Informe Final del Trabajo de Grado

Una Metaheurística Co-evolutiva
para el Problema del Viajante de Comercio

Alumna
Diana Holstein

Director
Pablo Moscato

Codirector
Ms. Ing. Oscar Bria

AGRADECIMIENTOS

Resultaría imposible mencionar a todas las personas que de una u otra manera colaboraron con este trabajo. Mencionando a algunas de ellas, quiero extender mi reconocimiento a todas las demás.

A Pablo Moscato, por ser una fuente inagotable de ideas, y por su confianza, que permitió la realización del trabajo. A Oscar Bria, por la inestimable ayuda y por estar disponible en todo momento. A Andrea Maciel, por sus aportes al comienzo de este proyecto. A Natalio Krasnogor, por colaborar con su experiencia y su tiempo. Al Ing. A. Quijano y al personal del CeTAD, por permitirme utilizar sus recursos. A Carina Santiago, por cederme tiempos tan valiosos. A mis padres, no sólo por su apoyo durante toda mi carrera. A David, que prescindió de su PC durante meses para que yo pudiera utilizarla. Y (no por último, menos importante) a Gustavo, que con su paciencia y su ayuda incondicional, fue indispensable para este trabajo, como lo es en tantos aspectos de la vida.

ÍNDICE

1. INTRODUCCIÓN

En este Trabajo de Grado se presenta una estrategia general (metaheurística) para la resolución del Problema del Viajante de Comercio.

Éste es un problema clásico de optimización combinatoria, cuyo conjunto de soluciones posibles es finito, pero demasiado numeroso para ser manejado en forma directa.

Dado un conjunto de ciudades, y una medida de "costo" entre ellas, el problema consiste en hallar un camino cerrado (tour) de costo mínimo, que visite cada ciudad exactamente una vez. El costo puede estar representado por la distancia entre las ciudades, o por cualquier otra medida, tal como el tiempo que las separa o el costo de un pasaje entre ellas.

Este problema puede aplicarse en muchas situaciones prácticas, tales como ruteo de vehículos, secuenciamiento de tareas, conexión de módulos electrónicos. Además reviste una importancia teórica para la Teoría de Complejidad, pues pertenece a la clase de los problemas combinatorios NP-Completos, para los cuales se conjetura que el tiempo de cómputo requerido para hallar la solución exacta crece al menos exponencialmente con el tamaño de la instancia considerada. Por esto es necesario buscar heurísticas que encuentren rápidamente tours cercanos al óptimo.

En el contexto de los problemas de optimización combinatoria, se pueden definir las heurísticas como técnicas que producen soluciones factibles rápidamente, en cuanto al tiempo de cómputo requerido, pero tales soluciones no son necesariamente óptimas. Estos procedimientos tienen una justificación intuitiva.

Las metaheurísticas son estrategias que generalmente guían otras heurísticas, y que no dependen de las características del problema a resolver. El desarrollo de metaheurísticas para resolver este problema, permite que las mismas técnicas puedan aplicarse a una gran variedad de problemas combinatorios.

El Problema del Viajante de Comercio se ha utilizado siempre para ensayar diferentes enfoques de optimización combinatoria, incluyendo las técnicas clásicas de optimización local, así como variantes más recientes: Búsqueda Tabú, Redes Neuronales, Algoritmos Genéticos. Es un dominio atípico desde el punto de vista teórico y experimental.

Luego de presentar el Problema del Viajante de Comercio, las secciones iniciales de este Trabajo describirán por separado los componentes de la metaheurística a desarrollar: heurísticas de búsqueda local, Búsqueda Local Guiada, Algoritmos Meméticos. En la Sección 6, se integrarán estos conceptos para diseñar una estrategia de resolución del Problema del Viajante de Comercio, que intenta aprovechar las mejores características de cada uno de estos enfoques. Las últimas secciones presentarán los resultados de los experimentos realizados y las conclusiones del trabajo.

2. EL PROBLEMA DEL VIAJANTE DE COMERCIO

2.1. Los problemas de optimización combinatoria

La optimización significa hallar el valor máximo o mínimo de una cierta función, definida en un dominio. En los problemas de decisión que generalmente se presentan en la vida empresarial, por lo general existen una serie de recursos escasos (personal, presupuesto, tiempo), o de requisitos mínimos a cumplir (producción, horas de descanso), que condicionan la elección de la estrategia más adecuada. Por lo general, el objetivo al tomar la decisión consiste en llevar a cabo el plan propuesto de una manera óptima: mínimos costos o máximo beneficio. La resolución de este tipo de problemas atrajo la atención de numerosos investigadores, principalmente desde la Segunda Guerra Mundial, con el florecimiento de la Investigación Operativa. [8]

Existe un tipo concreto de problemas de optimización, especialmente interesante. Lo forman los denominados problemas de optimización combinatoria. En ellos, las variables de decisión son enteras y por lo general el espacio de soluciones está formado por ordenaciones o subconjuntos de números naturales (de ahí su nombre). En el caso de la optimización combinatoria, se trata de hallar el mejor valor entre un número finito de posibilidades, pero la enumeración de este conjunto de posibilidades resulta prácticamente imposible, aún para instancias de tamaño moderado.

Las raíces históricas de la optimización combinatoria subyacen en problemas de economía: el planeamiento y gerenciamiento de operaciones, y el uso eficiente de recursos. Pronto comenzaron a modelizarse de esta manera aplicaciones más técnicas, y hoy vemos problemas de optimización discreta casi en cada campo del conocimiento: diseño de campañas de marketing, planeamiento de inversiones, división de áreas en distritos políticos, secuenciamiento de genes, clasificación de plantas y animales, diseño de nuevas moléculas, trazado de redes de comunicaciones, posicionamiento de satélites, determinación del tamaño de vehículos y las rutas de medios de transporte, diseño y producción de circuitos VLSI y tarjetas de circuitos impresos, asignación de trabajadores a tareas, construcción de códigos seguros. La lista de aplicaciones parece interminable; aún en áreas como deportes, arqueología o psicología, la optimización combinatoria es usada para responder importantes preguntas. En [17] se enumeran otros campos en los cuales pueden utilizarse técnicas de optimización combinatoria.

2.2. El Problema del Viajante de Comercio: definición y aplicaciones

Uno de los problemas más famosos, y quizás el más estudiado, en el campo de la optimización combinatoria, es el Problema del Viajante de Comercio (TSP, por sus siglas en inglés: Traveling Salesman Problem). A pesar de ser muy sencillo de plantear, el TSP ha causado numerosos problemas a los matemáticos en las últimas décadas, dado lo compleja que resulta su resolución.

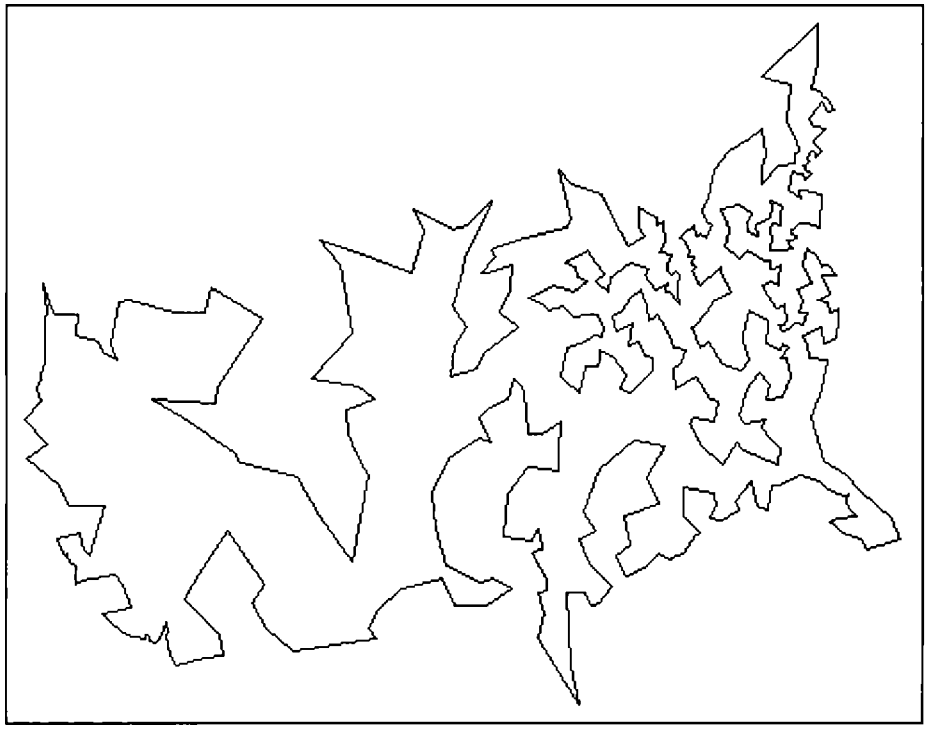
Su definición es la siguiente [34]:

Dadas N ciudades $\{c_1, c_2, ..., c_N\}$ y una matriz de distancias entre ellas D: N x N; $D_{i,j} \in R^+$ (donde $D_{i,j}$ indica la distancia entre las ciudades c_i y c_j), la tarea es encontrar un "tour", es decir una permutación de las N ciudades, que minimice la longitud total $L_{(Tour)}$, definida como la suma de las distancias:

$$L_{(Tour)} = \sum_{i=0}^{N-1} D_{Tour(i), Tour(i+1)} \qquad (2\text{-}1)$$

donde Tour(N) se identifica con Tour(0) para conseguir un ciclo cerrado.

Como ejemplo se presenta una instancia muy conocida del TSP, denominada att532. La figura representa el tour óptimo (camino de menor longitud) entre 532 ciudades de los Estados Unidos:



Si bien en este trabajo se utilizarán instancias cuya matriz de distancias es simétrica y euclídea, las heurísticas no dependerán de estas características de la matriz, y por lo tanto serán aplicables a cualquier tipo de instancias del TSP.

Según se detalla en [34], el TSP constituye un importante problema de testeo para heurísticas de optimización combinatoria, debido a los siguientes hechos:

a) Su problema de decisión asociado es representativo de la clase de los problemas NP-Completos, incluso para las versiones geométricas del TSP. En [22] se explican conceptos fundamentales de complejidad computacional, y se los aplica en particular al caso del TSP.

b) El problema tiene una larga historia, y muchos enfoques heurísticos son un verdadero desafío para la búsqueda de nuevos métodos. Pueden verse ejemplos en [17] y [20].

c) Grandes instancias del TSP han sido resueltas hasta la optimalidad, y están disponibles en la literatura (ver referencia [42]) y en Internet, donde Reinelt mantiene una base de datos de dominio público sumamente difundida, y fundamental para la comparación de resultados ([43]). En [26], [27] y [38], se presenta una forma de construir instancias aún mayores con algunas características específicas y con óptimos globales únicos.

d) Para instancias generadas con una cierta distribución de probabilidad, existen fórmulas asintóticas de la longitud esperada para el tour óptimo (límites empíricos), que permiten evaluar la calidad de los resultados obtenidos por una heurística, en el caso de no conocerse el óptimo global. Pueden consultarse al respecto las referencias [20], [21] y [38].

e) El estudio del desempeño de las heurísticas para grandes instancias del TSP está motivado tanto científica como tecnológicamente. Por ejemplo, ya se conocen instancias de tamaño 10^4, surgidas de aplicaciones de perforación de tarjetas de circuitos impresos, y cristalografía de rayos X ([43]). Métodos de búsqueda exitosos basados en el TSP tendrían impacto en la biología molecular y en procesos de VLSI.

El TSP euclídeo en el plano ha provisto gran parte de la motivación para el desarrollo de la Teoría de la Complejidad. Juega el rol de problema de testeo para muchas heurísticas generales destinadas a otros problemas de optimización, como ruteo de vehículos, secuenciamiento de tareas, etc; y para metaheurísticas generales como Búsqueda Tabú, Simulated Annealing, Algoritmos Genéticos y Meméticos. Además de su significación para evaluar algoritmos de aproximación y heurísticas, hay investigadores que usan instancias del TSP para ayudar a desarrollar métodos exactos, basados en técnicas de Branch-and-Bound y, más recientemente, Branch-and-Cut (investigación exhaustiva de las soluciones posibles, descartando en forma inteligente ciertas "direcciones de búsqueda").

La página de P. Moscato, citada como referencia [33], provee un listado muy amplio de publicaciones, reportes técnicos, etc, disponibles en Internet acerca del TSP y algunos problemas asociados.

3. LA HEURÍSTICA DE BÚSQUEDA LOCAL

Las heurísticas son herramientas desarrolladas para ayudar en la resolución de muchos tipos de problemas. Aunque la capacidad científica de la humanidad parece haber alcanzado un nivel que permite dar respuesta prácticamente a cualquier reto que se le plantee, lamentablemente no es ésa la realidad. En efecto, si pensamos en los problemas reales que suelen ser de interés, para los cuales el objetivo es encontrar la solución que optimiza algún tipo de criterio (por ejemplo, menor costo bajo determinadas condiciones), la situación es justamente la opuesta: tan sólo una pequeña parte de ellos pueden ser resueltos fácilmente. En estas circunstancias, las heurísticas adquieren cada vez una mayor importancia.

3.1. Complejidad computacional

Optimizar significa hallar el máximo o mínimo de una cierta función, definida en algún dominio. Las teorías clásicas de optimización (por ejemplo, el cálculo diferencial) tratan con el caso en que este dominio es infinito.

Desde este ángulo, los problemas de optimización combinatoria, donde el dominio es típicamente finito, pueden parecer triviales, pues existe siempre un procedimiento elemental para determinar la solución óptima buscada: realizar una exploración exhaustiva del conjunto de soluciones. Es decir, generar todas las soluciones factibles, calcular para cada una el costo asociado, y elegir finalmente la que haya dado lugar al mejor de ellos. Sin embargo, aunque este método teóricamente llega siempre a la solución buscada, no es eficiente, pues el tiempo de cálculo necesario crece exponencialmente con el número de ítems del problema. Las posibilidades a revisar pueden incluir todos los árboles de N nodos, o todos los circuitos hamiltonianos de un grafo completo, y por lo tanto listarlas todas para encontrar la mejor entre ellas es prácticamente imposible aún para instancias de tamaño moderado.

Existen problemas combinatorios para los cuales no se conocen algoritmos de resolución, más que aquéllos en los cuales se produce una explosión exponencial del tiempo de cálculo al aumentar el tamaño del problema. Son problemas computacionalmente difíciles de tratar. Por el contrario, para otros problemas combinatorios existen algoritmos que sólo crecen en tiempo polinomialmente con el tamaño del problema.

Matemáticamente se trató de caracterizar a uno y otro tipo de problemas. De aquéllos para los cuales se conocen algoritmos que necesitan un tiempo polinomial para ofrecer la solución óptima se dice que pertenecen a la clase P, y se considera que son resolubles eficientemente. Sin embargo, se comprobó que la mayoría de los principales problemas de optimización pertenecen a un superconjunto de P, la clase denominada NP, en la cual se incluyen aquellos problemas para los que no se conoce un

algoritmo polinomial de resolución[1], tales como el TSP y el QAP [45], coloreo de grafos [24], satisfactibilidad [22], etc.

Estos problemas "difíciles" en NP, para los cuales todos los algoritmos conocidos requieren tiempo exponencial, tienen la peculiaridad de que todos los problemas en NP pueden ser "reducidos" polinomialmente a ellos, es decir, si se puede dar una solución en tiempo polinomial para uno de ellos, se podría dar para todos los de NP. Esta propiedad ha sido usada para definir una subclase separada en NP: la de los problemas NP-hard. Se dice que un problema es NP-hard si cualquier problema en NP es polinomialmente transformable en él, aunque el problema en sí no pertenezca a NP. Si el problema además pertenece a NP, entonces se lo denomina NP-completo.

El hecho de que hasta hoy (luego de considerables esfuerzos a lo largo de muchos años) no se hayan podido encontrar algoritmos eficientes para problemas NP-completos lleva a pensar a la mayoría de los investigadores, que una vez que se demuestra que un problema pertenece a esa clase, ya no vale la pena tratar de buscar algoritmos eficientes para él. Lamentablemente, muchos de los problemas importantes que aparecen en Investigación Operativa son NP-completos.

3.2. Heurísticas

Dada la dificultad práctica para resolver en forma exacta una serie de importantes problemas combinatorios para los cuales, por otra parte, es necesario ofrecer alguna solución dado su interés práctico, comenzaron a aparecer algoritmos que proporcionan soluciones factibles (es decir, que satisfacen las restricciones del problema), las cuales, aunque no optimicen la función objetivo, al menos se acercan al valor óptimo en un tiempo de cálculo razonable. Podríamos llamarlas, en lugar de óptimas, "satisfactorias".

Este tipo de algoritmos se denominan heurísticas, del griego "heuriskein": encontrar. Aunque en un primer momento no fueron bien vistas en los círculos académicos, acusadas de escaso rigor matemático, su interés práctico como herramienta útil para dar soluciones a problemas reales les fue abriendo poco a poco las puertas, sobre todo a partir de los años setenta con la proliferación de resultados en el campo de la complejidad computacional.

Un método heurístico, también llamado algunas veces "algoritmo de aproximación", es un conjunto bien definido de pasos para identificar rápidamente una solución de alta calidad (aunque no necesariamente óptima) para un problema dado, donde una solución es un conjunto de valores para las incógnitas del problema. Las heurísticas son procedimientos simples, a menudo basados en el sentido común.

---

[1] Matemáticamente hablando, las clases P y NP no están compuestas por los problemas de optimización, sino por sus correspondientes "problemas de decisión", consistentes en determinar si para el problema de optimización existe una solución cuyo costo mejora un cierto valor.

Son varios los factores que pueden hacer interesante la utilización de algoritmos heurísticos para la resolución de un problema: [8]

a) Cuando no existe un método exacto de resolución. Ofrecer entonces una solución aceptablemente buena resulta de interés, frente a la alternativa de no tener ninguna solución.

b) Cuando no se necesita la solución óptima. Existen casos en que no se justifica el costo en tiempo y dinero para hallar una solución óptima que, por otra parte, no representará un beneficio importante con respecto a una que sea simplemente sub-óptima.

c) Cuando los datos son poco confiables, o bien cuando el modelo es una simplificación de la realidad. En estos casos puede carecer de interés buscar una solución exacta, dado que de por sí ésta no será más que una aproximación, al basarse en datos que no son los reales.

d) Cuando limitaciones de tiempo, memoria, espacio para almacenamiento de datos, etc., obligan al empleo de métodos de rápida respuesta, aún a costa de la precisión.

e) Como paso intermedio en la aplicación de otro algoritmo. A veces se usan soluciones heurísticas como punto de partida de algoritmos exactos.

Una importante ventaja que presentan las heurísticas respecto a las técnicas que buscan soluciones exactas, es que por lo general brindan una mayor flexibilidad para el manejo de las características del problema. Además, pueden ofrecer más de una solución, lo cual permite ampliar las posibilidades de elección, sobre todo cuando existen factores que no han podido ser añadidos en el modelo, pero que también deben ser considerados.

Existen diferentes tipos de heurísticas, según el modo en que buscan y construyen sus soluciones. Una posible clasificación las divide en heurísticas constructivas, y de mejoramiento iterativo.

a) Métodos constructivos: Consisten en añadir paulatinamente componentes individuales a la solución, hasta que se obtiene una solución factible. El más popular de estos métodos lo constituyen los algoritmos "codiciosos" (greedy), que construyen paso a paso la solución buscando el máximo beneficio en cada paso.

b) Métodos de mejoramiento iterativo o mejora local: Estos métodos no tratan de llegar a una solución factible, sino que parten de una de ellas (obtenida quizás mediante otra heurística), y mediante alteraciones de esa solución van pasando de forma iterativa y mientras no se cumpla un determinado criterio de fin, a otras también factibles pero de mejor costo.

Las heurísticas pueden ser simples o complejas. Los algoritmos simples tienden a tener reglas de terminación bien definidas, y se detienen en un óptimo local. Los algoritmos más complejos pueden no tener reglas de terminación estándar, y típicamente buscan soluciones mejores hasta alcanzar un punto de parada arbitrario. La mayoría de las metaheurísticas (Búsqueda Tabú, Simulated Annealing,

Algoritmos Genéticos, Redes Neuronales) son ejemplos de algoritmos más complejos; consultar [20] para una descripción de cada una de estas técnicas.

Los métodos heurísticos siempre han sido una parte de la resolución de problemas por los seres humanos. Las versiones matemáticas están creciendo en su rango de aplicaciones, así como en su variedad de enfoques. Nuevas tecnologías heurísticas están dando a los investigadores operativos, científicos de computación y profesionales, la capacidad de resolver rutinariamente problemas que eran demasiado grandes o complejos para generaciones previas de algoritmos.

En particular, para el TSP, en el caso asimétrico de N ciudades, hay (N-1)! tours posibles. Para el caso simétrico, existen la mitad de soluciones distintas, (N-1)! / 2. En cualquier caso, el número de soluciones se vuelve extremadamente grande para N grande, de modo que una búsqueda exhaustiva es impracticable.

Los algoritmos conocidos requieren tiempos de computación que crecen exponencialmente con N; incluso hay trabajos en teoría de la complejidad, que indican que problemas como el TSP probablemente sean inherentemente exponenciales. Los métodos heurísticos parecen ser una línea factible de ataque. Desde un punto de vista teórico, aunque generalmente no se puede probar la optimalidad de las soluciones, es posible obtener confianza estadística; para aplicaciones prácticas, frecuentemente lo que importa es obtener buenas respuestas en tiempos de ejecución factibles.[25]

Al pertenecer a la clase de problemas NP-completos, se conjetura que cualquier algoritmo para encontrar tours óptimos para el TSP tendrá un peor caso para el tiempo de ejecución que crezca más rápido que cualquier polinomio. Esto deja a los investigadores dos alternativas: buscar heurísticas que encuentren rápidamente tours "casi" óptimos, o desarrollar algoritmos de optimización que funcionen bien sobre casos "del mundo real", no tanto sobre instancias del peor caso. Por su simplicidad y aplicabilidad, el TSP ha servido por décadas como un campo de prueba inicial para nuevas ideas relacionadas con ambas alternativas.

El TSP es una de las historias de mayores éxitos para la optimización. Décadas de investigaciones sobre técnicas de optimización, combinadas con el rápido crecimiento en las velocidades de cómputo y capacidades de memoria, han llevado a un nuevo récord después de otro. Estos resultados debilitan el interés en heurísticas más costosas, al menos cuando el número de ciudades no es muy grande. Los éxitos de los enfoques tradicionales dejan menos espacio para el aporte de nuevas aproximaciones, como Búsqueda Tabú, Simulated Annealing, etc. Sin embargo, al menos uno de estos nuevos enfoques, el de Algoritmos Meméticos, tiene aún una contribución para hacer.[20]

3.3. Heurísticas de mejoramiento iterativo

Una aproximación básica a las heurísticas para problemas de optimización combinatoria, es el mejoramiento iterativo de una solución factible seleccionada al azar (heurística de mejoramiento iterativo). Este enfoque se describe en [25]:

1. Generar una solución factible inicial T.
2. Intentar hallar una solución factible mejorada T', por alguna transformación de T.
3. Si se encuentra una solución mejorada (es decir, que produzca un menor valor de la función objetivo), reemplazar T por T' y repetir desde el paso 2.
4. Si no puede encontrarse una solución mejorada, T es una solución localmente óptima.

