UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

EVALUACIÓN DE ESQUEMAS DE MEMORIA EN UN ALGORITMO MEMÉTICO PARA EL PROBLEMA DE ASIGNACIÓN CUADRÁTICA

HUGO MAURICIO MENESES PONCE

Profesor Guía: Dr. Mario Inostroza

Trabajo de titulación presentado
en conformidad a los requisitos
para obtener el Grado de Magíster
en Ingeniería en Informática

Santiago de Chile
2011

© **Hugo Mauricio Meneses Ponce**

Se autoriza la reproducción parcial o total de esta obra, con fines académicos, por cualquier forma, medio o procedimiento, siempre y cuando se incluya la cita bibliográfica del documento.

AGRADECIMIENTOS

Deseo agradecer a todos quienes tuvieron algo que ver con que esta tesis llegara a buen término.

En primer lugar agradezco a mi familia, que siempre ha confiado ciegamente en mi éxito, que ha estado presente en buenos y malos momentos a lo largo de los años y siempre ha velado por mi educación, procurando que ésta siempre fuera la mejor. Además no puedo dejar de dar gracias por la crianza recibida de mis padres, siempre con una confianza absoluta en mi persona y con gran respeto a mi libertad y opinión.

Agradezco a mi pareja y amigos, que constantemente fueron un fuerte apoyo en mi camino, sobre todo cuando era necesario recuperar la fuerza y la confianza en mí mismo. A mis compañeros que caminaron junto a mí en este viaje de preparación para el futuro.

Menciono aquí también a mi profesor guía Mario Inostroza, que siempre me otorgó ayuda y consejo cuando fue necesario, sobre todo cuando parecía difícil el avanzar.

Finalmente, quisiera dar mérito a la Universidad de Santiago de Chile, en particular al Departamento de Ingeniería Informática, por darme las herramientas posibles para el aprendizaje durante todos estos años, haberme otorgado la beca que me permitió cursar los estudios de Magíster en Ingeniería Informática y haberme dado la posibilidad de conocer a personas que han llegado a ser mis mejores amigos.

Dedico este trabajo de tesis a mi familia y amigos, en especial a mi sobrina Maite Sofía que está comenzando su viaje por este mundo.

RESUMEN

En las metaheurísticas existe el problema de la convergencia prematura, el cual ha sido evitado con diferentes técnicas, entre las que se encuentra el uso de memoria. Dentro de las metaheurísticas están los algoritmos meméticos (también conocidos como genéticos híbridos), los cuales han sido ampliamente usados para tratar problemas $NP - Hard$, como el problema de asignación cuadrática (QAP). Aunque el uso de memoria ha sido una importante técnica para evitar la convergencia prematura en metaheurísticas, es de interés comparar el uso de varios esquemas de memoria en alguna metaheurística específica.

En este trabajo se evalúa la mejora en calidad de soluciones de un algoritmo memético (MA) para QAP mediante el uso de esquemas de memoria. Primero, se proponen clasificaciones para estructuras de memoria, con la intención de unificar el análisis y definición de esquemas de memoria usados en algoritmos. Luego, se combinan 6 esquemas de memoria produciendo 23 versiones diferentes de un algoritmo de referencia, de acuerdo a los esquemas de memoria utilizados. Estas versiones son evaluadas usando instancias del problema encontradas en la literatura, y los resultados son dados en términos de calidad de soluciones, tiempo de convergencia y tasa de éxito. Dichos resultados son comparados a través de índices Copeland y diferenciados mediante los *p-values* dados por el Test de Wilcoxon.

La discusión de estos resultados muestra varios algoritmos que vencen al algoritmo memético original en términos de calidad de solución, rapidez y tasa de éxito. Este trabajo además, muestra la superioridad de un nuevo esquema de memoria propuesto, el cual es usado para evitar la repetición de cruzamientos en el MA. El mejor algoritmo encontrado reduce el error promedio del algoritmo original desde 0,150 % a 0,118 %.

PALABRAS CLAVE: Esquema de Memoria, Memoria, QAP, Algoritmo Memético, Convergencia

ABSTRACT

In metaheuristics exists the problem of premature convergence which have been avoided with different techniques, among them, the use of memory. Among the metaheuristics, there are the memetic algorithms (also known as hybrid genetic algorithms), which have been widely used to deal with $NP - Hard$ problems such as the quadratic assignment problem (QAP). Although the use of memory has been an important technique to avoid premature convergence in metaheuristics, it would be interesting to compare the use of several memory schemas in a specific metaheuristic.

This work is evaluates the improve in quality of solutions found by a memetic algorithm (MA) for QAP using memory schemas. First, several classifications for memory structures are proposed with the intention of unify the analysis and definition of memory schemes used in algorithms. Then, six memory schemas are combined producing 23 different versions of a reference MA, according to the memory schemas used. These versions are tested using instances of the problem found in the literature, and results are given in terms of quality of solutions, convergence time and sucess rate. Those results are compared using the Copeland index and differentiated using the *p-values* given by the Wilcoxon Test.

Results show several algorithms that defeat the original memetic algorithm in terms of quality, speed and sucess rate. This work also shows the superiority of a new proposed memory schema which is used to avoid to repeat crossovers into the MA. The best algorithm found in this research reduces the original average error from 0,150 % to 0,118 % in the selected instances.

KEYWORDS: Memory Schema, Memory Scheme, Memory, QAP, Memetic Algorithm, Convergence

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ALGORITMOS

CAPÍTULO 1. INTRODUCCIÓN

En el presente capítulo se analizan los aspectos relevantes al problema y su solución. Primero se revisa el contexto a modo de motivación, luego se explica en detalle en que consiste el problema, para el cual se otorga una solución y sus límites. Luego se plantean los objetivos de la solución mencionada y se expone la metodología de trabajo que se utiliza para esta investigación.

1.1 ANTECEDENTES Y MOTIVACIÓN

En la actualidad, se requiere que los sistemas informáticos tengan como atributos principales su eficiencia y eficacia, lo cual implica que den respuestas adecuadas en tiempos factibles.

Sin embargo, hay ocasiones en que se necesita que los sistemas anteriormente mencionados resuelvan problemas de una complejidad computacional muy alta, que para ser abordados requieren técnicas dadas por las diversas áreas de la optimización.

Dentro de estos problemas complejos, se tienen los de clase *NP-completo* y *NP-duro* (éstos últimos son al menos tan difíciles como el más difícil de los *NP-completo*). Estos problemas tienen la característica que no han podido ser resueltos de forma óptima en tiempo polinomial por una máquina determinista (computador), lo cual los hace actualmente irresolubles en un tiempo que sea considerable como eficiente. Algunos problemas que se ha demostrado que pertenecen a esta clase son *el problema de asignación cuadrática* Sahni & Gonzalez (1976) (QAP), *el vendedor viajero*, *problema de la mochila* (TSP) y el *problema del clique* Karp (1972), entre otros .

Dado que estos problemas no se han podido resolver de forma exacta se utilizan

diferentes técnicas para obtener soluciones cercanas al óptimo, entre las cuales se tienen las metaheurísticas. Luke (2009) describe las metaheurísticas como un sub-campo de la optimización estocástica, la cual corresponde a la clase de algoritmos y técnicas que utilizan el azar para encontrar la mejor solución posible a problemas en que se apliquen. Dentro de las metaheurísticas se tienen las denominadas *basadas en poblaciones*, las cuales para encontrar la mejor respuesta, mantienen un grupo de soluciones (*individuos*) las cuales se van modificando y evaluando. Algunos ejemplos de ellas son los Algoritmos Genéticos, Estrategias Evolutivas, Colonias de Hormigas, Búsqueda Esparcida (*Scatter Search)* y Algoritmos Meméticos.

Un problema conocido dentro de las metaheurísticas basadas en poblaciones es la *convergencia prematura* del algoritmo, es decir, en un momento temprano de su desarrollo se llega a un óptimo local que no se logra superar.

Se espera que al evitar la convergencia prematura de un algoritmo, éste sea capaz de alcanzar resultados mucho mejores y, por la naturaleza de las metaheurísticas, incrementar la probabilidad de alcanzar una solución de alta calidad del problema enfrentado, lo cual permitiría dar respuestas eficientes en tiempo y más eficaces en calidad, a problemas clasificados dentro de *NP-completo* y *NP-duro*.

Dado que este problema surge desde el nacimiento de los algoritmos basados en poblaciones, se ha enfrentado de diversas formas. Algunas técnicas mencionadas por Juan et al. (2000) son la cooperación entre algoritmos, probabilidad de selección y mutación adaptativa y modificación de la función de calidad. Otras formas utilizadas para confrontar la problemática mencionada son la utilización de poblaciones con gran cantidad de individuos Luke (2009) y el uso de memoria, tanto en casos puntuales como los expuestos por Wiering (2004); Louis & Li (1997), como en diferentes metaheurísticas Taillard et al. (2001).

1.2 DESCRIPCIÓN DEL PROBLEMA

La convergencia de una metaheurística se da al momento de alcanzar una solución de alta calidad la cual no logra ser superada dentro de los límites dados por los criterios de convergencia del algoritmo y parámetros en uso. Los criterios de término otorgan límites que restringen el tiempo de búsqueda dentro de un espacio de soluciones. Estos criterios, según Sivanandam & Deepa (2008) pueden ser los siguientes:

- Número máximo de generaciones.
- Tiempo de ejecución.
- Número de iteraciones/generaciones/tiempo sin mejora de calidad de la mejor solución encontrada.
- Número de iteraciones/generaciones/tiempo sin mejora de calidad de ninguna solución en tratamiento.

Cuando hay convergencia, el hecho que no se logre superar una solución puede ser por tres casos: se alcanzó el óptimo global, se logró un muy buen óptimo local el cual tiene probabilidades muy bajas de ser superado por las características de la instancia del problema que se intenta resolver o se tuvo una convergencia prematura.

Como se mencionó, en ocasiones es difícil superar ciertos óptimos locales. Esto se da por la naturaleza de los espacios de soluciones, algunos tipos son Luke (2009): unimodales, deceptivos, “aguja en un pajar” y ruidosos. Weinberger (1990) da directrices para ver la correlación de la distancia al óptimo con calidad de soluciones dentro del espacio de búsqueda, de manera de poder tener una noción de ver el grado de dificultad de una instancia de un problema.

Conocidos los tres casos de convergencia, corresponde diferenciar la *rápida* de la *prematura*. En la figura (1-1) se puede apreciar tanto la convergencia normal, como los dos casos mencionados anteriormente. En el eje vertical se tiene el error de la mejor solución de la población, y en el eje horizontal se tiene el tiempo, que puede ser visto también como número de iteraciones o generaciones del algoritmo poblacional.

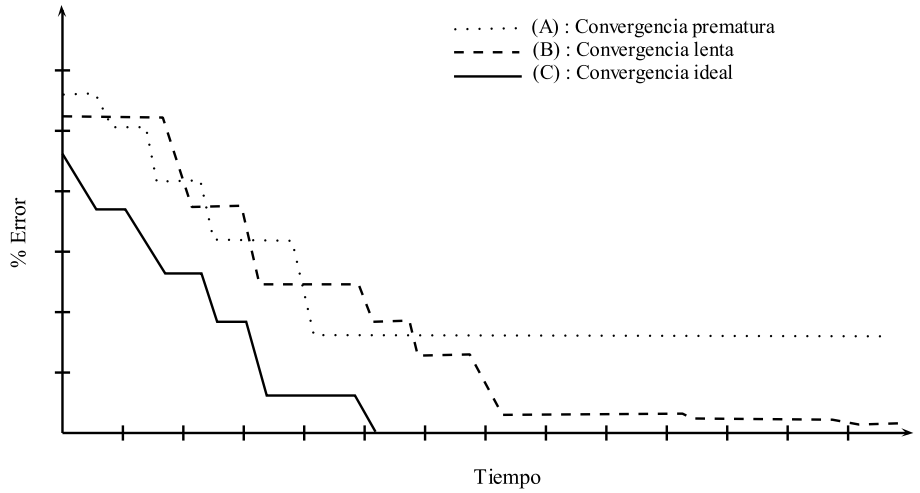

FIGURA 1-1: *Gráfico de tipos de convergencia*

En el caso mostrado en la figura (1-1 A) se puede ver que a pesar de tener inicialmente un rápido descenso en el error de la mejor solución encontrada, llega un punto en el cual no se puede superar su calidad durante el resto de la ejecución del algoritmo. En cambio en la figura (1-1 B), se produce una disminución continua del error del mejor individuo, llegando a un límite con un error cercano a cero. La situación mencionada ocurre con una lentitud mayor que en (1-1 A), es decir, se tiene una convergencia más lenta, pero que lleva a una mejor calidad de solución, lo cual es más deseable ya que en el uso de heurísticas es más importante la calidad de solución que la rapidez de cálculo Loiola et al. (2007), respetando un compromiso entre calidad de solución y rapidez de tiempo de llegar a ella. Finalmente se tiene el caso ideal dado por (1-1 C), es decir, se llega 0 % de error (óptimo

global) y en un tiempo menor.

Se han hecho estudios de la convergencia en diferentes metaheurísticas utilizando diversas técnicas para evitar la convergencia prematura. Misevičius & Kilda (2005) hacen una comparación de diferentes técnicas de cruzamiento para un algoritmo memético. Previamente Srinivas & Patnaik (1994) realizó estudios de la influencia de utilizar probabilidad adaptativa de selección y cruzamiento en algoritmos genéticos. En Fonlupt et al. (1997) se efectuó un estudio del efecto de los algoritmos genéticos por cooperación en la convergencia de éstos. Otros prefieren el uso de poblaciones grandes, pero el problema de esto es que cuando el problema tiene una función de calidad muy cara computacionalmente (como QAP), es poco viable utilizar poblaciones grandes por tiempo de cálculo, pero usar poblaciones muy pequeñas recae en el problema de la convergencia prematura.

Una técnica ampliamente utilizada para evitar la convergencia prematura es la inclusión de memoria. Por ejemplo en Wiering (2004) analizan el efecto de usar un esquema de memoria en un problema deceptivo, o en Louis & Li (1997), donde usan memoria en un algoritmo para resolver el problema del vendedor viajero. Sin embargo se piensa que la combinación de diferentes esquemas de memoria puede mejorar los resultados obtenidos por dichos esquemas aplicados independientemente, mejorando la convergencia en términos de calidad de solución. Por lo anterior, sería bueno comparar el uso de varios esquemas de memoria en una misma metaheurística específica, para un problema particular.

1.3 SOLUCIÓN PROPUESTA

La solución que se propone es la evaluación y mejora de convergencia de una metaheurística basada en poblaciones mediante el uso de diferentes esquemas de memoria. Dada la gran amplitud de metaheurísticas y problemas que abarcan, se trabajará sobre el algoritmo memético usado en Inostroza-Ponta (2008), el cual es usado para tratar QAP.

Los esquemas a usar se muestran a continuación:

- Esquemas aplicados a tareas de población:

  Población estructurada con búsqueda compartida Utilizado por Inostroza-Ponta (2008). Utilizan una estructura jerárquica poblacional de 13 *agentes*, la cual permite que los cruzamientos realizados se efectúen de manera que se mantenga una mejor diversificación poblacional. Además, en ese trabajo se utiliza *Tabu Search* (TS), y se comparten los movimientos Tabú entre soluciones de un mismo agente.

  Población estructurada sin búsqueda compartida Se mantiene la estructura poblacional jerárquica del trabajo anterior, pero se quita el uso compartido de la lista tabú, para evaluar si influye negativamente con otros esquemas de memoria.

  Matriz de cruzamientos Se recuerdan las parejas de padres que ya se han seleccionado para aplicar cruzamiento, de manera de generar una población más diversa. No es compatible con los esquemas anteriores porque éstos realizan un reemplazo de uno de los padres por el hijo creado, con lo cual la matriz de cruzamiento se vuelve inútil.

- Esquemas aplicados a la búsqueda local:

Lista tabú fija (LTF) Corresponde a la lista tabú usada por TS clásico Glover (1989, 1990), la cual es de tamaño fijo y va recordando movimientos previamente realizados para no repetirlos durante un cierto número de iteraciones. Es aplicada a QAP por Skorin-Kapov (1990).

Lista tabú variable (LTV) Es una modificación la lista tabú anteriormente mencionada, usada por Taillard (1991). Varía aleatoriamente (dentro de un rango) el tamaño de la lista tabú.

Reducción de espacio de búsqueda Usada por Merz & Freisleben (2000a). Cada individuo almacena las asignaciones comunes entre sus padres con el objetivo de reducir el espacio de búsqueda de soluciones al momento de hacer la búsqueda local.

Mejor vecino en Gradiente Descendiente Usada por Misevičius (2006) para perfeccionar la Búsqueda Tabú. Corresponde a un esquema simple, se almacena el mejor vecino encontrado durante la búsqueda local por Gradiente Descendiente.

Además de la implementación, la solución propuesta comprende análisis de los resultados que permiten comparar los algoritmos creados y los esquemas de memoria que ellos emplean.

Los algoritmos creados son comparados en los siguientes ámbitos:

- Calidad de mejor solución encontrada, expresada en porcentaje de error contra la mejor solución conocida.
- Tiempo de convergencia a la mejor solución encontrada.
- Cantidad de veces que cada algoritmo llega a la mejor solución encontrada por el mismo.

El análisis en estos ámbitos, se realiza tanto de forma global y particular (por cada instancia). Para analizar en forma global se compara el valor promedio obtenido en todos los experimentos. El análisis particular compara el desempeño de cada algoritmo contra los demás en cada instancia experimentada, reflejado a través del índice de Copeland, que es explicado más adelante.

Luego de ello, se realiza una comparación entre los esquemas de memoria aplicados, determinando su influencia en la mejora de los resultados. Esta comparación es realizada en términos de calidad de soluciones generadas.

1.3.1 Propósitos de la solución

El propósito principal de esta investigación es superar el rendimiento de un algoritmo basado en poblaciones usando esquemas de memoria, particularmente el algoritmo memético usado en Inostroza-Ponta (2008) para QAP.

Por otra parte, se desea continuar el avance científico respecto al estudio del enfrentamiento de convergencia prematura en metaheurísticas basadas en poblaciones usando esquemas de memoria.

Otro propósito de esta solución, es proponer una unificación de la forma de clasificar los esquemas de memoria usados en algoritmos, la cual no se había explicitado previamente.

Además, dado que esta investigación trabaja con algoritmos que tratan QAP, se aporta tanto a la comunidad que investiga este problema *NP-hard*, como a la mejora en la resolución de problemas de la vida diaria que contienen QAP como diseño de centros comerciales, procesos de comunicaciones, diseño de terminales de aeropuertos, diseño de circuitos electrónicos, etc.

1.3.2 Alcances y limitaciones de la solución

Dado que las metaheurísticas sirven para tratar una amplia variedad de problemas, se prueban los algoritmos generados específicamente en el *Problema de Asignación Cuadrática*. Además, como las metaheurísticas son muy diversas tanto por definición e hibridaciones, se tiene por base el algoritmo memético que enfrenta QAP dado por Inostroza-Ponta (2008). Por lo mismo, una limitante de esta investigación es que no será un resultado global, sino específicamente para algoritmos meméticos y QAP, ya que para otras metaheurísticas y problemas abordados pueden obtenerse resultados diferentes.

Por naturaleza de las metaheurísticas, las soluciones obtenidas por los algoritmos meméticos con diferentes esquemas de memoria no necesariamente son óptimas, pero buscan la mejor eficacia y eficiencia posible dentro de estos límites.

Dado que los esquemas de memoria que son posibles de implementar son ilimitados, se tendrá una cota de 6 esquemas de memoria y sus combinaciones factibles, de modo de abarcar una cantidad de algoritmos concordante al tiempo de trabajo de una tesis individual con el tiempo otorgado.

1.4 OBJETIVOS

En esta sección se darán a conocer los objetivos del presente trabajo de investigación, tanto en su enfoque general como en sus puntos específicos.

1.4.1 Objetivo general

Evaluar la mejora de convergencia del uso de esquemas de memoria en el algoritmo memético que enfrenta QAP dado por Inostroza-Ponta (2008).

1.4.2 Objetivos específicos

- Estudiar y analizar el algoritmo base propuesto por Inostroza-Ponta (2008).
- Investigar una definición basal de los esquemas de memoria.
- Revisar el estado del arte respecto a esquemas de memoria usados en metaheurísticas.
- Estudiar el estado del arte respecto a la convergencia en metaheurísticas poblacionales y QAP.
- Determinar en qué parte o partes del algoritmo memético es posible la inclusión de la memoria.
- Determinar los esquemas que serán agregados al algoritmo en cuestión.
- Realizar implementaciones del algoritmo memético con diferentes esquemas de memoria.
- Realizar evaluaciones y análisis de las implementaciones realizadas, en cuanto a calidad de soluciones, número de instancias en que cada algoritmo es superior y tiempo de convergencia a la mejor solución.
- Comparar los resultados de las evaluaciones, y determinar qué esquemas realizan mayores mejoras de desempeño.
- Determinar el mejor algoritmo encontrado en término de calidad de soluciones, compararlo detalladamente contra el original y la literatura actual.

1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS

Debido a que este trabajo tiene carácter de investigación de naturaleza científica, se utiliza el Método Científico. En este método se da una serie de pasos a seguir para realizar la investigación de manera correcta, los cuales aplicados a este trabajo son:

Observación del fenómeno, identificación y planteamiento del problema: Se define detalladamente cual es la problemática existente que se desea resolver que en este caso corresponde a la convergencia prematura. Se diferencia entre lo que se ha realizado en cuanto a mejora de convergencia en metaheurísticas poblacionales y lo que se desea hacer, especificado a los algoritmos meméticos usados para resolver QAP, correspondiente al uso y comparación de esquemas de memoria.

Formulación de la hipótesis: La hipótesis identificada es: "El uso de alguna combinación de esquemas de memoria ayuda a mejorar la convergencia del algoritmo memético de Inostroza-Ponta para el Problema de Asignación Cuadrática".

Revisión de la literatura existente: Adquirir un conocimiento amplio en los ámbitos relevantes al desarrollo de la solución, siendo en este caso: esquemas de memoria, QAP, algoritmos meméticos, metaheurísticas y convergencia en algoritmos poblacionales.

Análisis y estudio de trabajos ya realizados: Se analizan soluciones alternativas al problema planteado, se revisa el algoritmo que se desea mejorar y se extraen posibles ideas que pueden ser útiles para el diseño de la solución.

Planteamiento de la solución: Se identifican los esquemas de memoria a utilizar, se establecen los algoritmos posibles de implementar con la combinatoria factible de ellos.

Resultados y análisis: Se obtienen resultados de la implementación de la solución previamente planteada. Se analiza la convergencia de los algoritmos planteados mediante análisis de calidad de soluciones y tiempo de convergencia. También se analiza el número de veces que se supera el algoritmo original, se compara entre las implementaciones obteniendo el número de instancias en que se llega a un mejor promedio de ejecuciones.

Conclusiones: Se analiza los resultados obtenidos previamente, de manera de determinar si se acepta o rechaza la hipótesis planteada.

1.6 ORGANIZACIÓN DEL DOCUMENTO

En este capítulo se pudo revisar los ámbitos concernientes al problema, tanto en su definición, contexto y solución. En los siguientes capítulos se detalla cada uno de estos puntos, profundizando los puntos de vital importancia para el entendimiento de la investigación.

En el capítulo 2 se da una descripción detallada de los aspectos teóricos que conciernen al entendimiento de este trabajo, tanto en su planteamiento como en su solución. Se dan a conocer los algoritmos meméticos, el problema de asignación cuadrática y los esquemas de memoria. Luego se pasa a una revisión del estado del arte de la problemática, abarcando tanto la parte general de ésta (la convergencia en metaheurísticas poblacionales), como la parte específica (resolución de QAP).

En el capítulo 3 se detallan los esquemas de memoria utilizados y los algoritmos conformados con ellos, además de las herramientas utilizadas para su implementación e instancias del problema usadas para probar la efectividad de las implementaciones creadas. También se exponen los criterios de evaluación que serán usados para medir y analizar el

rendimiento de los algoritmos.

En el capítulo 4 se exponen los resultados obtenidos por los algoritmos presentados en el capítulo 3. Estos resultados son discutidos en el capítulo 5.

Finalmente, se dan las conclusiones del trabajo realizado en el capítulo 6, con respecto a los aportes que realiza, planteando el posible trabajo futuro a seguir.

Además, se adjuntan las referencias bibliográficas usadas en el documento y anexos necesarios para un entendimiento más profundo.

CAPÍTULO 2. FUNDAMENTOS TEÓRICOS

En este capítulo se detallan todos los aspectos concernientes al contexto teórico en que se encuentra el problema detallado en el capítulo anterior, y el estado actual del tratamiento de éste.

2.1 DOMINIO DEL PROBLEMA

En esta sección se dan a conocer todos los aspectos teóricos que conciernen en el entendimiento de la solución planteada, explicando el problema de asignación cuadrática, el uso de memoria en metaheurísticas y los algoritmos meméticos.

2.1.1 Problema de asignación cuadrática

Para explicar este problema, hay que tener en consideración la siguiente situación: se necesita asignar n instalaciones (*facilities*) a n localizaciones fijas (*locations*), de manera que cada localización posea asignada sólo una instalación, y viceversa. Además, las instalaciones tienen flujos entre ellas y las localizaciones, distancias; todos estos valores conocidos. Por ejemplo, en la figura 2-1 (a) se puede apreciar tanto las localizaciones como las instalaciones, y la existencia de distancias y flujos, con $n = 5$.

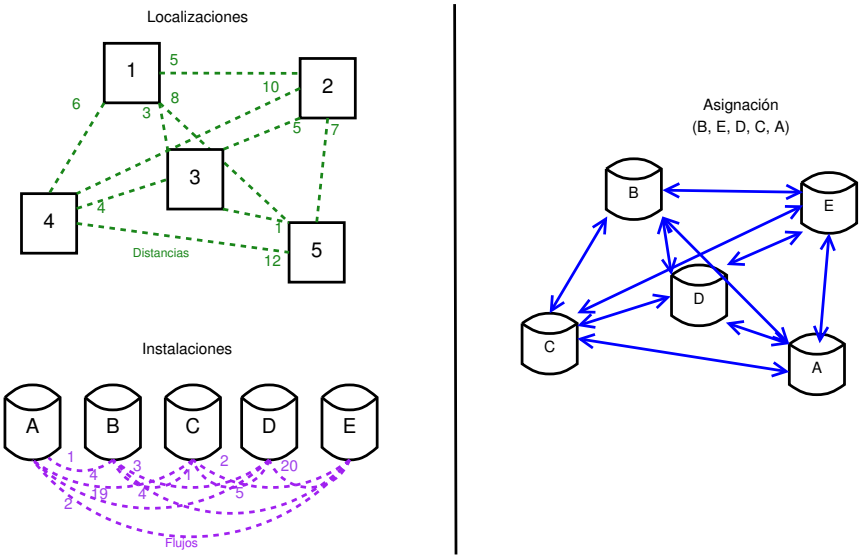

FIGURA 2-1: *(a) Representación gráfica de localizaciones e instalaciones (b) Asignación*

Es posible representar una solución válida por una permutación π, la cual indica que cada elemento $\pi(i)$ de ella corresponde a una instalación y la posición i a la localización en que se encuentra asignada. En la figura 2-1 (b) se muestra la asignación correspondiente a $\pi = (B, E, D, C, A)$ que corresponde a poner la instalación B en la localización 1, la E en la 2, etc.

Entonces, dada la matriz de distancias entre localizaciones D, la matriz de flujos entre instalaciones F (ambas de tamaño $n \times n$) y el espacio de soluciones S_n, el cual contendrá todas las permutaciones de tamaño n, con elementos entre 1 y n; el costo de una asignación π se da por:

$$C(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} F(i, j) \times D(\pi(i), \pi(j)); \pi \in S_n \tag{2.1}$$

El *problema de asignación cuadrática* (QAP, de sus siglas en inglés) consiste en encontrar la asignación π^* de instalaciones a localizaciones, de manera que el costo de ella sea mínimo.

$$C(\pi^*) = min \sum_{i=1}^{n} \sum_{j=1}^{n} F(i, j) \times D(\pi(i), \pi(j)); \pi \in S_n \tag{2.2}$$

QAP presentado por Koopmans & Beckmann (1957) como un problema de economía, y más tarde fue demostrado por Sahni & Gonzalez (1976) que pertenece a la clase *NP-Hard*, los cuales actualmente no pueden ser resueltos de forma óptima en tiempo polinomial por una máquina determinista. Por esto, se ha intentado abordar de diferentes formas tanto en su formulación como en su enfrentamiento, buscando soluciones aproximadas en tiempos manipulables.

En Loiola et al. (2007) se realiza un análisis de las formulaciones de QAP existentes a esa fecha, en las que mencionan: programación lineal entera (*integer linear programming*), programación lineal matemática mixta (*mixed integer linear programming*), formulación por permutaciones, formulación por trazas (*trace formulation*) y mediante grafos. Además, se tienen problemas relacionados como el Problema de Asignación Lineal (*Linear Assignment Problem*, LAP), problema de asigación de tres índices (*three-index assignment problem*, 3AP), problema de asignación cuadrática de cuello de botella (*The quadratic bottleneck assignment problem*, QBAP), problema de asignación bicuadrática (*The biquadratic assignment problem*, BiQAP), problema de asignación cuadrática tridimensional (*quadratic 3-dimensional assignment problem*, Q3AP), problema de semi-asignación cuadrática (*quadratic semi-assignment problem*, QSAP) y problema de asignación cuadrática multiobjetivo (*The multiobjective QAP*, mQAP).

Algunos de los problemas que se han modelado como QAP son el diseño de tableros de cableado, problemas económicos, problemas de planificación (*scheduling*), diseño de teclados y paneles de control, arqueología, análisis estadístico, diseño de parques y otros.

2.1.2 Memoria en metaheurísticas

La memoria en las metaheurísticas es un factor que siempre ha estado presente al momento de su diseño y desarrollo. Sin embargo muchas veces esto no ha sido explícito y se ha obviado su existencia. Por ejemplo, en los Algoritmos Genéticos no posee algún tipo de memoria explícita en particular, pero la población de soluciones sí constituye un almacenamiento de información importante para la resolución del problema; en cambio en Búsqueda Tabú se usa una lista tabú para la búsqueda local y otra memoria de larga duración para revisar si las soluciones fueron previamente encontradas, las cuales son declaradas de forma explícita para las tareas mencionadas Taillard et al. (2001).

Se dirá entonces que un esquema de memoria corresponde a una estructura de memoria (o varias, trabajando en conjunto) y su funcionamiento dentro de una o varias tareas dentro de un algoritmo. Las estructuras de memoria usadas en los esquemas pueden tener diferentes clasificaciones, las cuales son:

- **Clasificación por duración:** según el modelo biológico dado en Atkinson & Shiffrin (1968), la clasificación por duración de las estructuras de memoria es:
  - **Memoria sensorial:** memoria de pérdida casi instantánea. Se puede tomar como la memoria que arma el contexto en el pensamiento humano. Analogando esto a las metaheurísticas, se puede ejemplificar con las soluciones parciales que van apareciendo durante la búsqueda local o elementos intermedios que aparecen en medio de los procedimientos.
  - **Memoria de corto plazo:** memoria que dura un tiempo limitado. Se puede tener como ejemplo recordar una patente o un número de teléfono, algo que dura un tiempo corto en la mente y luego se pierde, pero que con refuerzo puede pasar a durar un plazo mayor.

Visto en el contexto de las metaheurísticas se puede analogar con los datos que se recuerdan por un procedimiento o tarea del algoritmo, y luego se pierden, por ejemplo *Tabu Search* usa la lista tabú que bloquea ciertos movimientos efectuados en un número limitado de iteraciones, hasta que llega a un óptimo local, tras lo cual la lista tabú se desecha.

- **Memoria de largo plazo:** memoria de larga duración que llega a durar toda la vida. Por ejemplo es probable que se recuerde el RUT o el nombre de los padres durante toda la vida. Asociado a las metaheurísticas se puede relacionar este tipo de memoria con los datos que se usan durante todo el algoritmo. Bajo esta definición, se puede considerar como ejemplo la población de los algoritmos genéticos y en general de cualquier metaheurística basada en poblaciones, ya que ésta dura durante toda la ejecución del algoritmo.

- **Clasificación por explicitud:** La explicitud se refiere a cómo el algoritmo trata la memoria, si es que la define como tal y la usa en una o varias tareas específicas Branke (1999).

  - **Explícita:** se almacenan datos específicos para la realización de alguna tarea particular dentro del algoritmo. Por ejemplo en la Optimización Basada en Cúmulos de Partículas (*Particle Swarm Optimization*, PSO) presentada por Kennedy & Eberhart (1995), cada partícula almacena la posición en que se encuentra y un vector de velocidad, el cual es usado para desplazarse en cada iteración.

  - **Implícita:** según Branke (1999) corresponde a la memoria existente en representaciones de algoritmos evolutivos que contienen más información de la necesaria para definir el fenotipo, ya que almacenan soluciones que pueden

ser reusadas en el futuro. Para esta investigación se varía esta definición, para que abarque más allá de los algoritmos evolutivos. Una memoria será implícita si almacena datos (tanto esenciales como redundantes o de poca importancia al problema) que no tienen una única tarea específica en la que son usadas, y son elementos existentes por la naturaleza de la solución. Por ejemplo, esto es realizado en las metaheurísticas poblacionales, ya que almacenan soluciones en poblaciones de individuos, pero no se explicita como una memoria, sino se tienen por la naturaleza de esos algoritmos. Además, los individuos son usados de diversas formas y para diferentes tareas.

- **Clasificación por variabilidad de tamaño:** se tienen dos tipos:

  - **Tamaño de memoria fijo:** la memoria tiene un largo limitado y estático. Corresponde a algoritmos que utilizan un tamaño fijo de memoria para la resolución de problemas, por ejemplo, en un algoritmo genético usualmente se da la definición de un tamaño fijo de población, en Yang (2005) usan una memoria fija de tamaño $0,1 \times n$ para almacenar soluciones y ser insertadas en la población cuando ocurren cambios de entorno, en *Fixed Tabu Search* Glover (1990, 1989) se tiene una lista tabú de tamaño fijo, etc.

