UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA
BARRANQUILLA

COMPARACIÓN Y APLICACIÓN DE LAS TÉCNICAS TRADICIONALES Y LAS TÉCNICAS HEURÍSTICAS- META HEURÍSTICAS

CELIN T. FREDDY

CONSUEGRA MOLINA INMACULADA C.

ESCOBAR SOLAEZ BEIRO JOSÉ

JIMÉNEZ DE LA CRUZ ADOLFO E.

Presentado como requisito parcial para optar al titulo de Ingeniero de Sistemas

UNIVERSIDAD SIMÓN BOLÍVAR
DIVISIÓN DE INGENIERÍAS
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
BARRANQUILLA
2006

INTRODUCCIÓN

Si un problema está formulado de forma unívoca, y el espacio de todas las soluciones posibles es conocido, un método de búsqueda trata de encontrar las soluciones o soluciones que satisfagan el enunciado del problema. Dicho así parece una tautología, pero no todos los problemas son problemas de búsqueda, ni todos los espacios de búsqueda se pueden definir también de forma unívoca.

Pero si resulta que efectivamente, un problema se puede efectuar como problema de búsqueda, en muchos casos se puede reducir a un problema de hallar un máximo o mínimo, o sea, un óptimo. Sólo va a funcionar esto en el caso de que se pueda calcular la bondad de una solución: la solución del problema será aquella, o aquellas, que optimicen una función de bondad, ajuste, evaluación o *fitness*; en muchos casos, por lo tanto, un problema de búsqueda se puede reducir a un problema de optimización (maximización o minimización).

Generalmente, para resolver un problema, del mundo real, no se trabaja directamente con el mundo real, es decir, al modelo se le extraen las características fundamentales del mismo. En todo caso, solo se puede resolver un problema si tiene un modelo del mismo; la solución al problema real será tanto más adecuada cuanto mas cercano sea el modelo al problema real, pero, en todo caso. La solución a un problema lo es sólo en el sentido que es una solución al modelo del problema.

En algunos casos, modelar el problema supone, de alguna forma, simplificarlo; incluso en casos en el que el problema este formulado

computacional mente de forma completa, hace falta hacer aproximaciones para hacer una resolución mas simple usando un método determinado.

Los problemas cuya solución varía con el tiempo son un ejemplo de problemas difíciles de solucionar usando técnicas tradicionales; sin embargo los problemas con restricciones tiene una larga tradición. Una restricción es simplemente una condición que tiene que cumplir la solución, generalmente formulada como una función.

En general todos los métodos de búsqueda se dividen en: métodos *globales y locales.* Los métodos globales tratan de encontrar el máximo global de un problema a diferencia de los locales que se centran en la vecindad de la solución generada inicialmente y, por tanto, necesitan de una técnica adicional, como comienzos múltiples para acercarse al máximo global.

La observación de la naturaleza ha sido una de las principales fuentes de inspiración para la propuesta de nuevos paradigmas computacionales. Así nacieron diversas técnicas de Inteligencia Artificial como: los Algoritmos Genéticos (*Genetic Algorithms*), Templado Simulado (*Simulated Annealing*), Redes Neuronales (*Neural Networks*), y entre estas técnicas, el sistema basado en Colonias de Hormigas

Hablaremos de dos técnicas, especialmente como lo son: Algoritmos Genéticos y Colonias de Hormigas (*Ant Colony System*) y contrastaremos una de estas técnicas con una técnica tradicional usando *Backtracking* que nos permitirá conocer que tan fuertes son estas técnicas a la hora de solucionar problemas de tipo NP.

Podemos decir que Los algoritmos genéticos se encuadran dentro de la clase de algoritmos que presentan ciertas analogías con los procesos

biológicos de la naturaleza. Están incluidos, por tanto, en el marco de la Bioinformática, área de especialización encargada de estudiar modelos y técnicas basándose en patrones biológicos y aprovechando las metodologías y técnicas informáticas. La bioinformática, trata de dar solución a una gran variedad de problemas de un amplio domino científico.

Dentro de este campo, nos encontramos con la Computación Evolutiva, que es un enfoque alternativo para abordar problemas complejos de búsqueda y aprendizaje a través de modelos computacionales de procesos evolutivos. Las implementaciones concretas de tales modelos se conocen como *ALGORITMOS EVOLUTIVOS*. El propósito genérico de los algoritmos evolutivos consiste en guiar una búsqueda estocástica haciendo evolucionar a un conjunto de estructuras y seleccionando de modo iterativo las más adecuadas.

La computación evolutiva parte de un hecho observado en la naturaleza: los organismos vivos poseen una destreza consumada en la resolución de los problemas que se les presentan, y obtienen sus habilidades, casi sin proponérselo, a través del mecanismo de la evolución natural. La evolución se produce, en casi todos los organismos, como consecuencia de dos procesos primarios: la selección natural y la reproducción sexual. La primera determina qué miembros de la población sobrevivirán hasta reproducirse (es un proceso sencillo: cuando un organismo falla una prueba de idoneidad, muere). La reproducción sexual garantiza la mezcla y recombinación de genes en la descendencia de un organismo.

Las condiciones que determinan un proceso evolutivo en la naturaleza son:

- Existe una población de entidades o individuos con capacidad para reproducirse.
- Existe alguna diferencia o variedad entre estos individuos.
- Esta variedad está relacionada con algunas diferencias en la habilidad para sobrevivir.

Como ya hemos mencionado anteriormente, *Los Algoritmos Genéticos* ocupan el lugar central dentro de la computación evolutiva. Las razones de esta preeminencia son tanto teóricas como prácticas, siendo las más importantes:

- Los algoritmos genéticos constituyen el paradigma más completo de la computación evolutiva, esto es, resumen de modo natural, todas las ideas fundamentales de dicho enfoque.
- Son muy flexibles, es decir, pueden adoptar con facilidad nuevas ideas, generales o específicas, que surjan dentro del campo de la computación evolutiva. Además se pueden hibridar fácilmente con otros paradigmas y enfoques, aunque no tengan ninguna relación con la computación evolutiva.

Los algoritmos genéticos (AGs) son el paradigma con mayor base teórica en la computación evolutiva. Además, dicha base teórica es sencilla en su desarrollo y con grandes posibilidades de ampliación.

Entre todos los paradigmas de la computación evolutiva son los que menos conocimiento específico necesitan para su funcionamiento, y en

consecuencia, los más versátiles. Pero es que además pueden incorporar conocimiento específico con poco esfuerzo adicional.