El corazón del procedimiento iterativo es, por supuesto, el paso 2, la transformación que trata de mejorar una solución dada. Un concepto clave es cómo realizar el paso de una solución factible a otra.

Para el TSP, se considerarán algoritmos basados en modificaciones simples del tour (heurísticas de intercambio). Tales algoritmos se especifican en términos de una clase de operaciones (intercambios o movidas), que pueden usarse para convertir un tour en otro. Dado un tour factible, el algoritmo ejecuta repetidamente operaciones de la clase dada, mientras va reduciendo la longitud del tour actual, hasta que se alcanza un tour para el cual ninguna operación produce una mejora. En [20] se describen varias clases de transformaciones de un tour en otro mejor.

Alternativamente, esto puede verse como un proceso de búsqueda en un "vecindario". El ingrediente básico de las heurísticas de mejora o de intercambio, es una regla que describe las posibles manipulaciones permitidas. Esta regla define implícitamente, para cada solución factible, el conjunto de otras soluciones factibles que pueden obtenerse por tal manipulación. El concepto de vecindario (neighborhood, en inglés) N(T) de una solución T, se refiere al conjunto de soluciones "cercanas" a ella, a las que puede llegarse por medio de una única movida u operación elemental a partir de T. Este método se basa por lo tanto en buscar entre los elementos del vecindario N(T) de la solución actual T, alguno que tenga un mejor valor, moverse a él, y repetir la operación hasta que se considere que no es posible hallar una mejor solución. En este caso se dice que la solución obtenida es localmente óptima. [8]

La existencia de óptimos locales es uno de los mayores inconvenientes que presentan estas técnicas. Se dice que una solución T es óptimo local si $\forall T' \in N(T)$, es $L_{(T)} \leq L_{(T')}$. Si a lo largo de la búsqueda se cae en un óptimo local, en principio la heurística no sabría continuar, pues quedaría detenida en ese punto. Una primera posibilidad para salvar esa dificultad consiste en reiniciar la búsqueda desde otra solución inicial, confiando en que pueda encontrar otros caminos. A esto se lo denomina técnica de comienzos múltiples o "multistart". Cuanto mejor sea la heurística, menor será el conjunto de óptimos locales, y mayor será la fracción de comienzos aleatorios que lleven al óptimo global. [25]

Otro de los problemas a los que se enfrentan estas heurísticas es la dependencia de la solución inicial de la que se parta. Obviamente, el punto inicial tiene una gran influencia sobre la posibilidad de caer o no en un óptimo local.

A pesar de que los métodos heurísticos sólo evalúan normalmente un número pequeño de alternativas del total posible, se ha atribuido su éxito a la aplicación de la regla 80/20 (el 80% de la

riqueza la posee un 20% de la población; con el 20% de esfuerzo se consigue el 80% de los resultados, etc.). Una heurística bien diseñada puede aprovechar esta propiedad y explorar exclusivamente las soluciones más interesantes.

3.4. La heurística 2-Opt

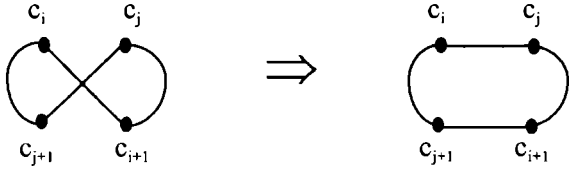
Para el TSP se considerará como heurística básica la siguiente: dado un tour T, se tratará de mejorarlo eliminando k arcos y reemplazándolos por un nuevo conjunto de k arcos que reconectan el tour. Estos borrados y reemplazos serán llamados k-Changes. Se dice que un tour es k-óptimo cuando no puede ser mejorado por un k-Change; es decir, es un mínimo local bajo la movida del k-Change.[34]

Para esta operación de intercambio, dos configuraciones son vecinas si existe un k-Change que transforma a una de ellas en la otra. Es fácil ver que existe una relación entre el tamaño del vecindario y el valor de k. En general, un paso del k-Opt mejora un tour borrando k arcos existentes, e insertando k nuevos arcos. El algoritmo k-Opt obvio requiere un tiempo de $O(N^k)$ para chequear por un único intercambio. Cuando se usa un mayor k, el mínimo local es mejor, pero el costo computacional es mayor. Se han desarrollado varias implementaciones rápidas para algoritmos k-Opt (se citan diversos ejemplos en [20]); sin embargo no se acostumbra usar un k muy grande. Las heurísticas más usadas de este tipo son las que producen tours 2-óptimos y 3-óptimos.

La movida del 2-Opt borra dos arcos, partiendo así el tour en dos subtours, y luego los reconecta de la otra manera posible. En el 3-Opt, el intercambio reemplaza hasta 3 arcos del tour actual. Aplicado a tours de distribución uniforme, el algoritmo 3-Opt produce tours sustancialmente más cortos que el 2-Opt, al costo de un significativo aumento en el tiempo de ejecución.

La operación de mejora del 2-Opt remueve dos arcos: (c_i, c_{i+1}) y (c_j, c_{j+1}), y en su lugar inserta los dos arcos (c_i, c_j) y (c_{i+1}, c_{j+1}). Los pares de arcos a intercambiar se eligen de manera que esta acción reduzca la longitud del tour, mientras mantiene un camino conectado entre todos los puntos. Al fin del proceso, cuando no pueden hacerse más movidas del 2-Opt, se dice que el tour es 2-óptimo.

La siguiente figura ilustra un único intercambio 2-Opt en un tour; esta operación mantiene el tour conectado y reduce su longitud. [3], [6]



Debe tenerse cuidado de no usar los arcos de reemplazo (c_i, c_{j+1}) y (c_j, c_{j+1}), ya que el tour se rompería en dos ciclos disjuntos.

El algoritmo del 2-Opt consiste en un ciclo que examina todos los pares de arcos. Cuando se

encuentra un par que puede intercambiarse, se ejecuta el 2-Change (intercambio 2-Opt). Si durante el ciclo se examinan todos los pares sin poder efectuar un intercambio, entonces el tour es 2-óptimo, y el algoritmo se detiene.

Una implementación obvia del 2-Opt repite el paso básico de considerar los N(N-1) / 2 pares de arcos en el tour, y ejecutar un intercambio del 2-Opt si se lo encuentra. Por lo tanto, cada intercambio 2-Opt toma un tiempo de $O(N^2)$; y el algoritmo podría requerir muchos intercambios para alcanzar el tour final 2-óptimo.

El algoritmo 2-Opt básico puede implementarse en una asombrosa variedad de maneras. Por ejemplo, ¿En qué orden se inspeccionan los arcos potenciales a ser considerados en los intercambios? ¿Se los considera en algún orden estático (por ejemplo, correspondiente al número identificatorio de las ciudades), o se recorre el tour? Y si se recorre el tour, ¿cómo se cambia el orden del recorrido luego de invertir un fragmento del mismo? Una vez elegido el primer arco a considerar en un intercambio, ¿cómo se elige el segundo arco a considerar? Si en un paso se descubren varios posibles intercambios 2-Opt, ¿cuál se toma? Podría elegirse el intercambio que otorgue la máxima ganancia (enfoque "codicioso" o "greedy"), con mínima ganancia (un enfoque más cauteloso para evitar mínimos locales), el primer intercambio que se descubre (para eficiencia en tiempo de corrida), o elegir uno arbitrariamente. Las posibilidades continúan.

La heurística básica elegida para los experimentos presentados en este trabajo, es una variante de la heurística discutida anteriormente. Podemos explicitarla usando una porción de seudocódigo.

Si A_k es el arco número k de un tour T formado por N ciudades (enumeradas arbitrariamente), entonces el procedimiento sería:

```
procedure BÚSQUEDA-LOCAL
begin
   INICIALIZAR (T)
   do
      Mejora = False
      for i = 0 to N-1 do
         for j = (i+2) to (N+i-2) do
            if (Δ (T, A_i, A_(j mod N)) < 0)
               begin
                  HACER-2-CHANGE (T, A_i, A_(j mod N))
                  Mejora = True
                  exit for              /* sale del loop interior sobre j */
               end
   while (Mejora)
end
```

donde INICIALIZAR (T) significa generar aleatoriamente un tour. Los arcos se etiquetan desde 0 hasta N-1, por lo tanto se necesita la expresión $A_{(j \bmod N)}$ (j módulo N), ya que j toma valores entre i+2 hasta N+i-2, e i va desde 0 hasta N-1.

La expresión Δ (T, A_i, $A_{(j \bmod N)}$) representa la variación en la longitud del tour T, resultante de efectuar un intercambio de 2-Change sobre los arcos A_i y $A_{(j \bmod N)}$ de T.

3.5. Mejoras de performance de la heurística 2-Opt

Para obtener tiempos subcuadráticos en la práctica, al correr el 2-Opt, debe hallarse un modo de evitar considerar todas las posibles movidas, al buscar una que disminuya el costo del tour. Hay una forma simple de hacer esto, aprovechando la estructura geométrica del problema. La clave a tener en cuenta es la siguiente observación simple:

> "En cualquier intercambio, al menos uno de los arcos incidentes en uno de los vértices decrece en longitud." [6]

Si los dos nuevos arcos crecieran en longitud, el 2-Change no podría reducir la longitud del tour. Así, la búsqueda del "segundo arco" en un 2-Change puede confinarse a una esfera centrada en un vértice del primer arco, con radio igual a la longitud de dicho arco.

Asumiendo una orientación fija del tour, cada posible 2-Change puede verse como correspondiente a una 4-tupla de ciudades $<c_1, c_2, c_3, c_4>$, donde (c_1, c_2) y (c_4, c_3) son los arcos orientados que se eliminan del tour, y (c_2, c_3) y (c_1, c_4) son los arcos que los reemplazan. Para el caso del TSP simétrico, la observación anterior permite afirmar que en la búsqueda, fijando c_1 y c_2, las posibilidades para c_3 pueden limitarse a las ciudades más cercanas a c_2 de lo que está c_1. (Notar que para la elección dada de c_3, hay una sola elección posible para c_4, tal que la movida produzca un tour en lugar de dos ciclos disjuntos). A medida que el algoritmo progresa, es probable que el conjunto de candidatos para c_3 sea cada vez más pequeño.

Estas afirmaciones se justifican detalladamente en [20], donde también se presenta un razonamiento análogo para el algoritmo 3-Opt. Otros autores (citados en [6]) han desarrollado ideas similares para acelerar la performance del 2-Opt, incluso en el caso de grafos no geométricos.

Para explotar estas observaciones, se necesita una estructura de datos que permita identificar rápidamente los candidatos posibles para c_3. Puede almacenarse para cada ciudad c, una lista de las ciudades restantes, en orden creciente de distancia desde c. Para considerar los candidatos para c_3, sólo se necesita comenzar al principio de la lista de "vecinos" de c_2, y recorrerla hasta encontrar una ciudad x con Dist $(c_2, x) \geq$ Dist (c_1, c_2). En el lenguaje de geometría computacional, se ejecutaría una búsqueda de "vecinos más cercanos con radio fijo", para examinar todos los puntos dentro del círculo centrado en c_2, con radio igual a Dist (c_1, c_2). [3], [20]

Este algoritmo sería sustancialmente más rápido que el algoritmo simple. Si bien resulta muy difícil analizar el tiempo de corrida teóricamente, se recurre a experimentos para comprobarlo. La desventaja de este enfoque es que tomaría un tiempo de $O(N^2 \log N)$ formar las listas de vecinos, y un espacio de $O(N^2)$ para almacenarlas.

Una forma de acelerar aún más la heurística 2-Opt, es implementar una "lista truncada de vecinos", tomando el razonable compromiso de almacenar listas ordenadas que contienen sólo los k vecinos más cercanos de cada ciudad para algún k fijo adecuado, típicamente k=20. En el caso de la 4-tupla anterior, se detendría la búsqueda para c_3, al alcanzar una ciudad que esté demasiado alejada, o al llegar al final de la lista. [20]

De esta manera se evita una desventaja importante de muchas heurísticas para el TSP: el gasto de una cantidad de tiempo innecesaria en considerar arcos "inútiles", es decir, conexiones entre dos ciudades que muy probablemente no estén contenidas en ningún tour razonable. Los arcos a los vecinos más cercanos de una ciudad, proveen candidatos promisorios a ser examinados. [41]

Aún con esta mejora, la construcción de la lista domina el tiempo general de corrida para la implementación del 2-Opt con lista de vecinos. Sin embargo, las listas de vecinos son estructuras estáticas, que pueden ser re-usadas por diferentes corridas del algoritmo sobre la misma instancia, amortizando así el costo de su construcción.

La pérdida de calidad del tour involucrada en el uso de listas truncadas de vecinos no es un punto importante en la práctica, aunque es posible que la solución final no sea verdaderamente un óptimo local. Por ejemplo, en [20] se establece que para instancias euclídeas aleatorias, pasar de k=20 a k=80 sólo tiende a mejorar el tour final en un 0.1 a 0.2 por ciento en promedio, a un costo significativo en cuanto al tiempo de corrida.

Otra idea para reducir el tiempo de ejecución es provista por el bit de "no-mirar". Éste es otro modo de aceptar un pequeño incremento en la posibilidad de que el tour final no sea localmente óptimo, para lograr una reducción importante en el tiempo de corrida, nuevamente basada en la idea de que sólo necesitan considerarse las movidas que mejoren. Se concentra la atención en los nodos que tienden a producir buenos intercambios del 2-Opt. Esta idea es presentada en [3], [6] y [20].

Si para una elección dada de c_1 no se pudo hallar previamente una movida que mejore, y si los vecinos de c_1 en el tour no han cambiado desde entonces, es improbable que pueda producirse una mejora, si se vuelve a empezar la búsqueda desde c_1. Esta observación se explota por medio de marcas o "bits" especiales para cada ciudad. Inicialmente, todos están apagados. El bit para la ciudad c se enciende siempre que falla la búsqueda de una movida que mejore con c_1=c, y se apaga siempre que se ejecuta una movida en la cual c es un extremo de uno de los arcos participantes. Al considerar los candidatos para c_1, ignoramos todas las ciudades cuyos bits están encendidos.

El número de nodos visitados continúa siendo el mismo, pero el número de nodos que se tienen en cuenta para analizar si sería conveniente un 2-Change, decrece considerablemente. Mientras el

verdadero algoritmo se vuelve sustancialmente más lento luego de la primera iteración, el algoritmo aproximado hace más intercambios efectivos. En general, el 2-Opt utilizando el bit de "no-mirar" parece ganar mucho tiempo, a costa de un aumento relativamente pequeño en la longitud del tour.

Además de las anteriores ideas algorítmicas, pueden utilizarse varios trucos que producen meramente mejoras de factor constante, tal como la caché de distancias considerada en[4].

Un cuello de botella en muchas aplicaciones es el cómputo de la función de distancia. En el caso del TSP simétrico, al haber N(N-1) / 2 distancias entre ciudades (o k.N distancias, si se utiliza la "lista truncada de vecinos"), usualmente es impráctico almacenarlas a todas. Utilizando la técnica de caché, se mantienen disponibles las distancias calculadas más recientemente. Cada vez que se necesita conocer la distancia entre un par de ciudades, se verifica si está disponible en la caché. Si es así, se la obtiene directamente; en caso contrario, se efectúa el cálculo correspondiente, y el resultado se almacena en la caché para su futuro uso.

Esta técnica probó ser efectiva, si el algoritmo subyacente despliega una gran cuota de localidad en su uso de la función de distancia. Al ejecutar el 2-Opt para grandes tours del TSP, por ejemplo, la tasa de uso de la caché es de alrededor del 75%. Acceder a la caché toma un tercio del tiempo que el cálculo efectivo de la distancia, por lo tanto, se logra reducir a la mitad el tiempo real utilizado para calcular distancias.

Todas las elecciones que deban hacerse en estas cuestiones de implementación, deben examinarse en detalle. Por ejemplo, cuando se puede elegir entre más de una movida que mejore, elegir la mejor movida parece una buena decisión, pero ésta puede ser costosa de encontrar, de modo que intervienen relaciones de costo-beneficio. Las implementaciones de lista de vecinos típicamente eligen el primer 2-Change posible, pero la búsqueda está sesgada de forma tal de encontrar primero las mejores movidas.

El análisis matemático es una técnica efectiva para comprender el comportamiento de algunos algoritmos. No es, sin embargo, el único método para la tarea. Los experimentos brindan otra herramienta que sirve para guiar la teoría, testear hipótesis, ayudar a efectuar elecciones de implementación, y proveer una visión del tema.

Bentley [5] ensayó numerosas variaciones algorítmicas para mejorar la performance de la heurística 2-Opt. Al no poder analizar los efectos matemáticamente, recurrió a los experimentos. Bentley esperaba que la versión aproximada del 2-Opt (usando los "bits" y la lista truncada de vecinos) fuera significativamente más rápida que la real, pero supuso que produciría tours sustancialmente más largos. Sin embargo, la conclusión fue otra. El tiempo de corrida del 2-Opt aproximado es más rápido y más predecible que el del verdadero 2-Opt. Los tours producidos por la versión verdadera son usualmente un poco más cortos que los aproximados, pero no siempre, y la variación entre un tour y otro es más grande que la diferencia debida al tipo de algoritmo.

Si bien al analizar la performance de estas heurísticas generalmente se considera que las ciudades se encuentran distribuidas en el espacio de manera uniforme, Bentley testeó el algoritmo sobre diferentes clases de distribuciones, incluyendo: uniforme sobre una línea horizontal, una línea diagonal, el borde de un círculo, normal bivariada, etc. Las distribuciones fueron elegidas tanto para modelizar aplicaciones reales, como para servir de contraejemplos del "peor caso" para los algoritmos. Todos los resultados fueron muy cercanos a sus valores en el caso uniforme. El algoritmo 2-Opt es muy robusto, y ninguno de sus valores clave (tiempo de ejecución, número total de nodos visitados, etc), varía ampliamente para distribuciones no uniformes.

4. BÚSQUEDA LOCAL GUIADA

La Búsqueda Local Guiada (GLS, por sus siglas en inglés: "Guided Local Search"), es una metaheurística adecuada para un amplio rango de problemas de optimización combinatoria.

Este método aprovecha información relacionada con la estructura del problema y con el curso de la búsqueda, para guiar la exploración del espacio de búsqueda. Esto es posible aumentando la función de costo del problema, para incluir un conjunto de términos de penalización.

Se realizan llamadas iterativas a un proceso de búsqueda local. Cada vez que queda atrapado en un mínimo local, los términos de penalización se modifican, y se llama nuevamente a la búsqueda local, para minimizar la función de costo modificada. Los términos de penalización reflejan la información previa o reunida durante la búsqueda. Esta información se traduce en restricciones que definen mejor el problema, reduciendo así el número de soluciones candidatas a ser consideradas.

La búsqueda local es confinada por los términos de penalización, y enfoca la atención en las regiones *promisorias* del espacio de búsqueda.

En un sentido general, los objetivos de GLS son similares a los de una clase más amplia de algoritmos de optimización combinatoria, conocidos como Búsqueda Tabú [20]. En realidad, GLS puede clasificarse como un método de Búsqueda Tabú, aunque hay muchas diferencias con los otros métodos desarrollados dentro de ese marco. La principal es que GLS es un algoritmo compacto que puede usarse sin modificaciones para un rango de problemas; esto contrasta con las variantes de Búsqueda Tabú que son casi siempre específicas para el problema. Las similitudes y diferencias entre GLS y Búsqueda Tabú se discuten en [47].

La presente Sección de este informe se basa en los trabajos de Voudouris y Tsang ([47] y [48]). Existe una página en Internet desarrollada por estos autores ([49]), dedicada especialmente a este tema.

4.1. Búsqueda local

La búsqueda local es la base de muchos métodos heurísticos para problemas de optimización combinatoria. Por sí sola, es un método iterativo simple para encontrar buenas soluciones aproximadas. Un ejemplo de búsqueda local es el algoritmo visto en la Sección 3, basado en la heurística 2-Opt.

La idea es la de ensayo y error. Consideremos una instancia de un problema de optimización combinatoria definida por el par *(S, g)* donde S es el conjunto de todas las soluciones factibles, y g es la función objetivo que mapea cada elemento s en S a un valor real. El objetivo es encontrar la solución s en S que minimice la función objetivo g. El problema se plantea como:

$$min\ g(s),\ s \in S \tag{4-1}$$

Como se indicó en la Sección 3, una solución x es llamada mínimo local de g con respecto al vecindario N si y sólo si:

$$g(x) \leq g(y), \ \forall y \in N(x) \tag{4-2}$$

La búsqueda local es el procedimiento que minimiza la función de costo g en un número de pasos sucesivos, en cada uno de los cuales la solución actual x es reemplazada por una solución y tal que:

$$g(y) < g(x), \quad y \in N(x) \tag{4-3}$$

En la mayoría de los casos, la búsqueda local comienza con una solución arbitraria y finaliza en un mínimo local. Hay muchas formas diferentes de conducir la búsqueda local. En el caso general, la complejidad computacional de un procedimiento de búsqueda local depende del tamaño del conjunto de vecindario, y también del tiempo necesario para evaluar una movida. Cuanto mayor sea el vecindario, mayor será el tiempo necesario para examinarlo, y mejores los mínimos locales.

Para el propósito de describir el GLS en el caso general, la búsqueda local se considera como un procedimiento general de la forma:

$$s_2 \leftarrow \textit{procedure}\ \textbf{BÚSQUEDA-LOCAL}\ (s_1, g)$$

donde s_1 es la solución inicial, s_2 la solución final (mínimo local), y g la función de costo a ser minimizada.

GLS hace llamadas iterativas a tal procedimiento, modificando la función de costo entre las sucesivas invocaciones. Antes de cada llamada, la función de costo del problema es aumentada para incluir un conjunto de términos de penalización, que permiten restringir dinámicamente las soluciones.

4.2. Características de la solución y la función de costo aumentada

GLS emplea características de las soluciones, como el medio para describirlas.

Una *característica de la solución* puede ser cualquier propiedad que satisfaga la simple restricción de no ser trivial, es decir, que no todas las soluciones tengan esta propiedad. Las características de la solución son dependientes del problema, y sirven como la interfase del algoritmo con una aplicación particular.

Las restricciones sobre características son introducidas o reforzadas sobre la base de información acerca del problema, y también acerca del curso de la búsqueda local.

La información perteneciente al problema es el costo de las características. El costo de las características representa el impacto directo o indirecto de las correspondientes propiedades de la solución sobre el costo de la solución. Los costos de las características pueden ser constantes o variables.

La información sobre el proceso de búsqueda pertenece a las soluciones que van siendo visitadas por la búsqueda local, y en particular los mínimos locales.