  - **Tamaño de memoria variable:** el tamaño de la memoria varía de acuerdo al desarrollo del algoritmo, entre cierto límite establecido *a priori*. Por ejemplo, esto es aplicado por Simoes et al. (2007) para afrontar entornos dinámicos y también se usa en *Robust Tabu Search* Taillard (1991), donde los autores usan una lista tabú de largo variable aleatoriamente dentro de un rango.

- **Clasificación por cobertura:** esta clasificación está dada específicamente para las metaheurísticas basadas en poblaciones, de acuerdo a la globalidad o particularidad

de su acceso, puede ser:

- **Poblacional:** la memoria es accesible para tareas de toda la población o parte de ella. Algunas de estas tareas pueden ser el cruzamiento, reemplazo u otras.

  Por ejemplo en Wiering (2004) se usa una memoria que almacena soluciones previas para reemplazar rápidamente las que salgan repetidas. Otro ejemplo es en Colonia de Hormigas, donde se usa un rastro de feromonas que es conocido por toda la población de hormigas.

- **Individual:** la memoria es accesible para tareas de los individuos, por ejemplo búsqueda local, mutación, etc. Por ejemplo en Merz & Freisleben (2000a) cada individuo recuerda los puntos comunes de cruzamiento entre sus padres, de manera de reducir el espacio de soluciones al hacer la búsqueda local.

2.1.3 Algoritmos meméticos

Dado que en la solución del problema se trabaja con esquemas de memoria aplicados a un algoritmo memético, corresponde definirlos y detallarlos para un mejor entendimiento de la solución creada.

2.1.3.1 Definición

Los algoritmos meméticos (AM) surgen a fines de la década de los 80 en el trabajo de Moscato (1989). Son técnicas de optimización que pertenecen a las denominadas *metaheurísticas basadas en poblaciones* y acuñan conceptos tanto de la computación evolutiva como de la búsqueda local Moscato & Cotta (2003). La denominación de *meméticos* surge de la palabra *meme*, el cual corresponde al análogo de gen en el ámbito

de la evolución cultural. Según Dawkins (1976):

> Ejemplos de "memes" son melodías, ideas, frases hechas, modas en la vestimenta, formas de hacer vasijas, o de construir bóvedas. Del mismo modo que los genes se propagan en el acervo genético a través de gametos, los "memes" se propagan en el acervo memético saltando de cerebro a cerebro en un proceso que, en un amplio sentido, puede denominarse imitación.

De esta misma definición se puede entender que los AM trabajan con todo el conocimiento disponible , y además de compartirlo, al momento de ser recibido éste es mejorado. Esta mejora y explotación del conocimiento que se tiene del problema de optimización enfrentado funciona acorde al teorema dado por Wolpert & Macready (1997) conocido como "*No Free Lunch*" el cual indica si un algoritmo funciona muy bien para una clase de problemas, para otras clases mostrará un peor rendimiento, por lo que el desempeño de un algoritmo se dará de acuerdo a la cantidad y calidad conocimiento de la globalidad del problema que incorpore en su enfrentamiento.

2.1.3.2 Forma de un AM

Los AM heredan muchos conceptos de la computación evolutiva, debido principalmente a la naturaleza poblacional que poseen. Un AM tiene una población de *agentes*, los cuales pueden considerarse como análogos a los *individuos* en los algoritmos evolutivos, con ciertas diferencias claves (por ejemplo un agente puede tener diversas soluciones u otros datos); los cuales interactúan tanto compitiendo como colaborando entre sí para llegar a mejores soluciones. Cuando se considera la población de agentes se agrupan algunos pasos en las llamadas *generaciones*.

En el algoritmo 1 se puede apreciar que en una generación se actualiza la población

de agentes, tras la generación de nuevos agentes por la aplicación de operadores de reproducción (como recombinación y mutación) a una selección de agentes padres existentes en la población. La selección (*Seleccionar()*, línea 1) elige un conjunto de padres, de forma aleatoria o sistemática (por ejemplo usando alguna función guía que permita priorizar a los agentes de mejor calidad), para ser operados. La reproducción (*Reproducir()*, línea 2) aplica entre los padres seleccionados un conjunto de operadores de manera de generar nuevos agentes. La actualización (*Actualizar()*, línea 3) se encarga de crear una nueva población, tomando agentes de la antigua y la nueva bajo criterios dependientes del algoritmo. Los criterios mencionados pueden ser: mantener los mejores agentes, mantener las soluciones más diferentes, azar, etc.

Algoritmo 1 Pasos de una generación

Entrada: *población*[]: población de Agentes, *operadores*[], conjunto de operadores a aplicar.

Salida: Población de Agentes actualizada.

1: *padres* ← Seleccionar(*población*);
2: *nuevaPoblación* ← Reproducir(*padres*, *operadores*);
3: *población* ← Actualizar(*población*, *nuevaPoblación*);
4: **retornar** *población*;

En la función de reproducción participan diferentes operadores con el objetivo de generar nuevos agentes, dentro de los cuales se tiene el *cruzamiento* y la *mutación*. El cruzamiento se encarga de realizar la cooperación entre agentes, combinando parte de la solución de cada padre de manera de generar una nueva. La mutación se encarga de incluir información externa a las nuevas soluciones, de manera de ampliar el espacio de soluciones abarcable por la población y sus nuevos agentes.

Tanto la función selección como la actualización, son pasos que efectúan la competencia entre los agentes de la población y la reproducción funciona como medio de cooperación entre ellos.

Como se puede ver en el algoritmo 2, la etapa de reproducción consiste en la aplicación sucesiva de un conjunto de operadores el cual genera un grupo de agentes a partir de la población original. Primero se crea un *buffer* (línea 3) que almacena la población tras la sucesiva aplicación de los operadores (*Aplicar()*, línea 6). Algunos de estos operadores son selección, cruzamiento y mutación.

Algoritmo 2 Pasos de la reproducción

Entrada: *población*[]: población de Agentes, *operadores*[], conjunto de operadores a aplicar.
Salida: Población con operadores aplicados.

```
1: largoPoblación ← Largo(población);
2: largoOperadores ← Largo(operadores);
3: buffer ← Agente[largoPoblación][largoOperadores];
4: buffer[0] ← población;
5: para i = 1 → largoPoblación hacer
6:    buffer[i] ← Aplicar(operadores[i], buffer[i − 1]);
7: fin para
8: retornar buffer[largoOperadores];
```

Moscato & Cotta (2003) indica que los AM tienen como característica diferenciadora frente a otras metaheurísticas el uso de un *metaoperador* correspondiente a la aplicación sucesiva de un operador de mutación a un agente, de manera de ir mejorando su calidad de solución, por lo que los denomina *optimizadores locales*.

Como se puede apreciar en el algoritmo 3, el optimizador local tiene un criterio de término (*TerminaciónLocal()*, línea 6) el cual es dado dependiendo del algoritmo aplicado pudiendo ser un número de iteraciones, tiempo de ejecución, variación de calidad de soluciones, etc. El trabajo de optimización consiste en aplicar un operador (*Aplicar()*, línea 2) una cantidad de veces dada por el criterio de término explicado anteriormente, de manera de ir quedando con una solución de cada vez mejor calidad (línea 3). Dado que el optimizador realiza una modificación de un agente, también puede ser considerado un operador dentro de la etapa de reproducción. La aplicación de la optimización local puede

ser en diversos puntos de la reproducción, ya sea luego de la recombinación, después de la mutación, al término de la reproducción, etc.

Algoritmo 3 Forma de un optimizador local

Entrada: *actual*: Agente a mejorar, *operador*, operador a aplicar para hacer la mejora local.
Salida: Agente mejorado.

```
1: repetir
2:    nuevo ← Aplicar(operador, actual);
3:    si EsMejor(nuevo, actual) entonces
4:       actual ← nuevo;
5:    fin si
6: hasta que TerminaciónLocal()
7: retornar actual;
```

Ya definido el proceso ocurrido en una generación, se puede ver el algoritmo memético (algoritmo 4) como tal, siendo una serie de repeticiones de estas generaciones, las cuales se realizan hasta cumplir con un criterio de término del algoritmo (*TérminoAlgoritmoMemético()*, línea 7). Además se pueden ver tres partes no mencionadas previamente: (a) el inicio de la población, (b) convergencia y (c) reinicio de la población.

(a) El inicio de la población (*IniciarPoblación()*, línea 1) puede ser realizado de una forma aleatoria o heurística.

(b) La convergencia de la población (*Convergencia()*, línea 4) es muy variada de comprobar. Puede ser medida por la distancia entre agentes de la población, número de generaciones con igual mejor solución, etc. Es un factor determinante del algoritmo memético, pues le permitirá escapar de óptimos locales y evitar la convergencia prematura.

(c) El reinicio de la población (*ReiniciarPoblación()*, línea 5) es aplicado cuando hay convergencia de la población a un óptimo local. El reinicio puede ser aplicado

de diferentes formas, por ejemplo el reinicio aleatorio de individuos, aplicarles mutación en alto grado, etc.

Luego de realizar los pasos generacionales permitidos por el criterio de término, la solución del algoritmo memético se obtiene del mejor agente de la población (*MejorAgente()*, línea 8).

Algoritmo 4 Algoritmo Memético

Entrada: *tamañoPoblación*: Tamaño de la población, *operadores*, operadores a aplicar.
Salida: Agente con la mejor solución.

```
1: población ← IniciarPoblación(tamañoPoblación);
2: repetir
3:    población ← PasoGeneracional(población, operadores);
4:    si Convergencia(población) entonces
5:       población ← ReiniciarPoblación(población);
6:    fin si
7: hasta que TérminoAlgoritmoMemético()
8: retornar MejorAgente(población);
```

Como se puede ver en la estructura expuesta anteriormente, hay muchos parámetros a definir en el algoritmo, los cuales son cruciales en su desempeño. Algunos de éstos son: tamaño de población, criterio de selección a utilizar, tipo de recombinación a usar, cantidad de nuevos agentes a crear, tipo de mutación a utilizar, probabilidad de mutación a emplear, criterio de reinicio, tipo de reinicio a utilizar, optimización local a aplicar y otros.

2.1.3.3 Aplicaciones de los AM

Moscato (2003) muestra una gran cantidad de aplicaciones que han tenido los AM desde sus orígenes a fines de los años 80. Algunas aplicaciones mencionadas son las siguientes: particionado de grafos Merz & Freisleben (2000b); Yeh (2000), *Bin-Packing* Reeves (1996), problemas de *Scheduling* Franca et al. (1999), problema de

asignación cuadrática Carrizo et al. (1992), vendedor viajero Holstein & Moscato (1999) y programación no lineal entera Taguchi et al. (1998).

Además del área de optimización, también existen aplicaciones en áreas como Robótica, Economía, Aprendizaje Artificial, Electrónica, Ingeniería, Medicina, Oceanografía y más Moscato (2003).

2.2 ESTADO DEL ARTE

En la presente sección se revisa el estado actual del problema descrito en el capítulo 1, la convergencia prematura, mostrando diferentes técnicas de enfrentamiento e investigaciones donde ello ha sido realizado.

2.2.1 Técnicas para evitar la convergencia prematura

Dado que ya fue presentado el problema de la convergencia prematura en metaheurísticas basadas en poblaciones, se puede mostrar diversas formas en que, a la fecha, se ha enfrentado el problema. Las principales técnicas para lidiar con esto están basadas en la parametrización de los algoritmos o en la selección de sus operadores.

En el ámbito de los operadores de cruzamiento se ha realizado diversas investigaciones. Picek & Golub (2010) realizan una exhaustiva comparación entre diversos cruzamientos para algoritmos genéticos que son codificados como arreglos booleanos, de manera de establecer conclusiones acerca de su desempeño. Ahmed (2010) propone un nuevo operador de cruzamiento denominado *Sequential Constructive Crossover* (SCX), el cual es aplicado a un algoritmo genético que enfrenta TSP, superando a varios operadores de cruzamiento comparados en el estudio. También para TSP se estudian las variaciones del cruzamiento ordenado (*Order Crossover*, OX) en Deep & Adane (2011) que intenta me-

jorar la eficiencia del algoritmo en el que se aplica. Otro estudio es realizado por Herrera et al. (2005), en donde estudia el efecto de los diferentes cruzamientos aplicados a algoritmos codificados en números reales, analizando cuando el desempeño es mejor mediante una combinación de cruzamientos que uno simple. También es importante mencionar el estudio de Misevičius & Kilda (2005) en que analizan el impacto de diferentes operadores de recombinación sobre algoritmos meméticos, viendo el efecto de ellos aplicado a la resolución de QAP para encontrar mejores soluciones.

Respecto a los operadores de mutación, también se han realizado variadas investigaciones. Korejo et al. (2009) analizan varios operadores de mutación adaptativa en un algoritmo genético, mostrando que generalmente son mejores los operadores de nivel de gen que de población. Hong et al. (2000) en su trabajo aplican varios tipos de mutación en un algoritmo genético, los cuales son elegidos para ser aplicados sobre los hijos de acuerdo a la calidad de las nuevas generaciones que van generando. Otro trabajo posible de mencionar es el realizado por Misevičius (2005) en el cual aplican variados operadores de mutación para el proceso de diversificación en un algoritmo de Búsqueda Tabú.

Un parámetro importante que también ha sido abarcado en estudios es el tamaño de la población, ya que con muchos individuos se hace más lenta la explotación y con muy pocos se produce convergencia prematura. Existen estudios antiguos como en Odetayo (1993) en los que se intenta buscar tamaños óptimos de poblaciones. Investigaciones más actuales como las realizadas por Affenzeller et al. (2007) y Tan et al. (2001) han explorado el uso de algoritmos genéticos con tamaños de población adaptativos, de manera de obtener mejores resultados en la aplicación de los algoritmos.

Otra técnica que se ha usado para evitar la convergencia prematura es mediante la exploración variable. Una forma en que se realiza esto es variando la profundidad de búsqueda como lo realiza Sudholt (2011) en algoritmos evolutivos, aplicados a los

problemas de corte mínimo (*MINCUT*), mochila (*KNAPSACK*), y máxima satisfacibilidad (*MAXSAT*). Otro tipo de exploración variable es el usado en la metaheurística propuesta por Mladenović & Hansen (1997) y sus variaciones, en las que varían el tamaño de la vecindad (dentro de un rango) para evitar atascarse en óptimos locales y con ello caer en convergencia prematura.

En Sudholt (2009) realizan un análisis matemático sobre un algoritmo memético simple que resuelve un problema de optimización semi-booleana, para determinar una correcta parametrización de aspectos como el tamaño de la población, la cantidad de descendientes por generación, la cantidad de iteraciones en la búsqueda local y otros, de manera de obtener los mejores resultados con soluciones de mejor calidad.

Un último aspecto que se quiere destacar, que ha sido usado en el tratamiento de la convergencia prematura es el uso de esquemas de memoria. Muchas veces el uso de *esquemas* como tal, ha sido implícito o asumido sin especificarlo, por lo que no se han hecho grandes comparaciones entre ellos siendo tratados como *memoria*. Wiering (2004) utilizó un esquema de memoria explícita en que almacenó soluciones previamente halladas con una tabla *hash*, para sacar de la población las soluciones que ya fueron exploradas previamente en un problema deceptivo, mostrando una mejora en los resultados contra el algoritmo sin esquema de memoria explícita. En Buriol et al. (2004) se utiliza un esquema de memoria poblacional implícito, el cual se utiliza en el largo plazo para mantener la diversidad de la población al momento que se llega a convergencia en las sub-poblaciones presentadas en ese algoritmo, junto con otro esquema explícito en el cual cada agente almacena una solución en un bolsillo (*pocket*). Ese mismo esquema de memoria poblacional es utilizado en Inostroza-Ponta (2008), sumándole el uso de la memoria que ofrece Búsqueda Tabú, usándola a largo plazo.

2.2.2 Enfrentamiento del problema de asignación cuadrática

Como la solución propuesta usará un algoritmo memético para QAP, es prudente mencionar algunos ejemplos de aplicaciones de metaheurísticas al problema nombrado.

- Algoritmos genéticos: Srinivas & Patnaik (1994); Tate et al. (1995); Wang & Okazaki (2005); Drezner (2005).

- Algoritmos híbridos/meméticos: Vazquez & Whitley (2000); Merz & Freisleben (2000a); Misevičius & Kilda (2005); Inostroza-Ponta (2008); Misevičius & Rubliauskas (2009)

- Colonia de hormigas: Gambardella et al. (1999); Tseng & Liang (2006); Ramkumar et al. (2009)

- Búsqueda Tabú, *Simulated Annealing* y otros: James et al. (2009); Song et al. (2009); Paul (2010); Rego et al. (2010); Hussin & Stutzle (2010); Fescioglu-Unver & Kokar (2011)

Finalmente, cabe destacar otras formas de lidiar con QAP mencionadas por Loiola et al. (2007): algoritmos exactos (*branch and bound*, programación dinámica, etc), heurísticas (métodos enumerativos y de mejora) y las ya mencionadas metaheurísticas, tanto basadas en procesos naturales (*Simulated Annealing*, Algoritmos Genéticos, etc), como basados en consideraciones teóricas y experimentales (*Tabu Search*, GRASP, etc).

Como se pudo ver, existen diversas formas en que se ha enfrentado el problema de la convergencia prematura, desde la búsqueda de una correcta parametrización hasta el análisis de operadores. Una de las técnicas empleadas es el uso de memoria, de manera de evitar repeticiones y hacer operaciones más sistemáticas. Sin embargo, algo que no se ha

realizado es la combinación de diferentes esquemas de memoria, de manera de encontrar posibles mejoras de desempeño en la resolución del algún problema particular.

En este capítulo pudo revisarse en profundidad los aspectos concernientes al contexto del problema, mostrando sus definiciones y características. Además, se mostró en detalle la actualidad de la problemática tratada, dando a conocer soluciones actuales a ella y trabajos en los que éstas son aplicadas. Ahora que se tiene conocimiento de los aspectos teóricos del problema se mostrarán las características de la solución, dadas por los algoritmos creados y métricas de evaluación a ellos.

CAPÍTULO 3. ALGORITMOS Y CRITERIOS DE EVALUACIÓN

En el presente capítulo se dan a conocer los algoritmos creados a partir de la inclusión de esquemas de memoria. Para ello, primero se muestra el algoritmo en el que se basa la investigación, tras lo que se presentan los esquemas de memoria utilizados. Luego, se dan a conocer los algoritmos creados y los criterios con que éstos son evaluados.

3.1 ALGORITMO MEMÉTICO PARA EL PROBLEMA DE ASIGNACIÓN CUADRÁTICA

El algoritmo memético en el que se basa la investigación corresponde al usado por Inostroza-Ponta (2008), el cual enfrenta el problema de asignación cuadrática. Este algoritmo está basado en la propuesta de Buriol et al. (2004), el cual posee una población estructurada organizando los agentes en subpoblaciones jerárquicas. Cada una de estas subpoblaciones posee un agente líder (*leader*) y agentes de soporte (*support*).

Como ejemplo, en la figura 3-1 la subpoblación $subpop^0$ es la principal, cuyo agente líder es el $agent^0$. Las subpoblaciones $subpop^1$, $subpop^2$ y $subpop^3$ son jerárquicamente inferiores, es decir, tienen soluciones de menor calidad. Notar que los agentes de soporte de $subpop^0$, $agent^1$, $agent^2$ y $agent^3$ son agentes líderes de las subpoblaciones $subpop^1$, $subpop^2$ y $subpop^3$ respectivamente. Además, el superíndice de cada agente corresponde al índice de subpoblación.

Cada uno de los agentes posee *pockets*, los cuales son usados para almacenar soluciones y usarlas al momento de realizar el cruzamiento, de manera de tener una población y generar soluciones con mayor diversificación.

La jerarquía poblacional se da en el hecho que en cada población los agentes líderes siempre tienen las mejores soluciones dentro de sus subpoblaciones, de manera que el agente líder de la subpoblación principal tiene la mejor solución encontrada. Cabe destacar que los agentes de soporte de una subpoblación pueden ser a su vez agentes líderes de las otras subpoblaciones jerárquicamente inferiores.

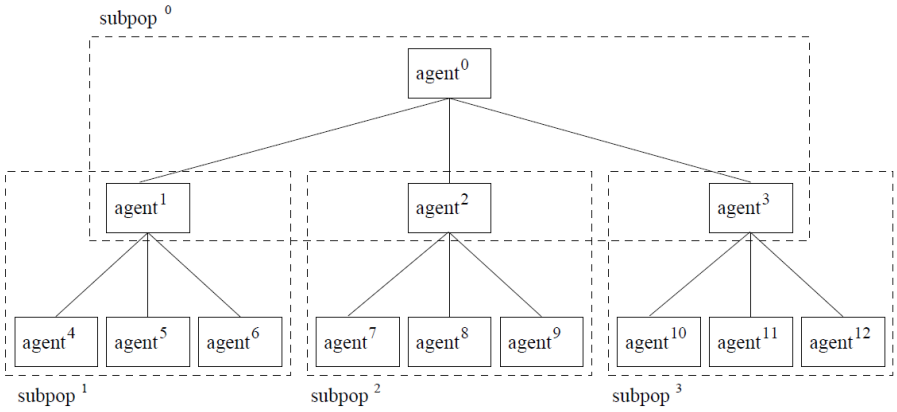

FIGURA 3-1: *Estructura poblacional usada por Inostroza-Ponta (2008)*

Esta estructura poblacional es utilizada para mantener la diversidad de la población general, ya que cada subpoblación va aplicando los cruzamientos internamente a menos que haya convergencia en ella, tras lo que se aplica cruzamiento con agentes de otras subpoblaciones.

A continuación se detallan los aspectos concernientes al algoritmo descrito: la representación de soluciones, los parámetros seleccionados y los operadores utilizados.

3.1.1 Representación de soluciones y población

La representación de las soluciones constituye un factor fundamental en el diseño del algoritmo memético.

Las instancias QAP probadas, que serán explicadas más adelante, poseen la misma cantidad n de instalaciones y localizaciones. Por lo anterior, se debe tener alguna estructura que represente la asignación de n instalaciones en n localizaciones.

Cada asignación es única y obligatoria: una instalación i puede ser asignada a una y sólo una localización j, y esa localización puede poseer una y sólo una instalación asignada.

Por lo anterior, el conjunto de asignaciones correspondientes a una solución válida se representan mediante una permutación π de n elementos. En la permutación π, un elemento $\pi(i)$ representa la instalación en la localización i.

Por ejemplo considerando $n = 5$ una posible solución es π = [2][5][1][4][3], la cual significa:

- $\pi(1) = 2$ indica que la instalación 2 está asignada en la localización 1.
- $\pi(2) = 5$ indica que la instalación 5 está asignada en la localización 2.
- $\pi(3) = 1$ indica que la instalación 1 está asignada en la localización 3.
- $\pi(4) = 4$ indica que la instalación 4 está asignada en la localización 4.
- $\pi(5) = 3$ indica que la instalación 3 está asignada en la localización 5.

La población de agentes se representa como una matriz de soluciones, las cuales tienen la organización mostrada en la figura 3-2. Dentro de esta matriz, cada celda representa una permutación π almacenada por un agente. Cada agente está dado por una fila de la matriz y cada *pocket* por una columna.

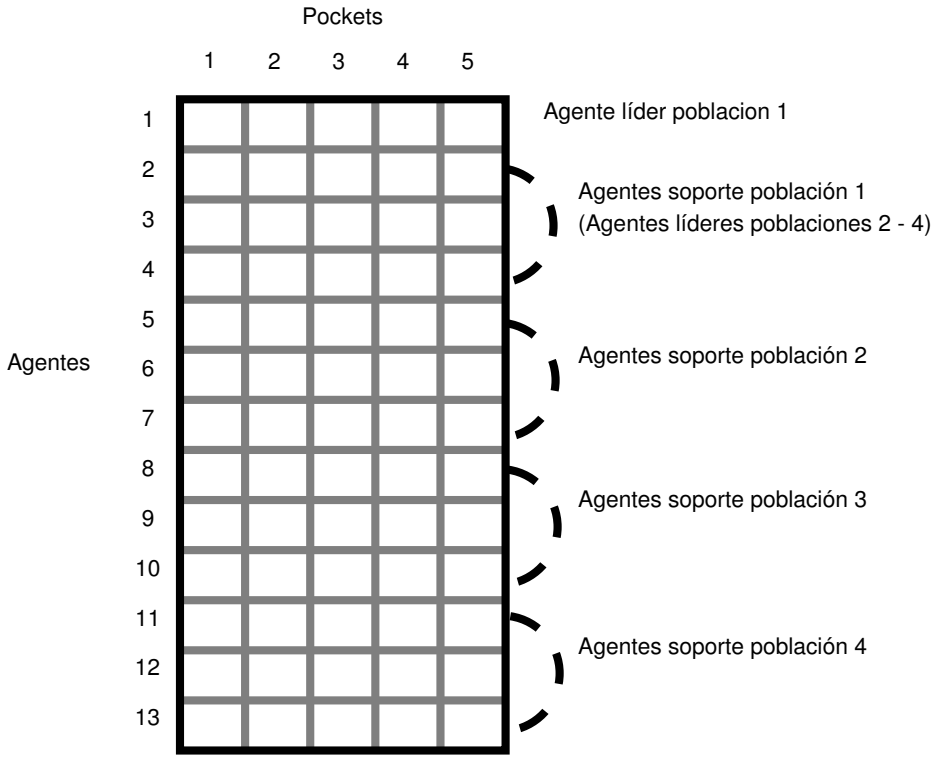

FIGURA 3-2: *Representación de la población estructurada en una matriz*

3.1.2 Operadores del algoritmo memético

i) Inicio de la población: Cada uno de los agentes es iniciado como uno nuevo aleatorio, tras lo cual se le aplica búsqueda local y se actualiza la población.

ii) Selección: El proceso de selección aprovecha la estructura poblacional, siendo realizada para cada subpoblación. En este proceso, se analiza la convergencia de la subpoblación que se pretende hacer la selección. De no haber convergencia en la subpoblación, se selecciona (y cruza) *internamente*, es decir, una solución del agente líder con una solución de un agente de soporte, elegidas al azar, como se muestra en el algoritmo 5. En este algoritmo, la función *elementoAleatorioEnPocketNoVacío()*

(línea 4), entrega un elemento aleatorio no vacío del agente indicado por el segundo parámetro (*padre*) dentro de la población dada por el primer parámetro (*población*). La función *cruzamiento()* aplica el cruzamiento entre los dos agentes padres seleccionados previamente.

Algoritmo 5 Selección y cruzamiento interno

Entrada: *poblacion*: población de agentes, *índice*: indice de la subpoblación.
Salida: *poblacion*: población con nuevas soluciones.

1: **para** $i = 1 \rightarrow 3$ **hacer**
2: $padre1 \leftarrow índice$
3: $padre2 \leftarrow (3 \times índice) + i$
4: $pocket1 \leftarrow$ elementoAleatorioEnPocketNoVacío(*población*, $padre1$);
5: $pocket2 \leftarrow$ elementoAleatorioEnPocketNoVacío(*población*, $padre2$);
6: $p1 \leftarrow población[pocket1][padre1]$
7: $p2 \leftarrow población[pocket2][padre2]$
8: $población[0][padre2] \leftarrow$ cruzamiento($p1$, $p2$);
9: **fin para**
10: **retornar** *población*

Si hay convergencia en la subpoblación, entonces se selecciona (y cruza) externamente, es decir, agentes de la subpoblación en cuestión con otros de subpoblaciones diferentes, como muestra el algoritmo 6. En este algoritmo, la función *aleatorioEntre()* (línea 2) entrega un número aleatorio entre los dos parámetros y diferente a *índice*. La función *tresAleatoriosDiferentesEntre()* (línea 3), entrega 3 números aleatorios diferentes entre los números pasados por parámetro. Las funciones *elementoAleatorioEnPocketNoVacío()* y *cruzamiento()* cumplen las mismas tareas del algoritmo 5.

Algoritmo 6 Selección y cruzamiento externo

Entrada: *población*: población de agentes, *índice*: indice de la subpoblación
Salida: *población*: población con nuevas soluciones

```
1:  agente1 ← índice
2:  agente2 ← aleatorioEntre(1, 3)≠ índice
3:  índices1[] ← tresAleatoriosDiferentesEntre(1,3);
4:  índices2[] ← tresAleatoriosDiferentesEntre(1,3);
5:  para i = 1 → 3 hacer
6:     padre1 ← (3 × agente1) + índices1[i]
7:     padre2 ← (3 × agente2) + índices2[i]
8:     pocket1 ← elementoAleatorioEnPocketNoVacío(población, padre1);
9:     pocket2 ← elementoAleatorioEnPocketNoVacío(población, padre2);
10:    p1 ← población[pocket1][padre1]
11:    p2 ← población[pocket2][padre2]
12:    población[0][padre1] ← cruzamiento(p1, p2);
13: fin para
14: retornar población
```

iii) Cruzamiento: El cruzamiento corresponde al cruzamiento cíclico o *cycle crossover* (CX). Este cruzamiento tiene la ventaja de no introducir mutaciones implícitas, pero genera poca variedad de hijos. La forma de no introducir mutaciones y así mantener la factibilidad de una solución, es detectar los "ciclos" entre dos padres. Un ciclo corresponde al conjunto de elementos entre dos padres que deben ser copiados en completitud desde alguno de ellos, de lo contrario dichos elementos no quedarán en posiciones iguales a alguna de las de su padre de origen y podrían repetirse u omitirse, dejando la solución infactible. En la figura 3-3 se representa un ciclo por cada número sobre el arreglo, comenzando desde la primera posición del primer padre.

En el proceso de cruzamiento primero se copian los elementos iguales de ambos padres, luego se va tomando una posición no asignada aleatoria y se copia el ciclo completo que la contiene en el hijo, desde uno de los padres. Esto se repite hasta que no queden posiciones no asignadas.

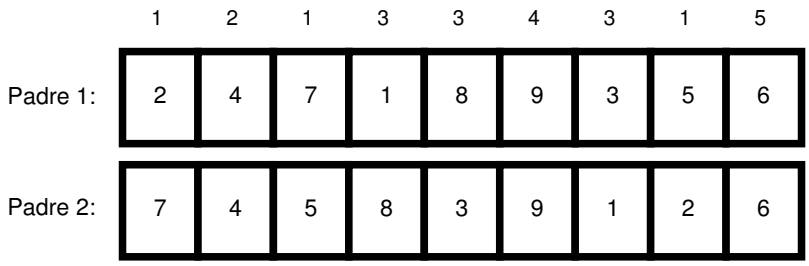

FIGURA 3-3: *Ciclos entre dos padres*

Por ejemplo, en la figura 3-4 se toma inicialmente la primera posición del primer padre, tras lo cual se copia el ciclo completo, correspondientes a los elementos 2, 7 y 5. Luego se comienza a copiar otro ciclo desde alguna posición no usada aleatoria el cual en este caso corresponde a, desde el segundo padre, los elementos 8, 1 y 3.

iv) Mutación: En esta investigación no utilizan un operador de mutación, de manera de preservar la información proveniente desde los padres.

v) Búsqueda local: Para la búsqueda local, se utiliza una variación de Búsqueda Tabú Robusta (*Robust Tabu Search*) presentada por Taillard (1991). El tamaño de la lista es variable. Los pasos que sigue se dan en el algoritmo 7. En dicho algoritmo, la función *azar()* (línea 3), devuelve un decimal entre los dos parámetros. La función *Costo()* (línea 4), entrega el costo de la solución pasada por parámetro. La función *encontrarMejorMovimiento()* (línea 6), entrega el mejor movimiento no tabú realizable en una solución, consistente en el intercambio de dos elementos que no están en tabú, a menos que superen la calidad de la mejor solución de la población. *intercambiar()* (línea 9), intercambia dos elementos dentro de un individuo.

vi) Actualización y reemplazo: La actualización de la población ocurre de manera que los agentes de líderes quedan con las mejores soluciones de su subpoblación, de

manera de mantener la jerarquía. El reemplazo depende del cruzamiento, como se vieron en los algoritmos 5 y 6: si se usa cruzamiento interno, se reemplaza (o agrega) la solución al agente de soporte; si se usa cruzamiento externo, se reemplaza una solución al azar de un agente de soporte de la población elegida.

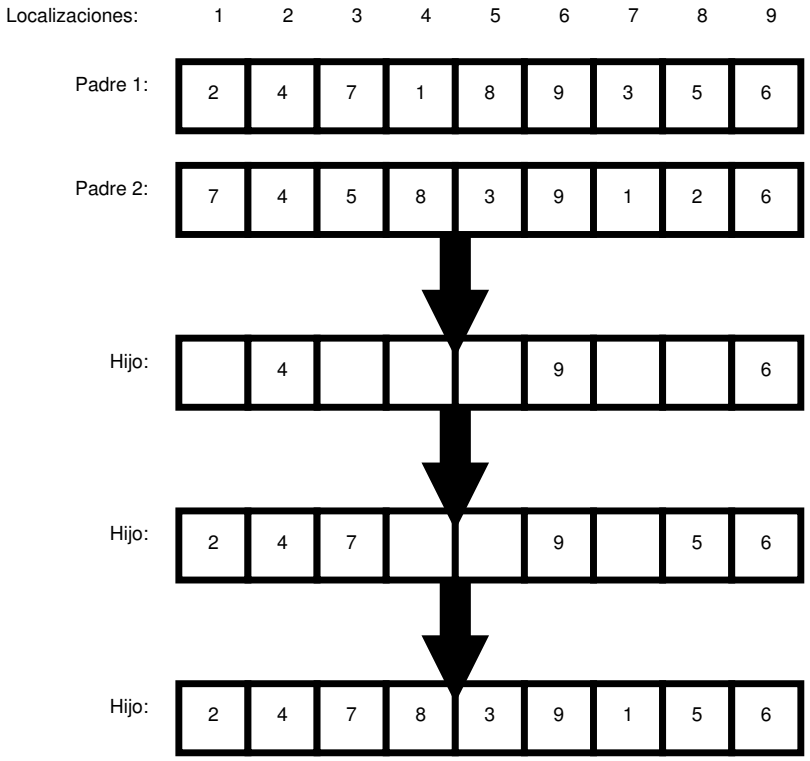

FIGURA 3-4: *Cruzamiento CX*

vii) Pérdida de diversidad: La pérdida de diversidad de una subpoblación se ve tomando tres soluciones al azar (una por cada agente soporte) dentro de la subpoblación, las cuales se van comparando elemento a elemento de sus permutaciones $0.2 \times n$ veces. Si todos los elementos comparados son iguales, hay convergencia de la subpoblación.

viii) Reinicio: Para reiniciar la población, se respalda la mejor solución encontrada a ese momento y cada uno de los agentes es iniciado como uno nuevo aleatorio, tras lo

cual se le aplica búsqueda local y se actualiza la población.