Son fácilmente implementables en computadores con capacidades medias, proporcionando resultados aceptables, en cuanto a precisión y recursos empleados, para una gran cantidad de problemas difícilmente resolubles por otros métodos.

Por tal razón la popularidad de los AGs se debe en parte a que la evolución es un método robusto y bien probado dentro de los sistemas biológicos naturales. Además son fácilmente paralelizables, lo que supone una ventaja gracias al abaratamiento actual de los costes en hardware. Por otra parte, los AGs pueden realizar búsquedas en espacios de hipótesis que contienen complejas interacciones entre las distintas partes, donde el impacto de cada parte sobre la función de evaluación es difícil de especificar.

Después de enumerar estas características podríamos definir los AGs, de forma general, como "métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas. En ellos se mantiene una población que representa un conjunto de posibles soluciones, la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos, y un proceso de selección sesgado en favor de los mejores candidatos".

Todo esto hace que los algoritmos genéticos proporcionen una mayor robustez a la búsqueda, esto es, más eficiencia sin perder generalidad.

Ahora bien por otro lado tenemos esta técnica conocida como Colonias de Hormigas ((*Ant Colony System*)), estos algoritmos son, sin ninguna duda, los más populares de todos, sobre todo en las últimas décadas. Parten

principalmente del estudio del comportamiento de las hormigas sociales, realizado principalmente por los mimercologos Pierre-Paul Grassé y Edward O. Wilson, que viene a entender la colonia como un solo organismo, cuyos componentes u órganos son las hormigas en sí.

Los componentes de una colonia se comportan de tal forma que el comportamiento de la misma es mucho mas que lo que cabria deducir de la suma de sus partes o incluso de la inteligencia de cada una de las partes.

Para explicar esto Pierre Paul Grassé introdujo el concepto de Stigmergia, que consiste en la sinergia por la comunicación que se realiza entre los miembros de la colonia a través del ambiente en el que se mueven de esta manera se puede explicar la realización de obras colectivas sin necesidad de la intervención de una autoridad central.

En general, la comunicación entre las hormigas es bastante escasa, y su entendimiento limitado; la mayor parte de la comunicación se realiza a través de los actos que otras hormigas han realizado, o bien a través de la acumulación de feromonas en el suelo Por ejemplo, si la colonia se dedica a acumular las larvas en un sitio determinado, no hay ninguna hormiga que decida de antemano en qué sitio se van a acumular: simplemente se cogen larvas de un lado, y se sueltan donde ya se detecte que hay otras larvas.

Casi todos los comportamientos de una colonia de hormigas, desde la construcción de estructuras, hasta la localización de comida se pueden usar para algún tipo de problema de búsqueda; toda esta familia de algoritmos se denomina *ant colony optimization* u *optimización por colonia de hormigas*. En general, tienen en común las *hormigas*, las *feromonas*, y un *entorno en el cual se desarrolla la labor* de la colonia.

Tanto las colonias de hormigas como los algoritmos que las imitan, se basan en tres principios: *retroalimentación positiva*, junto con *retroalimentación negativa*, lo cual lleva a una *amplificación de las fluctuaciones*, y el hecho de que sucedan *fluctuaciones múltiples*. Debe haber un equilibrio entre los dos tipos de retroalimentaciones; si no, se llegaría a una situación de caos, y las dos juntas hace que las fluctuaciones (por ejemplo, el que unas pocas hormigas elijan un camino en vez de otro) se amplifiquen. Por otro lado, estos comportamientos sólo pueden tener lugar si hay múltiples individuos e interacciones; en el caso de que haya solo unos pocos, los comportamientos serán más predecibles a corto plazo, y menos *sorpresivos*. Curiosamente, estos principios son comunes a la mayoría de los algoritmos bioinspirados.

Los algoritmos de hormigas se pueden usar también para clasificación de documentos.

También hablaremos de Backtracking, técnica conocida como "*fuerza bruta*", método sistemático que itera a través de todas las combinaciones posibles del espacio de búsqueda. Es una técnica general que debe ser adaptada para cada aplicación particular. Siempre puede encontrar todas las soluciones existentes.

Backtracking (o búsqueda atrás) es una técnica de programación para hacer búsqueda sistemática a través de todas las configuraciones posibles dentro de un espacio de búsqueda. Para lograr esto, los algoritmos de tipo backtracking construyen posibles soluciones candidatas de manera sistemática. En general, dado una solución candidata s:

1. Verifican si s es solución. Si lo es, hacen algo con ella (depende del problema).

2. Construyen todas las posibles extensiones de s, e invocan recursivamente al algoritmo con todas ellas.

A veces los algoritmos de tipo Backtracking se usan para encontrar una solución, pero otras veces interesa que las revisen todas (por ejemplo, para encontrar la más corta).

De esta forma la siguiente investigación aborda principalmente los Algoritmos Genéticos, Colonias de Hormigas y respectivamente la comparación entre una de estas técnicas con la técnica de Backtracking.

En el capitulo 1, se presentan los conceptos básicos de la heurística y la meta heurística. En el capitulo 2, una reseña histórica que nos remite a la teoría de la evolución y la relación de esta con los Algoritmos Genéticos. En el capitulo 3, se desarrolla los fundamentos teóricos de los Algoritmos Genéticos (AGs). En el capitulo 4, se hace referencia a los operadores utilizados para el desarrollo y funcionamiento de los AGs. En el capitulo 5 las ventajas y desventajas de los AGs frente a otras técnicas. En el capitulo 6, se hablará de los paquetes utilizados para desarrollar los AGs. En el capitulo 7, las áreas donde se han aplicado AGs. En el capitulo 8 hablaremos de las Colonias de Hormigas. En el capitulo 9, conoceremos sobre la técnica de Backtracking y finalmente en el capitulo 10, hablaremos de la comparación y que parámetros se utilizo para realizarla, (Colonias de Hormigas Vs Backtracking).

