10 cases. The performance increment of the MLSM algorithm could be explained because the multiple searches approach permitted to incorporate a better balance of intensification/diversification than the single search approach.

Additionally, with the proposed memetic algorithm MLSM 17 best known solutions were improved. These new best known solutions were obtained when the MLSM algorithm was carried out using a time limit of 600 seconds. Table 4 shows these new best known solutions.

Table 3. The Wilcoxon test (memetic algorithm by Schiavinotto [5] and MLSM)

Execution time limit of 10 seconds										
Prom.	RandA I(100)	RandA I(150)	RandA I (200)	RandA I(500)	RandA II(150)	RandA II(200)	RandB	XLOLIB (150)	XLOLIB (250)	Spec
N	25	25	25	25	25	25	20	39	39	7
N	7	24	25	25	4	12	20	39	33	3
R^+	23	282	246	325	10	55	210	749	558	4
R^-	5	18	79	0	0	23	0	31	3	2
R	26	219	236	236	NA	65	158	531	391	NA
Winner	none	SM	SM	SM	NA	none	SM	SM	SM	NA
Dif.Sig.?	No	Yes	Yes	Yes	NA	No	Yes	Yes	Yes	NA
Execution time limit of 600 seconds										
N	25	25	25	25	25	25	20	39	39	7
N	0	8	16	15	0	3	0	39	1	3
R^+	0	36	128	88	0	6	0	537	1	4
R^-	0	0	8	32	0	0	0	243	0	2
Ref	NA	33	107	95	NA	NA	NA	531	NA	NA
Winner	SM=MLSM	SM	SM	SM=MLSM	SM=MLSM	NA	S=MLSM	S	S=MLSM	NA
Dif.Sig.?	No	Yes	Yes	No	No	NA	No	Yes	No	NA

Table 4. New best solutions

Set	Instances	Old *best solutions*	New *best solutions*
Rand AI	N-t1d200.13	409234	409270
	N-t1d500.15	2411718	2412400
	N-t1d500.16	2416067	2416446
	N-t1d500.17	2401800	2402438
	N-t1d500.18	2421159	2421511
Spec	N-atp134	1796	1797
	N-atp163	2073	2075
XLOLIB	N- t75d11xx_150.mat	9642140	9643446
	N - tiw56r67_150.mat	2056347	2056446
	N - stabu2_250.mat	11500448	11500845
	N - stabu3_250.mat	11900315	11901939
	N - t59d11xx_250.mat	3841167	3841376
	N - t75d11xx_250.mat	25017059	25022475
	N - t75n11xx_250.mat	4524942	4525197
	N - tiw56n54_250.mat	2098726	2098877
	N - tiw56n62_250.mat	4142745	4143351
	N - tiw56n72_250.mat	11149706	11151094

6. Conclusions

In this work the linear ordering problem is approached. We propose a memetic algorithm that incorporates multiple local searches instead of a single local search. A combination of different local searches is evaluated for the process of improving the individuals.

The MLSM algorithm was tested on the UB-I instances, obtaining 0.12% in the average deviation from the best known solution, comparable with the memetic algorithm by Schiavinotto, and it outperforms the Tabu by Laguna by 48% in solution quality. The Wilcoxon non-parametric test shows that MLSM significantly outperforms the Tabu solution; but the memetic algorithm by Schiavinotto outperforms significantly the performance of MLSM in 6 of 10 cases for 10 seconds. In the test with 600 seconds MLSM was outperformed by SLSM only in 3 of 10 cases. The recovery of the MLSM performance could be explained

because the multiple local searches approach permits to get a better control of the balance of intensification/diversification than the single local search approach. With the proposed memetic algorithm MLSM, 17 best known solutions were improved. We are currently applying the multiple local searches approach in other metaheuristics.

Acknowledgements. This work was supported by PROMEP, CONACYT and COTACYT.

Also we want to thank Manuel Laguna and Abraham Duarte, for their support.