Algoritmo 7 Estructura de búsqueda tabú usada por Inostroza-Ponta (2008)

Entrada: *individuo*: solución a mejorar, $MAX\_ITERACIONES$: máximo de iteraciones sin cambios, *lista*: número de lista tabú a emplear
Salida: *individuo*: solución mejorada

```
1:  iteración ← 0;
2:  sinCambios ← 0;
3:  tamañoLista ← azar(0.1, 0.5) ×MAX_ITERACIONES;
4:  mejorCosto ← Costo(individuo);
5:  mientras sinCambios < MAX_ITERACIONES hacer
6:     (u, v) ← encontrarMejorMovimiento(iteración, individuo);
7:     listaTabu[lista][u][individuo[u]] ← iteración + tamañoLista
8:     listaTabu[lista][v][individuo[v]] ← iteración + tamañoLista
9:     individuo ← intercambiar(u, v, individuo);
10:    si mejorCosto > Costo(individuo) entonces
11:       mejorCosto ← Costo(individuo);
12:       sinCambios ← 0;
13:    si no
14:       sinCambios ← sinCambios + 1;
15:    fin si
16:    tamañoLista ← azar(0.1, 0.5) ×MAX_ITERACIONES;
17:    iteración ← iteración + 1;
18: fin mientras
19: retornar individuo
```

3.1.3 Parámetros del algoritmo memético

A continuación se detallan los parámetros usados en Inostroza-Ponta (2008).

i) Tamaño de población: El tamaño de población es de 13 agentes. Dentro de esta población existen 4 subpoblaciones de 4 agentes cada una, de los cuales uno es el líder y los demás son agentes de soporte. Una de las subpoblaciones es la principal, por lo que sus agentes de soporte son líderes de las demás subpoblaciones.

ii) Cantidad de soluciones por agente (*pockets*): Cada agente posee 5 *pockets*.

iii) Criterio de término algoritmo: El criterio de término del algoritmo es tiempo máximo de ejecución y/o un valor de función de costo a superar/igualar (*best known value*).

iv) Criterio de reinicio de población: El criterio de reinicio del algoritmo es la repetición de la misma mejor solución de la población una cantidad de $n/4$ veces.

v) Criterio de término búsqueda local: El criterio de término de la búsqueda local es 100 iteraciones sin mejorar la calidad de solución.

vi) Tamaño de lista tabú en búsqueda local: Inostroza-Ponta (2008) usa dos algoritmos (llamados *a* y *b* en la investigación), diferenciándose en los rangos de tamaño de las listas tabú usadas. Para esta investigación se consideran los parámetros de la versión *b*, dado que usa sólo un rango de variación de tamaño de lista tabú, el cual es $[0,1,0,5] \times MAX\_ITERACIONES$, el cual es usado en otras versiones del algoritmo que son explicadas más adelante. $MAX\_ITERACIONES$ es un parámetro que indica el máximo de iteraciones antes de terminar la búsqueda local, el cual fue dado anteriormente como 100.

3.2 ESQUEMAS DE MEMORIA SELECCIONADOS

Como se puede deducir de las clasificaciones de memoria presentadas en el capítulo de fundamentos teóricos, existen infinitas posibilidades de selección de esquemas, por lo cual se deber escoger algunos que se hayan utilizado con éxito previamente, o que presentan fundamentos bien establecidos.

A continuación se mencionan los esquemas seleccionados y usados en esta investigación, separados por su uso en tareas poblacionales como de búsqueda local.

3.2.1 Esquemas aplicados en tareas de población

Primero se dan a conocer los esquemas que utilizan estructuras de memoria para procedimientos particulares de la población usada por la metaheurística, como selección, cruzamiento y otros.

3.2.1.1 Población estructurada con búsqueda compartida

Corresponde a la estructura poblacional utilizada en el algoritmo en que se basa esta investigación, detallado previamente, proveniente del trabajo de Inostroza-Ponta (2008) y propuesta por Buriol et al. (2004), consistente en 13 agentes, organizados en 4 subpoblaciones jerárquicas traslapadas. Cada uno de los agentes posee una memoria de 5 soluciones que son usadas en diferentes etapas del algoritmo.

Entonces, corresponde clasificar cada una de las estructuras de memoria mostradas:

1. Jerarquía poblacional:
    - Duración: Largo plazo, la estructura perdura en la ejecución del algoritmo.
    - Explicitud: Implícita, los agentes conforman una población, no se ven como memoria propiamente tal y se usan para almacenar soluciones, cruzarlas, diversificar la exploración, etc.
    - Variabilidad: Tamaño fijo, 13 agentes de 5 soluciones cada uno.
    - Cobertura: Poblacional, se usa para tareas poblacionales, el almacenamiento de la población como tal.
2. Pockets:

- Duración: Largo plazo, los *pockets* no se pierden en la ejecución del algoritmo a pesar que vayan renovando las soluciones que almacenan.
- Explicitud: Implícita, se utilizan los bolsillos tanto para almacenar soluciones antiguas y poder usarlas más adelante, como para evitar repetirlas, diversificar los agentes, etc.
- Variabilidad: Tamaño fijo, cada agente tiene 5 bolsillos.
- Cobertura: Individual, los bolsillos son accesibles sólo por un agente.

3.2.1.2 Población estructurada sin búsqueda compartida

Es idéntico al esquema anteriormente mostrado, con la diferencia que no se utiliza la búsqueda compartida entre soluciones de un agente, ya que como se aplican más esquemas de memoria, está la posibilidad que se produzca una excesiva reducción del espacio de búsqueda, lo que debe ser evaluado. En el algoritmo 8 se puede ver que la principal diferencia se da en el uso de la matriz tabú no compartida en las líneas 7 y 8. Además, el tamaño de la lista tabú es dependiente del esquema de memoria de búsqueda local a utilizar. La estructura completa del algoritmo de búsqueda local se presenta en la siguiente sección.

Algoritmo 8 Estructura de búsqueda local no compartida

Entrada: *individuo*: solución a mejorar, $MAX\_ITERACIONES$: máximo de iteraciones sin cambios
Salida: *individuo*: solución mejorada

```
1:  iteración ← 0;
2:  sinCambios ← 0;
3:  mejorCosto ← Costo(individuo);
4:  mientras sinCambios < MAX_ITERACIONES hacer
5:     tamañoLista ← obtenerTamañoLista();
6:     (u, v) ← encontrarMejorMovimiento(iteración, individuo);
7:     listaTabu[u][individuo[u]] ← iteración + tamañoLista
8:     listaTabu[v][individuo[v]] ← iteración + tamañoLista
9:     individuo ← intercambiar(u, v, individuo);
10:    si mejorCosto > Costo(individuo) entonces
11:       mejorCosto ← Costo(individuo);
12:       sinCambios ← 0
13:    si no
14:       sinCambios ← sinCambios + 1
15:    fin si
16:    iteración ← iteración + 1
17: fin mientras
18: retornar individuo
```

3.2.1.3 Matriz de cruzamientos

El cruzamiento usado (*cycle crossover*, CX) posee la ventaja de no introducir mutaciones implícitas al realizar la recombinación de dos soluciones, con lo que se tiene un mayor control de los resultados obtenidos. El problema que presenta esta recombinación, es que del cruzamiento de dos padres se tiene una baja variedad de posibles soluciones generadas, determinado por los "ciclos" que existan entre ellos.

Como se detalla en el Apéndice B, al cruzar dos padres múltiples veces con CX sobre un 70 % de las veces se obtienen repeticiones de hijos y cerca de un 20 % genera hijos iguales a alguno de sus padres, debido a la ausencia de mutaciones implícitas. Es por esto que se ve la necesidad de establecer un esquema de memoria el cual recuerde, en el

proceso de selección y cruzamiento, cuales padres ya han sido elegidos, debido a la alta probabilidad que se repita una solución repetida. Para esto se usa una matriz booleana inicialmente en valores *falso*, la que se usa marcando como *verdadero* los pares de padres que se van cruzando, de manera de impedir su nuevo cruzamiento. En la figura 3-5 se puede apreciar como se marca el cruzamiento entre el padre $P1$ y $P2$ en las posiciones (1,2) y (2,1), con una matriz inicial falsa.

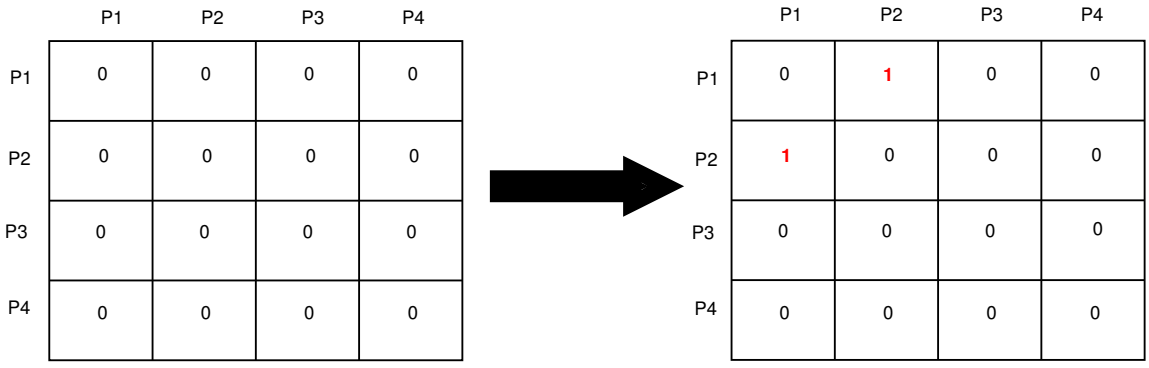

FIGURA 3-5: *Ejemplo de marcado de cruzamiento*

Cuando se realiza la selección de una pareja de padres, se revisa que no se hayan cruzado. Si ya se cruzaron, se repite la selección hasta que se escoja una pareja que no se haya cruzado previamente. De esta forma, la matriz de cruzamientos ayuda a generar una población más diversa, ya que al tener más diversidad en las parejas de padres, se tendrá más diversidad en los posibles hijos producto de la recombinación. Analizando la estructura de memoria descrita se cae en las siguientes clasificaciones:

- Duración: Corto plazo, la matriz de cruzamiento se usa sólo durante la selección/-cruzamiento, después se pierde.

- Explicitud: Explícita, se utiliza la matriz específicamente para cruzamientos realizados.
- Variabilidad: Tamaño fijo, siempre es de $P \times P$, con P tamaño de la población.
- Cobertura: Poblacional, se usa para la selección.

Además, cabe destacar que es incompatible con los dos esquemas de memoria previamente presentados, ya que en la política de reemplazo de ellos las nuevas soluciones van reemplazando a las antiguas en los agentes (cuando están completos), lo que hace que la matriz de cruzamientos se vuelva inútil. Es por esto que para este esquema poblacional se utiliza una población sin estructura jerárquica con los siguientes parámetros:

- El tamaño de población se definió estático en 40 soluciones. Esto surge de lo mencionado en Merz & Freisleben (2000a), donde indican que el tamaño ideal de población está entre 10 a 40 individuos.
- La búsqueda local depende del algoritmo en uso, se detalla más adelante.
- El tiempo máximo es parámetro al algoritmo. n y BKV (mejor valor conocido, *best known value*) dependen de la instancia en tratamiento.
- La cantidad de cruzamientos por generación es *tamañoPoblacion/5*. Esto es por la regla denominada *One-Fifth Rule*, mencionada en Luke (2009) la cual indica que si un quinto de los hijos obtenidos por búsqueda local son mejores que sus padres, entonces se está en una proporción correcta de trabajo de explotación y exploración.
- La selección en uso es por *ruleta*. Esta selección da una probabilidad de selección de un padre directamente proporcional a su calidad de solución, en comparación al total de individuos en la población como se detalla en Luke (2009).

- En la selección de padres se hace uso de la matriz de cruzamientos, como se muestra en el algoritmo 9. Allí, la variable *población*[] contiene los agentes que almacenan las soluciones encontradas por el algoritmo, de las cuales la mejor es almacenada en la variable $mejorSolución$. Además, la variable $padres$[][], corresponde a un arreglo de dimensión $2 \times cruzamientos$ el cual almacena los agentes seleccionados para el cruzamiento, el cual luego de ser aplicado genera agentes guardados en el arreglo $hijos$[]

- Para la actualización de la población, se utilizan los padres seleccionados como se muestra en la figura 9, luego se ordenan tanto los agentes en la población como los hijos de mayor a menor *fitness* (de menor a mayor costo) y se hace un *merge* de ambos arreglos, de manera de generar una población con una mayor diversidad en los costos y no sólo dejar los elementos de mejor calidad.

Algoritmo 9 Selección usando matriz de cruzamientos

Entrada: *población*: población de soluciones, $cruzamientos$: número de cruzamientos
Salida: $padres$: padres de la población seleccionados de a parejas

```
1:  matrizCruzamientos[][] ← booleano[tamaño(población)][tamaño(población)]
2:  para i = 1 → cruzamientos hacer
3:     padre1 ← seleccionRuleta(población)
4:     padres[i][1] ← población[padre1]
5:     noEncontrado ←verdadero
6:     mientras noEncontrado hacer
7:        padre2 ← seleccionRuleta(población)
8:        si matrizCruzamientos[padre1][padre2] = falso entonces
9:           noEncontrado ←falso
10:       fin si
11:    fin mientras
12:    padres[i][2] ← población[padre2]
13:    matrizCruzamientos[padre1][padre2] ←verdadero
14:    matrizCruzamientos[padre2][padre1] ←verdadero
15: fin para
16: retornar padres
```

- El criterio de reinicio es la repetición de la misma solución 30 iteraciones (obtenido de la investigación de Merz & Freisleben (2000a)) o la pérdida de diversidad en la población. Para determinar la pérdida de diversidad, se usó el criterio mostrado por Merz & Freisleben (2000a), el cual considera que la población converge si la distancia promedio de ella es igual o menor a 10, independientemente del tamaño de los individuos. Para ver esto se fueron comparando todos los individuos de la población contra la mejor solución, obteniendo una distancia promedio.

- El inicio de la población, cruzamiento y reinicio son iguales al algoritmo original.

También debe mencionarse que la misma población es una estructura de memoria no explicitada, ya que almacena soluciones que son usadas en el tiempo para llegar a otras soluciones. Las clasificaciones posibles de hacer corresponden a la siguientes:

- Duración: Largo plazo, la población dura toda la vida del algoritmo, aunque se reinicien sus individuos.

- Explicitud: Implícita, no se considera memoria como tal.

- Variabilidad: Tamaño fijo, en esta investigación se usa un tamaño fijo de 40 soluciones.

- Cobertura: Poblacional.

3.2.2 Esquemas aplicados en búsqueda local

Además de los esquemas enfocados a tareas de la población, se escogieron esquemas aplicados a la búsqueda local, los cuales son tanto individuales como poblacionales, dependiendo del contexto, y son explicados a continuación.

3.2.2.1 Lista tabú fija (LTF)

Corresponde a una adaptación de la memoria empleada por la Búsqueda Tabú clásica presentada por Glover (1989, 1990). La principal diferencia es que no se usa la memoria de largo plazo (*Long Term Memory*) del algoritmo mencionado, sino que sólo se utiliza la lista tabú para recordar las asignaciones de instalaciones a localizaciones.

Las asignaciones son recordadas por un número de iteraciones igual a $0.2 \times maxIter$, parámetro probado en Inostroza-Ponta (2008), donde no mostró buenos resultados; sin embargo, se tiene la hipótesis que será útil al usar otros esquemas de memoria. El recordar las asignaciones por esa cantidad de iteraciones permite que no se repitan nuevamente, a menos que se supere la calidad de la mejor solución encontrada. Las clasificaciones posibles de hacer corresponden a las siguientes:

- Duración: Corto plazo, la lista tabú se usa durante la búsqueda local, pero luego se pierde. Si se mezcla con el esquema de *población con búsqueda compartida* puede ser de largo plazo.
- Explicitud: Explícita, se utiliza la lista específicamente para recordar los movimientos realizados.
- Variabilidad: Tamaño fijo, la lista tabú no varía de tamaño.
- Cobertura: Individual, la lista es utilizada sólo por un individuo/agente.

3.2.2.2 Lista tabú variable (LTV)

Corresponde a la lista tabú proveniente de Búsqueda Tabú Robusta (*Robust Tabu Search*) presentada por Taillard (1991), con las modificaciones hechas en Inostroza-

Ponta (2008). El tamaño de la lista es variable y el rango de variación de tamaño es $[0.1, 0.5] \times MAX\_ITERACIONES$.

Las clasificaciones posibles de hacer corresponden a las siguientes:

- Duración: Corto plazo, la lista tabú se usa durante la búsqueda local, pero luego se pierde. Si se mezcla con el esquema de *población con búsqueda compartida* pasa a ser de largo plazo.
- Explicitud: Explícita, se utiliza la lista específicamente para recordar los movimientos realizados.
- Variabilidad: Tamaño variable, la lista tabú cambia de tamaño en cada iteración.
- Cobertura: Individual, la lista es utilizada sólo por un individuo/agente.

3.2.2.3 Reducción del espacio de búsqueda

Corresponde a una adaptación del esquema presentado por Merz & Freisleben (2000a). Cada individuo almacena las asignaciones comunes entre sus padres, las cuales no se permiten usar al momento inicial de hacer la búsqueda local, poniendo las asignaciones mencionadas como movimientos tabú por un cierto número de iteraciones. El objetivo de esto es reducir el espacio de búsqueda que recorrerá el algoritmo de búsqueda local inicialmente, quedando en su comienzo:

$$|N_{2opt}| = \frac{n(n-1)}{2} \longrightarrow |N| = \frac{d(d-1)}{2},\ d = d(\pi_1, \pi_2) \tag{3.1}$$

Donde $|N_{2opt}|$ corresponde al tamaño del espacio de búsqueda dada por el intercambio de dos elementos a partir de una permutación, y $d(\pi_1, \pi_2)$ es la distancia entre dos

permutaciones la cual se ve reflejada en los elementos diferentes de ellas en las mismas posiciones.

Las clasificaciones posibles de hacer corresponden a las siguientes:

- Duración: Corto plazo, se recuerda los puntos comunes entre los padres de una solución.
- Explicitud: Explícita, se utilizan los puntos recordados explícitamente para reducir el espacio de búsqueda.
- Variabilidad: Tamaño fijo, se recuerdan hasta *n* puntos comunes.
- Cobertura: Individual, los puntos comunes son recordados y usados sólo por un individuo.

3.2.2.4 Mejor vecino en Gradiente Descendiente (Steepest Descent, SD)

Corresponde al esquema de memoria usado en SD, un algoritmo goloso que avanza por los vecinos con mayor mejora de función de costo.

Usa una estructura de memoria simple: durante el recorrido entre los vecinos de una solución, se va almacenando el mejor encontrado, el cual si es de mejor calidad que la solución en revisión, la reemplaza para repetir el proceso. Estas repeticiones son realizadas hasta que no se encuentren vecinos de calidad superior a la solución, como muestra el algoritmo 10.

Algoritmo 10 Algoritmo de gradiente descendiente

Entrada: π, solución a mejorar.

```
1:  mejora ← verdadero ;
2:  mientras mejora = verdadero hacer
3:     mejorVecino ← ∅;
4:     para vecino_i ∈ vecindario_2opt(π) hacer
5:        si mejorVecino = ∅ ó Costo(vecino_i) < Costo(mejorVecino) entonces
6:           mejorVecino ← vecino_i;
7:        fin si
8:     fin para
9:     si Costo(mejorVecino) < Costo(π) entonces
10:       π ← mejorVecino
11:       mejora ← verdadero
12:    si no
13:       mejora ← falso
14:    fin si
15: fin mientras
16: retornar π
```

Este esquema por si solo no realiza una búsqueda local de buena calidad, por lo que se agrega como una mejora a la búsqueda local que se explica más adelante. El motivo de esta inclusión es que la mezcla de esquemas de memoria para la reducción del espacio de soluciones en la búsqueda local puede ocasionar una excesiva restricción de la vecindad de explotación. Por ello se optó por posibilitar la mejora por SD luego de aplicar la búsqueda tabú, similarmente a Misevičius (2006). Sin embargo el esquema presenta varias diferencias entre las que se tiene que los parámetros de tamaño de población, tamaño (o rango) de lista tabú, etc se mantienen constantes para todas las instancias que se evalúan (a excepción del tiempo máximo de ejecución).

Las clasificaciones posibles de hacer corresponden a las siguientes:

- Duración: Corto plazo, se recuerda el mejor vecino sólo durante la búsqueda local.
- Explicitud: Explícita, se recuerda el mejor vecino y se usa para avanzar en la búsqueda local.

- Variabilidad: Tamaño fijo, se recuerda sólo un vecino.
- Cobertura: Individual, se usa por sólo un individuo.

3.3 ALGORITMOS CREADOS

Los esquemas de memoria de los algoritmos que fueron presentados y formalizados, se detallan en esta sección. Primero, hay que revisar la compatibilidad entre los diferentes esquemas de memoria, presentados en la tabla 3.1, donde:

- PEBC: Población estructurada con búsqueda compartida.
- PE: Población estructurada sin búsqueda compartida.
- MC: Matriz de cruzamientos.
- LTV: Lista tabú fija.
- LTF: Lista tabú variable.
- R: Reducción de espacio de búsqueda.
- SD: *Steepest Descent*.

TABLA 3.1: *Compatibilidad de esquemas de memoria*

	PEBC	PE	MC	LTV	LTF	R	SD
PEBC	-	No	No	Sí	Sí	Sí	Sí
PE	-	-	No	Sí	Sí	Sí	Sí
MC	-	-	-	Sí	Sí	Sí	Sí
LTV	-	-	-	-	No	Sí	Sí
LTF	-	-	-	-	-	Sí	Sí
R	-	-	-	-	-	-	Sí
SD	-	-	-	-	-	-	-

La explicación de los esquemas incompatibles es la siguiente:

- PEBC y PE: no son compatibles debido a que PE corresponde a PEBC sin compartir los elementos reducidos en espacio de búsqueda *lista tabú*.
- MC y PEBC: no son compatibles pues el algoritmo original de PEBC Inostroza-Ponta (2008) realiza reemplazo directo de soluciones hijas sobre uno de sus padres, por lo que la matriz de cruzamientos se vuelve inútil.
- MC y PE: el mismo argumento anterior.
- LTF y LTV: son incompatibles debido a que uno mantiene la lista tabú de tamaño fijo y la otra variable en el tiempo.

En consecuencia de los algoritmos compatibles mostrados en la tabla 3.1, los algoritmos posibles de implementar quedan definidos por la tabla 3.2.

TABLA 3.2: *Algoritmos obtenidos por combinación de esquemas de memoria*

		PEBC	**PE**	**MC**
LTV	**-**	PEBC-LTV	PE-LTV	MC-LTV
	SD	PEBC-LTV-SD	PE-LTV-SD	MC-LTV-SD
	R	PEBC-LTV-R	PE-LTV-R	MC-LTV-R
	R-SD	PEBC-LTV-R-SD	PE-LTV-R-SD	MC-LTV-R-SD
LTF	**-**	PEBC-LTF	PE-LTF	MC-LTF
	SD	PEBC-LTF-SD	PE-SD	MC-SD
	R	PEBC-LTF-R	PE-R	MC-R
	R-SD	PEBC-LTF-R-SD	PE-R-SD	MC-R-SD

3.3.1 Estructura de algoritmos meméticos

Ahora que se sabe cuales son los algoritmos, es posible mostrar su estructura. En esta subsección se muestra la estructura de los algoritmos meméticos de forma global.

En el algoritmo 11 se muestra la estructura del algoritmo memético que usa matriz de cruzamientos. Los parámetros y operadores usados corresponden a los descritos en la sección anterior. En este algoritmo, la función *iniciarPoblación()* (línea 1) inicia la población de manera aleatoria. La función *búsquedaLocal()* (línea 6) realiza la optimización local de un agente, de la forma que se detalla más adelante. *seleccionarPadres()* (línea 10), entrega una matriz correspondiente a los pares de padres que se van a cruzar, seleccionados mediante *ruleta*. En *aplicarCruzamientos()* (línea 11) se cruzan los agentes obtenidos anteriormente, obteniendo un arreglo de agentes hijos. *mutación()* (línea 14) aplica mutación RPI a un agente una cantidad de veces indicada en el segundo parámetro. *actualizarPoblación()* mezcla los hijos obtenidos con la población actual. La función *pérdidaDiversidad()* (línea 29), determina si la distancia promedio entre los agentes de la población es menor a 10. *reiniciarPoblación()* reinicia los agentes de la población de manera aleatoria, exceptuando el mejor elemento.

Luego, se tiene el algoritmo 12, el cual muestra la estructura de los algoritmos que usan la población estructurada. Los parámetros y operadores son los mencionados al comienzo del presente capítulo. La optimización local que usa *búsquedaLocal()* (línea 4) es el descrito en el algoritmo 7. *cruzamientoInterno()* y *cruzamientoExterno()* corresponden a los descritos en los algoritmos 5 y 6 respectivamente.

Algoritmo 11 Estructura algoritmo memético que usa Matriz de Cruzamientos

Entrada: *tiempoMáximo*: tiempo máximo de ejecución, *tamañoPoblación*: tamaño de la población, *cruzamientos*: número de cruzamientos por cada generación, *n*: tamaño de la instancia, *BKV*: mejor costo de la instancia.

Salida: *mejorSolución*: mejor solución hallada.

```
1:  población ← iniciarPoblación(tamañoPoblación);
2:  mismaSolucion ← 0;
3:  padres ← Agente[cruzamientos][2];
4:  hijos ← Agente[cruzamientos];
5:  para i = 1 → tamañoPoblación hacer
6:      población[i] ← búsquedaLocal(población[i]);
7:  fin para
8:  mejorSolución ← obtenerMejorSolución(población);
9:  mientras tiempoTranscurrido < tiempoMáximo ó Costo(mejorSolución) > BKV
    hacer
10:     padres ← seleccionarPadres(población, cruzamientos);
11:     hijos ← aplicarCruzamientos(padres);
12:     para i = 1 → cruzamientos hacer
13:         si hijos[i] = padres[i][1] ó hijos[i] = padres[i][2] entonces
14:             hijos[i] ← mutación(hijos[i], n/3);
15:         si no
16:             hijos[i] ← mutación(hijos[i], 3);
17:         fin si
18:     fin para
19:     para i = 1 → cruzamientos hacer
20:         hijos[i] ← búsquedaLocal(hijos[i]);
21:     fin para
22:     población ← actualizarPoblación(población, hijos);
23:     si Costo(mejorSolución) > Costo(obtenerMejorSolución(población)) entonces
24:         mismaSolución ← 0;
25:         mejorSolución ← obtenerMejorSolución(población);
26:     si no
27:         mismaSolucion ← mismaSolución + 1
28:     fin si
29:     si mismaSolución = 30 ó pérdidaDiversidad(población) entonces
30:         población ← reiniciarPoblación(población);
31:         mejorSolución ← obtenerMejorSolución(población);
32:     fin si
33: fin mientras
34: retornar mejorSolución;
```

Algoritmo 12 *Cuerpo de algoritmo memético que usa población estructurada*

Entrada: *tiempoMáximo*: tiempo máximo de ejecución, n: tamaño de la instancia, BKV: mejor costo de la instancia

Salida: *mejorSolución*: mejor solución hallada

```
1:  población ← iniciarPoblación();
2:  mismaSolución ← 0;
3:  para i = 1 → 13 hacer
4:     población[1][i] ← búsquedaLocal(población[1][i], 1);
5:  fin para
6:  población ← actualizarPoblación(población);
7:  mejorSolución ← obtenermejorSolución(población);
8:  mientras tiempoTranscurrido < tiempoMáximo ó Costo(mejorSolución) > BKV
    hacer
9:     para k = 1 → 4 hacer
10:       si pérdidaDiversidad(población, k) entonces
11:          población ← cruzamientoExterno(población,k);
12:       si no
13:          población ← cruzamientoInterno(población,k);
14:       fin si
15:       para j = 1 → 3 hacer
16:          población[k][j] ← búsquedaLocal(población[k][j], k);
17:       fin para
18:    fin para
19:    población ← actualizarPoblación(población);
20:    si Costo(mejorSolución) > Costo(obtenermejorSolución(población)) entonces
21:       mismaSolución ← 1;
22:       mejorSolución ← obtenermejorSolución(población);
23:    si no
24:       mismaSolución ← mismaSolución + 1;
25:    fin si
26:    si mismaSolución > n/4 entonces
27:       población ← reiniciarPoblación(población);
28:       mejorSolución ← obtenermejorSolución(población);
29:    fin si
30: fin mientras
31: retornar mejorSolución
```

3.3.2 Estructura de la búsqueda local

En la presente subsección se muestra la estructura asociada a la búsqueda local. El algoritmo 13 muestra como se cambia el tamaño de la lista, el cual es usado en el algoritmo 14.

La búsqueda local considera los esquemas de memoria presentados previamente: tiene tamaño de lista estático o variable dependiente de la variable booleana LTV, aplica reducción de espacio de búsqueda dependiendo de la variable booleana *reducirEspacioBusqueda* y aplica Gradiente Descendiente dependiente de la variable *aplicarGradienteDescendiente*. Además, considera ambos tipos de búsqueda, posibilitando el uso de la matriz compartida, dependiendo de la existencia de la variable *lista*.

En el algoritmo 14, la función *cambiarTamañoLista()* (línea 3) varía el tamaño de la lista pasada por parámetro en el rango establecido anteriormente. *ponerTabu()* (línea 6) pone par de elementos de un individuo en lista tabú. En la línea 7 se agrega el parámetro correspondiente al número de la lista tabú en uso. Los elementos puestos son extraídos por la función *puntosComunesPadres()*, que saca los elementos comunes entre los padres de un individuo. *encontrarMejorMovimiento()* (línea 13) busca en los vecinos opt_2 que no estén en tabú y elige el mejor. La función *gradienteDescendiente()* realiza el algoritmo 10.

Algoritmo 13 Algoritmo de cambio de tamaño

Entrada: LTV: booleano que marca si se usa LTV o LTF, $MAX\_ITERACIONES$: máximo de iteraciones.
Salida: *tamañoLista*: tamaño de la lista tabú.

1: **si** LTV =**verdadero entonces**
2: $\quad tamañoLista \leftarrow \text{azar}(0.1, 0.5) \times MAX\_ITERACIONES$;
3: **si no**
4: $\quad tamañoLista \leftarrow 0.2 \times MAX\_ITERACIONES$;
5: **fin si**
6: **retornar** *tamañoLista*;

Algoritmo 14 Estructura de búsqueda local

Entrada: *individuo*: solución a mejorar, *MAX_ITERACIONES*: máximo de iteraciones sin cambios, *LTV*: booleano que marca si se usa LTV o LTF, *reducirEspacioBusqueda*: booleano que indica si debe reducirse el espacio de búsqueda, *aplicarGradienteDescendiente*: booleano que indica si debe aplicarse Gradiente Descendiente, *lista*: lista tabú a emplear.

Salida: *individuo*: solución mejorada.

```
1: iteración ← 0;
2: sinCambios ← 0;
3: tamañoLista ← cambiarTamañoLista(LTV, MAX_ITERACIONES);
4: si reducirEspacioBusqueda = verdadero entonces
5:    si lista = NULO entonces
6:       ponerTabu(listaTabu, tamañoLista, puntosComunesPadres(individuo));
7:    si no
8:       ponerTabu(listaTabu, tamañoLista, lista, puntosComunesPadres(individuo));
9:    fin si
10: fin si
11: mejorCosto ← Costo(individuo);
12: mientras sinCambios < MAX_ITERACIONES hacer
13:    (u, v) ← encontrarMejorMovimiento(iteración, individuo);
14:    si lista = NULO entonces
15:       listaTabu[u][individuo[u]] ← iteración + tamañoLista;
16:       listaTabu[v][individuo[v]] ← iteración + tamañoLista;
17:    si no
18:       listaTabu[lista][u][individuo[u]] ← iteración + tamañoLista;
19:       listaTabu[lista][v][individuo[v]] ← iteración + tamañoLista;
20:    fin si
21:    individuo ← intercambiar(u, v, individuo);
22:    si mejorCosto > Costo(individuo) entonces
23:       mejorCosto ← Costo(individuo);
24:       sinCambios ← 0;
25:    si no
26:       sinCambios ← sinCambios + 1;
27:    fin si
28:    tamañoLista ← cambiarTamañoLista(LTV, MAX_ITERACIONES);
29:    iteración ← iteración + 1;
30: fin mientras
31: si aplicarGradienteDescendiente =verdadero entonces
32:    individuo ← gradienteDescendiente(individuo);
33: fin si
34: retornar individuo;
```

3.4 CRITERIOS DE EVALUACIÓN

En la presente sección se dan los criterios para evaluar y comparar los algoritmos presentados. Según menciona Talbi (2009), para realizar el ánalisis de una metaheurística deben considerarse 3 pasos:

i) Diseño de experimentos: Se definen las metas de los experimentos, las instancias seleccionadas y otros factores de importancia.

ii) Medidas: Se seleccionan las medidas a computar y análisis estadísticos a aplicar sobre los datos obtenidos.

iii) Reporte: Se presentan los resultados de forma comprensible y se analizan de acuerdo a los objetivos del trabajo.