Una característica f_i es representada por una función indicador, de la siguiente manera:

$$I_i(s) = \begin{cases} 1, \text{ si la solución } s \text{ tiene la propiedad } i \\ 0, \text{ en caso contrario} \end{cases} \quad , s \in S \qquad (4\text{-}4)$$

Las restricciones sobre características se establecen aumentando la función de costo g del problema, para incluir un conjunto de términos de penalización. La nueva función de costo formada es llamada *función de costo aumentada*, y se define como sigue:

$$h(s) = g(s) + \lambda \sum_{i=1}^{M} p_i \cdot I_i(s) \qquad (4\text{-}5)$$

donde M es el número de características definidas sobre las soluciones, p_i es el parámetro de penalización correspondiente a la característica f_i, y λ es el *parámetro de regularización*. El parámetro de penalización p_i da el grado con el que es restringida la característica f_i de la solución. El parámetro de regularización λ representa la importancia relativa de las penalizaciones con respecto al costo de la solución, y es de gran importancia debido a que provee el medio para controlar la influencia de la información relativa al proceso de búsqueda.

GLS usa iterativamente la búsqueda local, y simplemente modifica el *vector de penalidades* $\boldsymbol{p}$ dado por:

$$\boldsymbol{p} = (p_1, \ldots, p_M) \qquad (4\text{-}6)$$

cada vez que la búsqueda local se detiene en un mínimo local. Las modificaciones se hacen sobre la base de información.

4.3. Búsqueda Local Guiada

El rol de las restricciones sobre características es guiar la búsqueda local, sobre la base de información no incorporada inicialmente en la función de costo, porque era ambigua o desconocida en ese momento. Dada una aplicación específica, a menudo se tiene alguna idea de lo que constituye una buena o mala solución; la función de costo del problema formula esto en un modo matemático. Sin embargo, hay muchos casos en los que no es posible incluir en la función de costo toda la información disponible acerca del problema. La información no incluida es principalmente de naturaleza incierta.

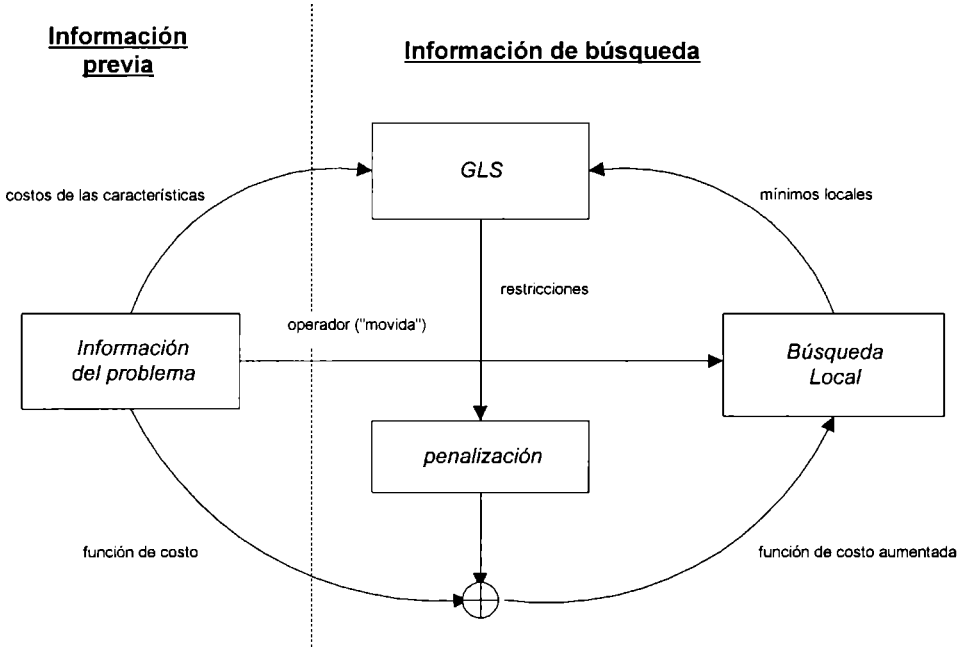
Por ejemplo, en el TSP, los arcos largos son características indeseables, aunque no pueden ser excluidos desde el principio ya que pueden ser necesarios para conectar grupos remotos de ciudades (*clusters*) en el problema. Además, se va recolectando información extra durante el proceso de

búsqueda. Por ejemplo, si una solución es visitada, ésta y otras soluciones (como las de mayor costo) pueden excluirse de las búsquedas futuras. Los algoritmos de branch-and-bound y otros métodos exactos hacen uso de esta observación.

GLS provee la manera de explotar tal información. La información se convierte en restricciones sobre características, que se incorporan entonces a la función de costo usando los términos de penalización modificables. Las restricciones confinan la búsqueda local a las soluciones promisorias con respecto a la información obtenida.

GLS provee un mecanismo simple para introducir o reforzar las restricciones sobre las características de las soluciones. En particular, cada vez que la búsqueda local queda atrapada en un mínimo local, GLS puede incrementar el parámetro de penalización de una o más de las características definidas sobre las soluciones. Si una característica es penalizada, entonces las soluciones que tienen esta característica son evitadas por la búsqueda local. La prioridad es evitar el mínimo local en el cual quedó atrapada la búsqueda local. Por lo tanto, la característica o características penalizadas deben estar entre las exhibidas por este mínimo. Además, las características de alto costo (malas características) deben tomar precedencia sobre las características de bajo costo (buenas características) para su penalización.

La siguiente figura da una visión general del enfoque de GLS:



Inicialmente, todos los parámetros de penalización son puestos en cero (es decir, no se restringe ninguna característica) y se hace una llamada al procedimiento de búsqueda local, para encontrar un mínimo local de la función de costo aumentada (que al ser nulos los términos de penalización, coincide por única vez con la función objetivo del problema). Después del primer mínimo local y de todos los subsiguientes, el algoritmo ejecuta una *acción* de modificación sobre la función de costo

aumentada, y nuevamente usa la búsqueda local comenzando desde el mínimo local previamente hallado. La acción de modificación consiste en incrementar en una unidad el parámetro de penalización de una o más de las características del mínimo local. La información se inserta gradualmente en la función de costo aumentada, seleccionando qué parámetros de penalización incrementar. Las fuentes de información son el costo de las características, y el mínimo local mismo (cuáles son las características que presenta).

A cada característica f_i definida sobre las soluciones se le asigna un costo c_i. Este costo puede ser constante o variable. Para simplificar el análisis, consideremos que los costos de las características son constantes y dados por el *vector de costos* $\boldsymbol{c}$:

$$\boldsymbol{c} = (c_1, \ldots\ c_M) \tag{4-7}$$

que contiene elementos positivos o cero.

Una solución particular que sea mínimo local exhibe un número de características. Si la característica f_i es exhibida por una solución s^* que sea mínimo local, se cumple lo siguiente:

$$I_i(s^*) = 1 \tag{4-8}$$

Así, el vector $\boldsymbol{b}$ con los valores de la función indicador en el mínimo local s*:

$$\boldsymbol{b}(s^*) = (I_1(s^*), \ldots, I_M(s^*)) \tag{4-9}$$

contiene toda la información relacionada con las características del mínimo local.

En un mínimo local s^*, los parámetros de penalización son incrementados en uno para todas las características f_i que maximizan la expresión de utilidad:

$$util\ (s^*, f_i) = I_i(s^*) \cdot \frac{c_i}{1 + p_i} \tag{4-10}$$

En otras palabras, incrementar el parámetro de penalización de la característica f_i se considera una acción con utilidad dada por (4-10). En un mínimo local, se seleccionan y ejecutan las acciones con máxima utilidad.

La función de utilidad hace un uso completo de la información contenida en los vectores $\boldsymbol{b}(s^*)$ y $\boldsymbol{c}$. El parámetro de penalización p_i es incorporado en la fórmula, para evitar que el esquema resulte totalmente sesgado hacia la penalización de características de alto costo. El parámetro de penalización cumple el rol de un contador, que registra la cantidad de veces que una característica fue penalizada. Si una característica es penalizada muchas veces durante un número de iteraciones, entonces el término de la expresión de utilidad en (4-10), c_i / (1+ p_i), decrece para esa característica, diversificando las elecciones y dando la oportunidad de que otras características también sean penalizadas. La política implementada es que las características sean penalizadas con una frecuencia proporcional a su costo. El esfuerzo de búsqueda es distribuido de acuerdo a la *promesa*, expresada por los costos de las características y por los mínimos locales ya visitados.

Algo que no se analizó hasta este momento es el parámetro de regularización λ en la función de costo aumentada (4-5). Este parámetro determina hasta qué grado las restricciones sobre las características van a afectar la búsqueda local. Examinemos cómo el parámetro de regularización va a afectar las movidas ejecutadas por un método de búsqueda local. Una movida altera la solución, añadiendo nuevas características y removiendo otras existentes, mientras deja otras características inalteradas. En el caso general, la diferencia Δh en el valor de la función de costo aumentada, debida a una movida, está dada por la siguiente ecuación:

$$\Delta h = \Delta g + \lambda \cdot \sum_{i=1}^{M} p_i \cdot \Delta I_i \qquad (4\text{-}11)$$

Como puede verse, si λ es mayor, entonces las movidas seleccionadas removerán solamente las características penalizadas de la solución, y la información obtenida durante la búsqueda determinará completamente el curso de la búsqueda local. Esto introduce riesgos, porque esta información puede ser errónea. A la inversa, si λ es 0, entonces la búsqueda local no podrá escapar de los mínimos locales. Sin embargo, si λ es pequeño y comparable con Δg, entonces las movidas seleccionadas apuntarán al objetivo combinado de mejorar la solución (teniendo en cuenta el gradiente) y también remover las características penalizadas (teniendo en cuenta la información reunida). Como la diferencia Δg es dependiente del problema y de la instancia considerada, el parámetro de regularización también lo es. Posteriormente se discutirán algunas formas de determinar valores apropiados para este parámetro.

El seudocódigo del GLS, tal como se lo describió hasta ahora, es el siguiente:

```
procedure GLS (S, g, λ, [I_1, ..., I_M], [c_1, ..., c_M], M)
begin
    k = 0
    s_0 = solución inicial en S
    for i=1 to M do
        p_i = 0
    while CriterioDeFin do
    begin
        h = g + λ * Σ p_i * I_i
        s_{k+1} = BÚSQUEDA-LOCAL (s_k , h)
        for i=1 to M do
            util_i = I_i (s_{k+1}) * c_i / (1+ p_i)
        for each i cuya utilidad util_i sea máxima do
            p_i = p_i + 1
        k = k + 1
    end
    s* = mejor solución hallada con respecto a la función de costo g
    devolver s*
end
```

En este algoritmo, el *CriterioDeFin* puede definirse como un límite sobre el número de iteraciones, un período de tiempo, un límite superior sobre el costo, o combinaciones de lo anterior.

Hay pequeñas alteraciones que pueden mejorar el método descripto. Por ejemplo, en lugar de calcular las utilidades de la fórmula (4-10) para todas las características, pueden tenerse en cuenta únicamente las características del mínimo local, porque para las características que no correspondan a mínimos locales la función de utilidad se vuelve nula al multiplicar por la función indicador. También debe considerarse el mecanismo de evaluación para las movidas. Usualmente, este mecanismo no evalúa directamente el costo de la nueva solución generada por la movida, sino que calcula la diferencia Δg causada sobre la función de costo. Esta diferencia en costo debe ser combinada con la diferencia en penalización, tal como se la muestra en (4-11). Esto puede hacerse fácilmente, y no produce un impacto significativo sobre el tiempo necesario para evaluar una movida. En particular, deben tenerse en cuenta sólo las características que cambian de estado (es decir, que son removidas o agregadas). Se suman los parámetros de penalización de las características removidas, y lo mismo se hace con los de las características agregadas. El cambio en la penalización debido a la movida, está dado entonces simplemente por la diferencia:

$$+ \sum_{\text{sobre las características j añadidas}} p_j \quad - \sum_{\text{sobre las características k removidas}} p_k \qquad (4\text{-}12)$$

4.4. Aplicación de GLS a la resolución del TSP

Un primer paso en el proceso de aplicar GLS a un problema, es encontrar un conjunto de características de la solución que sean responsables de parte del costo total de la solución. Para el TSP, un tour incluye un número de arcos, y el costo de la solución (longitud del tour) está dado por la suma de las longitudes de los arcos en el tour.

Los arcos son características ideales para el TSP. En primer lugar, pueden usarse para definir propiedades de la solución (un tour puede incluir o no un arco dado), y además, poseen un costo determinado por su longitud. Un conjunto de características puede definirse considerando todos los posibles arcos que pueden aparecer en un tour, con los costos de las características dados por las longitudes de los arcos. Una vez que se han definido las características y sus costos, el GLS tal como se describió en el inciso anterior de esta Sección, puede aplicarse al problema sin modificaciones.

Como puede verse en el seudocódigo de GLS, esta técnica no hace ninguna suposición acerca de los mecanismos de búsqueda local; y por lo tanto, puede combinarse con cualquier algoritmo de búsqueda local desarrollado para el problema que se esté enfrentando.

En este trabajo, tal como se describió en la Sección 3, se utilizará una variante del método 2-Opt aproximado, propuesto por Bentley en [6].

Esta heurística de búsqueda local, para ser integrada con GLS, sólo requiere ser implementada como un procedimiento que, a partir de un tour inicial que recibe como entrada, devuelva un tour localmente óptimo con respecto al vecindario considerado.

La matriz de distancia que usará la búsqueda local para evaluar las posibles movidas, es una matriz auxiliar:

$$D' = D + \lambda P = [d_{ij} + \lambda \, . \, p_{ij}] \qquad (4\text{-}13)$$

que combina las penalidades con la función de costo del problema, para formar la función de costo aumentada a ser minimizada por el proceso de búsqueda.

GLS modificará la matriz de penalidades P y a través de ella a D', cada vez que la búsqueda encuentre un mínimo local. Los arcos a penalizar en un mínimo local se seleccionan de acuerdo a la función de utilidad.

La matriz de distancias original D, sigue siendo requerida para determinar si las nuevas soluciones visitadas son mejores que las obtenidas hasta el momento. No es necesario llevar el control del valor de la función de costo aumentada, ya que la heurística de búsqueda local evalúa las movidas basándose simplemente en las diferencias de costo de un tour a otro.

En el caso particular de la heurística elegida, la interfase con el GLS requiere un ajuste adicional, relacionado con los bits de "no-mirar" presentados en la Sección anterior como una forma de reducir el tiempo de ejecución del algoritmo.

Cada vez que se penaliza un arco en GLS, se apagan los bits correspondientes a las ciudades en los extremos de ese arco, indicando que éstas pueden ser exploradas en las búsquedas subsiguientes. Luego de hallar el primer mínimo local, las llamadas a la heurística de búsqueda local examinarán un conjunto reducido de ciudades; en particular, las asociadas con los arcos recientemente penalizados. A lo largo de la búsqueda local pueden activarse nuevas ciudades como consecuencia de las movidas que se ejecutan, aunque en general, la búsqueda local rápidamente llega a otro mínimo local. Esto reduce drásticamente el tiempo de corrida de GLS, forzando a la búsqueda local a concentrarse en los intercambios que remueven arcos penalizados, en lugar de evaluar todas las movidas posibles.

El único parámetro de GLS que requiere cierto refinamiento especial antes de aplicar el algoritmo, es el parámetro de regularización λ. En los estudios realizados por Voudouris y Tsang ([47], [48]), el algoritmo se comportó bien para un rango de valores, dado aproximadamente por la siguiente ecuación paramétrica (aplicable en particular a cada instancia examinada):

$$\lambda = \alpha \, . \, \frac{g(\text{mínimo local})}{N}, \quad 0 \leq \alpha \leq 1 \qquad (4\text{-}14)$$

donde g(mínimo local) es el costo de un tour producido por la heurística de búsqueda local utilizada por GLS (por ejemplo, el primer mínimo local, antes de la aplicación de las penalidades), y N es el

número de ciudades de la instancia. El mejor comportamiento se obtuvo para valores de a alrededor de 0.3.

Para los resultados reportados en [48], el valor exacto de λ usado en las corridas finales fue determinado manualmente ejecutando un número de corridas de prueba, y observando la secuencia de soluciones generadas por el algoritmo. Un algoritmo con un λ adecuado genera una secuencia suave de soluciones que mejoran gradualmente. En caso contrario, el algoritmo o bien progresa muy lentamente (λ es menor de lo que debería ser), o muy rápidamente, por lo que encuentra sólo unos pocos mínimos locales (λ es mayor de lo que debería ser).

5. LOS ALGORITMOS MEMÉTICOS

5.1. Los Algoritmos Genéticos

En 1975, J. H. Holland publicó su libro "Adaptation in Natural and Artificial Systems", punto de partida de un campo luego llamado "Algoritmos Genéticos" (AG). Holland discutió el problema general de explorar un espacio de estructuras con *planes reproductivos*, que seleccionan y crean nuevas estructuras usando un conjunto de reglas denominadas *operadores genéticos*. La población de estructuras funciona bajo una presión selectiva, y como un resultado de estas reglas el objetivo es obtener, por evolución, mejores estructuras. [34]

La meta central era la de comprender los principios subyacentes en los sistemas adaptativos: aquéllos capaces de auto-modificación, en respuesta a las interacciones con el medio en el cual deben funcionar. Desde la perspectiva de Holland la clave de estos sistemas adaptativos robustos, en la naturaleza, estaba en el uso exitoso de la competición y la innovación para proveer la capacidad de responder dinámicamente a eventos anticipados y cambios ambientales[10].

Como se señala en [20], varias técnicas, tales como Búsqueda Tabú y Simulated Annealing, son motivadas por la observación de que no todas las soluciones localmente óptimas son necesariamente buenas soluciones. Así, sería deseable modificar un algoritmo puro de optimización local, mediante algún mecanismo que permita escapar del óptimo local y continuar la búsqueda. Uno de tales mecanismos podría consistir en ejecutar sucesivas corridas de un algoritmo de optimización local, usando una heurística de inicialización aleatoria para proveer diferentes soluciones iniciales. La ganancia en el desempeño por este enfoque de recomienzos aleatorios resulta limitada, sin embargo, y decrece al aumentar el tamaño de las instancias del problema.

Una razón para la limitada efectividad de este método es que no explota la posibilidad de que las soluciones localmente óptimas puedan estar "agrupadas" entre sí; es decir, para cualquier óptimo local dado, una solución mejor podría encontrarse cerca. Si esto fuera cierto, sería mejor recomenzar la búsqueda desde una ubicación cercana a la solución recién hallada, y no desde una ubicación elegida al azar. Esto es lo que se intenta conseguir con el enfoque de AG.

Para entender esta aproximación, puede plantearse el siguiente ejemplo, extraído de[24]:

Supongamos que nos enfrentamos al problema de encontrar el edificio más alto de Nueva York con los ojos vendados. Para establecer una analogía con un problema de optimización combinatoria, también supongamos que podemos caminar en una dirección determinada, entrar en un edificio y "computar" su altura. Tenemos que decidir cuál es la estrategia a seguir si no se pueden visitar todos los edificios (se descarta la enumeración exhaustiva como método válido). No obstante ello, vamos a suponer, por analogía con un algoritmo de computación, que no hay ningún tipo de "cansancio".

Hay que diseñar una estrategia. Por ejemplo, partiendo de un punto inicial elegido al azar, caminar diez kilómetros en una dirección elegida también al azar, medir la altura del edificio encontrado, y repetir este procedimiento por un cierto número fijo de iteraciones. Finalmente reportaremos la altura del edificio más alto encontrado durante todo el proceso. La eficiencia de esta estrategia es evidentemente muy pobre, debido a que podríamos dirigirnos a regiones de muy baja altura y permanecer allí por mucho tiempo.

Podemos entonces añadir una componente probabilística: al iterar, sólo aceptar un nuevo edificio como punto de referencia de un nuevo paso si satisface algún criterio dado.

Pero si en lugar de ir de un punto a otro utilizando pasos de diez kilómetros utilizamos pasos del orden de unos cientos de metros, entonces es más probable que el resultado final sea mejor que el del caso aleatorio. La razón es que no esperamos ninguna correlación fuerte entre las alturas de dos edificios que se encuentran a diez kilómetros. Debido a que las ciudades suelen tener los edificios más altos bastante localizados en áreas pequeñas, esperamos que la búsqueda utilizando pasos de cientos de metros sea más eficiente en la explotación de esta correlación.

Podemos decidir entonces, empezar a añadir más parámetros a la estrategia de búsqueda, como por ejemplo cambiar dinámicamente (es decir basados en la información encontrada hasta ese momento) parámetros tales como la longitud del paso. No descartando que se puedan obtener buenos resultados con estas extensiones, la introducción de nuevos parámetros sólo refleja nuestra ignorancia acerca de cómo resolver estas cuestiones, las cuales necesitan de una solución de tipo más fundamental.

Un cambio radical estaría dado por el uso de más de un "optimizador" para ese problema, es decir, de un método basado en una "población". Se debe considerar cuál es la mejor manera de organizar ese equipo para hacer más eficiente la búsqueda. Supongamos que comenzamos con cierto número de optimizadores dispersos en Nueva York. Cada uno de ellos mide la altura del edificio que está visitando, luego todos comunican este resultado y su posición a los otros. Entonces podemos hacer que estos optimizadores comiencen desde nuevas posiciones, que serán generadas por procesos que seleccionen más frecuentemente a aquellos optimizadores ubicados en los edificios más altos (esperando que exista una correlación). Cada vez que se desee generar una nueva dirección, se tomarán dos o más optimizadores (como "padres" de la nueva población a crear) y se establecerá una nueva posición por algún procedimiento aleatorio que sin embargo garantice que esa posición se encuentre, "en promedio", cerca de las soluciones "padres". El reposicionamiento de los optimizadores estará entonces fuertemente influenciado por aquellos que se encuentren mejor "posicionados".

La estrategia descripta anteriormente guarda una gran similitud con el enfoque de AG.

5.2. Analogías con la evolución

La clave de la evolución natural reside en una sencilla relación entre cambio y selección. Los

organismos se enfrentan a un ambiente que no pueden controlar; prosperarán aquellos individuos que presenten cambios que les supongan, a ellos y a sus descendientes, una ventaja, mientras que tenderán a morir los individuos que sufran cambios desfavorables. El ambiente natural selecciona a favor de los organismos mejor dotados y en contra de los peor dotados.

El éxito o el fracaso de un episodio de la evolución nunca se mide en un individuo aislado o de forma inmediata; se evalúa siempre en una población y en el transcurso de muchas generaciones[19].

Los AG pueden verse como técnicas de optimización basadas en una analogía con la evolución de los sistemas biológicos. Hacen uso de una "población" de individuos, cada uno de ellos "representado" por una configuración, y las configuraciones más "aptas" o con mejores características en cuanto a la función de costo evaluada, son preservadas con mayor probabilidad y utilizadas para generar nuevos individuos (que con una probabilidad apreciable son mejores que los anteriores). Estos algoritmos son parte de lo que constituye el campo de "programación evolutiva". [45]

En la naturaleza, tal como se observa en [24], la habilidad de una población de cromosomas para explorar el espacio de búsqueda "en paralelo" y combinar lo mejor que ha sido encontrado en él mediante el mecanismo de "crossover" es intrínseca a la evolución, y trata de ser imitada por los AG.