UNIVERSIDAD SIMON BOLIVAR
HEMEROTECA
BARRANQUILLA

ÍNDICE

1. PLANTEAMIENTO DEL PROBLEMA

1.1 Descripción del problema ----------18
1.2 Formulación de la Hipótesis ----------18

2. OBJETIVOS

2.1 Objetivo General ----------19
2.2 Objetivos Específicos ----------19

3. JUSTIFICACIÓN DEL PROYECTO

3.1 Descripción ----------20

4. MARCO DE REFERENCIA

4.1 Marco Teórico ----------23
4.2 Marco Conceptual ----------25
4.3 Marco Espacial ----------29
4.4 Marco Legal ----------29-30

5. METODOLOGÍA

5.1 Tipo de Estudio
5.2 Recursos ----------31
5.2.1 Recursos Humanos ----------32

5.2.2 Recursos Financieros ----------33
5.2.3 Recursos del Entorno ----------33

6. CAPÍTULOS

6.1 CAPITULO 1

Heurística y Meta heurística

6.1.1 Objetivo del Capitulo ----------34
6.1.2 Concepto de Heurística ----------34
6.1.3 Concepto de Meta heurística----------36
6.1.4 Necesidades de las Meta heurísticas----------38
6.1.5 Aspectos técnicos----------38
6.1.6 Procedimientos Meta heurísticos----------39
6.1.6.1 Algoritmos Genéticos ----------41
6.1.6.2 Simulated Anneling (Temple Simulado) ----------44
6.1.6.3 Tabú Search ----------44
6.1.6.4 GRASP ----------47

6.2 CAPITULO 2

Reseña Histórica

6.2.1 Objetivo del Capitulo ----------50
6.2.2 Teoría de la Evolución ----------50
6.2.3 Nacimiento de los Algoritmos Genéticos ----------52
6.2.4 Utilización de un Algoritmo Genético ----------55

6.3 CAPITULO 3

Fundamentos Teóricos

6.3.1 Objetivo del Capitulo ----------57
6.3.2 Estructura de un Algoritmo Genético ----------57

6.3.3 Codificación de datos de un Algoritmo genético ----------59
6.3.4 Procedimiento (seudo código) de un Algoritmo Genético ----------60
6.3.5 Parámetros utilizados para aplicar un Algoritmo Genético ---------65
6.3.6 Teorema del Esquema ----------66
6.3.7 Convergencia de un Algoritmo Genético ----------68

6.4 CAPITULO 4

Operadores Genéticos

6.4.1 Objetivo del Capitulo ----------71
6.4.2 Operador selección ----------71
6.4.2.1 Selección Directa ----------71
6.4.2.2 Selección Aleatoria ----------71
6.4.2.2.1 Selección Equiparable ----------72
6.4.2.2.2 Selección Estocástica ----------72
6.4.2.2.2.1 Selección Elitista ----------72
6.4.2.2.2.2 Selección Proporcional a la aptitud ----------73
6.4.2.2.2.3 Selección por Rueda de Ruleta. ----------72
6.4.2.2.2.4 Selección por Torneo ----------74
6.4.2.2.2.5 Selección por Rango ----------74
6.4.2.2.2.6 Selección Generacional ----------74
6.4.2.2.2.7 Selección por Estado Estacionario----------74
6.4.2.2.2.8 Selección de Búsqueda del Estado Estacionario ------75
6.4.2.2.2.9 Selección Jerárquica. ----------75
6.4.3 Operador Cross-Over ----------75
6.4.3.1 Cross-Over Uniforme ----------76
6.4.3.2 Cross-Over de n-puntos ----------77
6.4.3.3 Cross-Over Especializados ----------77
6.4.4 Operador Mutación ----------77
6.4.4.1 Mutación de Bit ----------78

6.4.4.2 Mutación Multibit ----78
6.4.4.3 Mutación de Gen ----79
6.4.4.4 Mutación Multigen ----79
6.4.4.5 Mutación de Intercambio ----79
6.4.4.6 Mutación de Barajado ----79
6.4.4.7 Creep ----80
6.4.5 Otros Operadores ----80
6.4.5.1 Cromosoma de Longitud Variable ----81
6.4.5.2 Operadores de Nicho Ecológico ----82
6.4.5.3 Operadores Especializados ----83
6.4.5.4 Zap ----83
6.4.5.5 Creep ----83
6.4.5.6 Transposición ----83

6.5 CAPITULO 5

Ventajas y Limitaciones de los Algoritmos Genéticos

6.5.1 Objetivo del Capitulo ----85
6.5.2 Ventajas de los Algoritmos Genéticos ----85
6.5.3 Limitaciones de los Algoritmos Genéticos ----87
6.5.4 Robustez de los Métodos de Búsqueda y Optimización tradicionales ----90
6.5.5 Diferencia entre los algoritmos Genéticos y los Métodos tradicionales ----91

6.6 CAPITULO 6

Áreas De Aplicación

6.6.1 Objetivo del Capitulo ----93
6.6.2 Acústica ----93
6.6.3 Ingeniería Aeroespacial ----93

6.6.4 Astronomía y Astrofísica ----94
6.6.5 Química ----94
6.6.6 Ingeniería Eléctrica ----94
6.6.7 Mercados Financieros ----95
6.6.8 Juegos ----95
6.6.9 Geofísica ----95
6.6.10 Ingeniería de Materiales ----95
6.6.11 Matemáticas y Algoritmia ----96
6.6.12 Ingeniaría de Sistemas ----96
6.6.13 Aplicaciones de Búsqueda y Optimización ----96
6.6.14 Aprendizaje Automático ----96
6.6.15 Aplicaciones de los AGs en la Computación Evolutiva ----100

6.7 CAPITULO 7

Paquetes para Resolver Algoritmos Genéticos

6.7.1 Objetivo del Capitulo ----105
6.7.2 Galopps ----105
6.7.3 Gags ----105
6.7.4 FORTRAN GA ----105
6.7.5 Galib ----106
6.7.6 Gas ----106
6.7.7 Geco ----106
6.7.8 GPdata ----106
6.7.9 GPJPP ----107
6.7.10 GP Kernel ----107
6.7.11 LIL-GP ----107
6.7.12 PGAPack ----107
6.7.13 SUGAL ----108
6.7.14 ADATE ----108

6.7.15 GPsys ----------108

6.8 CAPITULO 8

Colonias de Hormigas (*Ant Colony Optimization* u *Optimización por colonia de hormigas*).