3.4.1 Diseño de experimentos

Los experimentos tienen por objetivo determinar la mejora de rendimiento del algoritmo memético dado por Inostroza-Ponta (2008), el cual utiliza los esquemas de memoria:

- Población estructurada con búsqueda compartida.
- Lista tabú variable.

En dicha investigación, aparecen dos versiones del algoritmo memético, variando tan sólo en el rango en que se mueve el tamaño de la lista tabú. Para la presente investigación se trabaja con la versión MA_b (con el tamaño de lista tabú variable en el rango presentado en la sección anterior), ya que es la que presenta más instancias en las cuales muestra su

superioridad en la calidad de soluciones con respecto a los algoritmos contra los cuales compite.

Las instancias a evaluar son las mismas presentadas en Inostroza-Ponta (2008). Sin embargo, algunos tiempos de ejecución son variados porque en ese trabajo se realizan dos tipos de experimentos: unos de corta duración con 30 ejecuciones por instancia y otros de larga duración con 10 ejecuciones por instancia. Para esta investigación se trabaja con pruebas de larga duración, de manera que los tiempos de las pruebas cortas realizadas en la investigación mencionada no son los mismos.

Para hacer equitativa la comparación, tanto el algoritmo base (MA_b) como los obtenidos con otros esquemas de memoria son ejecutados con iguales tiempos máximos (dependientes de la instancia). Las instancias a probar y sus tiempos máximos de ejecución son los dados en la tabla 3.3. Cada instancia es enfrentada 10 veces por cada algoritmo.

TABLA 3.3: *Tiempo máximo de ejecución por instancia*

Instancia	Tiempo máximo [segundos]
chr25a	40
kra30a	76
nug30	76
ste36a	30
sko49	415
sko56	639
sko64	974
sko72	1415
sko81	2041
sko90	2825
sko100a	4000

Instancia	Tiempo máximo [segundos]
tai20a	26
tai25a	50
tai30a	87
tai35a	145
tai35b	147
tai40a	224
tai40b	240
tai50a	467
tai50b	480
tai60a	820
tai60b	855

Instancia	Tiempo máximo [segundos]
tai80a	2045
tai80b	2073
tai100a	4000
tai100b	4000
tho150b	5000
wil50	467
tai256c	6000
tai150b	5000
Total	**44657**

3.4.1.1 Instancias QAP

Las instancias QAP utilizadas fueron extraídas de QAPLIB[1] y se eligieron porque son las utilizadas en Inostroza-Ponta (2008). A continuación se dan datos de ellas:

- La instancia *chr25a* corresponde a los datos dados por una matriz de adyacencia de un árbol con pesos y la otra de un grafo completo. La solución disponible a comparar es la óptima.

- La instancia *kra30a* es un problema de la vida real cuyos datos fueron dados al construir la clínica Regensburg en Alemania. La solución disponible es la óptima.

- La instancia *nug30* contiene en su matriz de distancias las distancias manhattan en grillas rectangulares. La solución disponible es la óptima.

- La instancia *ste36a* contiene datos en distancia manhattan de tableros de cableado, multiplicadas por mil. Se tiene la solución óptima.

- Las instancias *sko** fueron generadas aleatoriamente. Se tienen mínimos obtenidos por límite inferior (*lower bound*) y metaheurísticas, pero no el óptimo.

- Las instancias *tai*a* son simétricas y las *tai*b* asimétricas, ambas generadas aleatoriamente. Las instancias *tai*c* son obtenidas en la generación de patrones grises. Para $n \leq 25$ se tienen las soluciones óptimas, pero para $n > 25$ se tiene sólo los valores dados por límite inferior y resultados de metaheurísticas.

- La instancia *tho150* tiene distancias rectangulares. No se tiene el óptimo, sólo mínimos de metaheurísticas y límite inferior.

[1] `http://www.seas.upenn.edu/qaplib/`

- La instancia *wil50* también tiene sus distancias rectangulares. No está el óptimo, sólo mínimos dados por metaheurísticas y límite inferior.

3.4.1.2 Implementación y condiciones de experimentación

Los experimentos contemplan la implementación de los algoritmos descritos previamente en Java 6, para ser ejecutados en una máquina con las siguientes características:

- Procesador: AMD Opteron(tm) Processor 6128, 16 núcleos, 2 Ghz cada uno, caché de 512 KB.
- RAM: 16 GB.

Es importante destacar que el código utiliza una sola hebra para la ejecución de cada algoritmo, por lo que los resultados obtenidos no son producto de alguna paralelización en ellos.

3.4.2 Medidas

Dado que se quiere analizar la convergencia de los algoritmos meméticos obtenidos, se debe considerar la calidad de la mejor solución encontrada por cada uno de ellos en cada instancia. Además, un factor importante en esto es el tiempo que toma llegar a estas soluciones. En la figura 3-6 se puede ver un ejemplo de convergencia, mostrando que las dos características que la determinan es la calidad (porcentaje de error) y el instante en que se encuentra.

Cabe mencionar que entre la calidad de la mejor solución y el tiempo en que ella fue encontrada, el primer factor es más importante, ya que el tiempo de convergencia de un

algoritmo puede ser mejorado por técnicas de programación como el paralelismo o con máquinas más rápidas de las usadas para los experimentos. Otro punto que favorece a la calidad de solución frente al tiempo de convergencia es que cuando ocurre el fenómeno de *convergencia prematura* el tiempo a pesar de ser bajo, ello no implica que el algoritmo sea de mejor calidad.

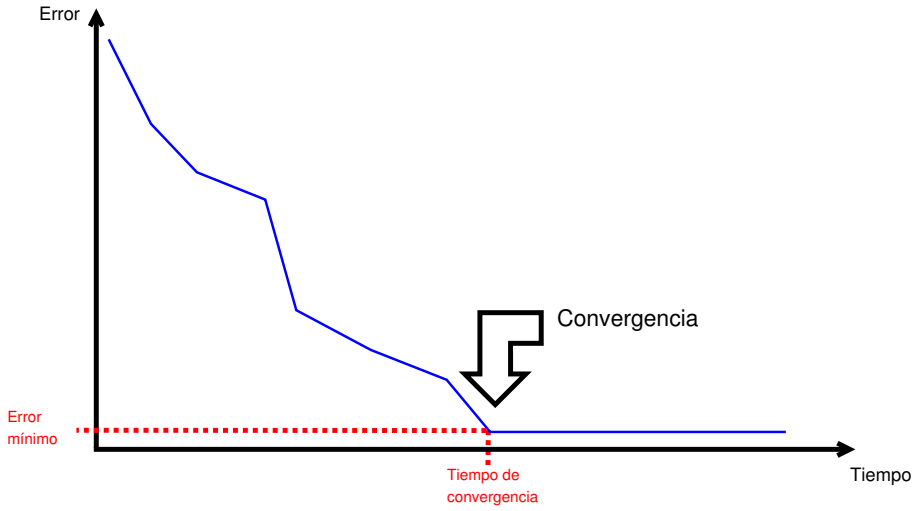

FIGURA 3-6: *Punto de convergencia*

La calidad de soluciones en esta investigación se da por el porcentaje de error de la mejor solución que encuentre cada algoritmo, comparado al valor *óptimo* o al *mejor valor encontrado* (*Best Known Value*, BKV) dado en QAPLIB. Esta calidad se obtendrá del promedio de 10 ejecuciones.

El tiempo de convergencia de un algoritmo se da por el intervalo que éste tarda en encontrar la mejor solución. Todas las implementaciones se ejecutan en la misma máquina y bajo las mismas condiciones. El tiempo se obtiene del promedio de 10 ejecuciones.

Un análisis estadístico a efectuar sobre los algoritmos es la *tasa de éxito* que tienen en el enfrentamiento de las instancias Talbi (2009). Esta tasa corresponde al número de ejecuciones exitosas (ejecuciones que alcanzan el mínimo encontrado por ese algoritmo)

sobre el número de pruebas realizadas.

$$Tasa\,éxito = \frac{Nro.\,ejecuciones\,exitosas}{Nro.\,ejecuciones} \tag{3.2}$$

3.4.3 Reporte

En el reporte se muestran los resultados obtenidos por las métricas expuestas anteriormente y además, se hacen análisis ordinales, de manera de determinar en la discusión qué algoritmo es mejor que otro y comparar los esquemas de memoria propuestos en esta investigación.

En Talbi (2009) sugieren hacer un *ranking* de los algoritmos de acuerdo a los índices obtenidos por el método de Copeland. Allí definen el índice Copeland de la siguiente forma:

> El índice Copeland σ es el número de veces que un método derrota otros métodos menos el número de veces que ese método pierde contra otros métodos, cuando éstos son comparados de a parejas. Por ejemplo, sea m y m' dos metaheurísticas y $c_{mm'}$ el número de veces que la metaheurística m derrota a m'. El índice Copeland para la metaheurística m será:

$$\sigma_m = \sum_{m'} c_{mm'} - c_{m'm} \tag{3.3}$$

De acuerdo a esta definición, los algoritmos que tienen un índice Copeland más alto son los mejores y viceversa.

En el reporte se incluye los índices Copeland para los algoritmos individualmente y agrupados por esquemas de memoria. Esto quiere decir:

- Para obtener los índices de algoritmos de forma individual se realiza la comparación de a pares para cada algoritmo en cada instancia contra todos los demás, y de ahí se realiza la sumatoria de los índices parciales.

- Para obtener los índices de algoritmos de forma grupal se realiza la comparación de a pares (en cada instancia) sólo contra los algoritmos que son diferentes en los esquemas comparados, y luego se suman los índices de los algoritmos que pertenecen al mismo grupo. Por ejemplo, si se está comparando LTF contra LTV, para el cálculo de índice Copeland los algoritmos que usan LTF se comparan sólo contra los que usan LTV (no entre ellos) y luego se suman los índices parciales, es decir se compara PE-LTV con PEBC-LTF-SD pero no con PEBC-LTV-SD.

Además de los índices Copeland, se aplica el denominado Test de Wilcoxon el cual es usado para determinar si dos poblaciones son diferentes (en el caso de este trabajo, se compara el rendimiento de algoritmos).

Derrac et al. (2011) describen el test de Wilcoxon de la siguiente forma:

> Sea d_i la diferencia entre el desempeño de dos algoritmos en el i-ésimo de n problemas. Las diferencias son ordenadas en un ranking de acuerdo a sus valores absolutos; en caso de empates se puede hacer varias técnicas (ignorar empates, asignar ranking más alto, asignar el ranking promedio, etc). Sea R^+ la suma de los rankings de los problemas en los cuales el primer algoritmo superó el segundo, y R^- la suma de los rankings para el caso contrario, quedando:

$$R^+ = \sum_{d_i>0} rank(d_i)$$

$$R^- = \sum_{d_i<0} rank(d_i)$$

Sea T el menor valor entre R^+ y $R-$, $T = min(R^+, R^-)$. Si T es menor o igual que el valor de la distribución de Wilcoxon para n grados de libertad, la hipótesis nula es rechazada, indicando que un algoritmo efectivamente supera al otro, con un $p - value$ asociado.

En la la presente investigación, el Índice Copeland es usado para realizar un *ranking* de los algoritmos y determinar superioridad entre ellos agrupados por esquemas de memoria o de forma individual. El Test de Wilcoxon es usado para determinar con qué certeza se diferencian los resultados de los algoritmos, mediante sus *p-values*. Como indican Derrac et al. (2011), el *p-value* es la probabilidad de obtener un resultado tan extremo como el observado, suponiendo la hipótesis nula H_0 como verdadera, por lo que mientras más pequeño el valor del *p-value*, más fuerte es la evidencia contra H_0. Para la realización de este test, se tiene que H_0 corresponde a la igualdad del rendimiento de los algoritmos.

Se puede concluir este capítulo indicando que se revisaron en profundidad los aspectos relacionados con la solución del problema, correspondiente a los algoritmos creados bajo la combinación de los diferentes esquemas de memoria que también fueron detallados. Además se dejaron establecidas las métricas que se muestran en el siguiente capítulo y que son aplicadas para hacer una discusión en el capítulo correspondiente a ello.

CAPÍTULO 4. RESULTADOS

En el presente capítulo se exponen los resultados obtenidos en los experimentos realizados. Cada uno de los 24 algoritmos (23 variaciones y el original), fueron ejecutados en 30 instancias QAP. Los resultados son expuestos en calidad de soluciones a través de su error al BKV, tiempo de convergencia en segundos y tasa de éxito. Además, se dan los índices Copeland para cada una de éstas métricas. Al final del capítulo se presentan los *p-values* obtenidos de la aplicación del Test de Wilcoxon a cada pareja de algoritmos.

Para todos los experimentos, se usan los parámetros dados por la tabla 4.1. La columna instancia contiene el nombre de ella, *n* corresponde al tamaño (cantidad de localizaciones e instalaciones), el tiempo máximo en segundos que puede correr el algoritmo si es que no llega al mejor valor encontrado, el cual está dado por la columna BKV (obtenido desde QAPLIB[1]).

TABLA 4.1: *Parámetros experimentos*

Instancia	**n**	**Tiempo máximo [segundos]**	**BKV**
chr25a	25	40	3796
kra30a	30	76	88900
nug30	30	76	6124
ste36a	36	30	9526
sko49	49	415	23386
sko56	56	639	34458
sko64	64	974	48498
sko72	72	1415	66256
sko81	81	2041	90998
sko90	90	2825	115534
sko100a	100	4000	152002
tai20a	20	26	703482
tai25a	25	50	1167256
tai30a	30	87	1818146
tai35a	35	145	2422002

Instancia	**n**	**Tiempo máximo [segundos]**	**BKV**
tai35b	35	147	283315445
tai40a	40	224	3139370
tai40b	40	240	637250948
tai50a	50	467	4938796
tai50b	50	480	458821517
tai60a	60	820	7205962
tai60b	60	855	608215054
tai80a	80	2045	13515450
tai80b	80	2073	818415043
tai100a	100	4000	21052466
tai100b	100	4000	1185996137
tho150b	150	5000	498896643
wil50	50	467	48816
tai256c	256	6000	44759294
tai150b	150	5000	498896643

[1] `http://www.seas.upenn.edu/qaplib/`

4.1 CALIDAD DE SOLUCIONES

Dado que se realizaron nuevamente las pruebas para el algoritmo propuesto por Inostroza-Ponta (2008), cabe mencionar que la calidad obtenida es muy cercana (y en la mayoría de los casos, de mejor costo) que la mostrada en esa investigación, de manera de hacer transparente y justa la comparación. En la tabla 4.2 se puede ver el resultado obtenido por Inostroza-Ponta y los nuevos valores con mayores tiempos de ejecución. Gran parte de los nuevos resultados son mejores que los publicados debido a que los nuevos tiempos de ejecución son iguales o mayores a los antiguos.

TABLA 4.2: *Porcentajes de error promedio publicados versus nuevos*

Instancia	**Nuevos**	**Inostroza-Ponta**	**Instancia**	**Nuevos**	**Inostroza-Ponta**
chr25a	**0,000**	**0,000**	tai35b	**0,000**	**0,000**
kra30a	**0,000**	**0,000**	tai40a	**0,290**	0,298
nug30	**0,000**	**0,000**	tai40b	**0,000**	**0,000**
ste36a	**0,000**	**0,000**	tai50a	**0,495**	0,743
sko49	**0,000**	**0,000**	tai50b	**0,000**	**0,000**
sko56	0,002	**0,000**	tai60a	**0,661**	**0,661**
sko64	**0,000**	**0,000**	tai60b	**0,000**	**0,000**
sko72	0,006	**0,003**	tai80a	0,782	**0,688**
sko81	**0,007**	0,014	tai80b	0,133	**0,082**
sko90	0,025	**0,019**	tai100a	**0,727**	0,761
sko100a	**0,041**	0,059	tai100b	**0,117**	0,162
tai20a	**0,000**	**0,000**	tho150b	**0,101**	0,148
tai25a	**0,000**	**0,000**	wil50	**0,000**	**0,000**
tai30a	**0,000**	**0,000**	tai256c	**0,093**	0,125
tai35a	0,051	**0,016**	tai150b	**0,534**	0,652

En las tablas 4.3 - 4.5 se detalla la calidad promedio obtenida por todos los algoritmos en todas las instancias. Por términos de mejora visual, se muestran agrupados por el esquema poblacional en uso.

Recordar que LTV es *lista tabú variable*, LTF es *lista tabú fija* y sus modificadores *R* y *SD* indican *reducción de espacio de búsqueda* y *Steepest Descent* (mejora por Gradiente Descendiente), respectivamente.

Cada una de las tablas indicadas anteriormente muestra el porcentaje de error promedio de 10 ejecuciones del algoritmo correspondiente sobre la instancia dada en la fila de la tabla. Los algoritmos se dan en las columnas de la tabla, cuyos nombres son formados por el esquema de memoria poblacional que utilizan (PEBC, PE o MC) y los esquemas de memoria usados en búsqueda local (LTF, LTV, SD, R) separados por guiones. El porcentaje de error para cada ejecución se calcula de la siguiente forma:

$$Porcentaje\,error = \frac{Costo_{obtenido} - BKV}{BKV} \times 100\,\% \tag{4.1}$$

Donde BKV es el mejor valor conocido para la instancia en enfrentamiento. Ahora, para calcular el error promedio se usa la siguiente fórmula:

$$Error_{promedio} = \frac{\sum_{i=1}^{Nro.\,ejecuciones} Porcentaje\,error_i}{Nro.\,ejecuciones} \tag{4.2}$$

Notar que en las tablas 4.3 - 4.5 los valores en negrita son los porcentajes de error promedio más bajos efectuados por los algoritmos de las tres tablas para esa instancia.

En la tabla 4.3 se muestra la calidad de los algoritmos que usan el esquema de memoria de matriz de cruzamientos (MC), con sus diversas variedades de esquemas aplicados a la búsqueda local. En las tablas 4.4 y 4.5 se muestra la calidad de los algoritmos que usan población estructurada sin búsqueda compartida (PE) y con búsqueda compartida (PEBC) respectivamente.

TABLA 4.3: *Calidad esquemas con MC en porcentaje de error*

	MC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
kra30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
nug30	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
ste36a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko49	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko56	**0,000**	0,001	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko64	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko72	**0,000**	**0,000**	**0,000**	0,001	0,001	**0,000**	**0,000**	**0,000**
sko81	0,025	0,015	0,011	0,012	0,016	0,015	0,015	0,018
sko90	0,010	0,007	0,006	0,013	0,009	0,011	**0,003**	0,012
sko100a	0,031	0,036	0,026	0,030	0,021	0,034	0,019	0,027
tai20a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai35a	**0,000**	0,010	**0,000**	**0,000**	**0,000**	**0,000**	0,016	**0,000**
tai35b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai40a	0,209	0,189	0,241	0,262	0,195	**0,181**	0,234	0,193
tai40b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai50a	0,550	0,543	0,536	0,586	0,568	0,560	0,549	0,526
tai50b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai60a	0,632	0,652	0,593	0,625	**0,503**	0,649	0,615	0,564
tai60b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	0,005
tai80a	0,776	0,802	0,839	0,780	0,793	0,729	0,716	0,745
tai80b	0,118	0,004	0,010	0,003	0,008	0,200	**0,000**	0,016
tai100a	0,713	0,691	**0,638**	0,679	0,643	0,692	0,681	0,744
tai100b	0,050	0,036	**0,014**	0,032	0,091	0,081	0,057	0,086
tho150b	0,110	0,108	**0,078**	0,079	0,112	0,105	0,103	0,113
wil50	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai256c	0,150	0,152	0,158	0,155	0,152	0,137	0,132	0,141
tai150b	0,506	0,404	0,382	0,450	0,425	0,406	0,458	0,360
Promedio	0,129	0,122	**0,118**	0,124	**0,118**	0,127	0,120	**0,118**

TABLA 4.4: *Calidad esquemas con PE en porcentaje de error*

	PE							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
kra30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
nug30	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
ste36a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko49	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko56	0,001	0,001	0,001	0,001	**0,000**	**0,000**	0,002	**0,000**
sko64	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko72	0,013	0,013	0,006	0,013	0,019	0,006	0,002	0,006
sko81	0,015	0,017	0,017	0,009	0,010	**0,007**	0,017	0,014
sko90	0,021	0,016	0,004	0,028	0,025	0,007	0,013	0,016
sko100a	0,051	0,030	0,041	0,031	0,037	0,021	0,033	0,029
tai20a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai30a	**0,000**	**0,000**	**0,000**	**0,000**	0,002	**0,000**	**0,000**	**0,000**
tai35a	0,060	0,042	0,017	0,067	0,034	0,072	0,037	0,026
tai35b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai40a	0,302	0,252	0,315	0,302	0,310	0,260	0,277	0,238
tai40b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai50a	0,519	0,558	0,621	0,610	0,635	0,594	0,605	0,550
tai50b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai60a	0,632	0,679	0,697	0,740	0,675	0,703	0,645	0,732
tai60b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai80a	0,818	0,849	0,783	0,864	0,770	0,770	0,766	0,779
tai80b	0,131	0,061	0,059	0,029	0,064	0,076	0,103	0,059
tai100a	0,640	0,711	0,723	0,735	0,738	0,687	0,783	0,759
tai100b	0,049	0,124	0,072	0,104	0,078	0,056	0,129	0,096
tho150b	0,091	0,094	0,082	0,080	0,087	0,080	0,079	0,101
wil50	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	0,002
tai256c	0,127	0,124	0,107	0,124	0,112	0,102	0,114	0,108
tai150b	0,527	0,58	0,479	0,578	0,513	0,477	0,585	0,613
Promedio	0,133	0,138	0,134	0,144	0,137	0,131	0,140	0,138

TABLA 4.5: *Calidad esquemas con PEBC en porcentaje de error*

	PEBC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
kra30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
nug30	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
ste36a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	0,010	**0,000**
sko49	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko56	0,002	**0,000**	0,001	0,002	**0,000**	**0,000**	**0,000**	**0,000**
sko64	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko72	0,006	**0,000**	**0,000**	0,007	0,014	0,006	**0,000**	0,006
sko81	**0,007**	0,016	0,010	0,013	0,016	0,024	0,020	0,019
sko90	0,025	**0,003**	0,018	0,004	0,018	0,010	0,015	0,008
sko100a	0,041	0,040	0,024	0,030	0,037	0,029	**0,015**	0,030
tai20a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai35a	0,051	0,017	0,040	0,019	0,034	0,041	0,053	0,045
tai35b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai40a	0,290	0,236	0,307	0,262	0,244	0,308	0,331	0,352
tai40b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai50a	**0,495**	0,660	0,565	0,620	0,683	0,636	0,625	0,634
tai50b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai60a	0,661	0,698	0,675	0,660	0,694	0,695	0,659	0,688
tai60b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai80a	0,782	0,901	0,840	0,733	0,820	0,817	0,778	0,846
tai80b	0,133	0,059	0,124	0,015	0,113	0,106	0,010	0,005
tai100a	0,727	0,770	0,717	0,710	0,703	0,674	0,741	0,677
tai100b	0,117	0,092	0,133	0,071	0,120	0,108	0,177	0,060
tho150b	0,101	0,147	0,082	0,130	0,110	0,083	0,140	0,094
wil50	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai256c	0,093	0,105	0,111	0,116	0,102	0,100	0,111	0,105
tai150b	0,534	0,47	0,667	0,592	0,555	0,569	0,497	0,608
Promedio	0,136	0,140	0,144	0,133	0,142	0,140	0,139	0,139

En la tabla 4.6, la cual resume todas las anteriores indicando:

- (A): Número de instancias en que el error es: $error = 0\,\%$.
- (B): Número de instancias en que el error es: $minimo < error \leq m\acute{i}nimo + 0,005\,\%$. Donde $m\acute{i}nimo$ corresponde al porcentaje de error más bajo para una instancia obtenido por uno de los algoritmos ejecutados.

- (C): Número de instancias en que el error es: $error = minimo; minimo \neq 0\,\%$

TABLA 4.6: *Resumen calidad de algoritmos en porcentaje de error*

		(A)	(B)	(C)	(A) + (B) + (C)	Error Promedio
MC	**LTV**	17	0	0	17	0,129
	LTV-R	15	3	1	19	0,122
	LTV-SD	17	3	3	**23**	**0,118**
	LTV-R-SD	16	4	0	20	0,124
	LTF	16	2	1	19	**0,118**
	LTF-R	17	0	1	18	0,127
	LTF-SD	17	1	1	19	0,120
	LTF-R-SD	16	1	1	18	**0,118**
PE	**LTV**	14	2	0	16	0,133
	LTV-R	14	1	0	15	0,138
	LTV-SD	14	3	0	17	0,134
	LTV-R-SD	14	3	0	17	0,144
	LTF	14	2	0	16	0,137
	LTF-R	15	1	1	17	0,131
	LTF-SD	14	3	0	17	0,140
	LTF-R-SD	14	1	0	15	0,138
PEBC	**LTV**	14	1	2	17	0,136
	LTV-R	16	0	1	17	0,140
	LTV-SD	15	3	0	18	0,144
	LTV-R-SD	14	2	0	16	0,133
	LTF	15	0	0	15	0,142
	LTF-R	15	1	0	16	0,140
	LTF-SD	15	0	1	16	0,139
	LTF-R-SD	15	2	0	17	0,139

4.2 TIEMPO CONVERGENCIA

En esta sección se presentan los tiempos de convergencia promedio de cada uno de los algoritmos para cada una de las instancias probadas, junto con el tiempo total de todas las instancias para cada algoritmo. Los resultados se dan en segundos y los tiempos máximos son los vistos en la tabla 4.1.

TABLA 4.7: *Tiempo de convergencia en segundos con MC*

	MC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	2,55	3,45	1,90	2,99	3,84	5,05	4,79	6,35
kra30a	0,66	1,01	0,69	0,78	1,54	0,78	1,05	1,50
nug30	0,67	1,29	1,59	0,90	0,90	1,38	0,75	1,19
ste36a	3,79	6,21	4,89	7,21	8,01	9,53	6,50	10,42
sko49	42,76	69,29	63,96	47,21	31,59	50,54	58,93	47,36
sko56	114,00	91,52	90,05	74,36	96,82	54,92	177,42	115,18
sko64	81,62	54,50	69,55	93,99	63,50	107,43	58,99	83,06
sko72	484,96	487,68	525,66	428,06	578,28	334,59	263,57	542,23
sko81	715,47	530,99	1.249,03	745,75	651,72	842,89	533,90	575,66
sko90	1.416,73	944,71	1.416,30	946,02	1.287,42	803,86	1.008,19	1.203,52
sko100a	1.622,20	955,32	1.327,02	1.219,43	1.164,06	1.177,51	2.022,01	904,38
tai20a	1,17	0,67	0,93	0,80	0,69	1,39	0,64	1,09
tai25a	1,78	2,34	3,54	2,07	2,91	3,33	2,10	2,86
tai30a	3,13	9,03	5,46	5,67	4,12	7,80	8,38	7,37
tai35a	46,95	36,05	43,31	24,82	26,76	35,81	23,60	23,25
tai35b	9,12	5,24	14,81	8,00	11,36	8,90	10,08	11,37
tai40a	96,03	86,96	104,20	116,25	131,38	126,78	111,44	102,02
tai40b	11,99	8,71	8,62	8,79	10,60	5,89	8,57	12,89
tai50a	268,31	222,94	197,83	207,09	281,43	194,83	172,33	246,99
tai50b	95,10	87,22	70,13	85,46	86,20	69,11	66,77	88,93
tai60a	419,42	411,89	359,26	366,05	343,17	367,51	446,33	259,03
tai60b	130,49	262,66	235,89	281,00	196,46	235,14	187,62	203,14
tai80a	1.016,50	891,67	1.093,28	936,03	963,73	803,27	1.095,59	965,49
tai80b	1.186,85	1.229,55	972,35	780,23	980,65	908,12	1.110,59	1.124,92
tai100a	1.726,09	1.363,75	1.756,49	1.617,56	2.065,33	2.308,28	1.661,27	1307,81
tai100b	2.065,50	1.432,43	1.274,43	1.540,79	2.251,09	2.051,45	1.580,30	1.778,16
tho150b	3.134,37	2.490,60	3.191,32	3.957,04	2.846,29	2.974,18	3.331,18	2.806,59
wil50	81,37	26,96	67,34	29,95	68,10	79,64	49,89	42,78
tai256c	3.776,33	3.930,27	3.536,60	3.457,39	4.125,05	4.389,86	3.543,87	3.056,92
tai150b	3.367,05	3.917,85	3.425,14	3.788,50	3.746,12	4.377,24	3.694,64	3.718,17
Total	21.922,95	19.562,70	21.111,55	20.780,20	22.029,11	22.336,97	21.241,28	19.250,62

TABLA 4.8: *Tiempo de convergencia en segundos con PE*

	PE							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	4,24	3,01	6,21	2,64	6,93	4,91	7,59	6,94
kra30a	1,78	0,83	1,60	0,71	1,38	1,39	1,60	1,16
nug30	1,54	1,00	1,56	1,21	2,20	1,24	1,16	1,36
ste36a	6,38	6,23	4,62	2,99	11,10	6,14	5,68	8,13
sko49	50,17	41,32	36,91	24,69	114,39	43,57	71,40	29,54
sko56	127,51	67,55	114,43	72,90	102,00	85,66	78,45	109,33
sko64	27,49	91,00	29,97	42,96	28,99	82,13	74,74	71,17
sko72	244,93	178,69	293,72	244,32	258,11	326,60	230,66	372,96
sko81	482,86	584,18	626,15	482,98	702,20	759,42	386,73	755,10
sko90	495,29	762,60	883,08	903,21	740,73	1.087,16	1.115,38	499,68
sko100a	487,77	1.422,30	1.387,45	1.434,15	1.338,68	1.416,95	1.191,97	1.144,91
tai20a	1,06	0,48	1,38	1,09	0,86	0,81	1,26	0,95
tai25a	3,98	4,37	5,40	4,50	7,69	3,73	4,42	4,84
tai30a	18,00	16,74	8,26	6,97	7,71	11,52	14,40	10,85
tai35a	34,16	62,93	32,60	46,95	54,67	61,59	28,97	74,30
tai35b	6,22	10,56	6,43	7,82	9,40	20,93	27,16	12,50
tai40a	98,49	100,11	108,37	87,09	81,82	142,33	91,04	69,57
tai40b	3,68	13,13	4,22	7,65	6,07	9,83	9,61	8,12
tai50a	176,30	205,83	185,27	197,05	157,12	206,34	196,99	143,43
tai50b	93,96	79,77	62,86	65,01	77,77	69,29	81,27	70,74
tai60a	275,78	141,95	420,71	291,24	332,48	257,87	241,62	387,90
tai60b	106,68	126,59	135,50	99,65	132,17	109,54	120,37	98,95
tai80a	785,55	593,09	714,53	700,22	558,82	913,31	709,85	766,72
tai80b	892,94	829,88	783,50	898,22	489,61	635,37	581,82	856,97
tai100a	1.190,28	1.492,63	1.062,57	1.406,40	1.232,76	1.237,15	965,90	1.596,71
tai100b	1.355,50	1.149,50	1.204,67	1.417,93	1.389,25	1.294,38	1.112,79	1.269,76
tho150b	2.383,02	2.913,73	2.257,28	3.916,18	2.272,78	2.402,49	2.418,48	2.996,78
wil50	35,88	54,47	69,23	70,52	82,04	88,55	86,28	36,20
tai256c	4.512,03	4.028,28	4.189,49	4.174,37	3.951,28	4.065,70	3.692,47	4.326,08
tai150b	3.270,81	4.465,52	3.893,88	3.282,60	4.299,30	4.051,01	4.016,06	3.748,25
Total	17.174,26	19.448,25	18.531,86	19.894,19	18.450,29	19.396,89	17.566,11	19.479,89

TABLA 4.9: *Tiempo de convergencia en segundos con PEBC*

	PEBC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	2,78	3,17	4,88	3,08	4,61	4,28	5,07	7,77
kra30a	1,28	1,44	1,22	0,71	1,30	1,57	0,51	2,04
nug30	1,01	1,08	1,81	0,79	1,37	1,34	1,80	1,35
ste36a	4,18	6,37	5,83	5,76	5,56	3,74	7,05	3,00
sko49	46,21	24,45	54,93	28,97	33,60	37,04	68,20	44,64
sko56	103,69	77,23	114,83	167,11	175,34	100,07	34,14	24,55
sko64	103,95	119,57	53,55	47,93	48,81	123,24	82,11	38,34
sko72	379,22	348,38	510,62	391,59	229,01	140,24	264,49	310,29
sko81	805,67	516,48	399,90	518,51	271,98	366,78	397,60	438,41
sko90	1.114,44	1.327,96	1.083,86	1.071,01	991,44	1.037,58	605,73	724,49
sko100a	838,68	1.361,12	1.342,39	822,65	762,13	1.462,79	1.892,45	760,25
tai20a	1,37	0,53	1,36	1,50	1,58	0,90	2,07	1,02
tai25a	2,57	2,96	3,59	2,58	5,95	5,33	5,51	3,51
tai30a	7,78	12,27	4,86	5,98	11,77	10,85	4,72	9,94
tai35a	35,18	47,50	52,96	56,93	58,99	48,05	51,38	33,58
tai35b	5,16	6,19	17,01	10,30	13,08	15,54	10,79	7,55
tai40a	119,66	109,11	84,60	93,93	98,06	80,43	95,24	81,24
tai40b	10,46	7,74	6,05	5,88	8,33	5,23	7,55	7,20
tai50a	210,67	180,54	221,77	200,65	236,86	114,24	149,70	207,09
tai50b	45,72	110,65	75,08	45,81	66,33	48,07	98,36	88,27
tai60a	232,12	318,10	446,22	229,67	474,95	145,55	382,88	360,62
tai60b	112,14	99,74	97,16	92,53	137,57	91,09	103,88	119,20
tai80a	634,49	585,86	504,85	619,91	824,42	804,96	786,19	492,91
tai80b	732,27	551,36	1.207,91	842,38	572,69	439,68	1.110,16	784,38
tai100a	1.618,59	1.106,84	1.428,56	2.049,99	1.438,28	2.075,10	1.396,26	1.025,23
tai100b	1.099,21	1.290,38	1.430,06	1.406,45	1.769,60	1.387,73	1.754,80	2.092,47
tho150b	1701,36	2292,36	2491,09	2836,99	2720,34	3103,44	3313,25	3274,59
wil50	106,52	51,30	86,25	41,93	70,76	67,14	90,13	59,39
tai256c	4.940,08	2.882,05	4.721,04	3.729,09	3.489,08	4.793,52	3.955,61	4.720,40
tai150b	3.577,40	4.237,64	3.064,77	3.918,47	4.003,94	3.836,99	3.416,86	4.036,21
Total	18.593,86	17.680,35	19.518,97	19.249,06	18.527,71	20.352,46	20.094,47	19.759,96

Finalmente, se presenta un resumen con la suma de los tiempos de convergencia de cada algoritmo en todas las instancias probadas.