Los primeros AG desarrollados eran simples, pero dieron soluciones satisfactorias a problemas que eran considerados como "difíciles" en ese momento. El campo de los AG ha evolucionado desde entonces, principalmente debido a las innovaciones introducidas en la década de 1980. Se han ido incorporando mecanismos cada vez más elaborados, motivados por la necesidad de resolver en forma aproximada una amplia variedad de problemas prácticos.

En [24] se enumeran algunas características de la evolución, enunciadas por L. Davis:

a) La Evolución es un proceso que opera sobre los cromosomas, no sobre los seres vivientes codificados por ellos.

b) Los procesos de la selección natural causan que aquellos cromosomas que codifican estructuras exitosas, se reproduzcan más frecuentemente que aquéllos que no lo son.

c) Las mutaciones pueden causar que los cromosomas de los hijos sean diferentes a los de los padres, y los procesos de recombinación pueden crear cromosomas bastante diferentes en los hijos, por la combinación de material genético de los cromosomas de los dos padres.

d) La evolución biológica no tiene memoria.

La premisa de los AG, y de los numerosos investigadores que los utilizan como una técnica de optimización, es que se pueden encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un proceso de "evolución simulada", en particular como un algoritmo matemático implementado en una computadora. En la descripción de estos algoritmos se utilizan

términos biológicos, ya que constituyen su fuente original de inspiración, y ayudan a comprenderlos intuitivamente.

Como ocurre en la evolución biológica, la evolución simulada estará diseñada para encontrar cada vez mejores cromosomas, mediante una manipulación "ciega" de sus contenidos. El término "ciega" se refiere al hecho de que el proceso no tiene ninguna información acerca del problema que está tratando de resolver, exceptuando el valor de la función objetivo. En la concepción original de los AG, la función objetivo es la única información por la que se evalúa el "valor" de un cromosoma.

Por lo tanto, los AG están basados en integrar e implementar eficientemente dos ideas fundamentales: la habilidad de representaciones simples para codificar las configuraciones del problema de optimización, y el poder de transformaciones simples para modificar y mejorar estas configuraciones.

Las mejoras son guiadas mediante un mecanismo de control que permite a una población de cromosomas evolucionar, tal como lo hacen las poblaciones de seres vivientes. Un AG puede ser visto como una estructura de control, que organiza o dirige un conjunto de transformaciones y operaciones diseñadas para simular estos procesos de evolución.

En la evolución de los seres vivientes, cada individuo enfrenta cotidianamente el problema de la supervivencia. Para ello cuenta con las habilidades innatas provistas por su material genético. A nivel de los genes, el problema a enfrentar es el de buscar aquellas adaptaciones beneficiosas en un medio ambiente hostil y cambiante. Debido en parte a la selección natural, cada especie gana una cierta cantidad de "conocimiento" que es codificado e incorporado en la nueva conformación de sus cromosomas. Esta conformación se ve alterada por las operaciones de reproducción. Algunas de ellas son las mutaciones aleatorias, y el "crossover", que es el intercambio de material genético proveniente de dos cromosomas padres.

En los AG, las mutaciones aleatorias proveen cierta variación, y ocasionalmente introducen alteraciones beneficiosas en los cromosomas. El crossover es el mecanismo responsable del intercambio de material genético entre dos padres para ser combinado en las estructuras hijos. Evidentemente, es la existencia del crossover lo que gobierna la eficiencia de los AG, y los destaca nítidamente de otro tipo de estrategias. Con él, las características de los padres pueden ser combinadas inmediatamente cuando se reproducen. Por lo tanto, la probabilidad de esta combinación se acrecienta cuando los padres tienen un buen valor de la función objetivo, debido a que en este caso se reproducen con mayor frecuencia.

5.3. Esquema básico de un Algoritmo Genético

Como se describe en [45], un AG es un procedimiento iterativo que mantiene una población con una cantidad generalmente fija de soluciones candidatas. En cada iteración, llamada "generación", se evalúan las configuraciones de la población actual, y sobre la base de esta evaluación se genera una

nueva población de soluciones candidatas. Expresado esto como seudocódigo (llamando S a la población de soluciones considerada):

```
procedure AG (S)
begin
    INICIALIZAR (S)
    EVALUAR (S)
    while NoConverge do
        begin
            SELECCIONAR (S)
            RECOMBINAR (S)
            EVALUAR (S)
        end
    Devolver la mejor solución en S
end
```

donde:

- Cada ejecución del ciclo consistente en SELECCIONAR, RECOMBINAR y EVALUAR la población, puede verse como el procesamiento de una única *generación* en el proceso evolutivo.
- INICIALIZAR (S), significa generar la población de k soluciones iniciales $S = \{S_1, ..., S_k\}$. Esto puede realizarse al azar, o mediante una heurística.
- EVALUAR (S), es dar a cada configuración ("individuo") un peso de importancia con respecto a las demás. Fundamentalmente se tiene en cuenta el valor de la configuración con respecto a la función objetivo.
- SELECCIONAR (S), significa elegir de la población las configuraciones que servirán de base para generar la población siguiente. Se eligen k' subconjuntos de S, generalmente de tamaño 2, como padres para formar la próxima generación. Este paso se denomina *estrategia de apareamiento*. En general, a mejores configuraciones de origen para la generación de la nueva población, se tendrán mejores configuraciones como resultado.
- RECOMBINAR (S), es utilizar las configuraciones seleccionadas en el paso anterior para generar la población siguiente. En este paso se hace uso de la técnica conocida en los AG como *crossover*, que ejecuta una operación sobre cada subconjunto de "padres", para obtener una nueva solución que refleje aspectos de todos ellos. Luego de llevar a cabo el crossover, se utiliza una *estrategia de selección* para elegir k sobrevivientes (entre la unión de las soluciones de S, con las generadas como descendencia), y se reemplaza el contenido de S por estos sobrevivientes.

En ocasiones, además de recombinar las soluciones entre sí, puede realizarse un paso adicional al que se denomina *mutación*, que introduce una variación aleatoria sobre subconjuntos seleccionados de

la población, generalmente de un elemento. En una mutación, alguna forma de alteración aleatoria introduce cambios localizados en las soluciones, dejando intacta la mayor parte de su estructura.

Se debe decidir cuándo se considera que S no puede mejorarse. Existen al menos dos tendencias en este sentido. Una de ellas es efectuar una gran cantidad de iteraciones, que aseguren la mayor calidad posible de las configuraciones. Esto implica decidir de antemano la cantidad de generaciones a realizar. La otra tendencia es decidir la finalización del proceso por un criterio basado en las configuraciones de la población "actual". En el primer caso se tiende a realizar más generaciones de las necesarias, y en el otro caso se debe elegir el criterio de finalización, y agregar el tiempo de cómputo necesario para verificar si este criterio se cumple o no en cada generación.

El esquema presentado refleja un modelo muy simple de evolución, que trata de incorporar los conceptos de supervivencia, selección del más apto y reproducción sexuada.

La operación sobre distintas soluciones puede ejecutarse en paralelo si se desea, por lo tanto a veces se lo llama el "AG Paralelo". [20]

Ésta es una de las primeras "Metaheurísticas" que se utilizaron para los problemas de optimización combinatoria. Se puede hablar de metaheurística en vez de heurística, porque es lo suficientemente general como para atacar cualquier problema de optimización, que no es lo que sucede, por ejemplo, con la heurística 2-Opt vista en la Sección 3.5 (específica para el TSP y otros problemas que involucran permutaciones). Un AG puede verse como una metaheurística de mejoramiento iterativo de una población de configuraciones, que a partir de una población inicial, intenta mejorarla sucesivamente eligiendo las mejores configuraciones y dando origen a otras nuevas en cada generación o iteración.

Gradualmente, durante los últimos años, un grupo creciente de investigadores comenzó a usar AG en algunas aplicaciones reales. En [34] se citan varias aplicaciones de este enfoque. En [24] se describe la aplicación de AG en diversos casos de estudio, tales como el problema de coloreo de grafos, el perceptrón binario, organización de cronogramas en una liga de hóckey. Además se citan muchísimos trabajos sobre técnicas basadas en poblaciones, aplicadas a casos muy variados en los campos de Investigación Operativa, "Scheduling" de Multiprocesadores, Electrónica y VLSI, Biología Molecular y Fisicoquímica, Ingeniería en Construcciones, Búsquedas en bases de datos, Geofísica, Redes Neuronales.

Las debilidades y fortalezas de esta estrategia se hacen más explícitas en la presencia de muchas diferentes aproximaciones "genéticas" para un único problema, y posiblemente el mejor ejemplo de este hecho sea el TSP.

5.4. Componentes a considerar

Gran número de investigadores se han dedicado a estudiar las numerosas variantes del esquema básico de AG, cambiando por ejemplo el número de individuos, el proceso de selección de los padres

de la nueva población, el proceso de creación de nuevas direcciones de búsqueda (recombinación), la introducción de pequeñas mutaciones al producto de esta recombinación.

Independientemente de cuán sofisticado pueda ser el diseño de un AG, existen algunas componentes que deben ser incluidas: [24]

- una forma de representar las configuraciones (individuos) del problema.
- una manera de crear las configuraciones de la población inicial.
- una función de evaluación, que permita ordenar los individuos de acuerdo a su valor asociado de la función objetivo.
- operadores "genéticos", que permitan alterar la composición de los nuevos individuos generados por los padres durante la reproducción.
- valores de los parámetros del AG: [45]
  - tamaño de población: cantidad de configuraciones que tendrá S.
  - cantidad de configuraciones que se utilizarán para aplicar el crossover.
  - cantidad de configuraciones que se reemplazarán en una generación.
  - estrategia de selección: forma de seleccionar los padres de una generación entre todos los individuos que componen la población.

Se llega entonces a tener un doble problema de optimización: el de los parámetros que gobiernan la heurística (que no son independientes entre sí), y el que se intenta resolver. [45]

Otro problema de los AG es el de la cantidad de iteraciones que se deben realizar. Esto depende de la decisión que se tome para el criterio de finalización del algoritmo. En la mayoría de las publicaciones se dan resultados de algunos problemas con cantidades fijas de generaciones, donde la población evoluciona según la cantidad de iteraciones realizadas. En otros casos se determina un criterio de fin basado en la evolución del AG; comúnmente se finaliza al producirse la convergencia del proceso evolutivo en un individuo particular.

El éxito de una implementación de un AG es altamente dependiente de la selección adecuada de cada uno de sus componentes. A continuación se tratarán algunos temas relevantes para el diseño de un AG.

La población inicial

En [24] se hacen algunas consideraciones sobre la población inicial de un AG. Ésta puede ser creada de diferentes maneras.

Al enfrentar un problema nuevo, se puede aprender mucho inicializando esta población de manera aleatoria. Hacer evolucionar una población que ha sido generada aleatoriamente hasta llegar a tener una "bien adaptada", es decir, con configuraciones satisfactorias que sean soluciones aproximadas del problema de optimización, es un buen modo de saber cómo está funcionando la implementación del AG. Las características esenciales de la solución final deben ser consecuencia de este proceso evolutivo, y no de los métodos usados para generar la población inicial.

Para algunas aplicaciones, especialmente las industriales, puede ser conveniente inicializar con métodos más directos. Estas técnicas incluyen el uso de algoritmos "greedy" (golosos), otro tipo de algoritmos constructivos, e inicialización por perturbación de soluciones generadas por algún experto en el problema.

El crossover

Como se explica en [24], el *crossover* es el procedimiento por el cual dos seres vivientes intercambian parte de su material genético para crear un nuevo organismo. En los AG, la técnica u "operador" de crossover recombina la información de ciertos individuos para generar otros nuevos.

El propósito de la selección de padres es incrementar la probabilidad de reproducir miembros de la población que tengan buenos valores de la función objetivo. Una vez que los padres han sido elegidos, se utiliza un operador genético como el crossover. Las configuraciones generadas se denominan "hijos" o "descendencia".

Las configuraciones surgidas por el crossover guardan ciertas similitudes con las configuraciones de las cuales se originan. Aquí es donde una configuración se ve desde el punto de vista genético (en sentido biológico) como una sucesión de genes. Cada configuración hijo tendrá algunos genes semejantes a una configuración padre, otros genes a otra y, eventualmente, algunos genes con valores distintos de todas las configuraciones que se tomen como padres. Definir un crossover implica definir, en el contexto de cada problema que se trate: [45]

a) Qué se toma como un "gen" de una configuración.

b) Cuántas configuraciones se tomarán como padres: generalmente se toman dos.

c) Cuántas configuraciones se generarán a partir de los padres: generalmente una y en algunas ocasiones dos, aunque no está, en principio, limitada la cantidad.

d) Qué genes toma cada hijo de cada padre: se tiende a que la mayoría de los genes de cada hijo provengan de alguno de los padres, y el resto tengan valor al azar.

Según se explica en [37], el crossover proporciona una forma de explorar el espacio de búsqueda. Se dice que un problema de optimización combinatoria tiene la característica de *bloques constructivos*, si las subcadenas que forman las soluciones óptimas (en el caso del TSP, las partes que componen el tour) están contenidas en otras buenas soluciones. En este caso, parece una buena estrategia generar nuevas soluciones uniendo subcadenas de las soluciones anteriores. Esto es exactamente lo que hace el operador de crossover.

Puede establecerse la siguiente relación: cuanto mejores sean las soluciones, más similares serán entre sí. Esta relación también se cumple si el problema tiene la característica de *bloques constructivos*, mencionada en [10] y [37], la cual es necesaria para el éxito del operador de crossover.

En [24] se describen diversos operadores genéticos de crossover, incluyendo técnicas especialmente diseñadas para el dominio de aplicación del TSP.

Existe un continuo flujo de propuestas para nuevos crossovers de dos padres (entre muchas otras, las descriptas en [13], [14], [20], [24], [34], [45]). Además están surgiendo desarrollos no tradicionales, que involucran crossovers con más de dos padres.

Pérdida de diversidad

En el contexto de los AG se puede hablar de la "diversidad de la población", que es un valor que cuantifica la diferencia (o semejanza) entre las configuraciones de la población [45]. El cálculo efectivo de este valor consiste usualmente en la comparación entre los genes de las configuraciones. La diversidad de la población da el criterio natural de finalización de un AG ya que, si la diversidad es reducida o nula, lo cual significa que todas las configuraciones son similares o iguales entre sí, no tiene sentido continuar iterando. Esto se basa en que el crossover no generará configuraciones suficientemente diferentes a las actuales, que permitan mejorar los valores de la función objetivo. Respecto de la diversidad de la población debería definirse:

a) Cuántas configuraciones de la población se tendrán en cuenta para comparar sus genes.

b) Qué genes de las distintas configuraciones se compararán.

c) Cuántos genes iguales de distintas configuraciones se consideran suficientes para afirmar que tales configuraciones son muy similares (no hay diversidad suficiente).

Sobre estas decisiones hay menos estudio que para el crossover, y en parte se debe a que, como se dijo anteriormente, muchas veces se usa como criterio de finalización de un AG el realizar una cantidad de iteraciones lo suficientemente grande como para no tener posibilidad de mejorar las configuraciones de la población.

Se puede agregar un comentario más acerca de la diversidad, y es la relación entre la diversidad (o la pérdida de la misma) y el tamaño de la población. Mientras mayor sea el tamaño de la población, más demorará ésta en perder diversidad. Cuando se tienen más individuos, se tienen más valores posibles para cada gen o combinaciones de genes cuando se haga el crossover y, por lo tanto, se realizarán más iteraciones (generaciones) hasta que todos los individuos de la población se parezcan lo suficiente como para afirmar que se perdió diversidad. Cuando el tamaño de la población es demasiado pequeño, se habla de "convergencia prematura", lo cual significa que la población perdió diversidad (por lo tanto no se puede mejorar sustancialmente) y los valores asociados a sus configuraciones aún no son lo suficientemente buenos.

5.5. Incorporación de conocimiento acerca del problema

En el campo del TSP, muchos estudios han indicado que los enfoques puros de AG son superados por las heurísticas convencionales de búsqueda local, particularmente cuando el tamaño del problema crece. La obtención de soluciones de más alta calidad parece requerir de los esfuerzos combinados de varios métodos. El deseo de mejorar el desempeño de los AG en aplicaciones del TSP, ha motivado a

varios investigadores a incorporar elementos heurísticos adicionales en estos algoritmos.

Diversos estudios parecen indicar que la única manera de obtener resultados comparables a las buenas técnicas convencionales de búsqueda local, es incorporar el conocimiento específico del TSP a los AG. En muchos casos, la eficiencia de búsqueda de los métodos generales se incrementa fuertemente cuando se les provee alguna forma de conocimiento del dominio acerca del TSP.[13]

Las principales propuestas para aumentar los AG con dicho conocimiento (información heurística) se enumeran en [14]:

a) En lugar de generar tours aleatorios, que forman los individuos de la población inicial de un AG, es posible usar una heurística de construcción de tours, tal como Nearest-Neighbour o varias heurísticas de inserción [20]. Como debe crearse una población completa, la heurística a usar debe ser capaz de producir diferentes tours.

b) La recombinación genética ofrece un gran potencial para el uso de elementos heurísticos. Pueden diseñarse operadores de crossover especiales, ligados a las características del TSP.

c) Es posible reemplazar la mutación por una de las heurísticas de mejoramiento de tours bien conocidas, tales como 2-Opt, 3-Opt o Lin-Kernighan ([6], [20], [25], [41]).

d) Pueden aplicarse heurísticas de mejoramiento iterativo a algunos o todos los individuos de la población actual. Estas heurísticas también pueden usarse para el post-procesamiento: sobre la población resultante, o solamente el mejor individuo, puede efectuarse una corrida adicional de un algoritmo de mejora del tour.

El siguiente esquema, dado en [14], resume las posibilidades de incorporar heurísticas del TSP en un AG, mostrando una "plantilla" general para la construcción de tal "*AG heurístico*":

Paso 1: Crear una población inicial por una heurística constructiva
Paso 2: Aplicar una heurística de mejoramiento iterativo a la población inicial
Paso 3: Evaluar la población resultante
Paso 4: Mientras (no converge)
  4.1 Selección: elegir los padres para el apareamiento
  4.2 Recombinación: ejecutar una heurística de crossover
  4.3 Aplicar una heurística de mejoramiento iterativo al resultado
  4.4 Mutación: modificar individuos con una probabilidad dada
  4.5 Reemplazo: reemplazar algunos padres con nuevos hijos
  4.6 Evaluar la población resultante
Paso 5: Ejecutar el post-procesamiento aplicando una heurística de mejoramiento iterativo
Paso 6: Evaluar la población resultante
Paso 7: Devolver la mejor solución de la población

Este esquema no sigue exactamente lo que se conocía como "Algoritmo Genético" en sus comienzos. En particular, la aplicación de optimización local a las soluciones individuales en los pasos 2 y 4.3 puede verse como un agregado casi herético: en el contexto de la motivación biológica original para el enfoque genético, involucra el desacreditado principio de Lamarck por el cual las habilidades adquiridas pueden ser heredadas [28]. Sin embargo, tales pasos de optimización local parecen ser esenciales para obtener resultados competitivos para el TSP. [20]

El método descripto puede verse como un paso más allá en la dirección de programación evolutiva.

Aunque la integración del conocimiento del dominio del TSP a los AG de acuerdo a este esquema parece un enfoque promisorio, varios intentos de usar heurísticas del TSP en pasos particulares de la "plantilla" no dieron buenos resultados. El uso de ciertas heurísticas de crossover, y de técnicas constructivas para generar la población inicial, produjeron soluciones incluso peores que las generadas por la heurística simple 2-Opt.

En cambio, los enfoques que utilizan un paso de búsqueda local dentro del AG (Paso 4.3), parecen ser la única manera de obtener resultados comparables a los de las técnicas convencionales de búsqueda local. Esta opinión se expresa en [14] y [20]. Un paso importante en esta dirección fue dado por Mühlenbein [37] con el desarrollo de "AG Paralelos", que permiten que los individuos en la población mejoren su valor de la función objetivo mediante procesos iterativos. También se permite que los individuos se seleccionen entre ellos por procesos "locales", lo que facilita su implementación en sistemas concurrentes. Mühlenbein justifica su aproximación genética con búsqueda local, haciendo un análisis del espacio de configuraciones del TSP. De sus observaciones, concluye que los arcos de los mejores tours 2-Opt están con gran probabilidad contenidos en otros tours 2-Opt, y que combinando dos buenos tours, la probabilidad de obtener un tour mejor es más alta que combinando dos tours 2-Opt arbitrarios. Freisleben y Merz [14] sugieren que esto también es válido para tours producidos por otras heurísticas de mejoramiento iterativo. Se dice que las superficies de costos del TSP exhiben una estructura de "gran valle", es decir, el óptimo global se encuentra cerca del centro de un único valle de soluciones casi óptimas.

De acuerdo con esta idea, cierto número de investigadores han reconocido que para el TSP existe una gran literatura en heurísticas rápidas de búsqueda local. Entonces han optado por apartarse de la aplicación de AG tradicionales, y comenzaron a combinar métodos de búsqueda local (inclusive basados en metaheurísticas como Búsqueda Tabú y Simulated Annealing), con métodos basados en el uso de una población, y la utilización de procesos de recombinación de individuos. [24]

Algunos investigadores definen esta estrategia como *métodos aumentados por conocimiento*. El campo que hace uso de heurísticas de búsqueda local con una estrategia basada en poblaciones ha sido denominado "Algoritmos Meméticos" (AM), debido a su analogía con la evolución *cultural* en lugar de biológica. Este nombre tiene su origen en la palabra "meme", introducida por Dawkins para significar la "unidad de imitación" en la transmisión cultural.

Aquí el término "memes" representa a aquellas estructuras indefinidas que son responsables por obtener tours de bajo costo mientras se realiza la búsqueda.

5.6. Los Algoritmos Meméticos

El primer uso del término "Algoritmos Meméticos" en la literatura apareció en 1989, en el paper de P. Moscato "On Evolution, Search Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". Este trabajo discutía una heurística desarrollada por el autor y por M. G. Norman, que usaba Simulated Annealing como método de búsqueda local, en el marco de una estrategia competitiva-cooperativa en una población de agentes. Esta estrategia utilizaba el TSP como un caso de prueba representativo. [32]

Los AM están ganando amplia aceptación, en particular en problemas bien conocidos de optimización combinatoria para los cuales grandes instancias han sido resueltas hasta la optimalidad, y donde otras metaheurísticas fallan. P. Moscato mantiene una página en Internet ([31]) dedicada a unificar la información sobre AM que está disponible en la Web para su acceso público.

Diversos resultados en el campo del TSP, citados en [34], muestran que hay evidencia computacional consistente con la hipótesis de que un grupo de procesos individuales en competencia y cooperación, que pasan por períodos de optimización individual, pueden superar la brecha que va desde la optimización local a la global.