6.8.1 Objetivo del capitulo ----------109

6.8.2 Fundamentos Teóricos ----------109

6.8.3 Colonias de hormigas Naturales ----------111

6.8.4 Algoritmos de Optimización Basados en Colonias de Hormigas

6.8.4.1 De La Hormigas Naturales a La Optimización de Colonias de Hormigas ----------113

6.8.4.2 La Hormiga Artificial ----------115

6.8.4.3 Similitudes y Diferencias Entre las Hormigas Naturales y Artificiales ----------116

6.8.4.4 Modo de Funcionamiento y Estructura Genérica de un Algoritmo de OCH ----------118

6.8.4.5 Pasos a seguir para Resolver Problemas mediante las OCH ----------122

6.8.4.6 Sistemas de Hormigas ----------123

6.8.4.7 Sistemas de Colonias de hormigas ----------124

6.8.4.8 Funciones adicionales de los algoritmos de OCH ----------125

6.9 CAPITULO 9

Técnica Backtracking (Fuerza Bruta)

6.9.1 Objetivo del capitulo ----------129

6.9.2 Fundamentos Teóricos ----------129

6.9.2.1 La vuelta del caballo ----------133

6.9.2.2 El problema de las ocho reinas ----------138

6.9.2.3 El problema de la mochila (selección Optima) ----------142

6.9.2.4 El problema del laberinto ----------145
6.9.3 Algoritmo Genérico Backtracking ----------148

6.10 CAPITULO 10

Aplicación a desarrollar ----------150

7. CONCLUSIONES ----------207
8. ANEXOS ----------208
9. BIBLIOGRAFIA ----------215

1. PLANTEAMIENTO DEL PROBLEMA

1.1 Descripción del problema

Determinar si el uso de técnicas meta heurísticas (Colonias de hormigas) en la solución de problemas de búsqueda ofrecen mejores resultados que los obtenidos en los métodos tradicionales (Backtracking).

1.2 Formulación de la Hipótesis

La utilización de las meta heurísticas en problemas de programación originan mejores soluciones porque trabajan probabilística mente, a diferencia de los Métodos Tradicionales que plantean soluciones eficaces, ya que trabajan deterministicamente.

2. OBJETIVOS

2.1 Objetivo General

Comparar técnicas tradicionales y técnicas Heurísticas- Meta Heurísticas para determinar cual de las dos dan mejores resultados en la solución de problemas de búsqueda.

2.2 Objetivos Específicos

- Identificar información de los métodos heuristicos y meta heurísticos.
- Recopilar información de la historia y surgimiento de los Algoritmos Genéticos, Colonia de hormigas y Backtracking.
- Analizar el funcionamiento de un algoritmo genético, Colonia de Hormigas y Backtracking.
- Identificar las ventajas de los algoritmos genéticos y colonias de Hormigas sobre los métodos tradicionales.
- Identificar las limitaciones de los algoritmos genéticos.
- Conocer las áreas donde se han utilizado los algoritmos genéticos y que resultados se han obtenido.
- Indagar sobre los lenguajes donde pueden ser desarrollados los algoritmos genéticos.
- Conocer los paquetes para la elaboración de algoritmos genéticos.
- Demostrar mediante una aplicación las ventajas que poseen las técnicas meta heurísticas sobre las técnicas tradicionales.

3. JUSTIFICACIÓN DEL PROYECTO

3.1 Descripción

En la actualidad la comunidad científica internacional ha demostrado un gran interés en nuevas técnicas llamadas y conocidas como Algoritmos Genéticos (AGs) y Colonia de Hormigas (Ant Colony System), la primera se fundamenta en la teoría de la evolución y la segunda en la imitación del comportamiento de las hormigas. Para ser más exactos un AG es una técnica de programación que imita la evolución biológica como estrategia para resolver problemas.

Dado un problema especifico a resolver, la entrada de los AGs es un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera y una métrica llamada función de aptitud que permite evaluar cuantitativamente a cada candidata.

El comportamiento de las hormigas que son insectos sociales que viven en colonias y que tienen un comportamiento dirigido al desarrollo de la colonia como un todo más que a un desarrollo individual.

Una característica interesante del comportamiento de las colonias de hormigas es cómo pueden encontrar los caminos más cortos entre el hormiguero y la comida.

En su recorrido, depositan una sustancia llamada feromona que todas pueden oler (*estimergia*). Este rastro permite a las hormigas volver a su hormiguero desde la comida.

UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA
BARRANQUILLA

Cada vez que una hormiga llega a una intersección, decide el camino a seguir de un modo probabilístico, por tanto las hormigas eligen con mayor probabilidad los caminos con un alto rastro de feromona.

La investigación de este proyecto esta dirigida hacia los AGs y Colonia de Hormigas para realizar respectivamente una comparación que permita ver que tan buenas son estas técnicas frente a las técnicas tradicionales en este caso Backtracking.

Se hace ya que en la Universidad Simón Bolívar no se cuenta con materiales y documentos que permitan conocer, indagar e investigar a los estudiantes y profesores sobre el tema, debido a esto se percibe una falencia en este tema por parte de la comunidad bolivariana en la parte de Inteligencia Artificial.

En la Alma Mater se encuentran libros sobre Inteligencia Artificial, pero estos no abarcan el tema de los AGs y las Colonias de Hormigas de una forma extensa para su investigación y posterior practica en problemas que se presenten en la actualidad.

Al realizar esta investigación además de conocer los AGs y tener conocimiento de Las Colonias de hormigas, también se pretende hacer una comparación entre las técnicas tradicionales con una de estas meta heurísticas, es decir, encontrar mediante el desarrollo de un algoritmo argumentos que permitan crear una diferenciación entre éstas.

De hecho lo que se quiere es que en la universidad en el área de la Inteligencia Artificial sea mas explorada para que así la comunidad comience a desarrollar estrategias que permitan plantear diferentes

alternativas de desarrollo que se puedan resolver con el uso del los AGs y las Colonias de Hormigas.

Es por eso que se hace necesaria una investigación ardua sobre el tema, sus interrogantes y una respectiva comparación entre los métodos para que los que accedan a este material puedan contrastar los diferentes conceptos y ver si su uso puede ser complementado y por consiguiente puesto en práctica.

4. MARCO DE REFERENCIA

4.1 Marco Teórico

Desde tiempo atrás la naturaleza siempre ha sido sabia, cualquier animal o vegetal a lo largo de su generación y existencia ha logrado adaptarse a cambios que se han presentado en su ambiente de desarrollo y por consiguiente en su estructura para poder defenderse de aquellos agentes o individuos que atenten con su supervivencia y desarrollo (Teoría de Darwin).

Ante esto, el hombre ha tenido muchos interrogantes, entre las cuales ¿Si la naturaleza es sabia, por que no hallar la forma o los mecanismo para poder simular su comportamiento?, ahora bien así como este interrogante surgen muchos mas y es donde aparecen los Algoritmos Genéticos (AGs) y las Colonias de Hormigas. Los AGs, son algoritmos de búsqueda basados en el mecanismo de selección natural donde combinan la supervivencia de las especies, mirándolas como supervivencia de secuencias mejores adaptadas con cambios aleatorios de información en cualquier generación.

Los algoritmos genéticos (AGs) proporcionan un método de aprendizaje basado en la analogía con la evolución de las especies y generan un conjunto de hipótesis mediante la mutación y recombinación de parte del conjunto de hipótesis conocido. En cada paso el conjunto de hipótesis conocido como "población actual" se renueva remplazando una proporción de esta población por los sucesores de las hipótesis más "adecuadas" (mediante el uso de una

función de evaluación o fitness). La popularidad de los AG se debe en parte a que la evolución es un método robusto y bien probado dentro de los sistemas biológicos naturales.