References

1. Laguna, M., Martí, R., Campos, V.: Intensification and diversification with elite tabu search solutions for the linear ordering problem. Computers and Operations Research 26, 1217-1230 (1998)
2. Karp, R., Miller, R., and Thatcher J.: Reducibility among Combinatorial Problems in: Complexity of computer computation. Eds. Plenum Press, 85-103 (1972)
3. Garey, M. R., Johnson, D.S.:Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co. (1975)
4. Leontief, Wassily W.W.: Input-Output Economics. Oxford University Press (1986)
5. Schiavinotto, T., Stützle, T.: Search Space Analysis of the Linear Ordering Problem. Darmstadt University of Technology, Intellectics Group, Darmstadt, Germany (2004)
6. Martí, R., Reinelt G., Duarte, A.: A Benchmark Library and a Comparison of Heuristic Methods for the Linear Ordering Problem. Computational optimization and applications (2010)
7. Moscato, P.,Cota, C.: A Modern Introduction to Memetic Algorithms. International Series in Operations Research & Management Science. M. Gendreau and J-Y Potvin (Eds.). Springer, Heidelberg 449-468 (2010)
8. Ong, Y.-S., Keane, A.J.: Meta-Lamarckian Learning in Memetic Algorithms. IEEE Trans. Evol. Comput. 8(2), 99–110 (2004)
9. Congram Richard K.: Polynomially Searchable Exponential Neighborhoods for Sequencing Problems in Combinatorial Optimization. Faculty of Mathematical Studies UK, University of Southampton (2000)

Referencias Bibliográficas

[Polya, 1957] G. Polya. How to Sove it? Priceton University Press, 1957.

[Karp, 1972] Karp, R., Miller, R. and Thatcher J.: Reducibility among combinatorial problems in: Complexity Of Computer Computations. Eds. Plenum Press, 85--103, (1972).

[Garey, 1975] Garey, M. R. and Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co. (1975).

[Dawkins, 1976] Dawkins, R.: The Selfish Gene. Clarendon, Oxford (1976).

[Zanakis, 1981] H. Zanakis, J.R. Stelios, and A. Evans. Heuristic optimization: Why, when and how to use it. Interfaces, 5(11):84–90, 1981.

[Leontief, 1986] Wassily, L.: Input-Output Economics. Oxford University Press (1986).

[Reinelt, 1985] Reinelt, G.: *The linear ordering problem: algorithms and applications*, research and exposition in mathematics 8, Heldermann, 1985.

[Glover, 1986] F. Glover. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13:533 549, 1986.

[Moscato,1989] P. Moscato. On evolution, search, optimization genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program, Report 826, California Institute of Technology, Pasadena, California, USA, 1989.

[Gart, 1995] Garth Isaak: Tournaments as Feedback Arc Sets, Department of Mathematics, (1995)

[Boese, 1996] K.D. Boese. Models for Iterative Global Optimization. PhD thesis, University of California, Computer Science Department, Los Angeles, CA, USA, 1996.

[Chanas, 1996] Chanas, S. and P. Kobylanski (1996) "A New Heuristic Algorithm Solving the Linear Ordering Problem," Computational Optimization and Applications, Vol. 6, pp. 191-205.

[Díaz, 1996] Adenso Diaz, Fred Glover, Hassan M. GH, J.L. González, Manuel Laguna, Pablo Mosscato, Fan T. Tseng; Optimización Heurística, Redes Neuronales; Editorial Paraninfo.S.A; 1996; ISBN: 84-283-2269-4.

[Kelly, 1996] J.P. Kelly and I.H. Osman. Meta-Heuristic: Theory and Applications. Kluwer Academic Publisher, 1996.

[Laguna, 1998] Laguna, M., Martí, R. and Campos, V.: Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem. Computers and Operations Research, vol. 26, pp. 1217—1230 (1998).

[Papadimitriou, 1998] C.H. Papadimitriou and K. Steiglitz. Combinatorial Optimization. Dover Publications, INC, 1998.

[Campos, 1999] Campos Vicente, Fred Glover, Manuel Laguna, Rafael Martí; An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem; Journal of Global Optimization, Volume 21 Issue 4, 1999.

[Festa, 1999] Festa, P. Pardalos, P.M., and Resende, M. G.C.: Feedback Set Problems. AT&T Labs. Research Technical Report: 99.2.2., (1999).

[Merz, 1999] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, pages 245–260.McGraw-Hill, London, 1999.