La idea básica de este enfoque es que el AM recombine información de tours provenientes del espacio de los mínimos locales, en lugar del espacio de búsqueda de todos los tours factibles.[14]

En cierto sentido, los AM pueden ser vistos como métodos sofisticados de descenso múltiple (multistart). El proceso de re-inicialización es gobernado en este caso por reglas genéticas, y la fase de descenso se lleva a cabo como de costumbre. El éxito de estos métodos puede ser atribuido a su balance entre tener una búsqueda rápida, y mantener una diversidad para evitar la convergencia prematura. [24]

En [45] se establecen las principales similitudes y diferencias entre los AG y los AM.

Se pueden enunciar al menos dos características propias de los AM que no se encuentran en los AG "tradicionales":

a) Cada configuración, antes de ser evaluada, se optimiza utilizando alguna heurística apropiada para el problema. Cada configuración evaluada será entonces un mínimo local. Es aquí donde se utilizan heurísticas como el 2-Opt (descripta en la Sección3).

b) La población se divide eventualmente en regiones, que establecen una "subpoblación". Se puede ver así una "población de poblaciones". La evaluación, selección y recombinación se realiza primero en las regiones en las cuales la población se divide, y luego entre individuos de

distintas regiones. Si la población no está dividida, entonces la evaluación, selección y recombinación se realiza como en los AG.

Las características comunes entre los AM y los AG son:

a) Hacen uso de una población de configuraciones.

b) Mejoran una población iterativamente.

c) En cada iteración seleccionan las mejores configuraciones, para generar nuevas configuraciones.

d) Generan nuevas configuraciones vía crossover.

Si se denomina S a la población de soluciones considerada, el seudocódigo de un AM podría expresarse de la siguiente manera:

```
procedure AM (S)
begin
    INICIALIZAR (S)
    OPTIMIZAR (S)
    EVALUAR (S)
    while NoConverge do
        begin
            SELECCIONAR (S)
            RECOMBINAR (S)
            OPTIMIZAR (S)
            EVALUAR (S)
        end
    Devolver la mejor solución en S
end
```

donde OPTIMIZAR (S), significa someter a cada individuo de la población S a una heurística de mejoramiento iterativo, para obtener una solución localmente óptima.

Cabe aclarar que en el seudocódigo anterior no se incluye la división de la población en subpoblaciones, ya que en este trabajo no se realizará dicha división.

Todas las configuraciones que constituyen la población, optimizan sus "tours actuales" con períodos de búsqueda local. Luego de alcanzar un óptimo local, se recombinan con un cierto conjunto de reglas dadas por el crossover. La población resultante de este proceso, será nuevamente optimizada. [34]

El esquema básico planteado deja varias operaciones y definiciones sin especificar. Una adaptación concreta del esquema al TSP necesita especificar los métodos para generar soluciones (tours) iniciales, el algoritmo de optimización local a usar, la estrategia de apareamiento, la naturaleza y cantidad de individuos involucrados en la operación de crossover, la estrategia de selección para las soluciones resultantes, y el criterio de convergencia del algoritmo [20]. Todos estos puntos se tratarán en la Sección 6, que abarca la implementación concreta de un AM realizada en este trabajo.

5.7. Factores para el éxito de los Algoritmos Meméticos

En los AM, al transformar cada configuración en un óptimo local utilizando una heurística, se incorporan mejores características aún a cada individuo (ya no es una configuración aleatoria proveniente de la recombinación de dos individuos de la generación previa). Además, se asegura de alguna manera que la cantidad de iteraciones de un AM para la búsqueda global resulte mucho menor que para un AG.

Como se explica en [45], desde el punto de vista de las heurísticas, se tiende tanto a subsanar la dependencia con respecto a las configuraciones iniciales, como a aprovechar los óptimos locales encontrados previamente cada vez que se aplica la heurística a una nueva configuración.

En cuanto a las configuraciones iniciales, la evolución de la población no es totalmente dependiente de ellas, porque si bien los descendientes toman la mayoría de los genes de alguno de sus padres, pueden tener algunos genes asignados al azar, y además las combinaciones de genes tomados de distintos padres serán distintas de las mismas combinaciones de genes en cada padre.

En lo que respecta a aprovechar todo lo hecho previamente cada vez que se aplica la heurística para encontrar un óptimo local, con el crossover se logra obtener información útil de los óptimos locales encontrados. Al asignar valores a los genes de la descendencia según los valores de los padres, se está aprovechando lo calculado por la aplicación de la heurística a configuraciones previas. Las configuraciones a partir de las cuales se buscarán nuevos óptimos locales, resultan de óptimos locales encontrados anteriormente utilizando un crossover. Como los crossovers se definen de acuerdo al problema, se tiende a que cada crossover sea lo más "inteligente" posible, es decir que utilice al máximo la información relevante del problema que se está intentando resolver.

Aún sin los pasos de optimización local, un AG puede considerarse como una variante de búsqueda local. Hay una estructura de vecindario implícita, en la cual los vecinos de un par de soluciones son aquellas soluciones que pueden alcanzarse por una operación de apareamiento (crossover). Agregando las heurísticas de búsqueda local, el esquema (AM) también puede verse simplemente como una variante del enfoque algorítmico "multistart": elegir la mejor entre varias corridas de optimización local. Aquí, en lugar de comienzos independientes aleatorios, se usan operaciones motivadas genéticamente para construir lo que se espera que sean mejores tours iniciales, ya que incorporan el conocimiento obtenido de corridas previas [20].

En [13], Freisleben y Merz también señalan otros enfoques, como el de *"Large Step Markov Chains"* y la técnica de *"Lin-Kernighan Iterada"*, como casos especiales de AM. Así, la propuesta de AM promete ser más poderosa que estas estrategias, las cuales están entre las mejores aproximaciones heurísticas sugeridas para la resolución del TSP.

La mayoría del trabajo realizado sobre AG y AM es difícil de extrapolar a otras instancias. Típicamente, los experimentos se ejecutan sobre unas pocas instancias específicas, y no se puede determinar cómo deben escalarse los parámetros claves (por ejemplo, el número de iteraciones), para reflejar el tipo o tamaño de las instancias. Los resultados de [20] ilustran la "negociación" que pueden lograrse entre el tiempo de corrida y la calidad de las soluciones, ajustando el número de iteraciones del algoritmo.

Diversos experimentos citados en [20], mostraron que mayores tamaños de población pueden conducir a mejores resultados para el TSP, en un lapso dado de tiempo secuencial. Además, la elección de diversas estrategias de apareamiento y selección puede influir significativamente en la calidad de los tours finales.

Sin embargo, lo más importante es determinar qué efectos tiene la elección del algoritmo de optimización local. En el contexto del TSP, se ha sugerido que éste es sin lugar a dudas el factor principal; sugestivamente, han podido desarrollarse algunos algoritmos "meméticos" con resultados muy promisorios, con un tamaño de población de k=1. (En términos biológicos, esto podría denominarse "Algoritmo Partenogenético").

En tales algoritmos, por supuesto no son posibles los apareamientos, y debe restringirse la atención a las mutaciones. En un AG común, las mutaciones funcionan como un modo de agregar diversidad a la población; en este caso, en cambio, constituyen la única manera de que el proceso siga funcionando.

En [37] se citan trabajos que investigaron la influencia del método de búsqueda local sobre la calidad de las soluciones, utilizando AM con tamaños de población muy pequeños.

Los resultados indicaron que los AM resultan más eficientes que la técnica de múltiples corridas. Además, la eficiencia de la búsqueda memética mejora con la calidad de la búsqueda local.

Se ha planteado la idea de que la verdadera contribución de la estrategia genética/memética, es que provee un método efectivo para generar adaptativamente tours iniciales cada vez mejores para su subrutina de optimización local. [20]

Los resultados del desempeño presentados para varias instancias del TSP, han mostrado que el enfoque de AM puede producir soluciones de alta calidad en un tiempo razonable. [13]

6. LA METAHEURÍSTICA

6.1. Combinación de Algoritmos Meméticos y Búsqueda Local Guiada

En las Secciones 4 y 5, se analizaron por separado los conceptos de Búsqueda Local Guiada (GLS) y Algoritmos Meméticos (AM). A continuación se explicará el esquema de trabajo desarrollado, donde se reúnen estos conceptos.

Se desarrolló una metaheurística que combina características de los AM y el GLS. Dentro del tema ampliamente difundido del Problema del Viajante de Comercio, se trata de un enfoque novedoso, ya que no se conocen trabajos publicados en este campo.

La base de la metaheurística es un AM que como vimos, partiendo de una población inicial de soluciones válidas del Problema del Viajante de Comercio, intenta mejorarlas iterativamente eligiendo las más aptas y generando otras nuevas en cada iteración. Antes de cada iteración se efectúa un paso de optimización local, por el cual cada individuo busca reunir las mejores características antes de transmitirlas.

La población va a evolucionar en primera instancia mediante el intercambio de características de sus individuos (soluciones). En el Problema del Viajante de Comercio, las características a intercambiar son los arcos que unen las ciudades. Se mantiene una cantidad fija de soluciones candidatas. En cada iteración, se evalúan estas soluciones, se seleccionan las que sirven de base para formar la población siguiente, y se genera dicha población mediante una técnica de recombinación (crossover).

Luego de este paso de recombinación, y antes de la optimización local, un individuo será sometido a mutación, con el objetivo de introducir diversidad adicional a la población.

La etapa de optimización local del AM, se realizará mediante GLS.

Tal como se explicó en la Sección 4, GLS efectúa llamadas iterativas a un proceso de búsqueda local. La heurística de búsqueda local que se utilizará en este trabajo, es una variante de la heurística 2-Opt explicada en la Sección 3.4, que incorpora las modificaciones detalladas en la Sección 3.5 para obtener mejoras de performance.

Si bien existen métodos de búsqueda local mucho más poderosos, que aprovechan propiedades de los tours o utilizan estrategias de geometría computacional, en este caso se eligió una heurística muy sencilla, ya que el acento del trabajo está puesto en investigar los posibles beneficios de combinar AM y GLS. Por lo tanto, se intenta reducir al mínimo cualquier otra variable que pueda influir en los resultados.

Empleando todo lo explicado anteriormente, el esquema general visto en la Sección 5.6, podría expresarse en forma más específica como:

```
procedure METAHEURÍSTICA (S)
begin
    INICIALIZAR (S)
    GLS (S)                                    /* paso de optimización */
    EVALUAR (S)
    while NoConverge do
        begin
            SELECCIONAR (S)
            CROSSOVER-CONFIGURACIONES (S)      /* paso de recombinación */
            MUTACIÓN-DE-UN-INDIVIDUO (S)
            GLS (S)                            /* paso de optimización */
            EVALUAR (S)
        end
    Devolver la mejor solución en S
end
```

6.2. Crossover de parámetros de la Búsqueda Local Guiada

El funcionamiento del esquema de GLS depende de una serie de parámetros.

Entre ellos, un parámetro fundamental es la importancia relativa que tendrán los términos de penalización en la función de costo modificada, con respecto a la función de costo original. Este parámetro, designado como λ en la Ecuación (4-5) que define la función de costo aumentada para GLS, indica cuánta es la influencia que tendrán las restricciones sobre características, durante el proceso de búsqueda local.

También pueden considerarse como parámetros muy importantes, los valores de penalización p_i, asociados a cada una de las características c_i de las soluciones, y que intervienen también en la Ecuación mencionada. En el caso del Problema del Viajante de Comercio, se asocia un parámetro de penalización a cada posible arco del tour, formándose así una matriz de penalidades.

Resulta crucial para la metaheurística, por lo tanto, la correcta determinación del valor de estos parámetros.

A diferencia de un AM tradicional, se introduce en este trabajo una segunda etapa de evolución, donde se intercambian los parámetros de GLS entre los individuos de la población.

Durante el proceso de evolución normal de la población, después de ejecutar GLS en cada individuo, se realiza el crossover entre las características de las soluciones obtenidas (arcos que conectan las ciudades), para formar la nueva generación de individuos. Cuando no se obtienen mejoras en esta evolución después de cierta cantidad de iteraciones, se pasa a una segunda instancia, en la cual se evalúa el comportamiento del proceso de GLS en cada individuo, de acuerdo a los

parámetros que lo gobiernan. En lugar de intercambiar características de sus individuos, la población intercambia los valores de los parámetros de GLS en cada uno de ellos. Con esto se espera que el mismo proceso determine en tiempo de corrida y en forma "inteligente", el mejor valor para dichos parámetros, de acuerdo a la instancia del problema que se esté considerando y a las configuraciones que se tengan como solución hasta el momento.

Al modificar el valor de estos parámetros cambia el comportamiento de GLS; y tiene sentido continuar con la evolución normal de la población, ya que pueden obtenerse soluciones diferentes a las anteriores.

El crossover de parámetros puede verse como una automatización del proceso de "tuning de parámetros", que constituye una parte esencial de la metaheurística. El refinamiento de los valores de los parámetros es un asunto crucial, tanto en el desarrollo científico como en el uso práctico de las heurísticas. En algunos casos, la intervención del usuario en este proceso hace difícil reproducir los resultados de una heurística, y el buen funcionamiento de las técnicas depende de las capacidades del usuario. Si el aprendizaje puede ser realizado "on-line" por el mismo algoritmo, la historia previa de la búsqueda determinará el balance apropiado entre diversificación e intensificación, a través de un loop interno con retroalimentación que permite transferir automáticamente la calidad de los resultados a diferentes instancias de un problema. [2]

En esta metaheurística se propone un enfoque híbrido, donde varios de los parámetros se determinan estáticamente por un proceso de "ensayo y error" o según los valores propuestos por publicaciones consultadas; en tanto que los parámetros de GLS son aprendidos "on-line" en base a la evolución de la búsqueda.

El seudocódigo de la Sección anterior, se modifica de la siguiente manera, para incluir el crossover de parámetros de GLS:

```
procedure METAHEURÍSTICA (S)
begin
    INICIALIZAR (S)
    GLS (S)
    EVALUAR (S)
    while NoConverge do
        begin
            SELECCIONAR (S)
            if (la población evolucionó en las últimas iteraciones)
                begin
                    CROSSOVER-CONFIGURACIONES (S)
                    MUTACIÓN-DE-UN-INDIVIDUO (S)
                end
```

```
            else
                begin
                    CROSSOVER-PARÁMETROS (S)
                end
            GLS (S)
            EVALUAR (S)
        end
    Devolver la mejor solución en S
end
```

6.3. Implementación de la metaheurística

En la Sección 6.2 se presentó el esquema de la metaheurística que constituye el objeto de este trabajo. A continuación, se explicarán cada una de las decisiones de implementación tomadas durante su construcción.

Estructura general de la metaheurística

En el contexto del Problema del Viajante de Comercio, los individuos que constituyen la población son los tours. La población no se divide en subpoblaciones; todos los individuos tienen igual jerarquía.

Se alternan etapas de optimización de la población, con etapas de crossover.

La metaheurística, luego de la inicialización (consistente en la creación de los individuos o tours que constituyen la primera generación del algoritmo), realiza una primera ejecución de GLS para cada uno de los individuos. La población pasa luego al crossover de configuraciones: se eligen parejas de "padres", para intercambiar sus genes (arcos) y dar origen a la próxima generación de individuos. Los pares de padres se forman siempre combinando el mejor individuo obtenido hasta el momento (el tour con el mínimo valor de la función objetivo), con otro individuo de la población, seleccionado al azar.

Así, la población aumenta su tamaño, quedando constituida por los padres más los hijos recientemente generados. A continuación, se elige al azar uno de los individuos de esta población formada por padres e hijos, y se lo somete a una mutación que altera algunos de sus genes. Con esto se busca introducir diversidad adicional en la población. Una salvedad es que no puede elegirse al mejor individuo para someterlo a mutación, pues se intenta preservar en todo momento a lo largo de la metaheurística las características del mejor tour.

El próximo paso es la etapa de optimización local del AM: se corre GLS sobre todos los individuos (padres y descendientes). Finalmente, se efectúan la evaluación, por la cual se ordenan los individuos en forma creciente de acuerdo a su valor asociado de la función objetivo; y la selección, consistente en determinar cuáles son los individuos que pasarán a la siguiente generación. En esta metaheurística se

seleccionan siempre los individuos con los mejores costos. Luego de este paso la población recupera su tamaño original.

Con la nueva generación así formada, se recomienza la alternancia entre etapas de recombinación (crossover de configuraciones) + mutación, y etapas de mejoramiento de los individuos (GLS).

Si durante una cantidad preestablecida de generaciones (tres) no se producen mejoras en el costo del mejor individuo de la población, se asume que la búsqueda ha quedado atrapada en un conjunto de configuraciones, formado por mínimos locales que no pueden ser mejorados por GLS. Para darle oportunidad de escapar de ellos, se introduce en este punto el crossover de parámetros de GLS.

Los pares de padres para el crossover de parámetros se seleccionan de manera similar al caso del crossover de configuraciones. Se combinan por pares, los parámetros del mejor individuo, con los de otros individuos elegidos al azar. En este caso, se considera mejor individuo a aquél cuyo costo haya variado más (comparativamente) desde el último crossover de parámetros realizado, lo cual indica que GLS sobre él produjo buenos resultados, y por lo tanto se espera que sus parámetros aporten características valiosas para gobernar el GLS sobre otros individuos de la población.

Al modificarse los parámetros de GLS, la búsqueda va a comportarse de manera diferente, por lo tanto puede esperarse la obtención de nuevos resultados. Así, la población recomienza su evolución (ciclos de crossover/mutación/optimización), en base a los parámetros recientemente determinados.

Dado que los AM constituyen una estrategia muy general, que brinda amplia libertad para variar los parámetros y técnicas utilizadas (ver Sección 5.4), es necesario explicitar con mayor detalle cada una de las decisiones de implementación tomadas durante la construcción de la metaheurística.

Muchas de las elecciones realizadas, se fundamentan en la experimentación o en la intuición; otras se basan en diferentes publicaciones consultadas. Reuniendo los diversos componentes que se enumeran a continuación, se construye la "instancia particular" de AM elegida para este trabajo.

Tamaño de población

Los parámetros relativos al tamaño de la población se tomaron de la referencia [14], que presenta un operador de crossover novedoso para el TSP, en el marco de un AM que lleva a cabo la etapa de optimización local de los individuos utilizando la heurística de mejoramiento iterativo de Lin-Kernighan ([25]).

La metaheurística trabaja sobre una población compuesta por 20 individuos, con una "tasa de crossover" de 0.5; es decir, por cada iteración se generan 10 hijos mediante crossover.

Estructuras de datos

La población se almacena mediante una estructura lineal (vector de individuos). Para cada individuo, además del tour propiamente dicho, debe conservarse otra información; esencialmente, los valores de los parámetros de GLS que evolucionarán (parámetro de regularización λ, matriz de

penalidades; ver Ecuación (4-5)), además de algunos datos de menor relevancia usados para dirigir la metaheurística.

Método de inicialización

Durante la etapa de programación se intentaron varias formas de inicialización de la población, tales como generar los individuos al azar, o mediante heurísticas constructivas (inicialización "greedy" o golosa). [20]

Finalmente se optó por la siguiente estrategia de inicialización:

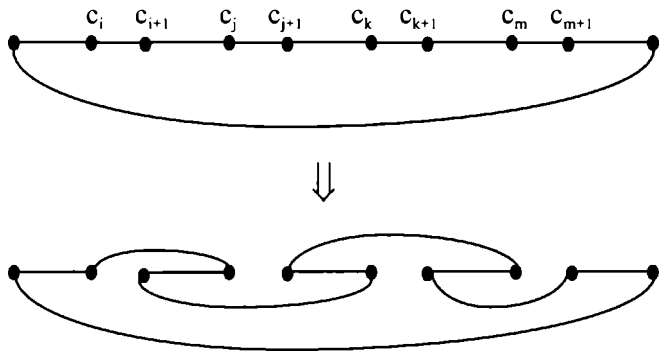
- Generar el individuo 1 como un tour aleatorio
- Mejorar el individuo 1 mediante una heurística simple de mejoramiento iterativo
- Generar los individuos i = 2, 3, ..., 20, por el siguiente procedimiento:
  - Ejecutar 2 mutaciones sobre el individuo i-1 para generar el individuo i
  - Mejorar el individuo i mediante una heurística simple de mejoramiento iterativo

Este método hace que la población inicial sea menos diversa que en el caso de N individuos creados aleatoriamente. Por lo tanto, resulta menor el período de transición hasta que la población comienza a evolucionar como un todo.

La heurística de mejoramiento iterativo que se aplica para formar cada uno de los individuos de la población inicial, es la misma que se utilizará como método de búsqueda local dentro del GLS (la variante del 2-Opt descripta en las Secciones 3.4 y 3.5).

La mutación usada durante la inicialización será llamada "cadena de 2-Changes". Consiste en remover cuatro arcos del tour: (c_i, c_{i+1}), (c_j, c_{j+1}), (c_k, c_{k+1}) y (c_m, c_{m+1}); e insertar en su lugar otros cuatro arcos: (c_i, c_j), (c_{i+1}, c_k), (c_{j+1}, c_m), (c_{k+1}, c_{m+1}). El primer arco a eliminar se elige al azar, y los otros tres se determinan a partir de él, de manera tal que al remover los cuatro arcos, el tour quede dividido en cuatro subcadenas de igual cantidad de ciudades (que luego se unirán al insertar los nuevos arcos).

La siguiente figura ilustra esta mutación:



El crossover

La estrategia diseñada para generar nuevos individuos a partir de dos existentes, se denomina

"Multiple Fragment - Nearest Neighbour Repair Edge Recombination", y fue usada por primera vez en este trabajo.

En resumen, este crossover busca preservar el material genético que es común a ambos padres; y de los genes en los cuales los padres se diferencian, transmitir al hijo aquéllos que le provean mejores características (es decir, los que produzcan un tour hijo de menor costo).

En el Problema del Viajante de Comercio, los individuos son los tours, y los "genes" que intercambiarán mediante el crossover son los arcos (por lo tanto, cada individuo está compuesto por N genes).

Se toman dos tours como padres, y se genera un tour como descendencia.

Dados los tours padres constituidos por dos conjuntos de arcos, a los que llamaremos A y B respectivamente, la construcción del tour hijo sigue varios pasos:

a) Se copian en primer lugar los arcos comunes a ambos padres, es decir, los arcos de $A \cap B$. En el caso de que ambos padres fueran iguales, esta operación genera un tour hijo igual a ambos, y el crossover finaliza. En el caso más común de que A y B sean diferentes, el tour hijo queda compuesto, hasta el momento, por una serie de subcadenas (sucesiones con menos de N ciudades).

b) A continuación se realiza la inserción de los arcos no comunes, que pertenecen sólo a uno de los padres: arcos de $(A - B) \cup (B - A)$.

Para esto, se forma una lista con dichos arcos, ordenados en forma creciente según su longitud. Se van tomando uno a uno los arcos de esta lista, y se los incluye en el tour hijo, siempre que la inserción no dé origen a bifurcaciones o subciclos.

Se produce una bifurcación, cuando se intenta insertar un arco para el cual una de las ciudades que lo forman, es interna a alguna de las subcadenas ya incluidas en el tour hijo. Se produce un subciclo, cuando se intenta insertar un arco para el cual las ciudades que lo forman, son puntos extremos ("endpoints") de alguna de las subcadenas ya incluidas en el tour hijo, y por lo tanto su inserción daría lugar a un subtour cerrado, con menos de N ciudades. En cualquiera de estos dos casos, se descarta el arco que produce el problema, y se pasa al siguiente arco de la lista de arcos no comunes.