La programación mediante algoritmos genéticos suponen un nuevo enfoque que permite abarcar todas aquellas áreas de aplicación donde no sepamos como resolver un problema, pero si seamos conscientes de que soluciones son buenas y cuales son malas. Desde aplicaciones evidentes, como la biología o la medicina, hasta otros campos como la industria (clasificación de piezas en cadenas de montaje). Los algoritmos genéticos poseen un importante papel en aplicaciones de búsqueda y optimización. La capacidad que poseen para favorecer a los individuos que explican bien los ejemplos, a costa de los que no lo hacen, consigue una nueva generación con mejores reglas y, por lo tanto el sistema será capaz de ir aprendiendo a conseguir mejores resultados.

Por otro lado las Colonias de Hormigas, se basan en el comportamiento colectivo de las hormigas en la búsqueda de alimentos para su subsistencia. Resulta fascinante entender como animales casi ciegos, moviéndose aproximadamente al azar, pueden encontrar el camino más corto desde su nido hasta la fuente de alimentos y regresar. Para esto, cuando una hormiga se mueve, deja una señal odorífera, depositando una sustancia denominada feromona, para que las demás puedan seguirla. En principio, una hormiga aislada se mueve esencialmente al azar, pero las siguientes deciden con una buena probabilidad seguir el camino con mayor cantidad de feromonas.

4.2 Marco Conceptual

- **Algoritmo Genético (AG):** técnica de búsqueda basada en la teoría de la evolución de Darwin, se basa en los mecanismos de selección que utiliza la naturaleza para la selección natural.
- **Individuo**: Representación de una solución del espacio de soluciones factibles. Puede ser una cadena de 0's y 1's como en los algoritmos genéticos, una representación de enteros, de números reales, o una combinación de varios de los anteriores. Un individuo es una estructura de dato.
- **Población:** conjunto de soluciones, que se espera que a través de la aplicación de operadores de cruce y mutación, se encuentren las soluciones optimas a l problema.
- **Inteligencia Artificial (IA):** estudio del comportamiento inteligente en las maquinas.
- **Fitness o Función Aptitud:** Es un valor dado a un individuo que mide que tan bien se ha adaptado al problema, en otras palabras, el fitness mide que tan bueno es un individuo dentro de la población en la que se encuentre. Este valor es el que posibilita que los mejores individuos tengan mayor probabilidad de ser seleccionados para el cruzamiento.
- **Métodos de selección:** provee la abstracción del mecanismo de la selección natural.
    - Selección por ruleta
    - Selección basado en rango
    - Selección por control esperado
    - Selección elitista

- Cross-over (n puntos, uniforme, especializado).

- **Métodos de Cruza:** forma de calcular el genoma del nuevo individuo en función del genoma padre y madre determina en gran medida la evolución de la población.
  - Cruce Básico
  - Cruce Multipunto
  - Cruce Segmentado
  - Cruce Uniforme
  - Cruce para permutación
- **Métodos de mutación:** se define como una variación de las informaciones contenidas en el código genético, habitualmente.
  - Mutación simple
  - Mutación adaptativa por convergencia
  - Mutación adaptativa por temperatura
  - Mutación de bit
  - Mutación multbit
  - Mutación de gen
  - Mutación multigen
  - Mutación de intercambio
  - Mutación de barajado
- **Criterios:** decisiones que debe tomarse para implementar AGs.
  - Criterio de codificación
  - Criterio de tratamiento no factible
  - Criterio de inicialización
  - Criterio de parada
  - Criterio de reemplazo
- **Operadores genéticos:** se emplean para determinar como va hacer la nueva generación (cruce y mutación).

- **Parámetros de funcionamiento:** son parámetros básicos y fundamentales para el funcionamiento de los AGs.
- **Genoma:** todos los parámetros que definen a todos los individuos de la población.
- **Genotipo:** descripción de un individuo.
- **Fenotipo:** expresión de un genotipo.
- **Gen:** serie de parámetros de un problema.
- **Problemas NP:** problemas con una complejidad computacional no polinomial.
- **C++:** Lenguaje de programación orientado a objetos creado por Bjarne Stroustrup como superconjunto del lenguaje C. Hoy en día es el lenguaje más popular entre los profesionales de la informática.
- Feromona: sustancia odorífera que dejan las hormigas para que las demás puedan seguirla.
- Backtracking (o búsqueda atrás): técnica de programación para hacer búsqueda sistemática a través de todas las configuraciones posibles dentro de un espacio de búsqueda.
- Colonia de Hormigas: heurística se basa en imitar el comportamiento de las hormigas.
- Heurística: deriva del Griego Heuriskein, que significa encontrar o descubrir, es una técnica que busca las soluciones de buena calidad a un costo computacional razonable sin garantizar la optimalidad de las mismas.

- **Meta heurística:** heurísticas de nivel más alto, estrategias que guían el proceso de búsqueda, incluyendo en general heurísticas subordinadas. Son de uso genérico, es decir no especificadas para un casi de problemas
- **Colonias de Hormigas Naturales**: Las hormigas son insectos sociales que viven en colonias y que, debido a su colaboración mutua, son capaces de mostrar comportamientos complejos y realizar tareas difíciles desde el punto de vista de una hormiga individual.
- **Algoritmos de Optimización de Colonias de Hormigas:** se inspiran directamente en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización combinatoria.
- **Hormiga Artificial:** La hormiga artificial es un agente computacional simple que intenta construir soluciones posibles al problema explotando los rastros de feromona disponibles y la información heurística.
- **Sistemas de Hormigas:** El SH, desarrollado por Dorigo, Maniezzo y Colorni en 1991, fue el primer algoritmo de OCH. El SH se caracteriza por el hecho de que la actualización de feromona se realiza una vez que todas las hormigas han completado sus soluciones.
- **Recocido simulado:** una de las meta heurísticas más antiguas que incorpora una estrategia explícita para impedir óptimos locales. Basada en la física de calentamiento de metales.
- **Búsqueda Tabú:** utiliza una búsqueda local con memoria a corto plazo que le permite escapar de mínimos locales y evita ciclos.

- **GRASP:** es una heurística que combina procedimientos constrictivos y de búsqueda local. Es un procedimiento iterativo de dos fase: una construcción de la solución (heurística) y otra de mejora.