[Congram, 2000] Congram Richard K.: Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation. Faculty of Mathematical Studies, (2000).

[Merz, 2000] P. Merz. Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000.

García González Carlos, Peréz Brito Dionisio; A Variable Neighborhood Search for Solving the Linear Ordering Problem; MIC'2001 - 4th Metaheuristics International Conference; pp. 181-185.

[Blum, 2003] Blum Christian, Roli Andrea: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, ACM Computing Surveys, Vol. 35, No. 3, September 2003, pp. 268–308.

[Huang, 2003] Huang, G. and Lim, A.: Designing a hybrid genetic algorithm for the linear ordering problem, in: Cantu-Paz, E. et al. (eds.): Proc. of Genetic and Evolutionary Computation – GECCO 2003, LNCS 2723, Springer, 2003, 1053{1064}.

[Moscato, 2003] P. Moscato, C. Cotta, Una Introducción a los Algoritmos Meméticos, Inteligencia Artificial 19 (2003) 131-148

[Resende, 2003] Resende G. C. Mauricio, González Velarde José Luis; GRASP: Greeedy Randomized Adaptive Search Procedures; Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.19 (2003), pp. 61-76 ISSN: 1137-3601.

[Shiavinotto, 2003] Tommaso Schiavinotto and Thomas Stutzle; Search Space Analysis of the Linear Ordering Problem; Applications of Evolutionary Computing Lecture Notes in Computer Science, 2003, Volume 2611/2003, 197-204.

[Belloni, 2004] Belloni, A., and Lucena, A.: Lagrangian Heuristic for the Linear Ordering Problem. In: Metaheuristics: Computer Decision-Making, Kluwer Academic Publishers, (2004).

[Chiarini, 2004] Chiarini, B.: New algorithm for the triangulation of input-output tables and the linear ordering problem. Thesis in the University of Florida (2004).

[Bertacco, 2005] Bertacco, L., L. Brunetta and M. Fischetti (2005) "The Linear Ordering Problem with Cumulative Costs", European Journal of Operational Research. Forthcoming, 2005.

[García, 2005] García Carlos G., Pérez-Brito Dionisio, Campos Vicente, Martí Rafael; Computers & Operations Research 33 (2006) 3549–3565; Available online 27 April 2005.

[Martí, 2006] Martí R.: LOP instances and best known solutions, www.uv.es/~rmarti/paper/lop.html, 2006.

[Cotta, 2007] C. Cotta, Una visión general de los algoritmos meméticos, Rect@ 3:139-166, 2007.

[Chiarbit, 2007] Charbit Pierre, Stéphan Thomassé and Anders Yeo: The minimum feedback arc set problem is NP-hard for tournaments, (2007).

[Cruz, 2008] Marco Antonio Cruz Chávez, Ocotlán Díaz-Parra: Un Mecanismo de Vecindad con Búsqueda Local y Algoritmo Genético para el Problema de Transporte con Ventanas de Tiempo, Universidad Autónoma del estado de Morelos, Programación Matemática y Software (2008)

[Duarte, 2008] Abraham Duarte Muñoz, Juan José Pantrigo Fernández, Micael Gallego Carrillo: Metaheurísticas; Universidad Rey Juan Carlos (2008).

[Garcia, 2008] García S., Molina D., Lozano F., Herrera F., A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real ParameterOptimization, Journal of Heurisatics, Springer Netherlands, Netherlands 2008.

[Chaovalitwongse, 2009] Chaovalitwongse, W. A., Pardalos, P.M., Resende, M.G.and Grundel, D.A.: Revised GRASP with path-relinking for the linear ordering problem, by appear in J. of Combinatorial Optimization (2009).

[Martí, 2009]Rafael Martí, Gerhard Reinelt and Abraham Duarte, Technical Report**,** A Benchmark Library and a Comparison of Heuristic Methods for the Linear Ordering Problem (2009).

[Martí, 2009] Rafael Martí, Gerhard Reinelt: LOLIB, Dept.de Estadística e Investigación Operativa, Universidad de Valencia, España (2009).

[Gendreau, 2010] Gendreau Michel, Jean-Yves Potvin; Handbook of Metaheuristics, second edition; Springer New York Dordrecht Heidelberg London; 2010.