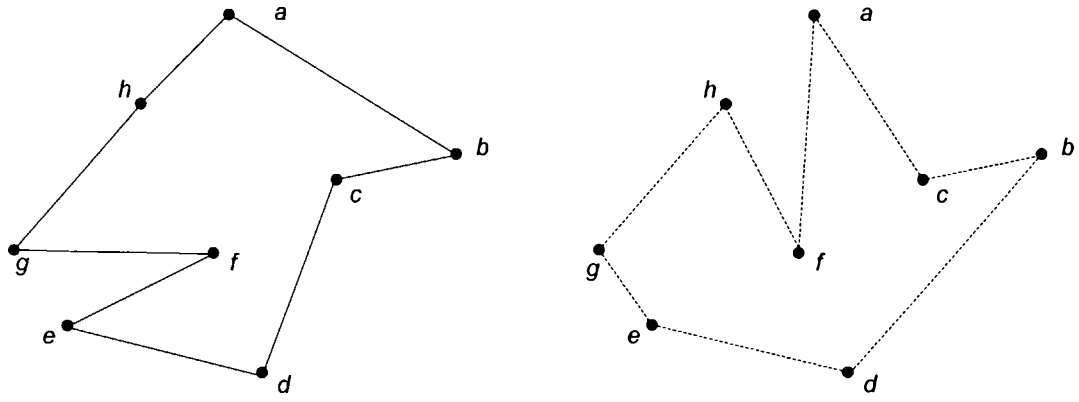
Al testear por la existencia de subciclos, debe permitirse el caso especial en que se forma un "subciclo" de N ciudades. Es decir, cuando el tour hijo es una subcadena compuesta de N-1 ciudades, y se inserta el arco que une los puntos extremos de esta cadena. En esta situación, el tour hijo queda completo y el crossover finaliza.

Si una vez recorrida toda la lista de arcos no comunes, no se pudo completar la construcción del tour hijo, se pasa al proceso de "patching" o unión de subcadenas. (Si hay ciudades que no participan de ninguna subcadena formada hasta el momento, serán consideradas para el "patching" como subcadenas compuestas por una sola ciudad).

c) El método de "patching" se inspira en la heurística constructiva de Nearest-Neighbour ([20]). El procedimiento es el siguiente: se toma una de las subcadenas del tour hijo, y partiendo desde uno de sus endpoints, se busca entre las demás subcadenas la que pueda unirse a la primera mediante el arco de menor longitud. Este arco se inserta en el tour hijo formando así una subcadena mayor, y se repite el procedimiento, partiendo ahora de un endpoint de la cadena recientemente formada.

Esta unión de subcadenas se repite, hasta obtener una única cadena (tour hijo) que incluya a las N ciudades; en este momento se unen los puntos extremos de esta cadena para completar el tour.[2]

Para ejemplificar este crossover, se consideran los siguientes tours padres A y B:



Los pasos a seguir para la formación del tour hijo son:

1. Insertar los arcos comunes a ambos padres, que se indicarán en el gráfico con línea más gruesa: (b, c), (d, e), (g, h)

2. Construir la lista de arcos no comunes, pertenecientes a uno solo de los padres, ordenada en forma creciente:

$\{ (e, g), (h, a), (e, f), (h, f), (f, g), (c, d), (a, c), (f, a), (a, b), (b, d) \}$

---

[2] Durante algunos experimentos que se realizaron previamente, en lugar de este crossover, se utilizó el crossover DPX, descripto en las referencias [13], [14] y [29]). Es interesante notar que el procedimiento detallado en estos trabajos no es completo. El DPX consiste en incorporar en el tour hijo los arcos comunes a ambos padres, y luego, unir los fragmentos utilizando arcos no pertenecientes a ninguno de los padres. Durante las pruebas sucedió varias veces que este procedimiento dio como resultado un conjunto de subcadenas en lugar de un tour completo, y para unir los puntos extremos de estas subcadenas se requerían arcos pertenecientes a uno solo de los padres ("prohibidos" de acuerdo al procedimiento del DPX). Los trabajos citados no explican cómo proceder en este caso. En las pruebas efectuadas, se dejaba de lado esta restricción y se permitía usar arcos pertenecientes a uno solo de los padres, cuando no había otra manera de unir los fragmentos del tour hijo.

Ejemplo: padres < 1, 2, 3, 4, 5 > y < 1, 2, 3, 5, 4>, el hijo hereda las subcadenas < 1, 2, 3 > y < 4, 5 >; y no tiene forma de unirlas sin utilizar arcos pertenecientes a alguno de los padres, ya sean (3, 4) y (5, 1), o bien (3, 5) y (4, 1).

3. Recorrer la lista insertando los arcos que no formen bifurcaciones ni subciclos (los arcos insertados se grafican en el tour hijo, con el estilo de líneas que identifica al padre del cual provienen):

- Se insertan (e, g) y (h, a)

- No se inserta (e, f) para evitar una bifurcación, pues e es interna a la subcadena formada hasta el momento: $< d, e, g, h, a>$

- No se insertan (h, f) ni (f, g) para evitar una bifurcación, pues h y g respectivamente, son internas a la misma subcadena

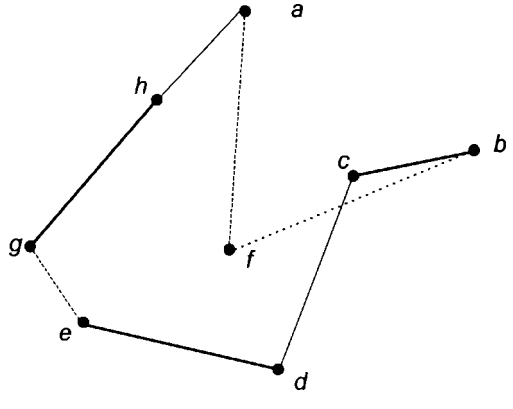
- Se inserta (c, d)

- No se inserta (a, c) para evitar una bifurcación, pues c es interna a la subcadena formada hasta el momento: $< b, c, d, e, g, h, a >$. Además, de insertar este arco, se cerraría un subciclo comprendiendo las ciudades: $< c, d, e, g, h, a >$.

- Se inserta (f, a)

No es necesario intentar la inserción de los arcos restantes de la lista, (a, b) y (b, d), pues al agregar (f, a) se obtiene una única cadena que incluye a las ocho ciudades: $< b, c, d, e, g, h, a, f >$. Por lo tanto se unen los puntos extremos de esta cadena mediante el arco (f, b), indicado con línea punteada, para completar el tour.

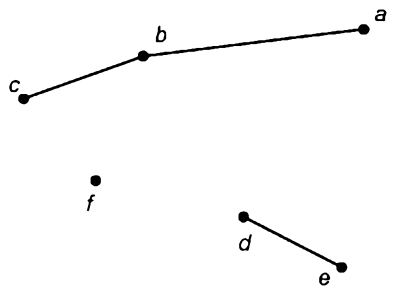
El tour hijo resultante es el siguiente:



Por tratarse de un ejemplo sencillo, no fue necesario utilizar el mecanismo de patching, pues se obtuvo directamente el tour hijo mediante la inserción de arcos pertenecientes a uno o ambos padres, más un arco adicional para cerrar la cadena resultante.

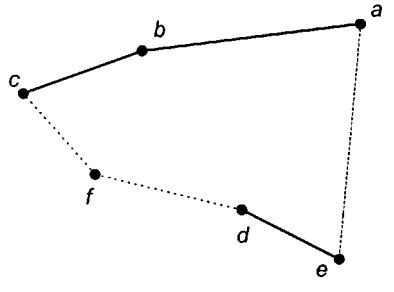
El siguiente es otro ejemplo, para analizar el método de patching propuesto:

Supongamos que luego de ejecutar el proceso de inserción de arcos de los padres, el tour hijo queda formado por tres subcadenas (identificadas en el gráfico por líneas gruesas): $< a, b, c >$, $< d, e >$, y $< f >$.



Se elige la primera de ellas como subcadena inicial para el patching, partiendo desde su endpoint *c*. Se evalúa la longitud de los arcos que podrían conectar a la ciudad *c* con un punto extremo de las restantes subcadenas, es decir: (*c*, *d*), (*c*, *e*), (*c*, *f*). El menor de estos arcos es (*c*, *f*); por lo tanto se lo inserta, y ahora el tour hijo queda compuesto por dos subcadenas: < *a*, *b*, *c*, *f* > y < *d*, *e* >.

Ahora se toma como punto de partida la ciudad final de la nueva subcadena, *f*, y se evalúan los arcos que pueden unirla a la otra subcadena: (*f*, *d*) y (*f*, *e*). El de menor longitud es (*f*, *d*), por lo tanto se lo incorpora al tour hijo. Éste consiste ahora de sólo una subcadena, < *a*, *b*, *c*, *f*, *d*, *e* >, que incluye a las seis ciudades; en este momento se unen los puntos extremos de esta cadena agregando el arco < *e*, *a* > , graficado con línea cortada, para completar el tour.



Hasta el momento se explicó el crossover elegido, que provee un método para intercambiar arcos ("genes") de dos tours y generar así un tour hijo entre ambos. Como se explicó anteriormente, con cada individuo de la población debe conservarse información acerca del parámetro de regularización (λ) y la matriz de penalidades. Por lo tanto, al generar un nuevo individuo como descendencia entre dos tours padres, también es necesario asignarle algún valor para estos parámetros. Los padres para el crossover son siempre el mejor individuo encontrado hasta el momento, y otro tour elegido al azar. Por lo tanto, al generar el tour hijo entre ellos, éste recibirá una copia del parámetro de regularización y la matriz de penalidades del padre elegido al azar (ya que si recibiera los del mejor padre, como éste interviene en todos los crossovers, todos los hijos generados tendrían el mismo valor para estos parámetros y se perdería variedad en la población).

Control de la diversidad de población

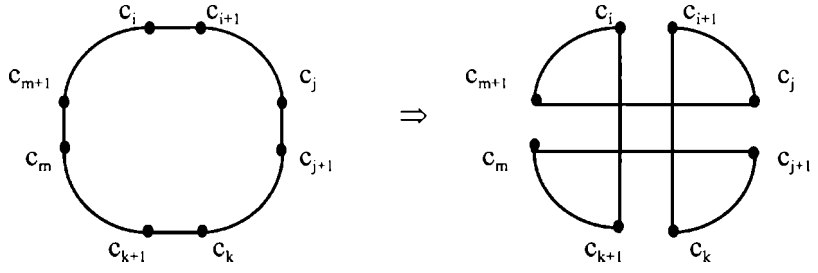
Se utilizan mutaciones en dos lugares diferentes de la metaheurística, para introducir variaciones en los individuos de la población, y así evitar el problema de la convergencia prematura, descripto en

la Sección 5.4.

Luego de cada etapa de crossover, se selecciona un individuo al azar, entre los padres y los descendientes (no se permite seleccionar al mejor individuo obtenido hasta el momento, para preservarlo de cambios). Este individuo se somete a una mutación conocida como "movida del doble puente", explicada en la referencia [13].

Esta operación consiste en remover cuatro arcos del tour: (c_i, c_{i+1}), (c_j, c_{j+1}), (c_k, c_{k+1}) y (c_m, c_{m+1}); e insertar en su lugar otros cuatro arcos: (c_i, c_{k+1}), (c_j, c_{m+1}), (c_k, c_{i+1}), (c_m, c_{j+1}). El primer arco a eliminar se elige al azar, y los otros tres se determinan a partir de él, de manera tal que al remover los cuatro arcos, el tour quede dividido en cuatro subcadenas de igual cantidad de ciudades (que luego se unirán al insertar los nuevos arcos).

Esta mutación puede graficarse de la siguiente manera:



Además, al salir de GLS (es decir, luego de cada etapa de optimización), si en la población hay dos o más tours con igual costo, se asume que se trata del mismo individuo, dado que la metaheurística se inicia con tours muy similares entre sí, y es factible que la búsqueda local produzca para varios de ellos el mismo resultado. De esta manera, se pierde diversidad en la población. Para evitar esta situación, se conserva sólo uno de los tours cuyos costos coinciden, y se introducen mutaciones en los restantes, para diferenciarlos del primero. En este caso se utiliza la misma mutación que se usó durante el paso de inicialización ("cadena de 2-Changes"), de esta manera:

- Detectar un conjunto de tours con el mismo costo
- Conservar uno de los tours y eliminar los restantes
- Construir los tours faltantes hasta obtener el tamaño original de la población, por el siguiente método:
    - Introducir dos mutaciones sobre el individuo preservado, para generar el siguiente individuo
    - Introducir dos mutaciones sobre el individuo recientemente generado, para formar el tercero
    - Y así sucesivamente, hasta completar la población

De la misma manera se procede para cada conjunto de tours de la población, para los cuales los costos sean iguales.

Búsqueda Local Guiada

Hay varios aspectos que deben decidirse para la implementación de GLS (además de elegir la heurística que funcionará como proceso de búsqueda local; en este caso, como ya se dijo, será la que se presentó en las Secciones 3.4 y 3.5, basada en el método del 2-Opt).

En primer lugar debe determinarse el valor del parámetro de regularización λ, asociado a cada uno de los tours que componen la población. Para el cálculo se utilizó una fórmula basada en la Ecuación (4-14), donde el factor *g(mínimo local)* se determina en base al costo obtenido para cada individuo luego del procedimiento de inicialización descripto en esta misma Sección. La constante *a* incluida en la fórmula se estableció en el valor 0.3, siguiendo los resultados determinados experimentalmente en [48]. Se multiplica además al resultado de esta fórmula, por un factor de "perturbación" (una pequeña constante que intenta introducir más variedad en los parámetros de regularización de los distintos individuos, para que resulte significativo intentar un crossover entre los valores de estos parámetros; si fueran todos muy similares entre sí no tendría sentido intentar que la población "aprendiera" un buen valor para ellos). Así, el λ para cada integrante de la población inicial se determina como:

$$\lambda = 0.3 \cdot \frac{g(\text{tour inicial})}{N} \cdot u, \qquad (6\text{-}1)$$

donde g(tour inicial) es el costo resultante del procedimiento de inicialización para cada tour de la población, y u es el factor de perturbación.

Durante el desarrollo de la metaheurística, en la etapa del crossover de parámetros de GLS, los integrantes de la población competirán entre sí, para determinar cuál de los valores del parámetro de regularización λ resulta más adecuado para guiar el comportamiento del proceso de búsqueda sobre la población (es decir, cuál de los valores de este parámetro refleja información más relevante reunida durante el transcurso de la búsqueda).

Otro parámetro muy importante que influye en el funcionamiento de GLS, es la cantidad de llamadas a la heurística simple de búsqueda local que se realizan dentro del proceso. Deben efectuarse suficientes llamadas, como para permitir que GLS "aprenda" a través de sucesivas modificaciones de la función de costo (incorporando términos de penalización), y logre concentrar sus esfuerzos en las regiones más promisorias del espacio de búsqueda.

En este trabajo, no se realizó un análisis detallado para establecer rangos "adecuados" de este parámetro. Se determinaron valores convenientes para cada instancia del TSP, en forma experimental, y luego estos valores se ingresaron al algoritmo a través de un archivo de entrada. En todos los casos, dichos valores fueron muy inferiores a los recomendables para un GLS "puro", pues la metaheurística desarrollada requiere ejecutar 30 corridas de GLS para cada generación del AM (una por cada uno de los 20 pobladores, y una por cada uno de sus 10 descendientes). Sumado esto al resto del tiempo de procesamiento requerido (para la inicialización, crossover de configuraciones, mutación, crossover de

parámetros), los tiempos de ejecución resultan muy difícilmente manejables para los recursos computacionales disponibles.

Crossover de parámetros

El crossover de configuraciones consiste en intercambiar información genética entre tours de la población, dando origen a nuevos individuos que compiten con sus padres por la supervivencia.

La situación es diferente durante el crossover de parámetros de GLS. En este caso, no se generan nuevos individuos, sino nuevos valores de parámetros para los individuos existentes. Así, en lugar de aumentar el tamaño de la población generando nuevos individuos, se mantendrá el tamaño de población original (20 individuos), pero los que sean sometidos al crossover, obtendrán nuevos valores para sus parámetros relevantes: el parámetro de regularización λ, y los elementos de la matriz de penalidades (ver Ecuación (4-5)).

Los pares de padres para el crossover de parámetros se seleccionan de manera similar al caso del crossover de configuraciones, combinando siempre el mejor individuo, con otros elegidos al azar. Como lo importante es tratar de buscar parámetros que permitan la evolución del proceso de búsqueda sobre los individuos, para este caso el mejor individuo no será el de mejor costo, sino aquél cuyo costo haya variado más comparativamente desde el último crossover de parámetros realizado. Esto indicará que los valores de los parámetros asociados con este individuo, le permitieron progresar más que al resto de la población, y por esto se los considera buenos parámetros, elegibles para basar en ellos el crossover.

Como ya se dijo, no se generan nuevos individuos. Se aparea en cada caso el mejor individuo (según el criterio antes descripto), con otro seleccionado al azar; se calculan los valores resultantes de los parámetros de GLS, y se los asigna al "padre" elegido al azar, es decir, al que supuestamente tenía un valor menos adecuado para estos parámetros.

El tour asociado al individuo cuyos parámetros varían con el crossover, permanece inalterado.

La cantidad de crossovers que se efectúan es la misma que para el crossover de configuraciones (es decir, se intercambian los valores de los parámetros del mejor individuo, con los de otros diez elegidos al azar).

El crossover del parámetro de regularización λ, que determina la importancia relativa que tienen los términos de penalización de la función de costo modificada con respecto a la función de costo original, se realiza de la siguiente manera:

Llamaremos λ_1 y λ_2 a los dos valores del parámetro que usaremos para el crossover, y asumiremos sin pérdida de generalidad que $\lambda_1 \leq \lambda_2$ (en caso contrario, se intercambian estos valores).

Entonces, sea dist = $\lambda_2 - \lambda_1$, la diferencia entre los valores; se genera como resultado del crossover un nuevo λ con distribución aleatoria uniforme en el intervalo $[\lambda_1 - \text{dist}/3, \lambda_2 + \text{dist}/3]$, y se asigna este valor del parámetro al padre elegido al azar. La constante 1/3 que amplía el intervalo en el cual se

determina el valor del parámetro, cumple la función de introducir cierta variación en los posibles valores de λ (su rol es análogo al de las mutaciones durante la evolución de los individuos).

Otro parámetro (o más precisamente, conjunto de parámetros) asociado a cada individuo, es la matriz de penalidades, usada para conservar la información aprendida por GLS durante el transcurso de la búsqueda, y acumulada en la forma de términos de penalización que modifican la función de costo original.

Dados dos individuos elegidos como padres, cada uno con su correspondiente matriz de penalidades, el crossover consiste en formar una nueva matriz, donde cada uno de sus elementos sea el máximo entre los elementos correspondientes de las matrices a intercambiar. Esta matriz se asigna al padre elegido al azar.

Si llamamos *MejorP* al mejor individuo elegido como padre para el crossover, y *OtroP* al restante individuo que interviene, esta operación puede expresarse como:

$$\text{Matriz}(OtroP)_{i,j} = \max\ (\text{Matriz}(MejorP)_{i,j},\ \text{Matriz}(OtroP)_{i,j}), \qquad i, j = 1, .. N$$

Mejoras de performance

Si bien no se prestó especial atención a los aspectos de eficiencia del algoritmo, se implementaron algunos "trucos" para evitar operaciones innecesarias a lo largo de toda la metaheurística.

En la sección 3.5 se presenta la utilización de bits de "no-mirar", como una estrategia para reducir sustancialmente el tiempo de corrida de la heurística 2-Opt, concentrando la atención en los nodos que tienen mayor probabilidad de producir buenos intercambios.

Esta idea puede extenderse a toda la metaheurística. A cada individuo de la población se le asocian tantos bits de "no-mirar", como ciudades compongan el tour correspondiente. Estos bits se conservan y actualizan durante el algoritmo, en forma análoga a la detallada para la heurística simple de búsqueda local: se encienden los bits para las ciudades desde las cuales no es posible ningún intercambio de arcos que reduzca el costo del tour, y se apagan para las ciudades que son extremos de algún arco introducido recientemente en el tour. En cada paso del AM se ignoran las ciudades cuyos bits están encendidos, reduciendo así el número de posibilidades a analizar.

Complementariamente, se mantiene durante toda la metaheurística un indicador asociado a cada tour, que sirve para determinar si ese individuo será sometido o no a GLS en cada iteración.

Inicialmente ese indicador está activo para todos los individuos de la población, ya que cada uno de ellos tiene posibilidades de mejorar en el transcurso de GLS. Si el costo de un individuo no disminuye al cabo de una corrida de GLS, su valor de indicador se invierte, para indicar que no debe intentarse volver a mejorar ese tour (ya que nada cambió desde la iteración anterior, y por lo tanto no tiene sentido someterlo nuevamente al mismo proceso de búsqueda).

A los individuos que se incorporan a la población, generados mediante el crossover de parámetros, se les asigna un indicador activo, lo cual permite que sean optimizados por GLS antes de competir con el resto de la población por la supervivencia. También se activa el indicador de los individuos cuyos parámetros de GLS son modificados por el crossover de parámetros, pues aunque no hayan mejorado durante la última corrida de GLS, los parámetros de la Búsqueda Local Guiada variaron, y esto implica la posibilidad de obtener resultados diferentes en la próxima oportunidad.

7. LOS EXPERIMENTOS

7.1. Instancias usadas

Disponer de un conjunto de instancias de testeo comunes para el TSP, es de suma importancia para comparar resultados. Gerhard Reinelt, del "Institut für Angewandte Mathematik", Universidad de Heidelberg, compila y mantiene una base de datos de dominio público llamada TSPLIB, que reúne muchas grandes instancias del TSP resueltas hasta la optimalidad.

TSPLIB es una librería de instancias de muestra para el TSP, y problemas relacionados de diversos tipos, ampliamente accesible a través de Internet. [42], [43]

Cada archivo de TSPLIB representa una instancia del TSP. Consiste en un encabezado que especifica el formato del archivo y características generales de la instancia (tales como cantidad de ciudades y tipo de distancia utilizada), y un cuerpo que contiene los datos de las ciudades (coordenadas de cada una, y matriz de distancias si es requerida).

La página Web del TSPLIB provee amplia documentación, que incluye la longitud óptima del tour para cada instancia, en caso de que sea conocida, o las cotas inferior y superior para dicha longitud, en caso contrario.

Esta base de datos se ha difundido ampliamente en el ambiente académico en todo el mundo, y es usual que sus instancias se utilicen como ejemplos en la mayoría de los trabajos de investigación sobre el TSP.

Un inconveniente con respecto a las instancias de TSPLIB, es su tamaño limitado; en especial, si se tiene en cuenta el avance actual en cuanto al desarrollo de algoritmos y la fabricación de supercomputadoras. Hay una clara necesidad de desarrollar un conjunto suplementario de instancias de testeo del TSP de mayores dimensiones, con soluciones óptimas conocidas. La principal motivación es el hecho de que algunas heurísticas para el problema son de baja complejidad, de modo que se requieren enormes casos de testeo.