4.3 Marco Espacial

La investigación será realizada para la Universidad Simón Bolívar, ubicada en la ciudad de Barranquilla, con sedes en Carrera 59 No. 59-92 PBX 344 4333. A.A. 50595 Barranquilla-Colombia, edificación situada en la calle 59 con carrera 54,

La Universidad Simón Bolívar, debe su existencia al espíritu creador del doctor José Consuegra Higgins, científico social, especializado en economía, ex decano de varias facultades de economía y ex rector de la Universidad del Atlántico, quien con el invaluable apoyo de su señora esposa doña Ana Bolívar de Consuegra, fundó la institución el 15 de octubre del 1972, iniciando sus labores académicas el 23 de marzo del 1973.

Desde entonces ha sido propósito de nuestra Universidad, promover un ambiente pedagógico que favorezca el desarrollo de la capacidad de aprendizaje y comprensión del estudiante y por eso sus profesores poseen una alta calidad intelectual, ética y moral, así como un dominio de la más avanzada metodología para la enseñanza moderna.

4.4 Marco Legal

El gobierno nacional propenderá porque el país se parte de los acuerdos o convenios internacionales, tendientes a evitar la doble

distribución en el pago de regalías por derechos de autor, correspondientes a las obras de carácter científico o cultural descritas en la ley.

Los autores de obras literarias, científicas o culturales conjuntamente con los editores de las mismas, tendrían derecho a participar de una remuneración o compensatoria por la reproducción de tales obras al amparo del artículo anterior.

El gobierno nacional propiciara la canalización de recursos para otorgar créditos, en condiciones favorables y a largo plazo a las personas naturales o jurídicas que inviertan en el ensanche o aperturas de nuevas librerías o de sucursales de las ya establecidas.

Para tener derecho a los beneficiarios establecidos en este articulo las librerías y sucursales según el caso se deben dedicar exclusivamente a la veta de libros. Folletos, revistas o coleccionables seriados de carácter científico o cultural.

UNIVERSIDAD SIMON BOLIVAR
HEMEROTECA
BARRANQUILLA

5. METODOLOGÍA

Para efectos de la siguiente investigación, con el propósito de elegir instrumentos para la recopilación de datos, fueron realizadas consultas principalmente por Internet, pues, este es un medio donde se encuentra información muy actualizada, dada la versatilidad y el alcance de este medio hoy día, hay mas variedad de temas y por sus carácter masivo es económico, rápido y eficaz.

5.1 Tipo de Estudio

Según el estudio a realizar se manifiesta dos tipos de investigación, cada uno de ellos tiene una estrategia diferente para su tratamiento metodológico. Estos son investigación Descriptiva y Exploratoria.

- **Investigación Descriptiva:** en el trabajo a desarrollar se pretende hacer un estudio sobre los Algoritmos Genéticos, Colonias de Hormigas, Backtracking y compararlos con los métodos tradicionales y ver que tan buenos pueden ser a la hora de resolver problemas de Búsqueda.
- **Investigación Exploratoria:** la realización de la comparación entre técnicas tradicionales y meta heurísticas, implica realizar indagaciones continuas sobre aquellos ejemplos de aplicaciones de los AGs en la actualidad y Colonias de Hormigas y calificar los resultados obtenidos en tales aplicaciones.

5.2 Recursos

5.2.1 Recursos Humanos

Estudiantes:

Freddy Celin Terán.

Inmaculada Consuegra Molina

Beiro Escobar Solaez

Adolfo Jiménez De La Cruz

Tutor Formativa I

Jorge Senior

Tutor Formativa II:

Juan Carlos Calabria

Tutor Formativa III:

Freddy Briceño

Tutor Formativa III:

Freddy Briceño

Tutor Asignado:

Néstor Alberto Torres Cuesta

5.2.2 Recursos Financieros

Gastos en el desarrollo de Formativa

Horas de Internet	$150.000
Copias	$80.000
Resma de papel	$60.000
Compra de CDS y disquetes	$30.000
TOTAL	$320.000

5.2.3 Recursos del Entorno

- **Hardware**
  - Un computador
  - Scanner
  - Impresora

- **Software**
  - C++

6. CAPÍTULOS

6.1 CAPITULO 1

Heurística y meta heurística

6.1.1 Objetivos del Capitulo

En este primer capitulo se comienza exponiendo los conceptos básicos para que las personas que acceda a este material puedan entender el concepto y el papel que juega la heurística y la meta heurística en la resolución de problemas de optimización y a la vez se familiaricen con los términos que en éstas se emplean.

También se pretende dar una pequeña introducción a las diferentes técnicas que hacen parte de las metaheuristicas y respectivamente profundizar en la que es de nuestro interés como los son los Algoritmos Genéticos, Colonia de Hormigas

6.1.2 Concepto de Heurística

Deriva del Griego Heuriskein, que significa encontrar o descubrir, es una técnica que busca las soluciones de buena calidad a un costo computacional razonable sin garantizar la optimalidad de las mismas.

La heurística trata de construir una solución eligiendo de manera interactiva los elementos y componentes de menor costo. Para algunos problemas con estructura particular, la solución construida es una solución óptima.

También según ANSI/IEEE Std 100-1984, la heurística trata de métodos o algoritmos exploratorios durante la resolución de problemas, en los cuales las soluciones se descubren por la evaluación del progreso logrado en la búsqueda de un resultado final.

Se puede definir también como:

Se suele usar actualmente como adjetivo, caracterizando técnicas por las cuales se mejora en promedio el resultado de una tarea resolutiva de problemas (parecido al uso de "método óptimo").

Se suele decir que hay búsquedas ciegas, búsquedas heurísticas (basadas en la experiencia) y búsquedas racionales (usando inteligencia)

Cambió varias veces de significado en la especialidad de inteligencia artificial, como:

- Estocástico
- Modelo de D'Alembert
- Resolución de problemas
- Búsqueda

Son métodos para resolver problemas complejos en forma aproximada, su complejidad computacional es reducida en relación a la de lo demás métodos exactos

Son flexibles y tienen como objetivo, encontrar soluciones de buena calidad en un tiempo computacional soportable, son métodos para resolver problemas complejos en forma aproximada. Su complejidad computacional es reducida en relación a los demás métodos exactos,

son flexibles y tiene como objetivo, encontrar soluciones de buena calidad en un tiempo computacional soportable.

6.1.3 Concepto de Meta heurística

Los métodos clásicos presentan problemas, entre ellos se encuentran que dichos métodos dependen de la naturaleza del espacio de soluciones, cuando el problema multiobjetivo es muy complejo estos métodos ya no son eficientes para encontrar una solución. Sin embargo existe una alternativa para solucionar los problemas de optimización multiobjetivo y en general, un problema de optimización. Esta alternativa se conoce como Meta Heurísticas.