TABLA 4.10: *Resumen tiempos totales de convergencia en segundos*

	MC	PE	PEBC
LTV	21.922,95	**17.174,26**	18.593,86
LTV-R	19.562,70	19.448,25	17.680,35
LTV-SD	21.111,55	18.531,86	19.518,97
LTV-R-SD	20.780,20	19.894,19	19.249,06
LTF	22.029,11	18.450,29	18.527,71
LTF-R	22.336,97	19.396,89	20.352,46
LTF-SD	21.241,28	17.566,11	20.094,47
LTF-R-SD	19.250,62	19.479,89	19.759,96

4.3 TASA DE ÉXITO

En esta sección se presentan los resultados de la tasa de éxito de cada algoritmo. Se recuerda que corresponde a la proporción entre ejecuciones exitosas (que se alcanza el mínimo logrado por el algoritmo) sobre la cantidad total de ejecuciones del algoritmo con esa instancia.

En la tabla 4.11 se tiene la tasa de éxito de cada algoritmo que usa matriz de cruzamientos para cada instancia, en la tabla 4.12 para los algoritmos que usan población estructurada sin busqueda compartida (PE) y en la tabla 4.13 con búsqueda compartida (PEBC). El número mostrado entre paréntesis corresponde al error mínimo encontrado por cada algoritmo en la instancia correspondiente.

TABLA 4.11: *Tasa de éxito para MC, además, se muestran entre paréntesis los valores mínimos de error logrados por cada algoritmo*

	MC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
kra30a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
nug30	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
ste36a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko49	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko56	1,0 (0,000)	0,9 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko64	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko72	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	0,9 (0,000)	0,9 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko81	0,5 (0,011)	0,3 (0,000)	0,3 (0,000)	0,4 (0,000)	0,2 (0,000)	0,4 (0,000)	0,3 (0,000)	0,2 (0,000)
sko90	0,5 (0,000)	0,5 (0,000)	0,3 (0,000)	0,5 (0,000)	0,6 (0,000)	0,4 (0,000)	0,7 (0,000)	0,5 (0,000)
sko100a	0,2 (0,000)	0,2 (0,016)	0,2 (0,000)	0,2 (0,000)	0,3 (0,000)	0,1 (0,000)	0,4 (0,000)	0,2 (0,000)
tai20a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai25a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai30a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai35a	1,0 (0,000)	0,9 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	0,9 (0,000)	1,0 (0,000)
tai35b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai40a	0,4 (0,074)	0,3 (0,074)	0,1 (0,074)	0,1 (0,074)	0,3 (0,074)	0,1 (0,000)	0,2 (0,074)	0,4 (0,074)
tai40b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai50a	0,1 (0,245)	0,1 (0,380)	0,1 (0,298)	0,1 (0,477)	0,1 (0,351)	0,1 (0,322)	0,1 (0,379)	0,1 (0,388)
tai50b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai60a	0,1 (0,380)	0,1 (0,481)	0,1 (0,172)	0,1 (0,382)	0,1 (0,349)	0,1 (0,529)	0,1 (0,443)	0,1 (0,273)
tai60b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	0,9 (0,000)	0,9 (0,000)
tai80a	0,1 (0,537)	0,1 (0,688)	0,1 (0,654)	0,1 (0,578)	0,1 (0,607)	0,1 (0,437)	0,1 (0,624)	0,1 (0,593)
tai80b	0,6 (0,000)	0,9 (0,000)	0,6 (0,000)	0,9 (0,000)	0,6 (0,000)	0,3 (0,000)	1,0 (0,000)	0,6 (0,000)
tai100a	0,1 (0,541)	0,1 (0,531)	0,1 (0,415)	0,1 (0,497)	0,1 (0,488)	0,1 (0,530)	0,1 (0,425)	0,1 (0,661)
tai100b	0,4 (0,000)	0,4 (0,000)	0,9 (0,000)	0,6 (0,000)	0,1 (0,000)	0,4 (0,000)	0,4 (0,000)	0,3 (0,000)
tho150b	0,1 (0,024)	0,1 (0,024)	0,1 (0,025)	0,1 (0,010)	0,1 (0,061)	0,1 (0,057)	0,1 (0,024)	0,1 (0,043)
wil50	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai256c	0,1 (0,129)	0,1 (0,104)	0,1 (0,088)	0,1 (0,135)	0,2 (0,126)	0,1 (0,107)	0,1 (0,066)	0,1 (0,078)
tai150b	0,1 (0,265)	0,1 (0,260)	0,1 (0,167)	0,1 (0,003)	0,1 (0,111)	0,1 (0,137)	0,1 (0,166)	0,1 (0,154)
Promedio	**0,68** (0,074)	0,67 (0,085)	0,67 (0,063)	**0,68** (0,072)	0,66 (0,072)	0,65 (0,071)	**0,68** (0,073)	0,66 (0,075)

TABLA 4.12: *Tasa de éxito para PE, además, se muestran entre paréntesis los valores mínimos de error logrados por cada algoritmo*

	PE							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
kra30a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
nug30	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
ste36a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko49	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko56	0,9 (0,000)	0,9 (0,000)	0,9 (0,000)	0,9 (0,000)	1,0 (0,000)	1,0 (0,000)	0,8 (0,000)	1,0 (0,000)
sko64	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko72	0,8 (0,000)	0,8 (0,000)	0,9 (0,000)	0,8 (0,000)	0,7 (0,000)	0,9 (0,000)	0,9 (0,000)	0,9 (0,000)
sko81	0,3 (0,000)	0,1 (0,000)	0,3 (0,000)	0,4 (0,000)	0,5 (0,000)	0,4 (0,000)	0,3 (0,000)	0,4 (0,000)
sko90	0,2 (0,000)	0,4 (0,000)	0,8 (0,000)	0,4 (0,000)	0,5 (0,000)	0,5 (0,000)	0,4 (0,000)	0,5 (0,000)
sko100a	0,1 (0,016)	0,3 (0,000)	0,1 (0,000)	0,3 (0,000)	0,1 (0,000)	0,4 (0,000)	0,4 (0,000)	0,1 (0,000)
tai20a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai25a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai30a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	0,9 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai35a	0,6 (0,000)	0,9 (0,000)	0,9 (0,000)	0,7 (0,000)	0,8 (0,000)	0,7 (0,000)	0,8 (0,000)	0,9 (0,000)
tai35b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai40a	0,2 (0,173)	0,3 (0,120)	0,1 (0,074)	0,1 (0,120)	0,2 (0,074)	0,2 (0,074)	0,2 (0,074)	0,1 (0,000)
tai40b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai50a	0,1 (0,000)	0,1 (0,053)	0,1 (0,490)	0,1 (0,452)	0,1 (0,386)	0,1 (0,382)	0,1 (0,495)	0,1 (0,000)
tai50b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai60a	0,1 (0,410)	0,1 (0,552)	0,1 (0,525)	0,1 (0,402)	0,1 (0,441)	0,1 (0,479)	0,1 (0,328)	0,1 (0,488)
tai60b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai80a	0,1 (0,537)	0,1 (0,649)	0,1 (0,587)	0,1 (0,753)	0,1 (0,600)	0,1 (0,621)	0,1 (0,572)	0,1 (0,576)
tai80b	0,7 (0,000)	0,7 (0,000)	0,8 (0,000)	0,9 (0,000)	0,7 (0,000)	0,7 (0,000)	0,6 (0,000)	0,8 (0,000)
tai100a	0,1 (0,356)	0,1 (0,524)	0,1 (0,523)	0,1 (0,680)	0,1 (0,625)	0,1 (0,452)	0,1 (0,674)	0,1 (0,668)
tai100b	0,5 (0,000)	0,3 (0,000)	0,4 (0,000)	0,2 (0,000)	0,4 (0,000)	0,5 (0,000)	0,2 (0,000)	0,5 (0,000)
tho150b	0,1 (0,010)	0,1 (0,051)	0,1 (0,014)	0,1 (0,008)	0,1 (0,001)	0,1 (0,022)	0,1 (0,030)	0,1 (0,006)
wil50	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	0,9 (0,000)
tai256c	0,1 (0,093)	0,1 (0,062)	0,1 (0,054)	0,1 (0,079)	0,1 (0,090)	0,1 (0,051)	0,1 (0,092)	0,1 (0,037)
tai150b	0,1 (0,383)	0,1 (0,231)	0,1 (0,152)	0,1 (0,131)	0,1 (0,108)	0,1 (0,110)	0,1 (0,199)	0,1 (0,353)
Promedio	0,63 (0,066)	0,65 (0,075)	0,66 (0,081)	0,65 (0,088)	0,65 (0,077)	0,67 (0,073)	0,64 (0,082)	0,66 (0,071)

TABLA 4.13: *Tasa de éxito para PEBC, además, se muestran entre paréntesis los valores mínimos de error logrados por cada algoritmo*

	PEBC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
chr25a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
kra30a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
nug30	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
ste36a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	0,9 (0,000)	1,0 (0,000)
sko49	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko56	0,8 (0,000)	1,0 (0,000)	0,9 (0,000)	0,8 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko64	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
sko72	0,9 (0,000)	1,0 (0,000)	1,0 (0,000)	0,9 (0,000)	0,8 (0,000)	0,9 (0,000)	1,0 (0,000)	0,9 (0,000)
sko81	0,4 (0,000)	0,4 (0,000)	0,5 (0,000)	0,2 (0,000)	0,4 (0,000)	0,1 (0,000)	0,2 (0,000)	0,3 (0,000)
sko90	0,5 (0,000)	0,6 (0,000)	0,4 (0,000)	0,6 (0,000)	0,5 (0,000)	0,7 (0,000)	0,6 (0,000)	0,7 (0,000)
sko100a	0,1 (0,000)	0,1 (0,000)	0,4 (0,000)	0,2 (0,000)	0,1 (0,000)	0,2 (0,000)	0,5 (0,000)	0,3 (0,000)
tai20a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai25a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai30a	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai35a	0,7 (0,000)	0,9 (0,000)	0,8 (0,000)	0,8 (0,000)	0,8 (0,000)	0,8 (0,000)	0,7 (0,000)	0,8 (0,000)
tai35b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai40a	0,1 (0,120)	0,1 (0,074)	0,2 (0,120)	0,2 (0,074)	0,1 (0,074)	0,1 (0,120)	0,1 (0,120)	0,1 (0,306)
tai40b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai50a	0,1 (0,231)	0,1 (0,328)	0,1 (0,290)	0,1 (0,430)	0,1 (0,551)	0,1 (0,441)	0,1 (0,477)	0,1 (0,317)
tai50b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai60a	0,1 (0,485)	0,1 (0,549)	0,1 (0,401)	0,1 (0,497)	0,1 (0,588)	0,1 (0,535)	0,1 (0,345)	0,1 (0,502)
tai60b	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai80a	0,1 (0,652)	0,1 (0,714)	0,1 (0,520)	0,1 (0,486)	0,1 (0,591)	0,1 (0,631)	0,1 (0,566)	0,1 (0,698)
tai80b	0,7 (0,000)	0,8 (0,000)	0,6 (0,000)	0,7 (0,000)	0,8 (0,000)	0,5 (0,000)	0,6 (0,000)	0,8 (0,000)
tai100a	0,1 (0,619)	0,1 (0,607)	0,1 (0,566)	0,1 (0,437)	0,1 (0,427)	0,1 (0,601)	0,1 (0,585)	0,1 (0,566)
tai100b	0,2 (0,000)	0,4 (0,000)	0,1 (0,000)	0,6 (0,000)	0,3 (0,000)	0,5 (0,000)	0,3 (0,000)	0,3 (0,000)
tho150b	0,1 (0,030)	0,1 (0,063)	0,1 (0,038)	0,1 (0,059)	0,1 (0,042)	0,1 (0,017)	0,1 (0,009)	0,1 (0,045)
wil50	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)	1,0 (0,000)
tai256c	0,1 (0,058)	0,2 (0,075)	0,1 (0,075)	0,1 (0,071)	0,1 (0,050)	0,1 (0,045)	0,1 (0,073)	0,1 (0,049)
tai150b	0,2 (0,175)	0,1 (0,271)	0,1 (0,308)	0,1 (0,438)	0,1 (0,301)	0,1 (0,105)	0,1 (0,236)	0,1 (0,397)
Promedio	0,64 (0,079)	0,67 (0,089)	0,65 (0,077)	0,66 (0,083)	0,65 (0,087)	0,65 (0,083)	0,65 (0,080)	0,66 (0,096)

Mostrado el detalle, se presenta un resumen de la tasa de éxito para cada algoritmo promediando todas las instancias utilizadas para hacer las pruebas en la tabla 4.14.

TABLA 4.14: *Resumen de tasa de éxito*

	MC	**PE**	**PEBC**
LTV	**0,68** (0,074)	0,63 (0,066)	0,64 (0,079)
LTV-R	0,67 (0,085)	0,65 (0,075)	0,67 (0,089)
LTV-SD	0,67 (**0,063**)	0,66 (0,081)	0,65 (0,077)
LTV-R-SD	**0,68** (0,072)	0,65 (0,088)	0,66 (0,083)
LTF	0,66 (0,072)	0,65 (0,077)	0,65 (0,087)
LTF-R	0,65 (0,071)	0,67 (0,073)	0,65 (0,083)
LTF-SD	**0,68** (0,073)	0,64 (0,082)	0,65 (0,080)
LTF-R-SD	0,66 (0,075)	0,66 (0,071)	0,66 (0,096)

4.4 ÍNDICES COPELAND

En esta sección se presentan los índices Copeland para cada algoritmo obtenido para cada una de las métricas expuestas anteriormente. Si se desea revisar en detalle la obtención de estos índices, revisar el Apéndice A.

El objetivo del uso de índices Copeland es poder comparar los diferentes algoritmos considerando la totalidad de los experimentos realizados.

4.4.1 Calidad de soluciones

En la tabla 4.15 se muestra el índice Copeland por cada algoritmo con respecto a la calidad de soluciones obtenida en todas las instancias. Mientras mayor es el número, indica que en más instancias superó a los demás, y mientras sea menor, lo contrario.

TABLA 4.15: *Índices Copeland calidad de soluciones*

	MC	**PE**	**PEBC**
LTV	40	-54	-67
LTV-R	94	-107	-41
LTV-SD	**196**	-40	-60
LTV-R-SD	126	-129	-9
LTF	106	-86	-109
LTF-R	78	83	-49
LTF-SD	190	-74	-78
LTF-R-SD	75	-47	-38

4.4.2 Tasa de éxito

En la tabla 4.16 se muestra el índice Copeland para cada algoritmo con respecto a la tasa de éxito obtenida en todas las pruebas de todas las instancias.

TABLA 4.16: *Índices Copeland tasa de éxito*

	MC	**PE**	**PEBC**
LTV	**75**	-74	-53
LTV-R	43	-41	64
LTV-SD	15	-4	-6
LTV-R-SD	60	-51	-8
LTF	37	-29	-31
LTF-R	-9	37	-23
LTF-SD	73	-55	-31
LTF-R-SD	2	-5	14

4.4.3 Tiempo de convergencia

Finalmente, en la tabla 4.17 se presentan los índices Copeland para el tiempo promedio de convergencia de los algoritmos en cada una de las instancias.

TABLA 4.17: *Índices Copeland tiempo de convergencia*

	MC	**PE**	**PEBC**
LTV	-93	124	58
LTV-R	8	36	114
LTV-SD	-72	49	-66
LTV-R-SD	13	**179**	161
LTF	-125	10	-60
LTF-R	-151	-108	69
LTF-SD	-40	15	-66
LTF-R-SD	-129	-7	81

4.4.4 Esquemas de memoria

En esta sección se darán resultados entre la comparación de los esquemas de memoria mediante índices Copeland respecto a la calidad de las soluciones obtenidas en cada una de los tipos de esquema de memoria.

4.4.4.1 Esquemas poblacionales

En la tabla 4.18 se puede ver los valores obtenidos de índices Copeland para cada tipo de estructura poblacional aplicada, y el promedio del porcentaje de error de ellas.

TABLA 4.18: *Índices Copeland esquemas poblacionales*

	MC	**PE**	**PEBC**
Copeland	905	-454	-451
Promedio	0,122	0,137	0,139

En la tabla 4.19 se pueden ver los valores de índices Copeland comparando sólo los esquemas poblacionales estructurados, para comparar entre el uso de búsqueda compartida y la ausencia de ella.

TABLA 4.19: *Índices Copeland esquemas poblacionales estructurados*

	PE	**PEBC**
Copeland	-10	10
Promedio	0,137	0,139

4.4.4.2 Tipo de Búsqueda Tabú

En la tabla 4.20 se puede ver los valores obtenidos de índices Copeland para cada tipo búsqueda tabú aplicada, y el promedio del porcentaje de error de ellas.

TABLA 4.20: *Índices Copeland tipos TS*

	LTV	**LTF**
Copeland	-51	51
Promedio	0,133	0,132

4.4.4.3 Reducción de espacio de búsqueda

En la tabla 4.21 se puede ver los valores obtenidos de índices Copeland para los algoritmos con y sin reducción del espacio de búsqueda, y el promedio del porcentaje de error de ellas.

TABLA 4.21: *Índices Copeland para reducción de espacio de búsqueda*

	Reducido	**No reducido**
Copeland	36	-36
Promedio	0,133	0,132

4.4.4.4 Mejora por Gradiente Descendiente

En la tabla 4.22 se puede ver los valores obtenidos de índices Copeland para los algoritmos con y sin gradiente descendiente, y el promedio del porcentaje de error de ellas.

TABLA 4.22: *Índices Copeland para mejora por gradiente descendiente*

	Mejorado	**No mejorado**
Copeland	112	-112
Promedio	0,133	0,133

4.5 TEST DE WILCOXON

Las tablas 4.23 - 4.23 muestran los *p-values* obtenidos tras la aplicación de el Test de Wilcoxon. Los detalles del test, mostrando evaluaciones positivas, negativas y empates

pueden ser encontradas en el archivo anexo *Wilcoxon.xlsx* del disco del trabajo presentado. Notar que los valores se repiten en las celdas que comparan dos algoritmos iguales, por ejemplo el *p-value* entre comparar MC-LTF y PE-LTV-R es igual al de comparar PE-LTV-R y MC-LTF. La repetición de valores se realiza con el fin de facilitar la lectura de las tablas.

TABLA 4.23: *P-Values obtenidos para el Test de Wilcoxon (1)*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	0,244	0,069	0,315	0,286	0,529	0,041	0,116
	LTV-R	0,244	-	0,139	0,641	0,865	0,826	0,683	0,569
	LTV-SD	0,069	0,139	-	0,068	0,925	0,173	0,33	0,925
	LTV-R-SD	0,315	0,641	0,068	-	0,363	1	0,32	0,426
	LTF	0,286	0,865	0,925	0,363	-	0,778	0,865	0,932
	LTF-R	0,529	0,826	0,173	1	0,778	-	0,311	0,925
	LTF-SD	0,041	0,683	0,33	0,32	0,865	0,311	-	0,842
	LTF-R-SD	0,116	0,569	0,925	0,426	0,932	0,925	0,842	-
PE	**LTV**	0,443	0,47	0,026	0,079	0,134	0,691	0,061	0,177
	LTV-R	0,147	0,012	0,003	0,02	0,013	0,088	0,002	0,033
	LTV-SD	0,453	0,05	0,015	0,011	0,083	0,205	0,009	0,149
	LTV-R-SD	0,078	0,012	0,003	0,004	0,011	0,066	0,003	0,035
	LTF	0,179	0,042	0,014	0,007	0,049	0,103	0,004	0,049
	LTF-R	0,955	0,173	0,069	0,244	0,158	0,532	0,047	0,244
	LTF-SD	0,121	0,109	0,015	0,019	0,021	0,108	0,005	0,012
	LTF-R-SD	0,201	0,055	0,03	0,063	0,115	0,155	0,006	0,023
PEBC	**LTV**	0,148	0,148	0,049	0,052	0,07	0,215	0,024	0,088
	LTV-R	0,149	0,008	0,008	0,021	0,014	0,056	0,006	0,012
	LTV-SD	0,065	0,045	0,011	0,036	0,03	0,14	0,011	0,056
	LTV-R-SD	0,438	0,079	0,03	0,103	0,098	0,205	0,003	0,407
	LTF	0,041	0,004	0,011	0,011	0,008	0,023	0,004	0,026
	LTF-R	0,109	0,056	0,011	0,018	0,015	0,156	0,013	0,049
	LTF-SD	0,164	0,03	0,022	0,014	0,016	0,053	0,003	0,036
	LTF-R-SD	0,306	0,093	0,009	0,019	0,057	0,349	0,018	0,301

	***P* <0,1**	5	12	18	15	13	5	17	11
	***P* <0,05**	2	7	15	11	9	1	16	9
	***P* <0,01**	0	2	4	2	1	0	10	0

TABLA 4.24: *P-Values obtenidos para el Test de Wilcoxon (2)*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	0,443	0,147	0,453	0,078	0,179	0,955	0,121	0,201
	LTV-R	0,47	0,012	0,05	0,012	0,042	0,173	0,109	0,055
	LTV-SD	0,026	0,003	0,015	0,003	0,014	0,069	0,015	0,03
	LTV-R-SD	0,079	0,02	0,011	0,004	0,007	0,244	0,019	0,063
	LTF	0,134	0,013	0,083	0,011	0,049	0,158	0,021	0,115
	LTF-R	0,691	0,088	0,205	0,066	0,103	0,532	0,108	0,155
	LTF-SD	0,061	0,002	0,009	0,003	0,004	0,047	0,005	0,006
	LTF-R-SD	0,177	0,033	0,149	0,035	0,049	0,244	0,012	0,023
PE	**LTV**	-	0,414	0,776	0,173	1	0,365	0,979	0,943
	LTV-R	0,414	-	0,396	0,196	0,722	0,266	0,887	0,408
	LTV-SD	0,776	0,396	-	0,088	0,135	0,147	0,46	0,394
	LTV-R-SD	0,173	0,196	0,088	-	0,554	0,009	0,438	0,256
	LTF	1	0,722	0,135	0,554	-	0,061	0,981	0,85
	LTF-R	0,365	0,266	0,147	0,009	0,061	-	0,07	0,28
	LTF-SD	0,979	0,887	0,46	0,438	0,981	0,07	-	0,394
	LTF-R-SD	0,943	0,408	0,394	0,256	0,85	0,28	0,394	-
PEBC	**LTV**	0,756	0,679	0,53	0,148	0,836	0,209	0,977	0,887
	LTV-R	0,737	0,938	0,682	0,66	0,717	0,281	0,642	0,909
	LTV-SD	0,349	0,865	0,397	0,426	0,955	0,07	0,379	0,344
	LTV-R-SD	0,877	0,256	0,363	0,063	0,08	0,535	0,211	0,276
	LTF	0,379	0,836	0,134	0,485	0,245	0,033	0,623	0,266
	LTF-R	0,569	0,679	0,222	0,289	0,535	0,044	0,501	0,594
	LTF-SD	0,758	0,981	0,266	0,356	1	0,098	0,538	0,776
	LTF-R-SD	0,379	0,802	0,46	0,244	0,979	0,221	0,66	0,755

	***P* <0,1**	3	7	6	11	8	9	6	5
	***P* <0,05**	1	6	3	7	6	4	5	3
	***P* <0,01**	0	2	1	4	2	1	1	1

TABLA 4.25: *P-Values obtenidos para el Test de Wilcoxon (3)*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	0,148	0,149	0,065	0,438	0,041	0,109	0,164	0,306
	LTV-R	0,148	0,008	0,045	0,079	0,004	0,056	0,03	0,093
	LTV-SD	0,049	0,008	0,011	0,03	0,011	0,011	0,022	0,009
	LTV-R-SD	0,052	0,021	0,036	0,103	0,011	0,018	0,014	0,019
	LTF	0,07	0,014	0,03	0,098	0,008	0,015	0,016	0,057
	LTF-R	0,215	0,056	0,14	0,205	0,023	0,156	0,053	0,349
	LTF-SD	0,024	0,006	0,011	0,003	0,004	0,013	0,003	0,018
	LTF-R-SD	0,088	0,012	0,056	0,407	0,026	0,049	0,036	0,301
PE	**LTV**	0,756	0,737	0,349	0,877	0,379	0,569	0,758	0,379
	LTV-R	0,679	0,938	0,865	0,256	0,836	0,679	0,981	0,802
	LTV-SD	0,53	0,682	0,397	0,363	0,134	0,222	0,266	0,46
	LTV-R-SD	0,148	0,66	0,426	0,063	0,485	0,289	0,356	0,244
	LTF	0,836	0,717	0,955	0,08	0,245	0,535	1	0,979
	LTF-R	0,209	0,281	0,07	0,535	0,033	0,044	0,098	0,221
	LTF-SD	0,977	0,642	0,379	0,211	0,623	0,501	0,538	0,66
	LTF-R-SD	0,887	0,909	0,344	0,276	0,266	0,594	0,776	0,755
PEBC	**LTV**	-	0,918	0,339	0,514	0,469	0,629	0,394	0,57
	LTV-R	0,918	-	0,932	0,196	0,47	0,798	0,82	0,433
	LTV-SD	0,339	0,932	-	0,127	0,755	0,453	0,776	0,979
	LTV-R-SD	0,514	0,196	0,127	-	0,187	0,266	0,129	0,426
	LTF	0,469	0,47	0,755	0,187	-	0,232	0,679	0,426
	LTF-R	0,629	0,798	0,453	0,266	0,232	-	0,938	0,706
	LTF-SD	0,394	0,82	0,776	0,129	0,679	0,938	-	0,938
	LTF-R-SD	0,57	0,433	0,979	0,426	0,426	0,706	0,938	-

	***P* <0,1**	5	7	8	6	9	7	8	5
	***P* <0,05**	2	6	5	2	9	6	6	3
	***P* <0,01**	0	3	0	1	3	0	1	1

CAPÍTULO 5. DISCUSIÓN

En este capítulo se analizarán los resultados presentados previamente, haciendo comparaciones de calidad de solución, tiempo de convergencia y robustez. Además, se determina un algoritmo superior dentro de los creados, el cual es comparado con el algoritmo original y con resultados actuales de la literatura.

5.1 COMPARACIÓN DE ALGORITMOS

Las pruebas realizadas utilizan las mismas instancias publicadas en Inostroza-Ponta (2008), pero se varían los tiempos máximos usando los presentes en el capítulo anterior.

Los resultados muestran que las nuevas pruebas tienen en promedio un 0,136 % de error con respecto al mejor valor conocido y los resultados publicados tienen un 0,150 %. Además comparando por cada instancia, se tiene que las nuevas pruebas superan a las publicadas en 10 instancias, el caso inverso en 6 instancias y 14 empates (todos en 0 % de error).

Por el hecho que se están considerando nuevos resultados de mejor calidad, tanto en promedio de error como en número de instancias superadas, se puede decir que la comparación de los resultados de otros algoritmos generados contra estos valores, es válida.

A continuación se analiza y compara la totalidad de algoritmos generados con respecto a calidad de soluciones, tiempo de convergencia y robustez, tanto en promedio como en índices Copeland.

5.1.1 Calidad de soluciones

Como se menciona en el capítulo 1, lo más deseable de una heurística es que se logre buena calidad de soluciones antes que su rapidez, por lo cual el análisis dado en esta sección es de vital importancia.

Para los algoritmos probados, se tiene que el rango promedio de variación del error va entre 0,118 % y 0,144 %. El algoritmo original posee en promedio un 0,136 % de error, con lo que se encuentran 12 variaciones que tienen un valor inferior. Estos algoritmos son los siguientes:

- Los algoritmos que usan MC en todas sus variedades (8 algoritmos). Dentro de éstos MC-LTV-SD, MC-LTF y MC-LTF-R-SD muestran la mejor calidad promedio con 0,118 % de error en las instancias tratadas.

- En los algoritmos que usan población estructurada sin búsqueda compartida se logran errores promedio inferiores en 3 casos comparando al algoritmo original: el que usa lista tabú variable sin reducción ni SD (PE-LTV) con 0,133 %, el mismo, pero usando SD (PE-LTV-SD) con 0,134 %; y el que usa lista tabú fija y reducción (PE-LTF-R) con 0,131 %.

- En los algoritmos que usan población estructurada con búsqueda compartida, sólo se supera en calidad promedio de soluciones en la variación que usa lista tabú variable, con reducción y mejora (PEBC-LTV-R-SD), con un 0,133 % de error.

Otra forma de visualizar la calidad de soluciones obtenidas por los algoritmos es contabilizar las instancias en las cuales se supera a todos los demás (o se empata con el mejor). Contabilizando:

- La cantidad de instancias en que se llega a 0 % de error contra el mejor valor encontrado.
- La cantidad de instancias en que se logra el mejor valor de error promedio, diferente de 0 %.
- La cantidad de veces que se llega muy cerca del menor valor alcanzado entre todos los algoritmos, visto como las veces que se cae en el rango entre $[\acute{m}inimo, \acute{m}inimo + 0,005]$.

Se tiene que el algoritmo original obtiene 17 instancias con dichas características. Empatando en ese valor se tienen 7 algoritmos:

- MC-LTV.
- PE-LTV-SD.
- PE-LTV-R-SD.
- PE-LTF-R.
- PE-LTF-SD.
- PEBC-LTV-R.
- PEBC-LTF-R-SD.

Los algoritmos que superan las 17 instancias son:

- MC-LTV-R (19).
- MC-LTV-SD (23).
- MC-LTV-R-SD (20).

- MC-LTF (19).
- MC-LTF-R (18).
- MC-LTF-SD (19).
- MC-LTF-R-SD (18).
- MC-PEBC-LTV-SD (18).

Con lo que se puede apreciar que el algoritmo que obtiene más instancias exitosas de forma global es MC-LTV-SD, el cual coincidentemente también tiene un error promedio mínimo con respecto a los analizados (0,118 %).

El análisis anterior permite ver la superioridad de un algoritmo de forma global, es decir, determinar cual de ellos logra una mayor cantidad de mejores valores en instancias particulares y un mejor promedio de todas ellas. Sin embargo, puede tenerse que un algoritmo posea porcentajes de error muy bajos para algunas instancias y muy altos para otras, por lo que conviene analizar la cantidad de veces que cada algoritmo en cada instancia supera y es superado por otros, lo cual es dado por el índice Copeland.

Dado que cada algoritmo se compara en 30 instancias con los demás 23, se tiene que el índice máximo para este primer análisis es 690 ($1_{algoritmo} \times 23_{otros} \times 30_{instancias} = 690$), el cual es dado por el total de comparaciones.

El algoritmo original presenta un índice de -67, lo cual implica que $victorias - derrotas = -67$. Obtenido este índice, se tienen 17 algoritmos que superan este valor, con índices mayores a -67. Sin embargo, los mejores algoritmos son los que poseen índices Copeland positivos, ya que implica que tienen menor porcentaje de error en mayor cantidad de ocasiones comparados contra los demás algoritmos. Los algoritmos que poseen índices Copeland mayores a cero son:

- Algoritmos que usan MC: todas sus variedades (8 algoritmos), con índices entre 40 y 196.
- Algoritmos que usan PE: sólo LTF-R, con índice de 83.
- Algoritmos que usan PEBC: ninguna variedad tiene índice positivo.

Dentro de todos los algoritmos el que tiene mayor índice Copeland es MC-LTV-SD (196). Esto quiere decir que hay 196 casos más de victorias que derrotas, comparando MC-LTV-SD con los demás algoritmos en cada instancia. Dado que el máximo de índice en este caso es 690, se tiene que la diferencia entre victorias y derrotas corresponde a un 28,4 % de las comparaciones. Notar que la distribución de las derrotas, victorias y empates puede ser variable, por ejemplo una distribución puede ser que de los 690 enfrentamientos sean 541 victorias, 149 derrotas y 0 empates, y otra distribución podría tener 196 victorias, 0 derrotas y 494 empates.

Cabe mencionar que el algoritmo MC-LTV-SD además de ser el vencedor por índice Copeland, coincide ser el mejor por promedio de porcentaje de error en instancias y número de veces que tiene un error promedio de instancia mejor que todos los otros algoritmos.

5.1.2 Tasa de éxito

Otro aspecto importante de analizar es la tasa de éxito de los algoritmos. Esto corresponde a la cantidad de veces que cada algoritmo llega al mínimo que es capaz de encontrar en un cierto número de pruebas.

En general el valor promedio de tasas de éxito para cada algoritmo es superior al original, el cual posee una tasa de éxito de 0,64. El único que no supera este éxito es PE-LTV con una tasa promedio de 0,63. Los demás van en promedio entre 0,64 y y 0,68, lo

que permite decir que en general los algoritmos presentan una estabilidad similar entre ellos.

Comparando particularmente cada algoritmo contra los demás, se tienen los índices Copeland para las tasas de éxito. Al igual que en los casos anteriores el máximo es de 690, donde el original tiene un índice de -53. En general este índice es superado por los demás algoritmos, excepto PE-LTV (-74) coincidente en tasa de éxito promedio más baja que el original, y PE-LTF-SD (-55).