Por ejemplo, Johnson [20] reportó estudios computacionales usando heurísticas O(N logN) para instancias del TSP de más de un millón de ciudades. Para testear las heurísticas con instancias de ese tamaño, Johnson se vio restringido a confiar en fórmulas asintóticas sobre la longitud esperada para el tour óptimo [21]; estas fórmulas sólo valen bajo la suposición de que las coordenadas de las ciudades se generan con una distribución probabilística dada, para la cual puede derivarse un resultado analítico.

En las referencias [26], [27] y [38], se presenta un método para construir algunas instancias del TSP euclídeo arbitrariamente grandes, y con tours óptimos conocidos. Las coordenadas de las ciudades se definen por procesos geométricos iterativos de construcción. A cada instancia se asocia un único tour de longitud mínima entre las ciudades, cuya optimalidad se demuestra por inducción basada

en la naturaleza del proceso constructivo. De esta forma se obtiene un conjunto de instancias con una cantidad arbitraria de ciudades, y con tours óptimos conocidos, lo que resulta de notable importancia para el testeo de heurísticas.

Estos ejemplos son conocidos como "instancias fractales" del TSP, y para su codificación se utilizan los L-Systems, que son sistemas de reglas para construir recursivamente curvas con propiedades de auto-similaridad. También pueden construirse del mismo modo otras instancias fractales, para las cuales no se ha demostrado que su tour asociado sea óptimo, pero que de todos modos resultan útiles como casos de testeo para el TSP.

La página de Internet [30], contiene referencias a publicaciones sobre la construcción de instancias fractales del TSP y temas relacionados, y acceso a un software que genera instancias en formato TSPLIB a partir de L-Systems.

En este trabajo se utilizan casos de prueba provenientes de ambas fuentes (TSPLIB e instancias fractales). Para las instancias fractales usadas, no se presenta la longitud del tour óptimo, ya que para algunas de ellas se desconoce, y para otras la optimalidad del tour asociado vale sólo en el caso de utilizar distancias reales. En cambio, este trabajo usa distancias enteras, calculadas de acuerdo a la documentación de TSPLIB.

7.2. Condiciones de los experimentos

Los algoritmos utilizados en este trabajo, fueron programados en su totalidad. No se intentó reutilizar código de otros autores, ya que una de las metas principales del trabajo consistía en comprender un tema nunca abordado anteriormente, y el ensayo de diferentes enfoques fue fundamental para este objetivo.

La metaheurística se construyó por etapas, incorporando en cada paso un nuevo aspecto; todos ellos fueron comentados en Secciones anteriores:

- Heurística simple de búsqueda local

  Se construyó la heurística detallada en la Sección 3.5, basada en la clásica heurística 2-Opt de mejoramiento iterativo.

- AM, con etapa de optimización de los individuos basada en la heurística simple

  Se desarrolló la base de la metaheurística: se definió la estructura de la población, la forma de inicialización, el crossover, y el método de control de la diversidad. Sin embargo, en esta etapa no se incorporó aún GLS, sino que durante la fase de optimización, se ejecutó la heurística simple mencionada en el punto anterior sobre cada individuo.

- AM, incorporando GLS como método de optimización

  En este paso, se agregó GLS a la metaheurística, para llevar a cabo la optimización de la población. En cada generación, los individuos fueron sometidos a la aplicación de GLS, que a su vez utilizó

como proceso de búsqueda local, la heurística construida en la primera etapa.

- AM con GLS, incluyendo crossover de parámetros

  Se completó la metaheurística, incluyendo el crossover de parámetros de GLS, como una nueva posibilidad de evolución para la población. Cuando ésta no mejora a lo largo de varias iteraciones, se intercambian por el crossover detallado en la Sección 6.2, los valores de los parámetros que gobiernan GLS para cada individuo.

Además, para propósitos de comparación, también se corrió GLS por sí solo (sin combinarlo con AM), sobre cada una de las instancias.

Se realizaron experimentos luego de cada etapa, para determinar la influencia de los distintos componentes de la metaheurística sobre los resultados.

Por cada instancia del conjunto de testeo, se ejecutaron inicialmente 100 corridas de la heurística de búsqueda local detallada en la Sección 3 (técnica multistart). El resultado que se reporta es el mejor obtenido entre los 100 intentos. A continuación, para las sucesivas pruebas a realizar con GLS y AM, se utilizó como criterio de fin una cota dada por la cantidad total de 2-Changes (movidas básicas de exploración del vecindario; ver Sección 3.4) sumada durante las 100 corridas de la heurística local. Es decir, si la ejecución multistart de la heurística simple para una instancia dada requirió W 2-Changes, las iteraciones de GLS y AM sobre esa instancia se detuvieron al alcanzar el mismo total de W 2-Changes.

Luego se presenta un segundo conjunto de resultados, obtenido repitiendo los experimentos descriptos, pero usando esta vez como criterio de fin la cota dada por la suma de 2-Changes durante 300 corridas de la heurística de búsqueda local.

Si bien en la Sección 6.3 se determinó un tamaño de población de 20 individuos para el AM, en las cuatro instancias con más cantidad de ciudades se usaron poblaciones más reducidas, debido a limitaciones de memoria: p654 (18 individuos), spnsk5 (16 individuos), koch1-4 y rat783 (14 individuos). En todos los casos se mantuvo la "tasa de crossover" en 0.5, lo cual significa que en cada generación surge por crossover, una cantidad de hijos igual a la mitad del tamaño de la población.

7.3. Resultados (comparación: 100 iteraciones de la heurística simple)

Los resultados del trabajo se muestran en dos tablas. La primera refleja los costos obtenidos para cada instancia de TSPLIB durante las diferentes etapas de implementación de la metaheurística, y los porcentajes de exceso de estos costos sobre los respectivos óptimos globales. Se incluye también un gráfico de estos porcentajes de exceso.

La segunda tabla presenta los resultados para las instancias fractales. Dado que para ellas no se conoce el óptimo global, como se mencionó en la Sección 7.1, en estos casos no pudieron calcularse los porcentajes de exceso, y solamente se muestran los costos para cada instancia.

Instancia	Cant. de ciudades	Óptimo global	Heurística simple		AM		AM + GLS		AM + GLS + Cross.Par.		GLS	
			Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.
att48	48	10628	10638	0,09	10638	0,09	10628	0,00	10628	0,00	10628	0,00
st70	70	675	687	1,78	683	1,19	676	0,15	676	0,15	678	0,44
eil76	76	538	557	3,53	538	0,00	538	0,00	538	0,00	539	0,19
rd100	100	7910	8184	3,46	8055	1,83	7910	0,00	7910	0,00	7910	0,00
kroe100	100	22068	22572	2,28	22130	0,28	22068	0,00	22068	0,00	22068	0,00
krod100	100	21294	22094	3,76	21536	1,14	21294	0,00	21294	0,00	21294	0,00
kroc100	100	20749	21473	3,49	20749	0,00	20749	0,00	20749	0,00	20749	0,00
krob100	100	22141	22468	1,48	22258	0,53	22141	0,00	22141	0,00	22141	0,00
kroa100	100	21282	21812	2,49	21305	0,11	21282	0,00	21282	0,00	21282	0,00
eil101	101	629	650	3,34	636	1,11	629	0,00	629	0,00	629	0,00
bier127	127	118282	120434	1,82	118423	0,12	118616	0,28	118616	0,28	118313	0,03
pr136	136	96772	100684	4,04	97516	0,77	96785	0,01	96785	0,01	96781	0,01
pr144	144	58537	59281	1,27	58570	0,06	58588	0,09	58588	0,09	58537	0,00
kroa150	150	26524	27372	3,20	26765	0,91	26524	0,00	26524	0,00	26524	0,00
u159	159	42080	44078	4,75	42080	0,00	42080	0,00	42080	0,00	42080	0,00
d198	198	15780	16390	3,87	15872	0,58	15818	0,24	15818	0,24	15833	0,34
krob200	200	29437	30774	4,54	30061	2,12	29465	0,10	29465	0,10	29437	0,00
lin318	318	42029	44825	6,65	42850	1,95	42150	0,29	42150	0,29	42157	0,30
pcb442	442	50778	53993	6,33	51262	0,95	50937	0,31	50937	0,31	50834	0,11
att532	532	27686	29638	7,05	28493	2,91	27896	0,76	27896	0,76	27790	0,38
u574	574	36905	39916	8,16	37959	2,86	37566	1,79	37460	1,50	36978	0,20
p654	654	34643	40421	16,68	36433	5,17	35647	2,90	35695	3,04	35903	3,64
rat783	783	8806	9532	8,24	8992	2,11	8868	0,70	8868	0,70	8822	0,18
Promedio				**4,45**		**1,16**		**0,33**		**0,32**		**0,25**
Desv. estándar				**3,44**		**1,27**		**0,69**		**0,69**		**0,75**

Tabla: Resultados de los experimentos

Instancias de TSPLIB - Comparación con 100 iteraciones de la heurística simple

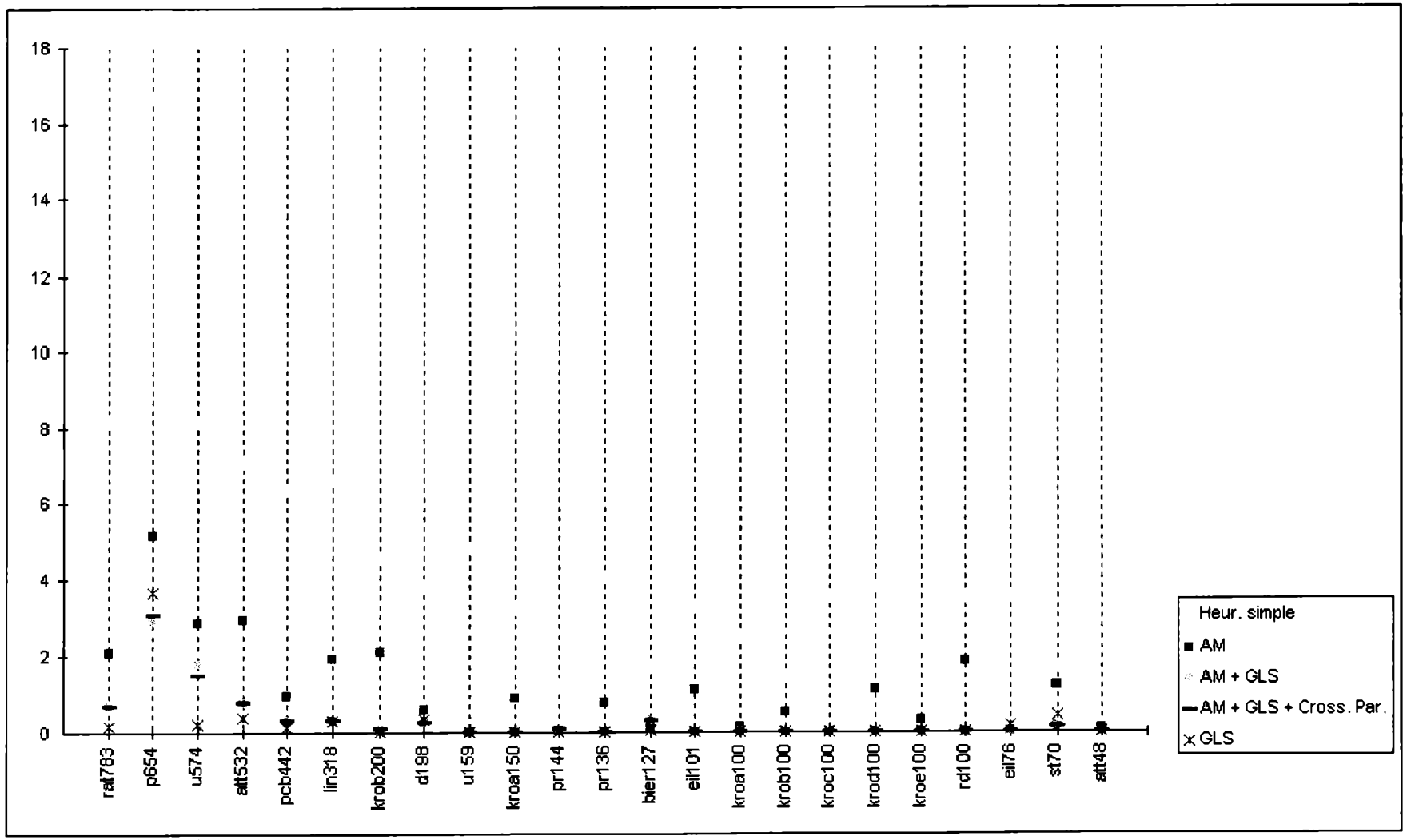


Gráfico: Resultados de los experimentos

Instancias de TSPLIB - Comparación con 100 iteraciones de la heurística simple

Instancia	Cant. de ciudades	Costo obtenido				
		Heurística simple	AM	AM + GLS	AM + GLS + Cross.Par.	GLS
snowfl-2	294	294	279	276	276	278
flowsnk2	294	258	255	254	254	254
fassp1-2	320	320	320	320	320	320
mnpeano3	364	459	428	441	441	460
daveven4	486	474	449	443	443	452
qtour2-3	500	516	501	500	500	503
spnsk5	729	808	715	715	715	722
koch1-4	768	939	715	715	715	715

Tabla: Resultados de los experimentos

Instancias fractales - Comparación con 100 iteraciones de la heurística simple

7.4. Resultados (comparación: 300 iteraciones de la heurística simple)

Se muestran los resultados de la misma manera que en la Sección anterior, pero utilizando un número mayor de iteraciones de la heurística simple, como criterio de fin para la terminación de las distintas etapas del experimento.

Puede verse que para las instancias fractales, los costos obtenidos son los mismos a los que se llegó en la Sección 7.3 (no se observaron mejoras en ninguno de los casos).

Además, para las instancias de TSPLIB se agrega un gráfico que muestra la cantidad de 2-Changes utilizados por la metaheurística (AM con GLS, incluyendo crossover de parámetros), para alcanzar su mejor resultado. Este número de 2-Changes realizados, puede verse como una función de la cantidad de ciudades de cada instancia.

Instancia	Cant. de ciudades	Óptimo global	Heurística simple		AM		AM + GLS		AM + GLS + Cross.Par.		GLS	
			Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.	Costo obtenido	% exceso sobre ópt.
att48	48	10628	10638	0,09	10628	0,00	10628	0,00	10628	0,00	10628	0,00
st70	70	675	687	1,78	681	0,89	675	0,00	675	0,00	675	0,00
eil76	76	538	555	3,16	538	0,00	538	0,00	538	0,00	539	0,19
rd100	100	7910	8131	2,79	8045	1,71	7910	0,00	7910	0,00	7910	0,00
kroe100	100	22068	22572	2,28	22073	0,02	22068	0,00	22068	0,00	22068	0,00
krod100	100	21294	21923	2,95	21536	1,14	21294	0,00	21294	0,00	21294	0,00
kroc100	100	20749	21018	1,30	20749	0,00	20749	0,00	20749	0,00	20749	0,00
krob100	100	22141	22323	0,82	22220	0,36	22141	0,00	22141	0,00	22141	0,00
kroa100	100	21282	21387	0,49	21282	0,00	21282	0,00	21282	0,00	21282	0,00
eil101	101	629	648	3,02	634	0,79	629	0,00	629	0,00	629	0,00
bier127	127	118282	120434	1,82	118361	0,07	118282	0,00	118282	0,00	118313	0,03
pr136	136	96772	99160	2,47	97009	0,24	96772	0,00	96772	0,00	96781	0,01
pr144	144	58537	59281	1,27	58537	0,00	58537	0,00	58537	0,00	58537	0,00
kroa150	150	26524	27361	3,16	26528	0,02	26524	0,00	26524	0,00	26524	0,00
u159	159	42080	43018	2,23	42080	0,00	42080	0,00	42080	0,00	42080	0,00
d198	198	15780	16368	3,73	15867	0,55	15808	0,18	15803	0,15	15816	0,23
krob200	200	29437	30774	4,54	29956	1,76	29437	0,00	29437	0,00	29437	0,00
lin318	318	42029	44508	5,90	42558	1,26	42029	0,00	42029	0,00	42062	0,08
pcb442	442	50778	53993	6,33	51242	0,91	50926	0,29	50926	0,29	50808	0,06
att532	532	27686	29566	6,79	28314	2,27	27744	0,21	27744	0,21	27716	0,11
u574	574	36905	39691	7,55	37112	0,56	37153	0,67	37147	0,66	36978	0,20
p654	654	34643	38522	11,20	35662	2,94	34674	0,09	34724	0,23	34879	0,68
rat783	783	8806	9468	7,52	8920	1,29	8834	0,32	8831	0,28	8809	0,03
Promedio				**3,62**		**0,73**		**0,08**		**0,08**		**0,07**
Desv. estándar				**2,75**		**0,83**		**0,16**		**0,16**		**0,15**

Tabla: Resultados de los experimentos

Instancias de TSPLIB - Comparación con 300 iteraciones de la heurística simple

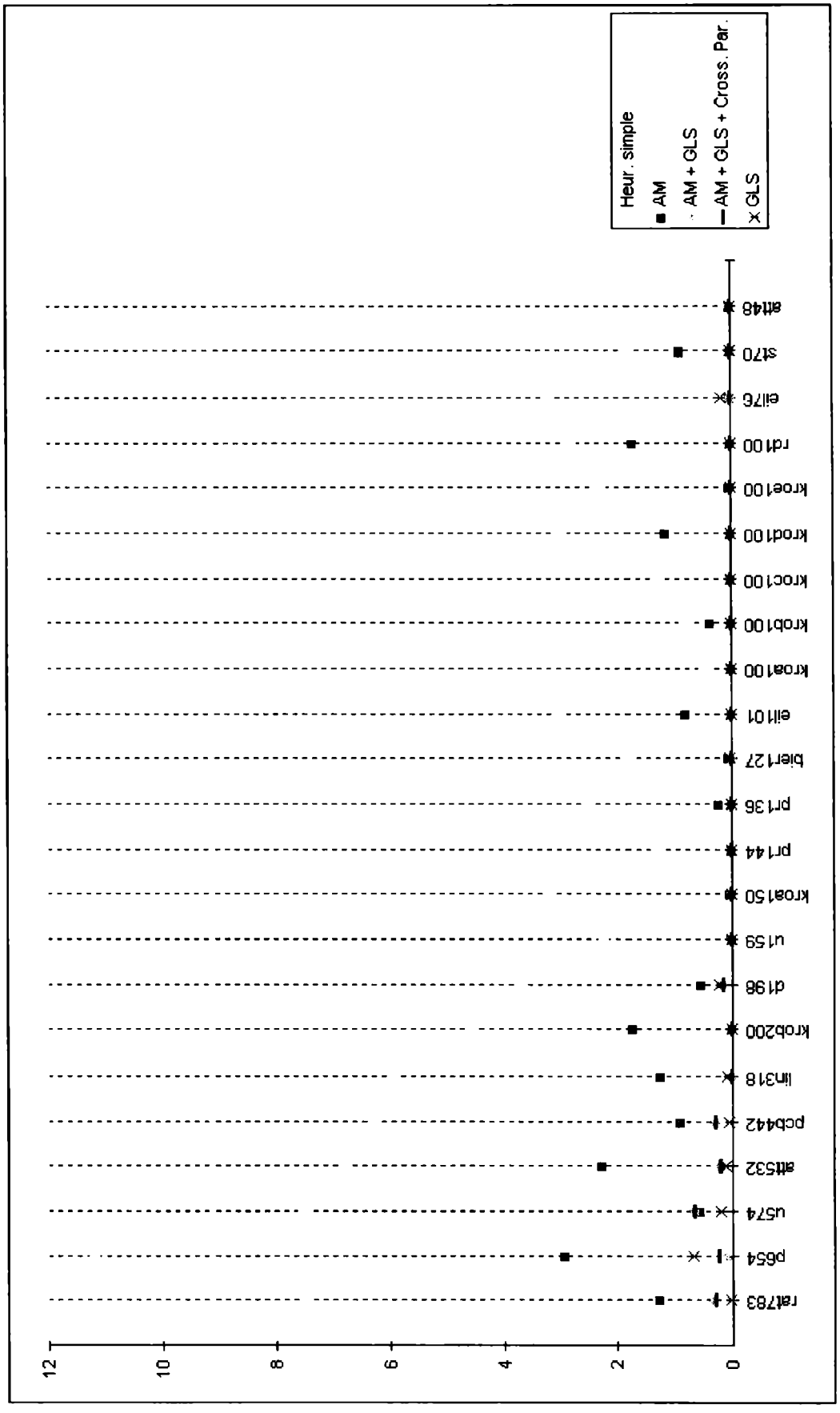


Gráfico: Resultados de los experimentos

Instancias de TSPLIB - Comparación con 300 iteraciones de la heurística simple

Instancia	Cant. de ciudades	Costo obtenido				
		Heurística simple	AM	AM + GLS	AM + GLS + Cross.Par.	GLS
snowfl-2	294	294	279	276	276	278
flowsnk2	294	258	255	254	254	254
fasspl-2	320	320	320	320	320	320
mnpeano3	364	459	428	441	441	460
daveven4	486	474	449	443	443	452
qtour2-3	500	516	501	500	500	503
spnsk5	729	808	715	715	715	722
kochl-4	768	939	715	715	715	715

Tabla: Resultados de los experimentos

Instancias fractales - Comparación con 300 iteraciones de la heurística simple

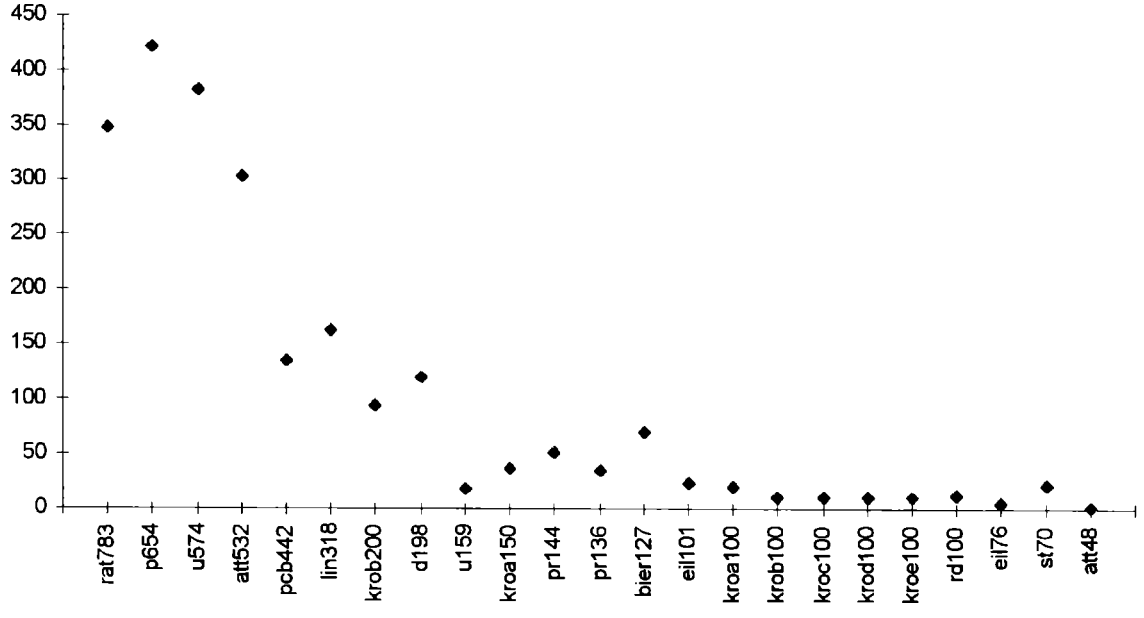


Gráfico: Cantidad de 2-Changes hasta el mejor resultado (en miles)

Metaheurística: AM + GLS + Crossover de parámetros

Instancias de TSPLIB - Comparación con 300 iteraciones de la heurística simple

8. CONCLUSIONES Y PROPUESTAS

8.1. Conclusiones

Para las pruebas realizadas y dentro del rango de instancias elegidas, se puede observar que (como era de esperarse) en todas sus variantes el AM se comporta mejor que la heurística simple de búsqueda local, produciendo resultados muy cercanos al óptimo global en todos los casos. El uso de GLS como método de optimización, durante la etapa de "aprendizaje" de la población del AM, origina también una mejora en el comportamiento, y aproxima notablemente los resultados al óptimo global. En cambio, la incorporación del crossover de parámetros de GLS, no parece realizar aportes al funcionamiento de la metaheurística.