En general, lo que buscan las meta heurísticas es una exploración "inteligente" del espacio de soluciones. Las meta heurísticas tienen la desventaja de que su resultado son puntos aproximados, es decir, es posible que teóricamente no sean las soluciones óptimas, aunque si se tiene un problema del tipo *NP*, como por ejemplo el problema del *Agente Viajero* o el problema de los *Arboles de Steiner*, el encontrar una muy buena solución es mejor a no tener ninguna.

Ésta facilita el desarrollo de métodos con una cierta rigidez matemática y con facilidades para incorporar nuevas situaciones. Logran superar algunas de las deficiencias presentes en los métodos heurísticos.

Se caracterizan además por ser estrategias que guían el proceso de búsqueda, incluyendo en general heurísticas subordinadas. Son de

uso genérico, no especificas para una clase de problemas, admiten descripción a nivel abstracto y deben instanciarse para cada clase de problemas.

Sus objetivos es encontrar soluciones factibles de buena calidad (valor cercano al óptimo), encontrar rápidamente soluciones factibles, pueden ser en si mismo un problema NP difícil y buscan reconocer el espacio de soluciones sin quedar "atrapados" en una zona. Noción de:

- Exploración del espacio.
- Explotación de las soluciones obtenidas.

Algunos problemas clásicos que se pueden resolver con las meta heurísticas son:

- Localización
- Problema del Agente Viajero,
- Recorrido de vértices (TSP)
- Problema del Cartero-recorrido de arcos
- Secuenciación
- Ruteamiento
- Distribución
- Redes de caminos

Este grupo de técnicas es la de mayor potencial para los problemas de empresas.

Muchas formulaciones de problemas reales usando estas técnicas generan modelos difíciles de resolver por los algoritmos clásicos de

solución (simplex, primal-dual, transporte, etc.), con una complejidad computacional de forma no polinomial (NP).

6.1.4 Necesidades de las Meta heurísticas

Una buena parte de los problemas tecno-científicos consiste en cuestiones de búsqueda, algunos de estos problemas poseen un elevado nivel de complejidad computacional (NP-arduos, NP complejos).

Encontrar una solución óptima en un tiempo computacional aceptable, para los problemas es, en muchos casos, posible solamente para tener instancias pequeñas.

A pesar del gran desarrollo teórico en el estudio de nuevos métodos para la solución de problemas complejos, este avance ha sido, en algunos casos, poco productivo en la solución de problemas reales. Los métodos de optimización exacta, se caracterizan por la rigidez de sus modelos matemáticos, dificultando la representación de situaciones reales cada vez más complejas y dinámicas.

El problema de esta falta de flexibilidad fue un poco, reducida, a partir del momento en que se paso a asociar técnicas de optimización con herramientas de inteligencia artificial, específicamente, las herramientas de búsqueda heurística.

6.1.5 Aspectos técnicos

Todo problema de optimización esta compuesto por un espacio de soluciones viables o espacio de búsqueda y una función a ser

optimizada. Cualquier algoritmo eficiente de optimización, debe usar dos técnicas para intentar encontrar un extremo global:

- La exploración para buscar en nuevas regiones del espacio de búsqueda.
- La exploración para hacer uso del conocimiento adquirido.

Estos métodos de búsqueda están inspirados en mecanismos usados por la naturaleza (evolución, adaptación, memoria, etc.) como principios inteligentes para la solución de problemas complejos. Esto incluye el proceso de crear estructuras para el uso ventajoso del conocimiento previo y el uso de este conocimiento para la obtención de mejores soluciones.

6.1.6 Procedimientos Meta heurísticos

A continuación se abordaran los procedimientos meta heurísticos mas utilizados y reconocidos en la optimización combinatoria, hablaremos de ellas someramente y profundizaremos en algoritmos genéticos y en Colonias de Hormigas, que es donde esta centrada nuestra atención e investigación.

Sin embargo antes de profundizar en las diferentes meta heurísticas se hablará de su clasificación que es la siguiente:

- **Inspiradas o no en la naturaleza (sistemas biológicos, físicos y sociales)**
  - Algoritmos Genéticos (evolución de las especies).

- Recocido simulado (enfriamiento de metales).
- Colonias de Hormigas.
- Búsqueda dispersa

- **Aleatorias Vs Deterministicas**
  - Algoritmos Genéticos, Recocido simulado, Colonias de Hormigas: aleatorias
  - Busqueda Tabù, Busqueda Dispersa: Deterministicas.

- **Basadas en un individuo Vs Basadas en Poblaciones**
  - Recocido Simulado, Búsqueda Tabú, Búsqueda Local y Variantes: Trayectorias.
  - GRASP, Colonias de Hormigas: Constructivas.

- **Con Memoria Vs sin Memoria**
  - Recocido Simulado, GRASP, Algoritmos Genéticos, Búsqueda local y Variantes: sin memoria.
  - Búsqueda Tabú, Colonias de Hormigas: con memoria.

UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA
BARRANQUILLA

Meta heurísticas	*Intensificación*	*Diversificación*
SA	Búsqueda local básica	Movimientos que empeoran la solución
TS	Búsqueda local básica	Lista Tabú (memoria a corto plazo)
ELS (GRASP)	Búsqueda local básica	Criterio de aceptación y perturbación
EC (GA)	Selección	Operadores de modificación (Ej.: mutación)
ANT (colonias de Hormigas)	Reglas de actualización de feromonas	Regla de construcción probabilísticas

Tabla 1. Comparación de las técnicas en intensificación y diversificación

Las más difundidas son la conocida como Búsqueda Local y Variantes (M/N/1), Recocido simulado (M/S/1), Búsqueda Tabú (A/N/1), GRASP (A/S/1), Algoritmos Genéticos (A/S/P) y Colonias de Hormigas (A/S/P).

6.1.6.1 Algoritmos Genéticos

Los Algoritmos Genéticos (AG) fueron introducidos por John Holland en 1970, inspirándose en el proceso observado en la evolución natural de los seres vivos.

Los biólogos han estudiado en profundidad los mecanismos de la evolución y aunque quedan parcelas por entender, muchos aspectos están bastante explicados. Podemos decir de manera muy general que en la evolución de los seres vivos el problema al que cada

individuo se enfrenta cada día es la supervivencia y para ello cuenta con las habilidades innatas provistas en su material genético.

Por tanto, la evolución tiene lugar en los cromosomas, en donde esta codificada la información el ser vivo, estas información varia de generación en generación en el proceso de formación de un nuevo individuo.