Los algoritmos que poseen índices Copeland mayores que el original y mayores a cero (más victorias que derrotas) son:

- Algoritmos que usan MC: todos excepto MC-LTV-R (-9). El máximo exponente es MC-LTV con índice 75.
- Algoritmos que usan PE: sólo PE-LTV-R (37) supera al original y es mayor a cero.
- Algoritmos que usan PEBC: hay dos casos, PEBC-LTV-R (64) y PEBC-LTF-R-SD (14).

Como se puede notar, los algoritmos que usan MC son los que poseen mayores índices Copeland para tasas de éxito. Además, cabe destacar que el algoritmo MC-LTV es el que posee mayor tasa de éxito tanto promedio como particular para cada instancia, ya que tiene su promedio e índice Copeland superiores a todos los demas algoritmos.

Según se puede desprender de esta sección, los algoritmos que usan MC además de tener más alta calidad de soluciones tienen más robustez, en comparación al algoritmos original y las demás modificaciones de éste.

5.1.3 Tiempo de convergencia

Un aspecto necesario a comparar corresponde al tiempo de convergencia de los algoritmos. Este aspecto es revisado tanto en el total de tiempo tardado en converger, como en índice Copeland de las comparaciones de a parejas.

Según los tiempos máximos dados en las métricas de evaluación, el total de tiempos máximos por instancia es de 44.657 segundos. Los algoritmos obtenidos se mueven entre 17.174,26 y 22.336,97 segundos, con lo que se puede ver que el peor caso tiene un tiempo total mayor en un 30 % al mejor caso. Sin embargo, el peor caso converge en promedio al 50 % del tiempo máximo límite del experimento y el mejor caso en 38,5 % del mismo.

El algoritmo original presenta un promedio de 18.593,86 segundos. El cual es superado con tiempos menores por 6 algoritmos:

- PE-LTV (17.174,26 segundos).
- PE-LTV-SD (18.531,86 segundos).
- PE-LTF (18.450,29 segundos).
- PE-LTF-SD (17.566,11 segundos).
- PEBC-LTV-R (17.680,35 segundos).
- PEBC-LTF (18.527,71 segundos).

Como se puede ver, el algoritmo PE-LTV presenta la convergencia promedio más rápida entre los 24 algoritmos analizados. Sin embargo, es prudente analizar a su vez las comparaciones de cada algoritmo en cada instancia, dados por el índice Copeland.

Al igual que la sección anterior, el índice máximo es de 690. El algoritmo original presenta un índice de 58, lo cual supera a la mayoría de los analizados. Los algoritmos que superan el índice Copeland del original son:

- PE-LTV (124).
- PE-LTV-R-SD (179).
- PEBC-LTV-R (114).
- PEBC-LTV-R-SD (161).
- PEBC-LTF-R (69).
- PEBC-LTF-R-SD (81).

Analizando estas dos comparativas de acuerdo al tiempo, se tiene que el algoritmo con convergencia promedio más rápida también posee mayor índice Copeland que el original: PE-LTV (17174,26 segundos, 124 Copeland). Sin embargo los demás algoritmos que tienen mayores victorias que derrotas que el original no poseen un tiempo promedio menor. Esto indica que hay unas pocas instancias en que esos algoritmos tienen un tiempo de convergencia muy alto lo que aumenta el promedio, pero no varía de sobremanera el índice Copeland.

Es importante destacar que los algoritmos que usan MC en general no presentan tiempos de convergencia menores que el original, y en particular sólo 2 algoritmos tienen índices Copeland mayores a cero: MC-LTV-R (8) y MC-LTV-R-SD (13). Esto indica que tienen una convergencia más lenta que el algoritmo original, sin embargo en la sección anterior se muestra que tienen soluciones de mejor calidad, lo que refleja que a pesar de tardar más tiempo llegan a mejores soluciones.

5.2 COMPARACIÓN DE ESQUEMAS DE MEMORIA

En esta sección se analiza y compara el efecto del uso de los diferentes esquemas de memoria que posee cada algoritmo en términos de calidad de soluciones dada por el error

promedio al BKV y el índice Copeland por cada análisis.

5.2.1 Esquemas de memoria poblacionales

Para comparar los esquemas poblacionales, se consideran los índices Copeland sólo de la comparativa entre algoritmos que los usan diferentes. Por ejemplo, para obtener el índice Copeland del esquema MC se consideran sólo los índices de MC contra algoritmos que emplean PE y PEBC, no se toman los demás del mismo tipo. En el apéndice A se cuenta con más detalle de estos datos.

El índice máximo para cada esquema es 3840, proveniente de las 30 instancias comparadas por cada uno de los 8 algoritmos de cada esquema poblacional, comparado contra los 16 restantes ($30_{instancias} \times 8_{esquemas} \times 16_{restantes} = 3840$). El menor índice máximo posible es 905, correspondiente al valor máximo existente para el esquema MC.

Se puede ver una amplia superioridad del esquema MC, teniendo 905 casos adicionales de victorias sobre derrotas, lo que indica que de las 3840 comparaciones hay 23,6 % más de victorias que derrotas. PE y PEBC muestran índices de -454 y -451 respectivamente.

Cabe mencionar que el esquema MC también supera en calidad promedio a los otros esquemas. MC tiene 0,122 % de error promedio, mientras que PE tiene 0,137 % y PEBC 0,139 %.

5.2.2 Uso de lista tabú fija y variable

De forma similar a lo anterior, se hizo comparación entre usar lista tabú de tamaño fijo y variable.

Esta vez el indice Copeland máximo es 4320, proveniente de la comparación de los 12 algoritmos con cada tipo de lista tabú contra los 12 otros, en 30 instancias cada uno

($12_{LTF} \times 12_{LTV} \times 30_{Instancias} = 4320$).

Se puede apreciar que la lista tabú de tamaño fijo supera a la variable por 51 casos, sin embargo esto no permite sacar conclusiones estadísticamente significativas respecto a la superioridad de un esquema sobre otro, ya que dentro de las 4320 comparaciones, hay un 1,18 % más de victorias que derrotas usando LTF en vez de LTV. Además, en promedio los esquemas que usan LTF tienen 0,132 % de error y las que usan LTV poseen 0,133 %, lo cual también impide concluir que LTF supera en totalidad a LTV.

5.2.3 Uso de reducción de espacio de búsqueda

Al igual que el caso anterior, el índice Copeland máximo es 4320. Se puede apreciar que el uso de la reducción del espacio de búsqueda supera su ausencia por 36 casos, sin embargo esto no permite sacar conclusiones estadísticamente significativas respecto a la superioridad de este esquema, ya que dentro de las 4320 comparaciones, hay un 0,83 % más de victorias que derrotas reducción comparado a no hacerlo. Además, en promedio los esquemas que usan reducción tienen 0,133 % de error y las que no, un 0,132 %, lo cual también impide concluir que la reducción del espacio de búsqueda supera en totalidad a no usarla.

5.2.4 Uso de *Steepest Descent*

Al igual que el caso anterior, el índice Copeland máximo es 4320. Se puede apreciar que el uso *Steepest Descent* supera a su ausencia por 112 casos, sin embargo esto no permite sacar conclusiones estadísticamente significativas con respecto a la superioridad del uso de esta técnica, ya que dentro de las 4320 comparaciones, hay un 2,59 % más de victorias que derrotas usando SD comparado a no hacerlo.

En promedio el uso de SD tiene 0,133 % de error y no usarlo un 0,133 %, lo cual no indica una superioridad significativa esta mejora (igual error promedio, 112 casos de 4320 comparados).

5.2.5 Uso de búsqueda compartida

En este caso el índice Copeland máximo es 1920, proveniente de la comparación de los 8 algoritmos con población estructurada con búsqueda compartida con los 8 algoritmos con población estructurada sin búsqueda compartida, 30 instancias en cada comparación ($8_{PE} \times 8_{PEBC} \times 30_{Instancias} = 1920$).

Se puede apreciar que el uso de la búsqueda compartida supera su ausencia por 10 casos, sin embargo esto no permite sacar conclusiones estadísticamente significativas respecto a la superioridad de este esquema, ya que dentro de las 1920 comparaciones, hay un 0,52 % más de victorias que derrotas usando búsqueda compartida comparado a no hacerlo. Además, en promedio los esquemas que usan búsqueda compartida 0,139 % de error y las que no, un 0,137 %, lo cual también impide concluir que el uso de búsqueda compartida supera en totalidad a no usarla.

5.3 DETERMINACIÓN Y ANÁLISIS DEL MEJOR ALGORITMO

Como se ha mencionado previamente, el factor más importante dentro de los mencionados para analizar los resultados de un algoritmo, es la calidad de las soluciones generadas.

Al revisar los resultados presentados previamente, se puede ver en términos de porcentaje de error al mejor valor conocido (calidad de solución) la superioridad del algoritmo MC-LTV-SD en:

- Promedio de porcentajes de error por instancia: tiene el valor mínimo correspondiente a 0,118 %.
- Cantidad de instancias con porcentaje de error mínimo, en comparación con los demás: tiene el máximo, correspondiente a 23 instancias.
- Cantidad de comparaciones victoriosas (índice Copeland): tiene el máximo, correspondiente a 196 de 690.

Determinado el algoritmo con mejor desempeño, corresponde analizar la mejora con respecto al algoritmo original que se tiene por objetivo superar y comparar la calidad de soluciones con respecto a resultados actuales de la literatura, de manera de tener una noción más completa de la calidad del algoritmo logrado.

5.3.1 Comparación con algoritmo original

En términos de calidad de solución el algoritmo MC-LTV-SD supera en todos los aspectos revisados al algoritmo original. El promedio de porcentaje de error es 0,118 % y el original tiene un valor de 0,136 %. La cantidad de instancias de porcentaje de error mínimo del algoritmo original es 17, superado por MC-LTV-SD con 23 instancias. Finalmente, el índice Copeland del algoritmo original es -67, siendo ampliamente superado por el nuevo algoritmo que posee un índice de 196.

Además, MC-LTV-SD está ubicado entre los que tienen mayor tasa de éxito de los comparados, con un promedio de 67,0 % de éxito (el máximo es 68,33 % y el mínimo 63,33 %), superando a 18 de los 23 otros algoritmos. Además, MC-LTV-SD tiene índice Copeland de 15 (el máximo es 75 y el mínimo -74), superando a 16 de los 23 otros algoritmos. En ambos aspectos se supera al algoritmo original, que tiene promedio de 64,0 % de éxito e índice de -53.

Otro aspecto a analizar es el tiempo de convergencia de MC-LTV-SD. En total se tienen 21111,55 segundos, lo cual corresponde a un 22,9 % extra del tiempo mínimo (17174,26 segundos) y un 13,5 % adicional del algoritmo original (18593,86 segundos). Además, por Copeland este algoritmo tampoco posee buen desempeño de rapidez, ya que tiene un índice de -72 contra un máximo de 179 y un mínimo de -129; y el original con 58.

Sin embargo, como se mencionó previamente el tiempo de convergencia no es de vital importancia, ya que se puede mejorar con técnicas de optimización a nivel de programación y corriendo el algoritmo en una máquina con *hardware* más potente.

Para visualizar la mejora de desempeño se tienen dos ejemplos:

- En la instancia *tai35a* MC-LTV-SD logra llegar el 100 % de las veces al BKV. En la figura 5-1 se aprecia la convergencia del algoritmo MC-LTV-SD y el memético original. El algoritmo original no logra llegar al valor con 0 % de error como la modificación realizada con MC, a pesar de tener una convergencia ligeramente más rápida que éste.

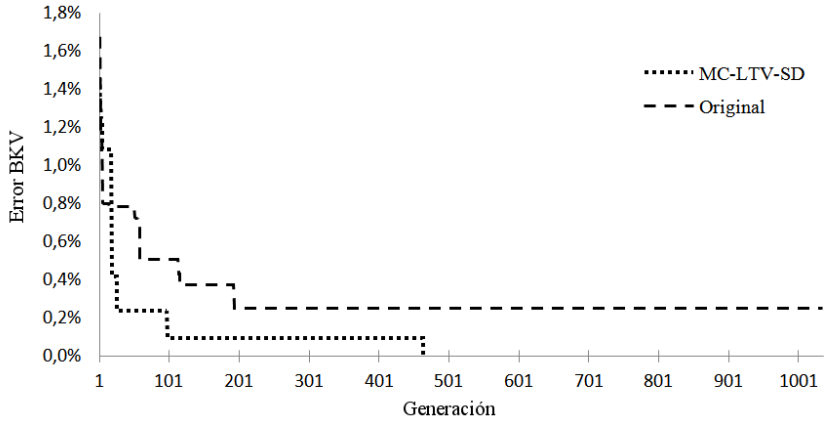

FIGURA 5-1: *Gráfico de convergencia para tai35a*

- En la instancia *tai100b* MC-LTV-SD logra llegar el 90 % de las veces al BKV. En la figura 5-2 se aprecia la convergencia del algoritmo MC-LTV-SD y el memético

original. El algoritmo original no logra llegar al valor con 0 % de error como la modificación realizada con MC.

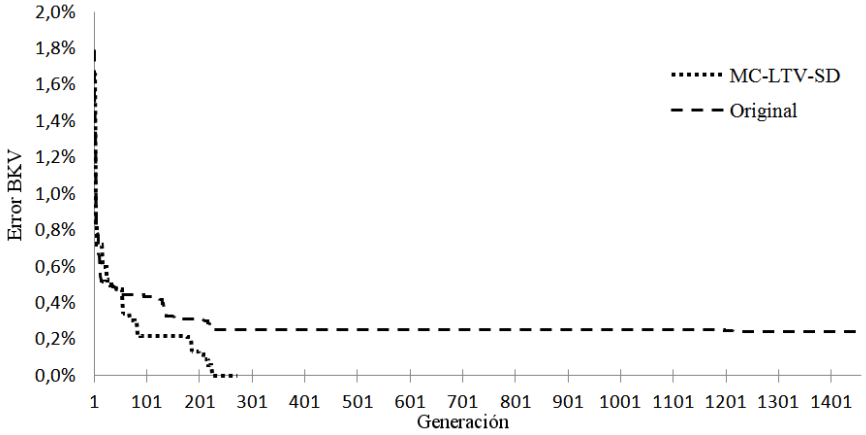

FIGURA 5-2: *Gráfico de convergencia para tai100b*

5.3.2 Comparación contra resultados actuales

De manera de validar la calidad de los datos anteriormente expuestos en el contexto actual de enfrentamiento a QAP, se compara el algoritmo memético que usa los esquemas MC-LTV-SD contra los resultados mostrados por Rego et al. (2010). En esa investigación crean tres variaciones de Búsqueda Tabú para enfrentar QAP, los resultados acá expuestos corresponden al algoritmo mencionado como EC3, el cual presenta mayor cantidad de instancias con menor porcentaje de error.

En la tabla 5.1 se pueden ver los resultados de la comparación de calidad de soluciones contra los resultados de Rego, que usan como criterio de término ejecución de 1 hora para las instancias cuyo $n \leq 40$ y 2 horas para las instancias cuyo $n > 40$. Los resultados de MC-LTV-SD son los mismos presentados en el capítulo anterior. Para *sko42* y para *sko100** se usaron 415 y 4000 segundos de tiempo máximo respectivamente.

TABLA 5.1: *Comparación de MC-LTV-SD contra resultados actuales*

Instancia	Rego	MC-LTV-SD
sko42	**0,000**	**0,000**
sko49	**0,039**	**0,000**
sko56	0,027	**0,000**
sko64	0,078	**0,000**
sko72	0,250	**0,000**
sko81	0,278	**0,011**
sko90	0,473	**0,006**
sko100a	0,340	**0,026**
sko100b	0,408	**0,002**
sko100c	0,543	**0,000**
sko100d	0,517	**0,038**
sko100e	0,460	**0,002**
sko100f	0,542	**0,041**
tai20a	**0,000**	**0,000**
tai25a	**0,000**	**0,000**
tai30a	**0,000**	**0,000**
tai35a	**0,000**	**0,000**
tai40a	**0,219**	0,241
tai50a	**0,514**	0,536
tai60a	0,657	**0,593**
tai80a	**0,730**	0,839
tai100a	0,729	**0,638**
Promedio	0,309	**0,135**

Como se puede apreciar de los resultados anteriores, el algoritmo MC-LTV-SD no sólo tiene buenos resultados con respecto al original, sino que además con respecto a soluciones actuales para QAP.

Con esta comparación se puede concluir la discusión de los resultados obtenidos de esta investigación. Primero se compararon los resultados de todos los algoritmos obtenidos en términos de calidad de soluciones, tasa de éxito y tiempo de convergencia, tanto en promedio como en índice Copeland. Luego se comparó los resultados obtenidos por la aplicación de cada esquema de memoria, siendo el esquema MC el único que mostró

una superioridad más notoria frente a los demás. Finalmente se determinó el esquema del mejor algoritmo, el cual fue comparado contra el algoritmo original y uno actual que también ataca QAP, mostrando superioridad en la mayoría de las instancias comparadas.

5.4 DIFERENCIA ENTRE ALGORITMOS

Las tablas presentadas en la sección 4.5 muestran los *p-values* entre las comparaciones de rendimiento de los algoritmos creados.

Como se puede apreciar, el algoritmo MC-LTV-SD elegido anteriormente como el de mejor rendimiento, tanto promedio (0,118 %) como en comparación mediante índices Copeland (196), coincide con el que tiene una mayor cantidad de diferencias significativas con (18 valores $p < 0,1$) al ser comparado contra los demás algoritmos. Esta coincidencia puede deberse a la similitud entre las formas de obtener el índice Copeland y los valores (R^+, R^-), ya que Copeland obtiene la diferencia entre las victorias y derrotas, y Wilcoxon obtiene sus R de acuerdo al ranking.

Se puede apreciar que el algoritmo MC-LTF-SD tiene la mayor cantidad de índices significativos con $p < 0,01$. A pesar de no tener el mejor promedio, está muy cercano a éste (0,120 % de error promedio) y posee el segundo mejor índice Copeland (190), coincidiendo además con ser el mejor algoritmo en cuanto a tasa de éxito (y segundo en índice Copeland en ese mismo aspecto).

De los algoritmos creados y sus comparaciones a parejas, 103 de las 276 comparaciones presentan diferencias significativas (*p-values* menores a 0,1). En estas comparaciones, la mayor diferencia se da entre los algoritmos con esquema MC contra los que usan PE o PEBC (de las 128 comparaciones en esta categoría, 81 presentan diferencias con *p-values* menores a 0,1). Esto puede deberse a que no hay diferencia entre el manejo de la pobla-

ción entre PE y PEBC, y se diferencian sólo en la forma de realizar sus búsquedas locales; en cambio MC utiliza una forma completamente diferente de estructura poblacional y su mantención.

Se puede apreciar que los algoritmos que usan un mismo esquema de memoria poblacional en general presentan poca diferencias entre ellos:

- Los algoritmos que usan MC tienen 3 pares de algoritmos con rendimiento significativamente diferente de las 28 comparaciones.
- Los algoritmos que usan PE tienen 3 pares de algoritmos con rendimiento significativamente diferente de las 28 comparaciones.
- Los algoritmos que usan PEBC tienen 6 pares de algoritmos con rendimiento significativamente diferente de las 28 comparaciones.

De los resultados analizados se puede ver que las mayores diferencias de rendimiento entre los algoritmos se dan por variaciones en los esquemas de memoria poblacionales o la composición de varios esquemas diferentes, ya que la comparación por esquemas aislados en general no otorga diferencias de rendimiento muy significativas como puede apreciarse en el Apéndice C.

CAPÍTULO 6. CONCLUSIONES

Los algoritmos meméticos, así como otras metaheurísticas basadas en poblaciones, son usados actualmente para enfrentar diversos problemas $NP-Completo$ y $NP-Hard$, ya que dan buena calidad de soluciones y en muchas ocasiones logran alcanzar resultados óptimos. Uno de los principales problemas que enfrentan estas técnicas de optimización es la convergencia prematura, la cual ha sido enfrentada con diversos métodos entre los que se tiene el uso de memoria.

En esta investigación, se propuso una definición de esquemas de memoria y clasificaciones a las estructuras que los conforman, de manera de solidificar la definición y análisis futuro de la memoria usada en algoritmos.

Luego de definidas las clasificaciones de estructuras de esquemas de memoria, se determinaron varios de ellos para ser aplicados y evaluados en un algoritmo memético para el problema de asignación cuadrática. Todos estos esquemas, con la excepción de *matriz de cruzamientos*, han sido usados exitosamente en investigaciones previas.

Dentro de los esquemas elegidos, no todos eran compatibles entre sí. En los esquemas compatibles de ser aplicados simultáneamente está la posibilidad tanto que se potencien en rendimiento, como que se perjudiquen; y los incompatibles hay unos que serán mejores que otros. Es por esto que se debió revisar varios aspectos: comparar el rendimiento de los algoritmos formados por sus diferentes combinaciones compatibles, determinar superioridad entre esquemas incompatibles y obtener una combinación de esquemas de memoria de rendimiento superior en términos de calidad de solución.

La aplicación de las diferentes combinaciones de esquemas de memoria, junto con la mejora a la búsqueda local mediante *Steepest Descent*, generó 23 variaciones de un

algoritmo memético original las que fueron evaluadas usando instancias existentes en QAPLIB y trabajadas previamente en Inostroza-Ponta (2008).

Los experimentos realizados mostraron superioridad de los algoritmos que usan el esquema MC, sobre los que usan PE y PEBC. Respecto al análisis de los algoritmos agrupados para comparar los otros esquemas incompatibles no se vio gran diferencia de rendimiento: al comparar LTF con LTV se obtuvieron valores muy cercanos en promedio y comparaciones individuales (índices Copeland), al igual que al comparar el uso y ausencia de reducción de espacio de búsqueda y mejora por SD, quedando demostrado por los *p-values* obtenidos al aplicar el test de Wilcoxon. Sin embargo, al revisar los resultados de todos los algoritmos, se obtuvieron 12 algoritmos superiores al original en términos de promedio de calidad de soluciones, 8 en cuanto a cantidad de instancias en que se obtiene el mejor valor promedio y 17 por índices Copeland. Además, se obtuvo un algoritmo superior, el cual correspondió al que usa matriz de cruzamientos con lista tabú variable y mejora de búsqueda local (MC-LTV-SD).

El algoritmo MC-LTV-SD se comparó contra el de la investigación realizada por Rego et al. (2010), de manera de mostrar la validez de los resultados obtenidos y ver cuanto mejor se está con respecto a la literatura.

El motivo de la poca diferencia entre las comparaciones descritas previamente puede deberse a diferentes factores. En cuanto al uso de LTF o LTV, el tamaño de las listas ($0,2 \times maxIteraciones$ para LTF y entre $[0,1; 0,5] \times maxIteraciones$ para LTV) al ser cercanos puede haber generado comportamientos similares. Con respecto a la reducción del espacio de búsqueda, no ocurrió una mejora sustancial por su aplicación, lo cual puede ser debido a la cantidad de iteraciones en que el algoritmo prohíbe el uso de asignaciones comunes entre los padres (cantidad igual al largo de la lista tabú correspondiente). Por lo mismo anterior, no parece haber una reducción excesiva de la cantidad de vecinos durante

la explotación por la aplicación conjunta de lista tabú y reducción, ya que el uso de SD luego de la búsqueda tabú no genera una mejora destacable de la calidad de soluciones encontradas. La poca diferencia de mejora por inclusión de reducción y SD puede ser debido a que los parámetros usados en los algoritmos (número de iteraciones, tamaños de listas, tamaño de población, número de bolsillos, etc) no permiten una mayor diferencia de la calidad de soluciones posibles de lograr.

Por lo anterior, entre otras cosas, se tienen diversos trabajos futuros posibles, por ejemplo:

- Explicar la influencia de los parámetros en los resultados de los algoritmos, de manera de parametrizarlos de mejor manera.
- Probar otros esquemas de memoria en el algoritmo memético analizado, tanto por separado como combinados a los tratados en esta investigación, de manera de buscar mayores mejoras.
- Probar los esquemas aplicados en otras metaheurísticas y problemas NP.
- Probar los esquemas aplicados en otras clasificaciones de duración, variabilidad de tamaño y cobertura. Por ejemplo, probar la matriz de cruzamientos a largo plazo, probar listas tabú poblacionales (en vez de sólo por agentes), etc.
- Aplicar los algoritmos logrados en un contexto no experimental, por ejemplo en desarrollo de *software* de algún ámbito que necesite enfrentar QAP.
- Está la posibilidad de hacer análisis matemáticos a las instancias QAP que enfrenta el algoritmo (determinación varianza de matrices de flujos/distancias, *random walks* en espacios de soluciones, dominancia de matrices, etc) de manera de hacer una parametrización adaptativa de acuerdo al tipo de instancia tratada.

- Mejora de los tiempos de ejecución de los algoritmos, mediante optimización de código, uso de un lenguaje más bajo nivel, uso de paralelismo, etc.

Finalmente, cabe destacar el cumplimiento de los objetivos propuestos para esta investigación, tanto en general como específicos. Se evaluó la mejora de convergencia por el uso de esquemas de memoria en el algoritmo memético que enfrenta QAP dado por Inostroza-Ponta (2008) en término de calidad de soluciones, cumpliendo para ello los objetivos específicos propuestos al inicio de la investigación. Además, con los resultados obtenidos se confirma la hipótesis planteada, ya que con varias combinaciones de esquemas de memoria se logra mejorar la calidad de soluciones del algoritmo memético original para QAP.

GLOSARIO

AM Algoritmo memético.

BKV Mejor valor encontrado (*Best Known Value*).

MC Matriz de cruzamientos.

GRASP Procedimiento de búsqueda adaptativa aleatoria golosa (*Greedy Randomized Adaptive Search Procedure*).

PE Población estructurada (sin búsqueda compartida).

PEBC Población estructurada con búsqueda compartida.

PSO Optimización basada en cúmulos de partículas (*Particle Swarm Optimization*).

QAP Problema de asignación cuadrática (*Quadratic Assignment Problem*).

SD Gradiente descendiente (*Steepest Descent*).

TS Búsqueda tabú (*Tabu Search*).

TSP Problema del vendedor viajero (*Traveling Salesman Problem*).

VNS Búsqueda en vecindad variable (*Variable Neighborhood Search*).

REFERENCIAS

Affenzeller, M., Wagner, S., & Winkler, S. (2007). Self-adaptive population size adjustment for genetic algorithms. En R. Moreno Diaz, F. Pichler, & A. Quesada Arencibia (Eds.) *Computer Aided Systems Theory*, vol. 4739 de *Lecture Notes in Computer Science*, 820–828. Springer Berlin / Heidelberg.

Ahmed, Z. H. (2010). Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. *International Journal of Biometrics & Bioinformatics (IJBB)*, *3*(6), 1985–2347.

Atkinson, R., & Shiffrin, R. (1968). Chapter: Human memory: A proposed system and its control processes. *The psychology of learning and motivation*, *2*, 89–195.

Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. En *IEEE Congress on Evolutionary Computation CEC99*, 1875–1882. IEEE Press.

Buriol, L., Franca, P., & Moscato, P. (2004). A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, *10*, 483–506.

Carrizo, J., Tinetti, F., & Moscato, P. (1992). A computational ecology for the quadratic assignment problem. En *21st Meeting on Informatics and Operations Research*.

Dawkins, R. (1976). *The Selfish Gene*. Clarendon Press, Oxford.

Deep, K., & Adane, H. (2011). New variations of order crossover for travelling salesman problem. *International Journal of Combinatorial Optimization Problems and Informatics*, *2*(1), 2–13.

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3–18.

Drezner, Z. (2005). Compounded genetic algorithms for the quadratic assignment problem. *Operations Research Letters*, *33*(5), 475 – 480.

Fescioglu-Unver, N., & Kokar, M. M. (2011). Self controlling tabu search algorithm for the quadratic assignment problem. *Comput. Ind. Eng.*, *60*, 310–319.

Fonlupt, C., Preux, P., & Robilliard, D. (1997). Preventing premature convergence via cooperating genetic algorithms. En *Proceedings Mandel'97*. Morgan Kaufmann.

Franca, P., Mendes, A., & Moscato, P. (1999). Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times. En *5th International Conference of the Decision Sciences Institute*, 1708–1710.

Gambardella, L., Taillard, D., & Dorigo, M. (1999). Ant colonies for the qap. *Journal of the Operational Research Society*, *50*, 167–176.

Glover, F. (1989). Tabu search–part i. *Informs Journal on Computing*, *1*(3), 190–206.

Glover, F. (1990). Tabu search–part ii. *Informs Journal on Computing*, *2*(1), 4–32.

Herrera, F., Lozano, M., & Sánchez, A. M. (2005). Hybrid crossover operators for real-coded genetic algorithms: an experimental study. *Soft Computing*, *9*, 280–298.

Holstein, D., & Moscato, P. (1999). *New Ideas in Optimization*, chap. Memetic algorithms using guided local search: a case study, 235–244. McGraw-Hill Ltd., UK.

Hong, T.-P., Wang, H.-S., & Chen, W.-C. (2000). Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics*, *6*, 439–455.

Hussin, M. S., & Stutzle, T. (2010). Tabu search vs. simulated annealing for solving large quadratic assignment instances. Reporte técnico IRIDIA/2010-020, Université Libre de Bruxelles.

Inostroza-Ponta, M. (2008). *An Integrated and Scalable Approach Based on Combinatorial Optimization Techniques for the Analysis of Microarray Data*. Tesis de doctorado, The University of Newcastle, Callaghan, Australia.

James, T., Rego, C., & Glover, F. (2009). Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE Transactions on Systems, Man and Cybernetics*, *39*, 579–596.

Juan, L., Zixing, C., & Jianqin, L. (2000). Premature convergence in genetic algorithm: analysis and prevention based on chaos operator. En *Proceedings of the 3rd World Congress on Intelligent Control and Automation.*, vol. 1, 495–499.

Karp, R. M. (1972). Reducibility among combinatorial problems. En R. E. Miller, & J. W. Thatcher (Eds.) *Complexity of Computer Computations*, IBM research symposia series, 85–103. Plenum Press.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. En *Proceedings of IEEE International Conference on Neural Networks*, vol. IV, 1942–1948.

Koopmans, T., & Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica*, *25*, 53–76.

Korejo, I., Yang, S., & Li, C. (2009). A comparative study of adaptive mutation operators for genetic algorithms. En *Proceedings of the 8th Metaheuristic International Conference*.

Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research*, *176*, 657–690.

Louis, S., & Li, G. (1997). Augmenting genetic algorithms with memory to solve traveling salesman problems. En *Proceedings of the Joint Conference on Information Sciences*, 108–111. Duke University Press.

Luke, S. (2009). *Essentials of Metaheuristics*, vol. 1. Lulu, 1 ed.

Merz, P., & Freisleben, B. (2000a). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, *4*(4), 337–352.

Merz, P., & Freisleben, B. (2000b). Fitness landscapes, memetic algorithms and greedy operators for graph bi-partitioning. *Evolutionary Computation*, *8*, 61–91.

Misevičius, A. (2005). A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, *30*, 95–111.

Misevičius, A. (2006). A fast hybrid genetic algorithm for the quadratic assignment problem. En *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, 1257–1264. New York, NY, USA: ACM.

Misevičius, A., & Kilda, B. (2005). Comparison of crossover operators for the quadratic assignment problem. *Information Technology and Control*, *34*(2), 109–119.

Misevičius, A., & Rubliauskas, D. (2009). Testing of hybrid genetic algorithms for structured quadratic assignment problems. *Informatica*, *20*, 255–272.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*, 1097–1100.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Reporte técnico 790, California Institute of Technology, Pasadena.

Moscato, P. (2003). A gentle introduction to memetic algorithms. En *Handbook of Metaheuristics*, 105–144. Kluwer Academic Publishers.

Moscato, P., & Cotta, C. (2003). An introduction to memetic algorithms. *Revista Iberoamericana de Inteligencia Artificial*, *19*, 131–148.

Odetayo, M. (1993). Optimal population size for genetic algorithms: an investigation. En *IEE Colloquium on Genetic Algorithms for the Control Systems Engineering*, 1 –4.

Paul, G. (2010). Comparative performance of tabu search and simulated annealing heuristics for the quadratic assignment problem. *Operations Research Letters*, *38*(6), 577–581.

Picek, S., & Golub, M. (2010). Comparison of a crossover operator in binary-coded genetic algorithms. *World Scientific and Engineering Academy and Society Transactions on Computers*, *9*(9), 1064–1073.

Ramkumar, A., Ponnambalam, S., & Jawahar, N. (2009). A population-based hybrid ant system for quadratic assignment formulations in facility layout design. *The International Journal of Advanced Manufacturing Technology*, *44*, 548–558.

Reeves, C. (1996). Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, *63*, 371–396.

Rego, C., James, T., & Glover, F. (2010). An ejection chain algorithm for the quadratic assignment problem. *Networks*, *56*(3), 188–206.

Sahni, S., & Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM*, *23*, 555–565.

Simoes, A., Costa, E., Pedro, R., & Nora, N. Q. (2007). Variable-size memory evolutionary algorithm to deal with dynamic environments. En *M. Giacobini et al. (Eds.): EvoWorkshops 2007, Applications of Evolutionary Computing, LNCS 4448*, 617–626. Springer Verlag.

Sivanandam, S., & Deepa, S. (2008). *Introduction to Genetic Algorithms*. Springer Berlin / Heidelberg.

Skorin-Kapov, J. (1990). Tabu search applied to the quadratic assignment problem. *Informs Journal on Computing*, *2*(1), 33–45.

Song, L., Lim, M., Suganthan, P., & Doan, V. (2009). Ensemble for solving quadratic assignment problems. En *International Conference of Soft Computing and Pattern Recognition*, 190 –195.

Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *Transactions on Systems, Man and Cybernetics*, *24*, 656–667.

Sudholt, D. (2009). The impact of parametrization in memetic evolutionary algorithms. *Theoretical Computer Science*, *410*(26), 2511–2528.

Sudholt, D. (2011). Hybridizing evolutionary algorithms with variable-depth search to overcome local optima. *Algorithmica*, *59*, 343–368.

Taguchi, T., Yokota, T., & Gen, M. (1998). Reliability optimal design problem with interval coeficients using hybrid genetic algorithms. *Computers & Industrial Engineering*, *35*, 373–376.

Taillard, E. D. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, *17*, 443–455.