También pudo observarse que ejecutando el AM sin GLS, los costos continuaban descendiendo hasta último momento (es decir, al llegar al límite de 2-Changes determinado como cota, el algoritmo aún tenía posibilidades de seguir progresando). Por el contrario, al incorporar GLS, los mejores resultados de la metaheurística se obtenían en las primeras iteraciones del AM, y a partir de allí no se conseguía disminuirlos. El agregado de GLS colaboró con el AM produciendo rápidamente muy buenos tours para la población, y dejando poco margen para que la evolución introdujera mejoras posteriores. Los logros de la combinación AM-GLS son más remarcables, si se tiene en cuenta que la búsqueda local se basa simplemente en la heurística del 2-Opt, con algunas modificaciones que contribuyen a la performance del algoritmo pero empeoran ligeramente sus resultados.

El AM es una técnica que requiere un cierto número de iteraciones iniciales, para que la población se estabilice y la evolución comience a influir en su composición. Como en la naturaleza y tal como se indica en [19], los procesos evolutivos sólo se perciben a lo largo de varias generaciones. En cambio, GLS utiliza una única corrida para producir sus resultados. Dado que estos resultados fueron desde el comienzo muy buenos, la evolución prácticamente no tuvo oportunidad de introducir cambios en la población, pues resultó difícil generar nuevos individuos cuyas características superaran a las de aquéllos ya existentes.

En este trabajo, el último agregado a la metaheurística (crossover de parámetros de GLS) no parece influir en los resultados. Como se indica en [47] y [48], GLS es una técnica suficientemente robusta, como para que pequeñas variaciones en los parámetros no alcancen a desviar el comportamiento general de la búsqueda. Dado que los resultados de GLS utilizando parámetros fijos resultaron muy cercanos al óptimo global para las diferentes instancias, la evolución de estos parámetros no resultó suficiente como para modificar el funcionamiento del algoritmo.

Los experimentos realizados no bastan para concluir que el crossover de parámetros de GLS no es de utilidad. Podría probarse con diferentes instancias del TSP, principalmente de mayores dimensiones, e introducir cambios en el conjunto de parámetros a evolucionar, y la naturaleza de los

operadores de crossover para estos parámetros. Todas estas variantes no fueron experimentadas por limitaciones de tiempo y recursos computacionales.

En las últimas columnas de las tablas de resultados de las Secciones 7.3 y 7.4, se presentan (con propósitos de comparación) los costos obtenidos ejecutando GLS sobre las distintas instancias, tal como lo describen sus autores, sin combinarlo con AM. Para los casos de TSPLIB, los resultados de GLS fueron sorprendentemente buenos. Comparando con 100 corridas de la heurística simple, GLS funciona en general mejor que el AM, en todas sus variantes. Contra 300 corridas de la heurística simple (con más tiempo para que la población del AM alcance estabilidad), la combinación de AM + GLS comienza a dar resultados muy similares a los de GLS.

Para el tamaño de instancias y la cantidad de iteraciones consideradas, puede concluirse que GLS supera a la metaheurística propuesta en este trabajo, pues produce resultados similares en base a una técnica más sencilla y con un costo computacional mucho menor. Sin embargo, se plantea la hipótesis de que a medida que aumente el tamaño de las instancias a resolver, la contribución del AM a la metaheurística será mayor, y podrá superar la aplicación de GLS puro.

Esta suposición se fundamenta en el esquema de penalización de GLS, que restringe en cada iteración, un pequeño conjunto de características del tour (aquellos arcos para los cuales la expresión de utilidad dada por la Ecuación (4-10) es máxima). Para instancias del TSP con un número considerablemente mayor de ciudades, sin duda la efectividad de esta técnica decrecerá, pues se requerirá una cantidad muy grande de iteraciones para desplazarse a regiones distantes en el espacio de búsqueda, a través de paulatinas modificaciones de los términos en la función de costo (ver Ecuación (4-5)). En esta situación puede ser importante el aporte de la técnica de crossover, que recombinando información de dos tours "alejados" entre sí, nos lleve rápidamente a explorar nuevas regiones.

De hecho, la mayor instancia analizada por los autores de GLS en [47] y [48] se compone de 2392 ciudades; no se conocen resultados sobre ejemplos mayores del TSP.

En este trabajo no pudo testearse la hipótesis expresada con respecto al buen funcionamiento de la metaheurística sobre instancias de superiores dimensiones, debido a las restricciones de recursos.

Para las instancias fractales, en cambio, GLS produce en general peores resultados que el AM, si bien supera a la heurística simple de búsqueda local. Una posible explicación para este resultado, está dada también por el método que utiliza GLS para modificar la función de costo.

Debido a la naturaleza del proceso constructivo de las instancias fractales, las ciudades se encuentran distribuidas a distancias regulares, por lo tanto es probable que varios arcos de cualquier tour entre dichas ciudades tengan la misma longitud. Así, la expresión de utilidad en la Ecuación (4-10) podría coincidir para varios de ellos, y muchos términos de la función de costo aumentada se alterarían en cada iteración, al penalizar numerosos arcos por vez. Por consiguiente, no se obtendría

una secuencia suave de soluciones que mejoren gradualmente (tal como proponen los autores de GLS), sino una sucesión de cambios bruscos.

Esto también debería verificarse con más experimentos sobre instancias fractales de distintos tamaños.

En resumen, la combinación de AM y GLS ofrece ventajas sobre un AM "puro", y promete muy buenos frutos sobre un rango de instancias mayores que las analizadas en este trabajo. Sin embargo, podría intentarse una mejor manera de combinar estas dos técnicas para que en el resultado influyan en forma pareja las mejores características de ambas (como se indica en el siguiente inciso), ya que en los experimentos realizados es más determinante la contribución de GLS.

Resulta especialmente aplicable aquí la afirmación hecha en [22]:

> "Tal vez, el principal mérito de un trabajo de investigación radique no en las respuestas que logra encontrar sino en las preguntas que deja planteadas"

En este caso, desde el punto de vista de la investigación, la mayor parte de lo que se obtuvo como conclusión del proyecto son preguntas y líneas abiertas para continuarlo; y desde el punto de vista personal, una valiosa experiencia de aprendizaje.

8.2. Propuestas

Antes de introducir variantes o nuevos objetivos para este trabajo, sería posible mejorarlo efectuando en él algunos cambios destinados a aumentar la eficiencia, para que resultara comparable a otras publicaciones existentes. Además, disminuyendo los tiempos de ejecución y los requerimientos de memoria, cabría la posibilidad de experimentar con instancias mayores del TSP, que someterían a mayor esfuerzo a las técnicas utilizadas, permitiendo evaluar con más claridad su funcionamiento.

Una ganancia inmediata en la performance estaría dada por la implementación del AM en paralelo, tal como se sugiere en [20] y se implementó en [45]. La concurrencia se aplicaría durante la etapa de optimización local del AM, ya que los procesos de búsqueda sobre cada individuo (en este caso, las corridas de GLS) son independientes entre sí, y por lo tanto, disponiendo de los recursos computacionales necesarios, se podrían encontrar simultáneamente tantos óptimos locales como individuos tenga la población.

También podrían probarse estructuras de datos más eficientes, diseñadas específicamente para el TSP, tales como las presentadas en [12] y [15]; y heurísticas más poderosas de búsqueda local para ser utilizadas como "núcleo" de GLS, por ejemplo, las descriptas en [20] y [25]. En el presente trabajo se eligieron, como ya se ha mencionado, una heurística y estructuras de datos sumamente sencillas, para poder discernir con mayor claridad los aportes de cada técnica (AM y GLS) sobre el funcionamiento del algoritmo. Esta decisión fue acertada, ya que aún con elementos tan simples, los resultados fueron

suficientemente cercanos a los óptimos globales como para dificultar la evaluación de cada componente de la metaheurística.

Si bien el último punto considerado en este trabajo, la evolución de los parámetros de GLS en base a la información recopilada durante el proceso de búsqueda, no tuvo una influencia perceptible, esto parece deberse principalmente a los buenos resultados conseguidos por la metaheurística con sus parámetros determinados estáticamente, que no brindaron un margen de mejora para que esta última etapa efectuara su aporte. Por consiguiente no se descarta el interés de esta innovación; pero sería deseable antes de afrontar este tema, investigar en más detalle la interacción entre AM y GLS, pues queda mucho aún por progresar en este campo antes de intentar avanzar un paso más.

Ambas técnicas dependen en gran medida de la manera de los valores elegidos para el numeroso conjunto de parámetros que las gobiernan. Como se menciona en [45] respecto al AM, se enfrenta en realidad un doble problema de optimización: el de los parámetros que dirigen la metaheurística (que no son independientes entre sí), y el que se intenta resolver. Cuando el AM se combina con GLS, técnica de la cual puede decirse algo similar, el problema aumenta. Por lo tanto, quizás el objetivo de este trabajo fue demasiado abarcativo, teniendo en cuenta la cantidad de variantes que entran en juego en los dos enfoques que fueron combinados. Así, antes de intentar que los parámetros de GLS evolucionen, sería conveniente encontrar un buen conjunto de parámetros para la combinación de AM y GLS, de manera que ambas técnicas puedan aportar sus beneficios. Una idea que surge en forma inmediata, sería reducir el número de llamadas a la heurística local que se realizan en cada iteración de GLS, para corregir la excesiva influencia del paso de optimización local en el AM, y lograr un peso más determinante de la etapa de recombinación de la población.

Una vez logrado un buen balance en este aspecto, y ya en condiciones de retomar la experimentación sobre el crossover de parámetros de GLS, podría intentarse que las variaciones introducidas en dichos parámetros por la evolución fueran más "bruscas", para que afectaran en mayor medida el comportamiento de la metaheurística (teniendo en cuenta que GLS demostró ser suficientemente robusto como para no resultar sensible a cambios menores en sus parámetros). También podrían ensayarse operadores de crossover diferentes para estos parámetros.

Sería interesante incluir otros parámetros de gran importancia, dentro del conjunto de los que van a someterse a evolución a lo largo del AM. Por ejemplo, el número de llamadas a la heurística de búsqueda local dentro de cada corrida de GLS. El poder de GLS radica en su capacidad para reunir información desde varias fuentes, y explotarla para guiar la búsqueda local a partes promisorias del espacio de búsqueda. Por eso es importante llamar al proceso de búsqueda local un número suficiente de veces como para que GLS pueda reunir información relevante sobre el espacio de búsqueda, a través de los datos que va recopilando en forma acumulativa en cada iteración; y no un número excesivo, que lleve a la metaheurística a alejarse de las regiones de interés. Por la gran importancia de

este parámetro, resultaría útil ir "aprendiendo" su valor más adecuado para cada instancia del problema, usándolo como un parámetro más a incluir en el crossover.

Este trabajo ya cuenta con posibilidades concretas de prosecución. En la Universidade Estadual de Campinas (Brasil) se proyecta modificar la metaheurística planteada, para el caso del TSP únicamente con distancias 1 y 2, apuntando a resolver el problema de hallar circuitos hamiltonianos en un grafo. En BioCom, parte del grupo de Teoría de la Computación del L.I.F.I.A., Universidad Nacional de La Plata, existe interés en abordar algunas de las propuestas mencionadas para la continuación del trabajo.

9. BIBLIOGRAFÍA

[1] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende y W. R. Stewart, "Designing and Reporting on Computational Experiments with Heuristic Methods", 'Journal of Heuristics', Vol. 1, 9-32, Kluwer Academic Publisher, Junio 1995.

[2] R. Battiti, "Machine Learning Methods for Parameter Tuning in Heuristics", versión preliminar, 5th DIMACS Challenge Workshop: Experimental Methodology Day, Rutgers University, USA, 1996.

[3] J. L. Bentley, "Experiments on Geometric Traveling Salesman Heuristics", 'First Annual ACM-SIAM Symposium on Discrete Algorithms', 91-99, San Francisco, CA, Enero 1990.

[4] J. L. Bentley, "K-d Trees for Semidynamic Point Sets", 'Sixth Annual Symposium on Computational Geometry', 187-197, Berkeley, CA, Junio 1990.

[5] J. L. Bentley, "Tools for Experiments on Algorithms", 'Proceedings of the CMU 25th Anniversary Simposium', Pittsburg, PA, Septiembre 1990.

[6] J. L. Bentley, "Fast algorithms for Geometric Traveling Salesman Problems", 'Operational Research Society of America', Vol. 4 N° 4, 1992.

[7] K. D. Boese, "Cost Versus Distance in the Traveling Salesman Problem", Technical Report CSD-950018, UCLA Computer Science Department, Los Angeles, USA, Mayo 1995.

[8] A. Díaz y F. T. Tseng, "Introducción a las Técnicas Heurísticas", Cap. 1 del libro 'Optimización Heurística y Redes Neuronales', editado por B. A. Díaz, Editorial Paraninfo, Madrid, España, 1996.

[9] B. Edmonds, "A Brief Overview and History of Memetics", 'Journal of Memetics - Evolutionary Models of Information Transmission', Noviembre 1996,
http://www.cpm.mmu.ac.uk/jom-emit/overview.html

[10] S. Esquivel y R. H. Gallard, "Computación Evolutiva: Conceptos y Aplicaciones", Universidad Nacional de San Luis. Apuntes del Curso 'Computación Evolutiva: Conceptos y Aplicaciones', M.Sc. R. Gallard, I Escuela Internacional de Informática CACIC, UNLP, Octubre 1997.

[11] C. Fleurent y J. A. Ferland, "Algorithmes Génétiques Hybrides pour l'Optimisation Combinatoire", Université de Montreal, Canadá, Noviembre 1994.

[12] M. L. Fredman, D. S. Johnson, L. A. McGeoch y G. Ostheimer, "Data Structures for Traveling Salesmen", 'Journal of Algorithms', Vol. 18 N° 3, 432-479, Mayo 1995.

[13] B. Freisleben y P. Merz, "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems", 'Proceedings of the 1996 IEEE International Conference on Evolutionary Computation', Nagoya, Japón, 616-621, 1996.

[14] B. Freisleben y P. Merz, "New Genetic Local Search Operators for the Traveling Salesman Problem", 'Proceedings of the 4th Conference on Parallel Problem Solving from Nature - PPSN IV', editado por H. M. Voigt, W. Ebeling, I. Rechenberg y H. P. Schwefel, 'Lecture Notes in Computer Science', Vol. 1141, 890-899, Springer, 1996.

[15] M. Gorges-Schleuter, "Asparagos96 and the Traveling Salesman Problem", 'Proceedings of the IEEE International Conference in Evolutionary Computation', IEEE Press, 1997.

[16] G. Grant, "Memes: Introduction", 'Principia Cybernetica Web', http://pespmc1.vub.ac.be/MEMIN.html

[17] M. Grötschel y L. Lóvasz, "Combinatorial Optimization: A Survey", Reporte Técnico 93-29, DIMACS, Mayo 1993.

[18] W. E. Hart, "Adaptive Global Optimization with Local Search", Tesis para el grado de Ph.D., University of California, San Diego, Junio 1994.

[19] M. Hoagland, "Las Raíces de la Vida. Genes, Células y Evolución", Biblioteca Científica Salvat, Barcelona, 1985.

[20] D. Johnson y L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization", a publicarse como un capítulo del libro 'Local Search in Combinatorial Optimization', editado por E. H. L. Aarts y J. K. Lenstra, 1995.

[21] D. S. Johnson, L. A. McGeoch y E. E. Rothberg, "Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound", 'Proceedings of the 7th Anual ACM-SIAM Symposium on Discrete Algorithms', 341-350, 1996.

[22] N. Krasnogor, "Heurísticas para el TSP-2D Euclídeo y Simétrico Basadas en la Triangulación de Delaunay y sus Subgrafos", Trabajo de Grado para la Licenciatura en Informática, UNLP, Abril 1998. Directores: P. Moscato y G. Baum.

[23] N. Krasnogor, P. Moscato y M. G. Norman, "A New Hybrid Heuristic for Large Geometric Traveling Salesman Problems based on the Delaunay Triangulation", 'Anales del XXVII Simposio Brasileiro de Pesquisa Operacional', Vitoria, Brasil, Noviembre 1995.

[24] M. Laguna y P. Moscato, "Algoritmos Genéticos", Cap. 3 del libro 'Optimización Heurística y Redes Neuronales', editado por B. A. Díaz, Editorial Paraninfo, Madrid, España, 1996.

[25] S. Lin y B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem", 'Operations Research' 21 (1973), 498-516.

[26] A. Mariano, P. Moscato y M. G. Norman, "Using L-Systems to Generate Arbitrarily Large Instances of the Euclidean Traveling Salesman Problem with Known Optimal Tours", 'Anales del XXVII Simposio Brasileiro de Pesquisa Operacional', Vitoria, Brasil, Noviembre 1995.

[27] A. Mariano, P. Moscato y M. G. Norman, "Arbitrarily Large Planar ETSP Instances with Known Optimal Tours", versión extendida del trabajo presentado en Vitoria, Brasil, en 1995. Aceptado para su publicación en 'Pesquisa Operacional'.

[28] E. Mayr, "Evolución", 'Investigación y Ciencia'.

[29] P. Merz y B. Freisleben, "Genetic Local Search for the TSP: New results", 'Proceedings of the 1997 IEEE International Conference on Genetic Algorithms (ICGA'97)', editado por Thomas Bäck, Morgan-Kaufmann, 465-472, East Lansing, USA, 1997.

[30] P. Moscato, "Fractal Instances of the Traveling Salesman Problem", http://www.densis.fee.unicamp.br/~moscato/FRACTAL_TSP_home.html

[31] P. Moscato, "Memetic Algorithms' Home Page", http://www.densis.fee.unicamp.br/~moscato/memetic_home.html

[32] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms", Caltech Concurrent Computation Program, C3P Report 826, 1989.

[33] P. Moscato, "TSPBIB Home Page", http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html

[34] P. Moscato y F. Tinetti, "Blending Heuristics with a Population-Based Approach: A 'Memetic' Algorithm for the Traveling Salesman Problem", Diciembre 1992, revisado Julio 1994.

[35] H. Mühlenbein, "Evolution in Time and Space - The Parallel Genetic Algorithm", 'Foundations of Genetic Algorithms', editado por G. Rawlins, 316-337, Morgan-Kaufman, 1991.

[36] H. Mühlenbein, "How Genetic Algorithms Really Work: Mutation and Hill-Climbing", 'Parallel Problem Solving from Nature - PPSN II', editado por Männer y Manderick, 15-26, North-Holland, 1992.

[37] H. Mühlenbein, "Parallel Genetic Algorithms in Combinatorial Optimization", 'Computer Science and Operations Research', editado por O. Balci, R. Sharda y S. Zenios, 441-456. Pergamon Press, Nueva York, 1992.

[38] M. G. Norman y P. Moscato, "The Euclidean Traveling Salesman Problem and a Space-Filling Curve", 'Chaos, Solitons and Fractals', Vol. 6, 389-397, 1995.

[39] M. H. Noschang, "The Traveling Salesman Problem - A Review of Theory and Current Research", University of Cincinatti. http://www.ececs.uc.edu/~mnoschan/sale.html

[40] M. Padberg y G. Rinaldi, "Optimization of a 532-city Symmetric TSP by Branch and Cut", Elsevier Science Publishers B.V., North-Holland, 1987.

[41] G. Reinelt, "Fast Heuristics for Large Geometric Traveling Salesman Problems", 'ORSA Journal on Computing', Vol. 4 Nº 2, 206-217, 1992.

[42] G. Reinelt, "TSPLIB: A Traveling Salesman Problem Library", 'European Journal of Operations Research', 52 (1991), 125.

[43] G. Reinelt, "TSPLIB", http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html

[44] J. E. Smith y T. C. Fogarty, "Adaptively Parameterised Evolutionary Systems: Self Adaptive Recombination and Mutation in a Genetic Algorithm", 'Parallel Problem Solving from Nature IV', editado por Voigt, Ebeling, Rechenberg y Schwefel, Springer Verlag, 1996.

[45] F. Tinetti, "Metaheurísticas para Problemas de Optimización Combinatoria", Trabajo de Grado para la Licenciatura en Informática, UNLP, Septiembre 1992. Director: P. Moscato.

[46] E. P. K. Tsang, J. E. Borrett y A. C. M. Kwan, "An Attempt to Map the Performance of a Range of Algorithm and Heuristic Combinations", Department of Computer Science, University of Essex.

[47] C. Voudouris, "Guided Local Search for Combinatorial Optimisation Problems", Tesis para el grado de Ph.D., Department of Computer Science, University of Essex, 1997.

[48] C. Voudouris y E. P. K. Tsang, "Guided Local Search", Department of Computer Science, University of Essex, Agosto 1995.

[49] C. Voudouris y E. P. K. Tsang, "Guided Local Search", http://cswww.essex.ac.uk/Research/CSP/gls.html

[50] D. Whitley, "Modeling Hybrid Genetic Algorithms", 'Genetic Algorithms in Engineering and Computer Science', editado por G. Winter, J. Periaux, M. Galan y P. Cuesta, 191-201, John Wiley, 1995.

[51] D. Whitley, V. S. Gordon y K. Mathias, "Lamarckian Evolution, the Baldwin Effect and Function Optimization", 'Parallel Problem Solving from Nature - PPSN III', editado por H. P. Schwefel y R. Manner, 6-15, Springer-Verlag, 1994.

[52] M. Zachariasen y M. Dam, "Tabu Search on the Geometric Traveling Salesman Problem", Department of Computer Science, University of Copenhagen, presentado en Metaheuristics International Conference 95, Breckenridge, Colorado, USA, 1995.