Taillard, E. D., Gambardella, L. M., Gendreau, M., & Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, *135*(1), 1–16.

Talbi, E.-G. (2009). *Metaheuristics : from design to implementation*, vol. 10 de *The Sciences Po series in international relations and political economy*. John Wiley & Sons.

Tan, K., Lee, T., & Khor, E. (2001). Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, *5*(6), 565–588.

Tate, D., Tate, D. M., Smith, A. E., & Smith, A. E. (1995). A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, *22*, 73–83.

Tseng, L.-Y., & Liang, S.-C. (2006). A hybrid metaheuristic for the quadratic assignment problem. *Computational Optimization and Applications*, *34*, 85–113.

Vazquez, M., & Whitley, L. D. (2000). A hybrid genetic algorithm for the quadratic assignment problem. En *in GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, 169–178. Morgan Kaufmann.

Wang, R.-L., & Okazaki, K. (2005). Solving facility layout problem using an

improved genetic algorithm. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *E88-A*, 606–610.

Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, *63*, 325–336.

Wiering, M. (2004). Memory-based memetic algorithms. En *Proceedings of the Thirteenth Belgian-Dutch Conference on Machine Learning*, 191–198. Benelearn.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.

Yang, S. (2005). Memory-based immigrants for genetic algorithms in dynamic environments. En *Proceedings of the 2005 conference on Genetic and evolutionary computation*, 1115–1122. ACM.

Yeh, W. (2000). A memetic algorithm for the min k-cut problem. *Control and Intelligent Systems*, *28*, 47–55.

APÉNDICE A. DETALLE ÍNDICES COPELAND

En este apéndice se presenta el detalle de obtención de los índices Copeland de la calidad de soluciones, tasa de éxito de algoritmos, tiempo de ejecución de ellos y esquemas de memoria. En las tablas A.1 - A.9 se ve el detalle de obtención de índices Copeland. En las columnas se tienen los algoritmos a evaluar y en las filas los algoritmos contra los cuales se compara. Es decir, al comparar con la fórmula:

$$\sigma_m = \sum_{m'} c_{mm'} - c_{m'm} \tag{A.1}$$

Los algoritmos de las columnas serán m y los de las filas serán m'. Además, el total está dado por la suma de los valores de la columna, correspondiendo a σ_m.

A.1 DETALLE DE ÍNDICES COPELAND PARA CALIDAD DE SOLUCIONES

En las tablas A.1 - A.3 se aprecia el detalle de obtención de índices para a calidad de soluciones.

TABLA A.1: *Detalle de índices Copeland para calidad de soluciones para MC*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	0	3	7	2	2	1	8	4
	LTV-R	-3	0	7	4	-3	2	4	0
	LTV-SD	-7	-7	0	-8	-2	-7	-2	-2
	LTV-R-SD	-2	-4	8	0	3	0	3	3
	LTF	-2	3	2	-3	0	4	5	-1
	LTF-R	-1	-2	7	0	-4	0	7	-4
	LTF-SD	-8	-4	2	-3	-5	-7	0	-5
	LTF-R-SD	-4	0	2	-3	1	4	5	0
PE	**LTV**	3	4	10	10	4	1	5	3
	LTV-R	4	9	14	9	10	6	12	7
	LTV-SD	4	7	10	12	6	6	12	3
	LTV-R-SD	7	7	12	12	10	6	10	7
	LTF	6	7	10	10	6	6	10	5
	LTF-R	-3	1	7	3	2	-1	7	0
	LTF-SD	6	6	12	10	10	6	12	9
	LTF-R-SD	5	5	10	6	6	4	10	9
PEBC	**LTV**	8	6	8	10	6	6	8	5
	LTV-R	4	9	8	7	8	8	11	7
	LTV-SD	7	6	9	6	6	5	9	4
	LTV-R-SD	2	6	10	8	6	4	12	1
	LTF	6	12	11	11	10	11	13	6
	LTF-R	2	6	11	7	11	3	9	6
	LTF-SD	5	8	8	8	8	9	11	6
	LTF-R-SD	1	6	11	8	5	1	9	2
Total		40	94	196	126	106	78	190	75

TABLA A.2: *Detalle de índices Copeland para calidad de soluciones para PE*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-3	-4	-4	-7	-6	3	-6	-5
	LTV-R	-4	-9	-7	-7	-7	-1	-6	-5
	LTV-SD	-10	-14	-10	-12	-10	-7	-12	-10
	LTV-R-SD	-10	-9	-12	-12	-10	-3	-10	-6
	LTF	-4	-10	-6	-10	-6	-2	-10	-6
	LTF-R	-1	-6	-6	-6	-6	1	-6	-4
	LTF-SD	-5	-12	-12	-10	-10	-7	-12	-10
	LTF-R-SD	-3	-7	-3	-7	-5	0	-9	-9
PE	**LTV**	0	-2	3	-3	1	6	2	3
	LTV-R	2	0	4	-3	1	6	-1	6
	LTV-SD	-3	-4	0	-3	-5	7	-1	-3
	LTV-R-SD	3	3	3	0	-1	11	2	5
	LTF	-1	-1	5	1	0	9	1	1
	LTF-R	-6	-6	-7	-11	-9	0	-6	-7
	LTF-SD	-2	1	1	-2	-1	6	0	3
	LTF-R-SD	-3	-6	3	-5	-1	7	-3	0
PEBC	**LTV**	0	0	2	-6	0	6	3	1
	LTV-R	-2	-4	-2	-2	0	1	-6	-3
	LTV-SD	-1	-1	0	-3	-3	8	2	3
	LTV-R-SD	-2	-5	-3	-6	-7	6	-3	-3
	LTF	2	-2	2	-2	2	8	0	6
	LTF-R	-2	-6	3	-4	0	6	2	0
	LTF-SD	-1	-1	5	-3	-1	6	3	-1
	LTF-R-SD	2	-2	1	-6	-2	6	2	-3
Total		-54	-107	-40	-129	-86	83	-74	-47

TABLA A.3: *Detalle de índices Copeland para calidad de soluciones para PEBC*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-8	-4	-7	-2	-6	-2	-5	-1
	LTV-R	-6	-9	-6	-6	-12	-6	-8	-6
	LTV-SD	-8	-8	-9	-10	-11	-11	-8	-11
	LTV-R-SD	-10	-7	-6	-8	-11	-7	-8	-8
	LTF	-6	-8	-6	-6	-10	-11	-8	-5
	LTF-R	-6	-8	-5	-4	-11	-3	-9	-1
	LTF-SD	-8	-11	-9	-12	-13	-9	-11	-9
	LTF-R-SD	-5	-7	-4	-1	-6	-6	-6	-2
PE	**LTV**	0	2	1	2	-2	2	1	-2
	LTV-R	0	4	1	5	2	6	1	2
	LTV-SD	-2	2	0	3	-2	-3	-5	-1
	LTV-R-SD	6	2	3	6	2	4	3	6
	LTF	0	0	3	7	-2	0	1	2
	LTF-R	-6	-1	-8	-6	-8	-6	-6	-6
	LTF-SD	-3	6	-2	3	0	-2	-3	-2
	LTF-R-SD	-1	3	-3	3	-6	0	1	3
PEBC	**LTV**	0	2	0	3	-2	1	-1	1
	LTV-R	-2	0	-1	2	-2	-1	-1	2
	LTV-SD	0	1	0	2	3	0	-1	-2
	LTV-R-SD	-3	-2	-2	0	-4	-2	-3	-1
	LTF	2	2	-3	4	0	5	0	3
	LTF-R	-1	1	0	2	-5	0	0	-2
	LTF-SD	1	1	1	3	0	0	0	2
	LTF-R-SD	-1	-2	2	1	-3	2	-2	0
Total		-67	-41	-60	-9	-109	-49	-78	-38

A.2 DETALLE DE ÍNDICES COPELAND PARA TASA DE ÉXITO

En las tablas A.4 - A.6 se puede ver el detalle de obtención de índices Copeland para la tasa de éxito.

TABLA A.4: *Detalle de índices Copeland para tasa de éxito para MC*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	0	-3	-2	-1	-1	-5	-1	-3
	LTV-R	3	0	0	2	1	-1	2	-1
	LTV-SD	2	0	0	1	1	-1	1	-1
	LTV-R-SD	1	-2	-1	0	1	-3	1	-2
	LTF	1	-1	-1	-1	0	-2	2	-1
	LTF-R	5	1	1	3	2	0	1	1
	LTF-SD	1	-2	-1	-1	-2	-1	0	-3
	LTF-R-SD	3	1	1	2	1	-1	3	0
PE	**LTV**	5	5	4	7	4	2	4	2
	LTV-R	5	4	1	5	4	2	5	3
	LTV-SD	4	3	3	5	1	2	3	0
	LTV-R-SD	5	3	0	4	3	2	6	2
	LTF	4	4	1	4	4	-1	4	1
	LTF-R	1	0	-2	1	0	-3	1	-2
	LTF-SD	6	6	1	4	2	2	5	3
	LTF-R-SD	4	2	2	5	3	0	3	1
PEBC	**LTV**	5	5	1	4	2	1	5	2
	LTV-R	1	-1	-1	1	-1	-2	1	-3
	LTV-SD	4	3	-1	1	2	-1	3	2
	LTV-R-SD	2	4	2	2	3	-1	5	0
	LTF	5	4	1	5	3	1	5	1
	LTF-R	3	2	4	3	4	-1	4	2
	LTF-SD	3	3	2	2	1	1	5	0
	LTF-R-SD	2	2	0	2	-1	1	5	-2
Total		75	43	15	60	37	-9	73	2

TABLA A.5: *Detalle de índices Copeland para tasa de éxito para PE*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-5	-5	-4	-5	-4	-1	-6	-4
	LTV-R	-5	-4	-3	-3	-4	0	-6	-2
	LTV-SD	-4	-1	-3	0	-1	2	-1	-2
	LTV-R-SD	-7	-5	-5	-4	-4	-1	-4	-5
	LTF	-4	-4	-1	-3	-4	0	-2	-3
	LTF-R	-2	-2	-2	-2	1	3	-2	0
	LTF-SD	-4	-5	-3	-6	-4	-1	-5	-3
	LTF-R-SD	-2	-3	0	-2	-1	2	-3	-1
PE	**LTV**	0	2	2	3	1	6	1	4
	LTV-R	-2	0	3	-1	-1	4	-2	3
	LTV-SD	-2	-3	0	-1	-2	2	-3	1
	LTV-R-SD	-3	1	1	0	2	5	1	2
	LTF	-1	1	2	-2	0	2	-2	2
	LTF-R	-6	-4	-2	-5	-2	0	-4	-1
	LTF-SD	-1	2	3	-1	2	4	0	3
	LTF-R-SD	-4	-3	-1	-2	-2	1	-3	0
PEBC	**LTV**	-2	1	3	0	2	3	-1	2
	LTV-R	-5	-5	-3	-4	-4	-2	-6	-3
	LTV-SD	-3	1	0	-3	1	1	-2	0
	LTV-R-SD	-3	0	2	-1	-3	0	-1	-1
	LTF	-3	-1	2	-2	0	2	-2	2
	LTF-R	-2	0	1	-2	-2	2	1	0
	LTF-SD	0	-1	3	-2	2	2	-1	1
	LTF-R-SD	-4	-3	1	-3	-2	1	-2	0
Total		-74	-41	-4	-51	-29	37	-55	-5

TABLA A.6: *Detalle de índices Copeland para tasa de éxito para PEBC*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-5	-1	-4	-2	-5	-3	-3	-2
	LTV-R	-5	1	-3	-4	-4	-2	-3	-2
	LTV-SD	-1	1	1	-2	-1	-4	-2	0
	LTV-R-SD	-4	-1	-1	-2	-5	-3	-2	-2
	LTF	-2	1	-2	-3	-3	-4	-1	1
	LTF-R	-1	2	1	1	-1	1	-1	-1
	LTF-SD	-5	-1	-3	-5	-5	-4	-5	-5
	LTF-R-SD	-2	3	-2	0	-1	-2	0	2
PE	**LTV**	2	5	3	3	3	2	0	4
	LTV-R	-1	5	-1	0	1	0	1	3
	LTV-SD	-3	3	0	-2	-2	-1	-3	-1
	LTV-R-SD	0	4	3	1	2	2	2	3
	LTF	-2	4	-1	3	0	2	-2	2
	LTF-R	-3	2	-1	0	-2	-2	-2	-1
	LTF-SD	1	6	2	1	2	-1	1	2
	LTF-R-SD	-2	3	0	1	-2	0	-1	0
PEBC	**LTV**	0	6	2	3	2	2	1	4
	LTV-R	-6	0	-3	-3	-5	-2	-5	-3
	LTV-SD	-2	3	0	-1	0	-2	0	0
	LTV-R-SD	-3	3	1	0	-2	-2	-2	3
	LTF	-2	5	0	2	0	2	-1	2
	LTF-R	-2	2	2	2	-2	0	0	2
	LTF-SD	-1	5	0	2	1	0	0	3
	LTF-R-SD	-4	3	0	-3	-2	-2	-3	0
Total		-53	64	-6	-8	-31	-23	-31	14

A.3 DETALLE DE ÍNDICES COPELAND PARA TIEMPO DE CONVERGENCIA

En las tablas A.7 - A.9 se puede ver el detalle de obtención de índices Copeland para el tiempo de convergencia de los algoritmos.

TABLA A.7: *Detalle de índices Copeland para tiempo de convergencia para MC*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	0	6	0	4	0	0	-2	0
	LTV-R	-6	0	0	0	-12	-2	-2	-8
	LTV-SD	0	0	0	2	0	4	2	2
	LTV-R-SD	-4	0	-2	0	-9	-9	-2	-8
	LTF	0	12	0	9	0	-6	12	2
	LTF-R	0	2	-4	9	6	0	6	6
	LTF-SD	2	2	-2	2	-12	-6	0	-6
	LTF-R-SD	0	8	-2	8	-2	-6	6	0
PE	**LTV**	-10	-4	-10	-4	-8	-16	-12	-10
	LTV-R	-4	0	-4	-4	-10	-8	-4	-6
	LTV-SD	-4	-4	-6	-2	-2	-12	-6	-4
	LTV-R-SD	-15	-10	-8	-12	-12	-14	-14	-11
	LTF	-4	0	0	0	-4	-4	-4	-4
	LTF-R	0	4	2	4	-2	-2	8	-8
	LTF-SD	-8	-2	-2	0	-4	-4	4	-6
	LTF-R-SD	-8	4	-4	0	0	-6	0	-4
PEBC	**LTV**	-2	-6	-6	2	-10	-14	-2	-12
	LTV-R	-8	-4	-8	-4	-12	-10	-6	-8
	LTV-SD	2	2	4	4	-2	4	-2	-6
	LTV-R-SD	-6	-8	-6	-4	-20	-16	-8	-16
	LTF	-2	6	-4	2	-6	-6	-6	-6
	LTF-R	-8	2	-4	2	2	-6	-4	-4
	LTF-SD	2	2	2	-2	-2	-2	-2	-6
	LTF-R-SD	-10	-4	-8	-3	-4	-10	-2	-6
Total		-93	8	-72	13	-125	-151	-40	-129

TABLA A.8: *Detalle de índices Copeland para tiempo de convergencia para PE*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	10	4	4	15	4	0	8	8
	LTV-R	4	0	4	10	0	-4	2	-4
	LTV-SD	10	4	6	8	0	-2	2	4
	LTV-R-SD	4	4	2	12	0	-4	0	0
	LTF	8	10	2	12	4	2	4	0
	LTF-R	16	8	12	14	4	2	4	6
	LTF-SD	12	4	6	14	4	-8	-4	0
	LTF-R-SD	10	6	4	11	4	8	6	4
PE	**LTV**	0	0	-4	-4	-8	-4	2	-2
	LTV-R	0	0	-2	2	2	-4	2	-6
	LTV-SD	4	2	0	4	0	-10	3	-2
	LTV-R-SD	4	-2	-4	0	-4	-12	-4	-10
	LTF	8	-2	0	4	0	-6	0	-2
	LTF-R	4	4	10	12	6	0	8	2
	LTF-SD	-2	-2	-3	4	0	-8	0	2
	LTF-R-SD	2	6	2	10	2	-2	-2	0
PEBC	**LTV**	0	2	-6	8	0	-2	-4	-2
	LTV-R	-4	-4	-4	2	-4	-12	-4	-6
	LTV-SD	8	0	8	10	2	-4	0	4
	LTV-R-SD	2	-10	0	1	-2	-16	-4	-10
	LTF	8	0	10	18	4	-8	-4	6
	LTF-R	0	4	-4	4	-8	-8	-8	1
	LTF-SD	14	2	4	10	2	-2	8	2
	LTF-R-SD	2	-4	2	-2	-2	-4	0	-2
Total		124	36	49	179	10	-108	15	-7

TABLA A.9: *Detalle de índices Copeland para tiempo de convergencia para PEBC*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	2	8	-2	6	2	8	-2	10
	LTV-R	6	4	-2	8	-6	-2	-2	4
	LTV-SD	6	8	-4	6	4	4	-2	8
	LTV-R-SD	-2	4	-4	4	-2	-2	2	3
	LTF	10	12	2	20	6	-2	2	4
	LTF-R	14	10	-4	16	6	6	2	10
	LTF-SD	2	6	2	8	6	4	2	2
	LTF-R-SD	12	8	6	16	6	4	6	6
PE	**LTV**	0	4	-8	-2	-8	0	-14	-2
	LTV-R	-2	4	0	10	0	-4	-2	4
	LTV-SD	6	4	-8	0	-10	4	-4	-2
	LTV-R-SD	-8	-2	-10	-1	-18	-4	-10	2
	LTF	0	4	-2	2	-4	8	-2	2
	LTF-R	2	12	4	16	8	8	2	4
	LTF-SD	4	4	0	4	4	8	-8	0
	LTF-R-SD	2	6	-4	10	-6	-1	-2	2
PEBC	**LTV**	0	-2	-2	2	-6	0	0	4
	LTV-R	2	0	-4	2	-8	-4	-6	0
	LTV-SD	2	4	0	10	-2	10	-2	10
	LTV-R-SD	-2	-2	-10	0	-12	0	-10	-2
	LTF	6	8	2	12	0	10	2	8
	LTF-R	0	4	-10	0	-10	0	-10	-4
	LTF-SD	0	6	2	10	-2	10	0	8
	LTF-R-SD	-4	0	-10	2	-8	4	-8	0
Total		58	114	-66	161	-60	69	-66	81

A.4 DETALLE DE ÍNDICES COPELAND PARA ESQUEMAS DE MEMORIA

En esta sección se verá el detalle de la obtención de los índices Copeland para los esquemas de memoria aplicados. Para esto, se tomarán las tablas A.1 - A.3 y se quitarán los elementos que no se deben comparar para un determinado análisis.

Para obtener los índices de los esquemas poblacionales, deberán quitarse los que comparen un mismo tipo de esquema. Por ejemplo no se deben considerar los que

comparen algoritmos que usen MC entre ellos. De manera similar deben quitarse luego los que tengan igual tipo de búsqueda tabú, usen o no usen reducción de espacio de búsqueda y gradiente descendiente.

A.4.1 Esquemas poblacionales

En esta sección se analizan los esquemas aplicados a tareas de población dentro del algoritmo memético. Primero se comparan los 3 esquemas probados y luego se compara el beneficio de usar búsqueda compartida.

A.4.1.1 Todos los esquemas poblacionales

En las tablas A.10 - A.12 se puede apreciar los índices Copeland para las parejas de algoritmos en sus 30 instancias probadas. Notar que hay elementos ausentes, marcados con "-", los cuales son omitidos para no considerar la comparación entre algoritmos que usan el mismo tipo de esquema de memoria poblacional.

TABLA A.10: *Detalle de índices Copeland para esquemas poblacionales (1)*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-	-	-	-	-	-	-
	LTV-R	-	-	-	-	-	-	-	-
	LTV-SD	-	-	-	-	-	-	-	-
	LTV-R-SD	-	-	-	-	-	-	-	-
	LTF	-	-	-	-	-	-	-	-
	LTF-R	-	-	-	-	-	-	-	-
	LTF-SD	-	-	-	-	-	-	-	-
	LTF-R-SD	-	-	-	-	-	-	-	-
PE	**LTV**	3	4	10	10	4	1	5	3
	LTV-R	4	9	14	9	10	6	12	7
	LTV-SD	4	7	10	12	6	6	12	3
	LTV-R-SD	7	7	12	12	10	6	10	7
	LTF	6	7	10	10	6	6	10	5
	LTF-R	-3	1	7	3	2	-1	7	0
	LTF-SD	6	6	12	10	10	6	12	9
	LTF-R-SD	5	5	10	6	6	4	10	9
PEBC	**LTV**	8	6	8	10	6	6	8	5
	LTV-R	4	9	8	7	8	8	11	7
	LTV-SD	7	6	9	6	6	5	9	4
	LTV-R-SD	2	6	10	8	6	4	12	1
	LTF	6	12	11	11	10	11	13	6
	LTF-R	2	6	11	7	11	3	9	6
	LTF-SD	5	8	8	8	8	9	11	6
	LTF-R-SD	1	6	11	8	5	1	9	2
Total		67	105	161	137	114	81	160	80

TABLA A.11: *Detalle de índices Copeland para esquemas poblacionales (2)*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-3	-4	-4	-7	-6	3	-6	-5
	LTV-R	-4	-9	-7	-7	-7	-1	-6	-5
	LTV-SD	-10	-14	-10	-12	-10	-7	-12	-10
	LTV-R-SD	-10	-9	-12	-12	-10	-3	-10	-6
	LTF	-4	-10	-6	-10	-6	-2	-10	-6
	LTF-R	-1	-6	-6	-6	-6	1	-6	-4
	LTF-SD	-5	-12	-12	-10	-10	-7	-12	-10
	LTF-R-SD	-3	-7	-3	-7	-5	0	-9	-9
PE	**LTV**	-	-	-	-	-	-	-	-
	LTV-R	-	-	-	-	-	-	-	-
	LTV-SD	-	-	-	-	-	-	-	-
	LTV-R-SD	-	-	-	-	-	-	-	-
	LTF	-	-	-	-	-	-	-	-
	LTF-R	-	-	-	-	-	-	-	-
	LTF-SD	-	-	-	-	-	-	-	-
	LTF-R-SD	-	-	-	-	-	-	-	-
PEBC	**LTV**	0	0	2	-6	0	6	3	1
	LTV-R	-2	-4	-2	-2	0	1	-6	-3
	LTV-SD	-1	-1	0	-3	-3	8	2	3
	LTV-R-SD	-2	-5	-3	-6	-7	6	-3	-3
	LTF	2	-2	2	-2	2	8	0	6
	LTF-R	-2	-6	3	-4	0	6	2	0
	LTF-SD	-1	-1	5	-3	-1	6	3	-1
	LTF-R-SD	2	-2	1	-6	-2	6	2	-3
Total		-44	-92	-52	-103	-71	31	-68	-55

TABLA A.12: *Detalle de índices Copeland para esquemas poblacionales (3)*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-8	-4	-7	-2	-6	-2	-5	-1
	LTV-R	-6	-9	-6	-6	-12	-6	-8	-6
	LTV-SD	-8	-8	-9	-10	-11	-11	-8	-11
	LTV-R-SD	-10	-7	-6	-8	-11	-7	-8	-8
	LTF	-6	-8	-6	-6	-10	-11	-8	-5
	LTF-R	-6	-8	-5	-4	-11	-3	-9	-1
	LTF-SD	-8	-11	-9	-12	-13	-9	-11	-9
	LTF-R-SD	-5	-7	-4	-1	-6	-6	-6	-2
PE	**LTV**	0	2	1	2	-2	2	1	-2
	LTV-R	0	4	1	5	2	6	1	2
	LTV-SD	-2	2	0	3	-2	-3	-5	-1
	LTV-R-SD	6	2	3	6	2	4	3	6
	LTF	0	0	3	7	-2	0	1	2
	LTF-R	-6	-1	-8	-6	-8	-6	-6	-6
	LTF-SD	-3	6	-2	3	0	-2	-3	-2
	LTF-R-SD	-1	3	-3	3	-6	0	1	3
PEBC	**LTV**	-	-	-	-	-	-	-	-
	LTV-R	-	-	-	-	-	-	-	-
	LTV-SD	-	-	-	-	-	-	-	-
	LTV-R-SD	-	-	-	-	-	-	-	-
	LTF	-	-	-	-	-	-	-	-
	LTF-R	-	-	-	-	-	-	-	-
	LTF-SD	-	-	-	-	-	-	-	-
	LTF-R-SD	-	-	-	-	-	-	-	-
Total		-63	-44	-57	-26	-96	-54	-70	-41

A.4.1.2 Esquemas poblacionales estructurados

En las tablas A.13 y A.14 se pueden ver los índices parciales comparando los algoritmos que tienen población estructurada y tienen tanto uso como ausencia de búsqueda compartida entre soluciones de cada agente.

TABLA A.13: *Detalle de índices Copeland para esquemas poblacionales estructurados (1)*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
PE	**LTV**	-	-	-	-	-	-	-	-
	LTV-R	-	-	-	-	-	-	-	-
	LTV-SD	-	-	-	-	-	-	-	-
	LTV-R-SD	-	-	-	-	-	-	-	-
	LTF	-	-	-	-	-	-	-	-
	LTF-R	-	-	-	-	-	-	-	-
	LTF-SD	-	-	-	-	-	-	-	-
	LTF-R-SD	-	-	-	-	-	-	-	-
PEBC	**LTV**	0	0	2	-6	0	6	3	1
	LTV-R	-2	-4	-2	-2	0	1	-6	-3
	LTV-SD	-1	-1	0	-3	-3	8	2	3
	LTV-R-SD	-2	-5	-3	-6	-7	6	-3	-3
	LTF	2	-2	2	-2	2	8	0	6
	LTF-R	-2	-6	3	-4	0	6	2	0
	LTF-SD	-1	-1	5	-3	-1	6	3	-1
	LTF-R-SD	2	-2	1	-6	-2	6	2	-3
Total		-4	-21	8	-32	-11	47	3	0

TABLA A.14: *Detalle de índices Copeland para esquemas poblacionales estructurados (2)*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
PE	**LTV**	0	2	1	2	-2	2	1	-2
	LTV-R	0	4	1	5	2	6	1	2
	LTV-SD	-2	2	0	3	-2	-3	-5	-1
	LTV-R-SD	6	2	3	6	2	4	3	6
	LTF	0	0	3	7	-2	0	1	2
	LTF-R	-6	-1	-8	-6	-8	-6	-6	-6
	LTF-SD	-3	6	-2	3	0	-2	-3	-2
	LTF-R-SD	-1	3	-3	3	-6	0	1	3
PEBC	**LTV**	-	-	-	-	-	-	-	-
	LTV-R	-	-	-	-	-	-	-	-
	LTV-SD	-	-	-	-	-	-	-	-
	LTV-R-SD	-	-	-	-	-	-	-	-
	LTF	-	-	-	-	-	-	-	-
	LTF-R	-	-	-	-	-	-	-	-
	LTF-SD	-	-	-	-	-	-	-	-
	LTF-R-SD	-	-	-	-	-	-	-	-
Total		-6	18	-5	23	-16	1	-7	2

A.4.2 Tipo de Búsqueda Tabú

En las tablas A.15 - A.17 se ven los índices de cada pareja de algoritmos, omitiendo las comparaciones entre algoritmos que usan el mismo tipo de búsqueda tabú. Esto quiere decir que no se comparan las parejas de algoritmos que usen LTF y lo mismo para LTV.

TABLA A.15: *Detalle de índices Copeland para tipos TS (1)*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-	-	-	2	1	8	4
	LTV-R	-	-	-	-	-3	2	4	0
	LTV-SD	-	-	-	-	-2	-7	-2	-2
	LTV-R-SD	-	-	-	-	3	0	3	3
	LTF	-2	3	2	-3	-	-	-	-
	LTF-R	-1	-2	7	0	-	-	-	-
	LTF-SD	-8	-4	2	-3	-	-	-	-
	LTF-R-SD	-4	0	2	-3	-	-	-	-
PE	**LTV**	-	-	-	-	4	1	5	3
	LTV-R	-	-	-	-	10	6	12	7
	LTV-SD	-	-	-	-	6	6	12	3
	LTV-R-SD	-	-	-	-	10	6	10	7
	LTF	6	7	10	10	-	-	-	-
	LTF-R	-3	1	7	3	-	-	-	-
	LTF-SD	6	6	12	10	-	-	-	-
	LTF-R-SD	5	5	10	6	-	-	-	-
PEBC	**LTV**	-	-	-	-	6	6	8	5
	LTV-R	-	-	-	-	8	8	11	7
	LTV-SD	-	-	-	-	6	5	9	4
	LTV-R-SD	-	-	-	-	6	4	12	1
	LTF	6	12	11	11	-	-	-	-
	LTF-R	2	6	11	7	-	-	-	-
	LTF-SD	5	8	8	8	-	-	-	-
	LTF-R-SD	1	6	11	8	-	-	-	-
Total		13	48	93	54	56	38	92	42

TABLA A.16: *Detalle de índices Copeland para tipos TS (2)*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-	-	-	-6	3	-6	-5
	LTV-R	-	-	-	-	-7	-1	-6	-5
	LTV-SD	-	-	-	-	-10	-7	-12	-10
	LTV-R-SD	-	-	-	-	-10	-3	-10	-6
	LTF	-4	-10	-6	-10	-	-	-	-
	LTF-R	-1	-6	-6	-6	-	-	-	-
	LTF-SD	-5	-12	-12	-10	-	-	-	-
	LTF-R-SD	-3	-7	-3	-7	-	-	-	-
PE	**LTV**	-	-	-	-	1	6	2	3
	LTV-R	-	-	-	-	1	6	-1	6
	LTV-SD	-	-	-	-	-5	7	-1	-3
	LTV-R-SD	-	-	-	-	-1	11	2	5
	LTF	-1	-1	5	1	-	-	-	-
	LTF-R	-6	-6	-7	-11	-	-	-	-
	LTF-SD	-2	1	1	-2	-	-	-	-
	LTF-R-SD	-3	-6	3	-5	-	-	-	-
PEBC	**LTV**	-	-	-	-	0	6	3	1
	LTV-R	-	-	-	-	0	1	-6	-3
	LTV-SD	-	-	-	-	-3	8	2	3
	LTV-R-SD	-	-	-	-	-7	6	-3	-3
	LTF	2	-2	2	-2	-	-	-	-
	LTF-R	-2	-6	3	-4	-	-	-	-
	LTF-SD	-1	-1	5	-3	-	-	-	-
	LTF-R-SD	2	-2	1	-6	-	-	-	-
Total		-24	-58	-14	-65	-47	43	-36	-17

TABLA A.17: *Detalle de índices Copeland para tipos TS (3)*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-	-	-	-6	-2	-5	-1
	LTV-R	-	-	-	-	-12	-6	-8	-6
	LTV-SD	-	-	-	-	-11	-11	-8	-11
	LTV-R-SD	-	-	-	-	-11	-7	-8	-8
	LTF	-6	-8	-6	-6	-	-	-	-
	LTF-R	-6	-8	-5	-4	-	-	-	-
	LTF-SD	-8	-11	-9	-12	-	-	-	-
	LTF-R-SD	-5	-7	-4	-1	-	-	-	-
PE	**LTV**	-	-	-	-	-2	2	1	-2
	LTV-R	-	-	-	-	2	6	1	2
	LTV-SD	-	-	-	-	-2	-3	-5	-1
	LTV-R-SD	-	-	-	-	2	4	3	6
	LTF	0	0	3	7	-	-	-	-
	LTF-R	-6	-1	-8	-6	-	-	-	-
	LTF-SD	-3	6	-2	3	-	-	-	-
	LTF-R-SD	-1	3	-3	3	-	-	-	-
PEBC	**LTV**	0	2	0	3	-2	1	-1	1
	LTV-R	-2	0	-1	2	-2	-1	-1	2
	LTV-SD	0	1	0	2	3	0	-1	-2
	LTV-R-SD	-3	-2	-2	0	-4	-2	-3	-1
	LTF	2	2	-3	4	-	-	-	-
	LTF-R	-1	1	0	2	-	-	-	-
	LTF-SD	1	1	1	3	-	-	-	-
	LTF-R-SD	-1	-2	2	1	-	-	-	-
Total		-39	-23	-37	1	-45	-19	-35	-21

A.4.3 Reducción de espacio de búsqueda

En las tablas A.18 - A.20 se ven los índices de cada pareja de algoritmos, omitiendo las comparaciones entre algoritmos que usan reducción de espacio de búsqueda y lo mismo para el caso contrario. Por ejemplo, no se compara LTV-R con LTF-R, ya que ambos aplican reducción de espacio de búsqueda.

TABLA A.18: *Detalle de índices Copeland para reducción de espacio de búsqueda (1)*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	3	-	2	-	1	-	4
	LTV-R	-3	-	7	-	-3	-	4	-
	LTV-SD	-	-7	-	-8	-	-7	-	-2
	LTV-R-SD	-2	-	8	-	3	-	3	-
	LTF	-	3	-	-3	-	4	-	-1
	LTF-R	-1	-	7	-	-4	-	7	-
	LTF-SD	-	-4	-	-3	-	-7	-	-5
	LTF-R-SD	-4	-	2	-	1	-	5	-
PE	**LTV**	-	4	-	10	-	1	-	3
	LTV-R	4	-	14	-	10	-	12	-
	LTV-SD	-	7	-	12	-	6	-	3
	LTV-R-SD	7	-	12	-	10	-	10	-
	LTF	-	7	-	10	-	6	-	5
	LTF-R	-3	-	7	-	2	-	7	-
	LTF-SD	-	6	-	10	-	6	-	9
	LTF-R-SD	5	-	10	-	6	-	10	-
PEBC	**LTV**	-	6	-	10	-	6	-	5
	LTV-R	4	-	8	-	8	-	11	-
	LTV-SD	-	6	-	6	-	5	-	4
	LTV-R-SD	2	-	10	-	6	-	12	-
	LTF	-	12	-	11	-	11	-	6
	LTF-R	2	-	11	-	11	-	9	-
	LTF-SD	-	8	-	8	-	9	-	6
	LTF-R-SD	1	-	11	-	5	-	9	-
Total		12	51	107	65	55	41	99	37

TABLA A.19: *Detalle de índices Copeland para reducción de espacio de búsqueda (2)*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-4	-	-7	-	3	-	-5
	LTV-R	-4	-	-7	-	-7	-	-6	-
	LTV-SD	-	-14	-	-12	-	-7	-	-10
	LTV-R-SD	-10	-	-12	-	-10	-	-10	-
	LTF	-	-10	-	-10	-	-2	-	-6
	LTF-R	-1	-	-6	-	-6	-	-6	-
	LTF-SD	-	-12	-	-10	-	-7	-	-10
	LTF-R-SD	-3	-	-3	-	-5	-	-9	-
PE	**LTV**	-	-2	-	-3	-	6	-	3
	LTV-R	2	-	4	-	1	-	-1	-
	LTV-SD	-	-4	-	-3	-	7	-	-3
	LTV-R-SD	3	-	3	-	-1	-	2	-
	LTF	-	-1	-	1	-	9	-	1
	LTF-R	-6	-	-7	-	-9	-	-6	-
	LTF-SD	-	1	-	-2	-	6	-	3
	LTF-R-SD	-3	-	3	-	-1	-	-3	-
PEBC	**LTV**	-	0	-	-6	-	6	-	1
	LTV-R	-2	-	-2	-	0	-	-6	-
	LTV-SD	-	-1	-	-3	-	8	-	3
	LTV-R-SD	-2	-	-3	-	-7	-	-3	-
	LTF	-	-2	-	-2	-	8	-	6
	LTF-R	-2	-	3	-	0	-	2	-
	LTF-SD	-	-1	-	-3	-	6	-	-1
	LTF-R-SD	2	-	1	-	-2	-	2	-
Total		-26	-50	-26	-60	-47	43	-44	-18

TABLA A.20: *Detalle de índices Copeland para reducción de espacio de búsqueda (3)*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-4	-	-2	-	-2	-	-1
	LTV-R	-6	-	-6	-	-12	-	-8	-
	LTV-SD	-	-8	-	-10	-	-11	-	-11
	LTV-R-SD	-10	-	-6	-	-11	-	-8	-
	LTF	-	-8	-	-6	-	-11	-	-5
	LTF-R	-6	-	-5	-	-11	-	-9	-
	LTF-SD	-	-11	-	-12	-	-9	-	-9
	LTF-R-SD	-5	-	-4	-	-6	-	-6	-
PE	**LTV**	-	2	-	2	-	2	-	-2
	LTV-R	0	-	1	-	2	-	1	-
	LTV-SD	-	2	-	3	-	-3	-	-1
	LTV-R-SD	6	-	3	-	2	-	3	-
	LTF	-	0	-	7	-	0	-	2
	LTF-R	-6	-	-8	-	-8	-	-6	-
	LTF-SD	-	6	-	3	-	-2	-	-2
	LTF-R-SD	-1	-	-3	-	-6	-	1	-
PEBC	**LTV**	-	2	-	3	-	1	-	1
	LTV-R	-2	-	-1	-	-2	-	-1	-
	LTV-SD	-	1	-	2	-	0	-	-2
	LTV-R-SD	-3	-	-2	-	-4	-	-3	-
	LTF	-	2	-	4	-	5	-	3
	LTF-R	-1	-	0	-	-5	-	0	-
	LTF-SD	-	1	-	3	-	0	-	2
	LTF-R-SD	-1	-	2	-	-3	-	-2	-
Total		-35	-15	-29	-3	-64	-30	-38	-25

A.4.4 Mejora por *Steepest Descent*

En las tablas A.21 - A.23 se ven los índices de cada pareja de algoritmos, omitiendo las comparaciones entre algoritmos que usan mejora por SD y lo mismo para el caso contrario. Por ejemplo, no se compara LTV-SD con LTF-SD, ya que ambos aplican mejora por SD.

TABLA A.21: *Detalle de índices Copeland para búsqueda mejorada (1)*

		MC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-	7	2	-	-	8	4
	LTV-R	-	-	7	4	-	-	4	0
	LTV-SD	-7	-7	-	-	-2	-7	-	-
	LTV-R-SD	-2	-4	-	-	3	0	-	-
	LTF	-	-	2	-3	-	-	5	-1
	LTF-R	-	-	7	0	-	-	7	-4
	LTF-SD	-8	-4	-	-	-5	-7	-	-
	LTF-R-SD	-4	0	-	-	1	4	-	-
PE	**LTV**	-	-	10	10	-	-	5	3
	LTV-R	-	-	14	9	-	-	12	7
	LTV-SD	4	7	-	-	6	6	-	-
	LTV-R-SD	7	7	-	-	10	6	-	-
	LTF	-	-	10	10	-	-	10	5
	LTF-R	-	-	7	3	-	-	7	0
	LTF-SD	6	6	-	-	10	6	-	-
	LTF-R-SD	5	5	-	-	6	4	-	-
PEBC	**LTV**	-	-	8	10	-	-	8	5
	LTV-R	-	-	8	7	-	-	11	7
	LTV-SD	7	6	-	-	6	5	-	-
	LTV-R-SD	2	6	-	-	6	4	-	-
	LTF	-	-	11	11	-	-	13	6
	LTF-R	-	-	11	7	-	-	9	6
	LTF-SD	5	8	-	-	8	9	-	-
	LTF-R-SD	1	6	-	-	5	1	-	-
Total		16	36	102	70	54	31	99	38

TABLA A.22: *Detalle de índices Copeland para búsqueda mejorada (2)*

		PE							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-	-4	-7	-	-	-6	-5
	LTV-R	-	-	-7	-7	-	-	-6	-5
	LTV-SD	-10	-14	-	-	-10	-7	-	-
	LTV-R-SD	-10	-9	-	-	-10	-3	-	-
	LTF	-	-	-6	-10	-	-	-10	-6
	LTF-R	-	-	-6	-6	-	-	-6	-4
	LTF-SD	-5	-12	-	-	-10	-7	-	-
	LTF-R-SD	-3	-7	-	-	-5	0	-	-
PE	**LTV**	-	-	3	-3	-	-	2	3
	LTV-R	-	-	4	-3	-	-	-1	6
	LTV-SD	-3	-4	-	-	-5	7	-	-
	LTV-R-SD	3	3	-	-	-1	11	-	-
	LTF	-	-	5	1	-	-	1	1
	LTF-R	-	-	-7	-11	-	-	-6	-7
	LTF-SD	-2	1	-	-	-1	6	-	-
	LTF-R-SD	-3	-6	-	-	-1	7	-	-
PEBC	**LTV**	-	-	2	-6	-	-	3	1
	LTV-R	-	-	-2	-2	-	-	-6	-3
	LTV-SD	-1	-1	-	-	-3	8	-	-
	LTV-R-SD	-2	-5	-	-	-7	6	-	-
	LTF	-	-	2	-2	-	-	0	6
	LTF-R	-	-	3	-4	-	-	2	0
	LTF-SD	-1	-1	-	-	-1	6	-	-
	LTF-R-SD	2	-2	-	-	-2	6	-	-
Total		-35	-57	-13	-60	-56	40	-33	-13

TABLA A.23: *Detalle de índices Copeland para búsqueda mejorada (3)*

		PEBC							
		LTV	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**LTF-SD**	**LTF-R-SD**
MC	**LTV**	-	-	-7	-2	-	-	-5	-1
	LTV-R	-	-	-6	-6	-	-	-8	-6
	LTV-SD	-8	-8	-	-	-11	-11	-	-
	LTV-R-SD	-10	-7	-	-	-11	-7	-	-
	LTF	-	-	-6	-6	-	-	-8	-5
	LTF-R	-	-	-5	-4	-	-	-9	-1
	LTF-SD	-8	-11	-	-	-13	-9	-	-
	LTF-R-SD	-5	-7	-	-	-6	-6	-	-
PE	**LTV**	-	-	1	2	-	-	1	-2
	LTV-R	-	-	1	5	-	-	1	2
	LTV-SD	-2	2	-	-	-2	-3	-	-
	LTV-R-SD	6	2	-	-	2	4	-	-
	LTF	-	-	3	7	-	-	1	2
	LTF-R	-	-	-8	-6	-	-	-6	-6
	LTF-SD	-3	6	-	-	0	-2	-	-
	LTF-R-SD	-1	3	-	-	-6	0	-	-
PEBC	**LTV**	-	-	0	3	-	-	-1	1
	LTV-R	-	-	-1	2	-	-	-1	2
	LTV-SD	0	1	-	-	3	0	-	-
	LTV-R-SD	-3	-2	-	-	-4	-2	-	-
	LTF	-	-	-3	4	-	-	0	3
	LTF-R	-	-	0	2	-	-	0	-2
	LTF-SD	1	1	-	-	0	0	-	-
	LTF-R-SD	-1	-2	-	-	-3	2	-	-
Total		-34	-22	-31	1	-51	-34	-35	-13

APÉNDICE B. ANÁLISIS DE CRUZAMIENTO CX

En el cruzamiento CX se da la repetición de muchos hijos, saliendo iguales a sus padres o a otros hijos. Es por esto que se desea saber con qué frecuencia salen hijos iguales a sus padres y cuántos de los hijos generados salen redundantes mediante el cruzamiento CX, de manera de ver si es importante impedir la repetición de cruzamiento entre dos padres.

Dado que para permutaciones de tamaños más grandes hay una mayor cantidad de hijos posibles de generar, el experimento debe considerar la comparación de una cantidad de hijos proporcional a n. Es por esto que el experimento considera lo siguiente: para cada n dentro de un rango se generan dos padres al azar y se realizan n cruzamientos entre ellos, tras los cuales se van contando los hijos iguales a alguno de los padres y los redundantes (igual a un padre u otro hijo previamente generado). Los resultados aquí expuestos consideran n entre 12 hasta 256, por los tamaños de instancias de QAPLIB. El resultado resumido es el mostrado en la tabla B.1.

TABLA B.1: *Resumen experimento cruzamiento CX*

Rango n	Promedio % hijos iguales a padres	Promedio % hijos repetidos
12-40	35,53 %	75,14 %
41-70	23,43 %	75,71 %
71-100	18,19 %	80,40 %
101-130	20,09 %	83,93 %
131-160	12,99 %	76,52 %
161-190	16,82 %	78,46 %
191-220	11,53 %	80,92 %
221-256	10,74 %	82,51 %
10-256	**18,47 %**	**79,33 %**

Estos valores son debidos a los ciclos que se producen dentro de las permutaciones, usadas por el cruzamiento CX para no introducir mutación.

A continuación se muestra el detalle de las cantidades de hijos repetidos e iguales a los padres por cada n, desde 12 hasta 256.

TABLA B.2: *Resultados cruzamientos CX (1)*

n	Hijos iguales a padres	Hijos repetidos
12	7 (58,33 %)	10 (83,33 %)
13	3 (23,08 %)	11 (84,62 %)
14	3 (21,43 %)	10 (71,43 %)
15	2 (13,33 %)	10 (66,67 %)
16	16 (100,00 %)	16 (100,00 %)
17	5 (29,41 %)	15 (88,24 %)
18	2 (11,11 %)	9 (50,00 %)
19	2 (10,53 %)	8 (42,11 %)
20	2 (10,00 %)	11 (55,00 %)
21	0 (0,00 %)	6 (28,57 %)
22	0 (0,00 %)	10 (45,45 %)
23	23 (100,00 %)	23 (100,00 %)
24	16 (66,67 %)	22 (91,67 %)
25	9 (36,00 %)	20 (80,00 %)
26	6 (23,08 %)	20 (76,92 %)
27	5 (18,52 %)	14 (51,85 %)
28	4 (14,29 %)	18 (64,29 %)
29	5 (17,24 %)	23 (79,31 %)
30	6 (20,00 %)	24 (80,00 %)
31	31 (100,00 %)	31 (100,00 %)
32	3 (9,38 %)	18 (56,25 %)
33	33 (100,00 %)	33 (100,00 %)
34	8 (23,53 %)	28 (82,35 %)
35	35 (100,00 %)	35 (100,00 %)
36	6 (16,67 %)	30 (83,33 %)
37	19 (51,35 %)	35 (94,59 %)
38	5 (13,16 %)	26 (68,42 %)
39	10 (25,64 %)	33 (84,62 %)
40	7 (17,50 %)	28 (70,00 %)
41	11 (26,83 %)	35 (85,37 %)
42	5 (11,90 %)	36 (85,71 %)
43	10 (23,26 %)	38 (88,37 %)
44	9 (20,45 %)	38 (86,36 %)
45	14 (31,11 %)	39 (86,67 %)
46	25 (54,35 %)	44 (95,65 %)
47	8 (17,02 %)	41 (87,23 %)
48	25 (52,08 %)	46 (95,83 %)
49	3 (6,12 %)	26 (53,06 %)
50	19 (38,00 %)	48 (96,00 %)
51	26 (50,98 %)	49 (96,08 %)
52	3 (5,77 %)	28 (53,85 %)
53	1 (1,89 %)	40 (75,47 %)
54	7 (12,96 %)	48 (88,89 %)
55	15 (27,27 %)	49 (89,09 %)

n	Hijos iguales a padres	Hijos repetidos
56	3 (5,36 %)	42 (75,00 %)
57	57 (100,00 %)	57 (100,00 %)
58	3 (5,17 %)	35 (60,34 %)
59	2 (3,39 %)	23 (38,98 %)
60	3 (5,00 %)	33 (55,00 %)
61	3 (4,92 %)	23 (37,70 %)
62	3 (4,84 %)	13 (20,97 %)
63	14 (22,22 %)	57 (90,48 %)
64	15 (23,44 %)	58 (90,63 %)
65	6 (9,23 %)	41 (63,08 %)
66	3 (4,55 %)	41 (62,12 %)
67	5 (7,46 %)	40 (59,70 %)
68	68 (100,00 %)	68 (100,00 %)
69	2 (2,90 %)	43 (62,32 %)
70	17 (24,29 %)	64 (91,43 %)
71	2 (2,82 %)	29 (40,85 %)
72	19 (26,39 %)	66 (91,67 %)
73	19 (26,03 %)	67 (91,78 %)
74	20 (27,03 %)	68 (91,89 %)
75	10 (13,33 %)	61 (81,33 %)
76	12 (15,79 %)	63 (82,89 %)
77	11 (14,29 %)	63 (81,82 %)
78	13 (16,67 %)	72 (92,31 %)
79	23 (29,11 %)	73 (92,41 %)
80	10 (12,50 %)	66 (82,50 %)
81	40 (49,38 %)	79 (97,53 %)
82	3 (3,66 %)	39 (47,56 %)
83	13 (15,66 %)	69 (83,13 %)
84	5 (5,95 %)	55 (65,48 %)
85	3 (3,53 %)	56 (65,88 %)
86	3 (3,49 %)	40 (46,51 %)
87	46 (52,87 %)	85 (97,70 %)
88	7 (7,95 %)	60 (68,18 %)
89	7 (7,87 %)	62 (69,66 %)
90	11 (12,22 %)	76 (84,44 %)
91	17 (18,68 %)	85 (93,41 %)
92	49 (53,26 %)	90 (97,83 %)
93	3 (3,23 %)	47 (50,54 %)
94	13 (13,83 %)	80 (85,11 %)
95	4 (4,21 %)	65 (68,42 %)
96	23 (23,96 %)	90 (93,75 %)
97	23 (23,71 %)	91 (93,81 %)
98	8 (8,16 %)	84 (85,71 %)
99	24 (24,24 %)	93 (93,94 %)

TABLA B.3: *Resultados cruzamientos CX (2)*

n	Hijos iguales a padres	Hijos repetidos
100	26 (26,00 %)	94 (94,00 %)
101	7 (6,93 %)	72 (71,29 %)
102	25 (24,51 %)	96 (94,12 %)
103	48 (46,60 %)	101 (98,06 %)
104	18 (17,31 %)	90 (86,54 %)
105	3 (2,86 %)	56 (53,33 %)
106	18 (16,98 %)	100 (94,34 %)
107	4 (3,74 %)	77 (71,96 %)
108	26 (24,07 %)	102 (94,44 %)
109	109 (100,00 %)	109 (100,00 %)
110	13 (11,82 %)	96 (87,27 %)
111	3 (2,70 %)	44 (39,64 %)
112	10 (8,93 %)	98 (87,50 %)
113	33 (29,20 %)	107 (94,69 %)
114	17 (14,91 %)	100 (87,72 %)
115	31 (26,96 %)	109 (94,78 %)
116	4 (3,45 %)	86 (74,14 %)
117	4 (3,42 %)	65 (55,56 %)
118	13 (11,02 %)	104 (88,14 %)
119	32 (26,89 %)	113 (94,96 %)
120	13 (10,83 %)	106 (88,33 %)
121	16 (13,22 %)	107 (88,43 %)
122	122 (100,00 %)	122 (100,00 %)
123	4 (3,25 %)	94 (76,42 %)
124	19 (15,32 %)	110 (88,71 %)
125	16 (12,80 %)	111 (88,80 %)
126	14 (11,11 %)	112 (88,89 %)
127	18 (14,17 %)	113 (88,98 %)
128	37 (28,91 %)	122 (95,31 %)
129	31 (24,03 %)	123 (95,35 %)
130	5 (3,85 %)	74 (56,92 %)
131	4 (3,05 %)	101 (77,10 %)
132	6 (4,55 %)	104 (78,79 %)
133	2 (1,50 %)	57 (42,86 %)
134	13 (9,70 %)	120 (89,55 %)
135	31 (22,96 %)	129 (95,56 %)
136	30 (22,06 %)	130 (95,59 %)
137	32 (23,36 %)	131 (95,62 %)
138	5 (3,62 %)	108 (78,26 %)
139	19 (13,67 %)	125 (89,93 %)
140	33 (23,57 %)	134 (95,71 %)
141	6 (4,26 %)	85 (60,28 %)
142	3 (2,11 %)	112 (78,87 %)
143	27 (18,88 %)	129 (90,21 %)

n	Hijos iguales a padres	Hijos repetidos
144	17 (11,81 %)	130 (90,28 %)
145	4 (2,76 %)	87 (60,00 %)
146	1 (0,68 %)	60 (41,10 %)
147	5 (3,40 %)	93 (63,27 %)
148	33 (22,30 %)	142 (95,95 %)
149	2 (1,34 %)	59 (39,60 %)
150	81 (54,00 %)	148 (98,67 %)
151	3 (1,99 %)	94 (62,25 %)
152	19 (12,50 %)	138 (90,79 %)
153	3 (1,96 %)	65 (42,48 %)
154	3 (1,95 %)	60 (38,96 %)
155	33 (21,29 %)	149 (96,13 %)
156	76 (48,72 %)	154 (98,72 %)
157	14 (8,92 %)	128 (81,53 %)
158	22 (13,92 %)	144 (91,14 %)
159	1 (0,63 %)	64 (40,25 %)
160	45 (28,13 %)	154 (96,25 %)
161	21 (13,04 %)	147 (91,30 %)
162	2 (1,23 %)	104 (64,20 %)
163	8 (4,91 %)	133 (81,60 %)
164	23 (14,02 %)	150 (91,46 %)
165	5 (3,03 %)	136 (82,42 %)
166	5 (3,01 %)	107 (64,46 %)
167	4 (2,40 %)	108 (64,67 %)
168	168 (100,00 %)	168 (100,00 %)
169	8 (4,73 %)	111 (65,68 %)
170	23 (13,53 %)	156 (91,76 %)
171	6 (3,51 %)	81 (47,37 %)
172	0 (0,00 %)	24 (13,95 %)
173	8 (4,62 %)	116 (67,05 %)
174	43 (23,37 %)	178 (96,74 %)
175	3 (1,62 %)	126 (68,11 %)
176	22 (11,83 %)	172 (92,47 %)
177	19 (10,16 %)	173 (92,51 %)
178	93 (49,47 %)	186 (98,94 %)
179	5 (2,65 %)	89 (47,09 %)
180	5 (2,63 %)	93 (48,95 %)
181	26 (13,61 %)	177 (92,67 %)
182	13 (6,77 %)	162 (84,38 %)
183	99 (51,30 %)	191 (98,96 %)
184	13 (6,70 %)	164 (84,54 %)
185	21 (10,77 %)	181 (92,82 %)
186	9 (4,59 %)	166 (84,69 %)
187	5 (2,54 %)	136 (69,04 %)

TABLA B.4: *Resultados cruzamientos CX (3)*

n	Hijos iguales a padres	Hijos repetidos
188	49 (24,75 %)	192 (96,97 %)
189	17 (8,54 %)	169 (84,92 %)
190	2 (1,00 %)	97 (48,50 %)
191	3 (1,49 %)	98 (48,76 %)
192	9 (4,46 %)	141 (69,80 %)
193	8 (3,94 %)	143 (70,44 %)
194	26 (12,75 %)	190 (93,14 %)
195	95 (46,34 %)	203 (99,02 %)
196	10 (4,85 %)	147 (71,36 %)
197	1 (0,48 %)	65 (31,40 %)
198	11 (5,29 %)	178 (85,58 %)
199	99 (47,37 %)	207 (99,04 %)
200	7 (3,33 %)	150 (71,43 %)
201	17 (8,06 %)	181 (85,78 %)
202	27 (12,74 %)	198 (93,40 %)
203	15 (7,04 %)	183 (85,92 %)
204	18 (8,41 %)	184 (85,98 %)
205	28 (13,02 %)	201 (93,49 %)
206	24 (11,11 %)	202 (93,52 %)
207	10 (4,61 %)	158 (72,81 %)
208	7 (3,21 %)	158 (72,48 %)
209	29 (13,24 %)	205 (93,61 %)
210	8 (3,64 %)	161 (73,18 %)
211	30 (13,57 %)	207 (93,67 %)
212	33 (14,86 %)	208 (93,69 %)
213	19 (8,52 %)	193 (86,55 %)
214	2 (0,89 %)	117 (52,23 %)
215	11 (4,89 %)	195 (86,67 %)
216	18 (7,96 %)	212 (93,81 %)
217	14 (6,17 %)	197 (86,78 %)
218	55 (24,12 %)	222 (97,37 %)
219	2 (0,87 %)	70 (30,57 %)
220	61 (26,52 %)	224 (97,39 %)
221	28 (12,12 %)	217 (93,94 %)
222	14 (6,03 %)	202 (87,07 %)

n	Hijos iguales a padres	Hijos repetidos
223	59 (25,32 %)	227 (97,42 %)
224	3 (1,28 %)	173 (73,93 %)
225	15 (6,38 %)	205 (87,23 %)
226	15 (6,36 %)	206 (87,29 %)
227	6 (2,53 %)	133 (56,12 %)
228	55 (24,12 %)	222 (97,37 %)
229	2 (0,87 %)	70 (30,57 %)
230	61 (26,52 %)	224 (97,39 %)
231	28 (12,12 %)	217 (93,94 %)
232	14 (6,03 %)	202 (87,07 %)
233	59 (25,32 %)	227 (97,42 %)
234	3 (1,28 %)	173 (73,93 %)
235	15 (6,38 %)	205 (87,23 %)
236	15 (6,36 %)	206 (87,29 %)
237	6 (2,53 %)	133 (56,12 %)
238	34 (14,29 %)	224 (94,12 %)
239	63 (26,36 %)	233 (97,49 %)
240	14 (5,83 %)	210 (87,50 %)
241	37 (15,35 %)	227 (94,19 %)
242	122 (50,41 %)	240 (99,17 %)
243	2 (0,82 %)	138 (56,79 %)
244	6 (2,46 %)	135 (55,33 %)
245	68 (27,76 %)	239 (97,55 %)
246	1 (0,41 %)	81 (32,93 %)
247	20 (8,10 %)	217 (87,85 %)
248	8 (3,23 %)	186 (75,00 %)
249	32 (12,85 %)	235 (94,38 %)
250	7 (2,80 %)	189 (75,60 %)
251	14 (5,58 %)	221 (88,05 %)
252	7 (2,78 %)	190 (75,40 %)
253	30 (11,86 %)	239 (94,47 %)
254	51 (20,08 %)	248 (97,64 %)
255	7 (2,75 %)	196 (76,86 %)
256	12 (4,69 %)	226 (88,28 %)

APÉNDICE C. *P-VALUES* ENTRE ALGORITMOS POR ESQUEMAS DE MEMORIA

En este apéndice se muestran los *p-values* obtenidos de la sección 4.5 agrupados por esquemas de memoria y de a parejas.

TABLA C.1: *Comparación por esquemas poblacionales*

MC	**LTV**	0,443	**LTV**	**PE**
	LTV-R	0,012	**LTV-R**	
	LTV-SD	0,015	**LTV-SD**	
	LTV-R-SD	0,004	**LTV-R-SD**	
	LTF	0,049	**LTF**	
	LTF-R	0,532	**LTF-R**	
	LTF-SD	0,005	**LTF-SD**	
	LTF-R-SD	0,023	**LTF-R-SD**	
PE	**LTV**	0,756	**LTV**	**PEBC**
	LTV-R	0,938	**LTV-R**	
	LTV-SD	0,397	**LTV-SD**	
	LTV-R-SD	0,063	**LTV-R-SD**	
	LTF	0,245	**LTF**	
	LTF-R	0,044	**LTF-R**	
	LTF-SD	0,538	**LTF-SD**	
	LTF-R-SD	0,755	**LTF-R-SD**	
PEBC	**LTV**	0,148	**LTV**	**MC**
	LTV-R	0,008	**LTV-R**	
	LTV-SD	0,011	**LTV-SD**	
	LTV-R-SD	0,103	**LTV-R-SD**	
	LTF	0,008	**LTF**	
	LTF-R	0,156	**LTF-R**	
	LTF-SD	0,003	**LTF-SD**	
	LTF-R-SD	0,301	**LTF-R-SD**	

TABLA C.2: *Comparación por esquema de reducción de espacio de búsqueda*

MC	**LTV**	0,244	**LTV-R**	**MC**
	LTV-SD	0,068	**LTV-R-SD**	
	LTF	0,778	**LTF-R**	
	LTF-SD	0,842	**LTF-R-SD**	
PE	**LTV**	0,414	**LTV-R**	**PE**
	LTV-SD	0,088	**LTV-R-SD**	
	LTF	0,061	**LTF-R**	
	LTF-SD	0,394	**LTF-R-SD**	
PEBC	**LTV**	0,918	**LTV-R**	**PEBC**
	LTV-SD	0,127	**LTV-R-SD**	
	LTF	0,232	**LTF-R**	
	LTF-SD	0,938	**LTF-R-SD**	

TABLA C.3: *Comparación por esquema de Steepest Descent*

MC	**LTV**	0,069	**LTV-SD**	**MC**
	LTV-R	0,641	**LTV-R-SD**	
	LTF	0,865	**LTF-SD**	
	LTF-R	0,925	**LTF-R-SD**	
PE	**LTV**	0,776	**LTV-SD**	**PE**
	LTV-R	0,196	**LTV-R-SD**	
	LTF	0,981	**LTF-SD**	
	LTF-R	0,28	**LTF-R-SD**	
PEBC	**LTV**	0,339	**LTV-SD**	**PEBC**
	LTV-R	0,196	**LTV-R-SD**	
	LTF	0,266	**LTF-SD**	
	LTF-R	0,679	**LTF-R-SD**	

TABLA C.4: *Comparación por largo de lista fijo o variable*

MC	**LTV**	0,286	**LTF**	**MC**
	LTV-R	0,826	**LTF-R**	
	LTV-SD	0,33	**LTF-SD**	
	LTV-R-SD	0,426	**LTF-R-SD**	
PE	**LTV**	0,173	**LTF**	**PE**
	LTV-R	0,722	**LTF-R**	
	LTV-SD	0,147	**LTF-SD**	
	LTV-R-SD	0,438	**LTF-R-SD**	
PEBC	**LTV**	0,469	**LTF**	**PEBC**
	LTV-R	0,798	**LTF-R**	
	LTV-SD	0,776	**LTF-SD**	
	LTV-R-SD	0,426	**LTF-R-SD**	

APÉNDICE D. DESVIACIONES ESTÁNDAR

En este apéndice se muestran las desviaciones estándar de los resultados de las 10 ejecuciones de cada algoritmo en cada instancia.

TABLA D.1: *Desviaciones estándar de algoritmos en instancias (1)*

	MC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**I-LTF**	**LTF-R-SD**
chr25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
kra30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
nug30	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
ste36a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko49	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko56	**0,000**	0,004	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko64	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko72	**0,000**	**0,000**	**0,000**	0,002	0,002	**0,000**	**0,000**	**0,000**
sko81	0,019	0,014	0,011	0,014	0,014	0,016	0,015	0,015
sko90	0,016	0,012	0,005	0,018	0,016	0,015	**0,004**	0,017
sko100a	0,022	0,015	0,017	0,021	0,020	0,017	0,022	0,023
tai20a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai35a	**0,000**	0,030	**0,000**	**0,000**	**0,000**	**0,000**	0,052	**0,000**
tai35b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai40a	0,122	0,112	0,088	0,114	0,113	0,138	0,123	0,118
tai40b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai50a	0,170	0,110	0,143	0,083	0,132	0,106	0,094	0,115
tai50b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai60a	0,129	0,116	0,173	0,146	0,106	0,081	0,086	0,174
tai60b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	0,001	0,015
tai80a	0,172	0,067	0,110	0,119	0,096	0,129	**0,056**	0,124
tai80b	0,236	0,012	0,012	0,008	0,012	0,308	**0,000**	0,031
tai100a	0,096	0,130	0,132	0,102	0,116	0,096	0,114	0,052
tai100b	0,049	**0,042**	0,045	0,047	0,076	0,099	0,065	0,100
tho150b	0,053	0,050	0,044	0,047	0,052	0,045	0,059	0,083
wil50	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai256c	0,016	0,033	0,031	0,018	0,019	0,018	0,037	0,030
tai150b	0,152	**0,119**	0,172	0,248	0,144	0,189	0,161	0,172
Promedio	0,042	**0,029**	0,033	0,033	0,031	0,042	0,030	0,036

TABLA D.2: *Desviaciones estándar de algoritmos en instancias (2)*

	PE							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**I-LTF**	**LTF-R-SD**
chr25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
kra30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
nug30	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
ste36a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko49	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko56	0,004	0,004	0,004	0,004	**0,000**	**0,000**	0,005	**0,000**
sko64	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko72	0,027	0,027	0,020	0,027	0,031	0,020	0,006	0,020
sko81	0,014	0,013	0,016	0,010	0,013	**0,006**	0,016	0,015
sko90	0,018	0,019	0,012	0,044	0,039	0,012	0,017	0,019
sko100a	0,018	0,026	0,031	0,023	0,020	0,023	0,029	0,021
tai20a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai30a	**0,000**	**0,000**	**0,000**	**0,000**	0,005	**0,000**	**0,000**	**0,000**
tai35a	0,081	0,132	0,054	0,109	0,071	0,117	0,082	0,081
tai35b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai40a	0,072	0,109	0,119	0,077	0,130	0,144	0,161	0,123
tai40b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai50a	0,231	0,213	0,109	**0,079**	0,148	0,106	0,080	0,221
tai50b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai60a	0,114	0,094	0,106	0,193	0,111	0,123	0,183	0,150
tai60b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai80a	0,160	0,132	0,137	0,094	0,104	0,108	0,118	0,095
tai80b	0,273	0,177	0,178	0,093	0,180	0,225	0,300	0,178
tai100a	0,147	0,135	0,117	**0,051**	0,077	0,116	0,076	0,075
tai100b	0,082	0,111	0,070	0,073	0,107	0,060	0,092	0,124
tho150b	0,062	0,036	0,046	0,041	0,044	0,056	**0,020**	0,052
wil50	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	0,005
tai256c	0,026	0,027	0,031	0,029	**0,013**	0,024	0,016	0,042
tai150b	0,152	0,216	0,178	0,188	0,203	0,256	0,200	0,200
Promedio	0,049	0,049	0,041	0,038	0,043	0,047	0,047	0,047

TABLA D.3: *Desviaciones estándar de algoritmos en instancias (3)*

	PEBC							
Instancia	**LTV**	**LTV-R**	**LTV-SD**	**LTV-R-SD**	**LTF**	**LTF-R**	**I-LTF**	**LTF-R-SD**
chr25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
kra30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
nug30	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
ste36a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	0,033	**0,000**
sko49	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko56	0,005	**0,000**	0,004	0,005	**0,000**	**0,000**	**0,000**	**0,000**
sko64	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
sko72	0,020	**0,000**	**0,000**	0,023	0,029	0,020	**0,000**	**0,020**
sko81	**0,006**	0,017	0,013	0,012	0,017	0,014	0,015	0,016
sko90	0,033	**0,004**	0,020	0,008	0,019	0,017	0,020	0,016
sko100a	0,026	0,018	0,034	0,023	0,023	0,022	0,019	0,025
tai20a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai25a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai30a	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai35a	0,091	0,054	0,087	0,040	0,080	0,102	0,093	0,111
tai35b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai40a	0,072	0,099	0,101	0,139	0,125	0,076	0,088	**0,044**
tai40b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai50a	0,177	0,138	0,161	0,096	0,113	0,095	0,093	0,129
tai50b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai60a	0,127	0,095	0,135	0,103	**0,074**	0,125	0,181	0,140
tai60b	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai80a	0,119	0,101	0,142	0,175	0,170	0,120	0,106	0,085
tai80b	0,275	0,178	0,251	0,031	0,238	0,302	0,013	0,010
tai100a	0,071	0,075	0,110	0,136	0,131	0,080	0,094	0,090
tai100b	0,093	0,106	0,097	0,115	0,114	0,157	0,132	0,063
tho150b	0,071	0,056	0,032	0,054	0,056	0,047	0,092	0,053
wil50	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**	**0,000**
tai256c	0,028	0,025	0,021	0,028	0,036	0,031	0,024	0,029
tai150b	0,254	0,126	0,254	0,120	0,155	0,256	0,205	0,174
Promedio	0,049	0,036	0,049	0,037	0,046	0,049	0,040	0,034