Los AG establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en un string (vector binario) a modo de cromosoma, es decir como dijo Holland:
"Se pueden encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un proceso de evolución simulada"

A tal efecto se introduce una función de evaluación de los cromosomas, que llamó calidad o fitness y que esta basada en la función objetivo del problema, al igual introduce un mecanismo de selección de manera que los cromosomas con mejor evaluación sean escogidos para reproducirse en mayor cantidad que aquellos que no cumplen con el objetivo de la función.

Los algoritmos primeramente desarrollados por Holland eran sencillos pero con buenos resultados en problemas que para ese entonces eran considerados difíciles. Los AG primordialmente tienen y están basados en dos ideas fundamentales: las representaciones simples como strings binarios de las soluciones del problema y la realización de las

trasformaciones simples para modificar y mejorar estas representaciones.

Esta técnica utiliza operadores como:

- De un punto: se elige aleatoria mente un punto de ruptura en los padres y se intercambian sus bits.
- De dos puntos: se eligen dos puntos de ruptura al azar para intercambiar.
- Uniforme: en cada bit se elige al azar un padre para que contribuya con su bit al del hijo, mientras que el segundo hijo recibe el bit del otro padre.
- PMX, SEX: son operadores mas sofisticados fruto de mezclar y aleatorizar los anteriores.

También utilizan el de mutación, que consiste en reemplazar con cierta probabilidades valor de un bit, su papel es el de introducir un factor de diversificación, pues muchas veces la convergencia del procedimiento hacia buenas soluciones es prematura y puede quedar atrapada en un optimo local.

Otra forma de introducir nuevos elementos en una población es recombinar elementos tomados al azar sin considerar su fitness.

Más adelante profundizaremos más en este tema que tiene muchas cosas para descubrir y poner en práctica.

6.1.6.2 Simulated Anneling (Temple Simulado)

Kirkpatrick, Gelatt, y Vecchi proponen en 1983 un procedimiento para obtener soluciones aproximadas a problemas de optimización, este procedimiento se basa en una analogía con el comportamiento de un sistema físico al someterlo a un baño de agua caliente. Simulated Annealing (SA) ha sido aprobado con éxito en numerosos problemas de optimización, mostrando gran habilidad para evitar quedar atrapado en óptimos locales. Debido a su sencillez de implementación así como de los buenos resultados que iban apareciendo, experimentó un gran auge en la década de los 80.

Es así, como SA es un procedimiento basado en búsqueda local en donde todo movimiento de mejora es aceptado y se permiten movimientos de no mejoras de acuerdo con unas probabilidades, dichas probabilidades están basadas en analogías con el proceso físico de enfriamiento y se obtienen como función de la temperatura del sistema.

En la actualidad se están diseñando numerosos algoritmos híbridos en donde la búsqueda local se realiza con un procedimiento basado en SA.

6.1.6.3 Tabú Search

Los orígenes de esta técnica son de los 70, su forma actual fue presentada por Glover en 1986, y los primeros modelos teóricos son de 1992. No se conoce ninguna prueba clara de la convergencia pero la técnica ha mostrado una eficacia notable en muchos problemas.

Tabú Search es una técnica para resolver problemas combinatorios de gran dificultad que está basada en principios generales de Inteligencia Artificial. En esencia es un meta heurística que puede ser utilizada para guiar cualquier procedimiento de búsqueda local en búsqueda agresiva del óptimo del problema, es decir, la estrategia de evitar que la búsqueda quede atrapada por así decir en un optimo local que no sea global, por tal razón, esta técnica toma el concepto de memoria de la inteligencia artificial y lo pone a funcionar mediante estructuras simples con el objetivo de dirigir la búsqueda, sin olvidar la secuencia e historia de ésta.

Este algoritmo se caracteriza, por tanto, de una "memoria" en la cual se pueden almacenar los últimos movimientos realizados, y que puede ser utilizada para "recordar" aquellos movimientos que hacen caer de nuevo en soluciones ya exploradas.

A continuación presentaremos la estructura del algoritmo utilizado para realizar la búsqueda:

```
LT <-- {0} {lista tabú vacía al inicio}
x* <-- x0 {solución inicial}
F* <-- F(x0) {valor óptimo inicial}
Q MIENTRAS ((n_iter_total - n_iter_cambio) < N_MAX_ITER) HACER
        * {si nº de iteraciones sin mejorar es menor que un máximo}
        * n_iter_total <-- n_iter_total+ 1 {nº total de iteraciones}
        * xn <-- x* {solución actual}
        * F' <-- 4 {coste mínimo de solución cercana}
        * +Q PARA Iteracion=1 HASTA Card(V(xn)) HACER
            {para todas las cercanas}
            * * x <-- V(xn) {se genera solución cercana}
            * * +Q SI ( F(x)<F' Y ( [xn->x]óLT O ([xn->x]0LT Y F(x)<Fa) ) )
            ENTONCES
                    * * * {si es la de menor coste y no es tabu o siendo tabu
                    es aceptada}
                    * * * x' <-- x {mejor solución cercana}
                    * * * F' <-- F(x) {coste mínimo cercano}
            * * .Q FINSI
        * .Q FINPARA
    * xn+1 <-- x' {mejor solución cercana}
    * LT <-- LT + [xn+1-->xn] - [1°_si_llena] {incorpora como tabú}
    * +Q SI F'< F* ENTONCES {si la mejor solución cercana es mínima}
            * * x* <-- x' {nueva solución óptima}
            * * F* <-- F' {nuevo valor óptimo}
    * .Q FINSI
Q FINMIENTRAS
Imprimir_Optimo(x*) {se llega a la mejor}
```

Figura 1. Algoritmo de tabú para realizar la búsqueda

Si deseas tener mas información y profundizar más en este tema puedes acceder a las siguientes direcciones y disipar tus dudas si este tema de interesa en profundización:

- http://www.fiec.espol.edu.ec/investigacion/topico/tabusearch.pdf
- http://tornado.dia.fi.upm.es/caepia/numeros/19/glover.pdf
- http://www.azc.uam.mx/publicaciones/enlinea2/num1/1-3.htm#direcmag
- http://www.cs.sandia.gov/opt/survey/ts.html
- http://w3.mor.itesm.mx/~emorales/Cursos/Busqueda04/node68.html
- http://www.terra.es/personal/faustopedro.gar/files/A5.pdf

6.1.6.4 GRASP (Greedy Randomized Adaptive Search Procedures)

La palabra GRASP proviene de las siglas de **Greedy Randomized Adaptive Search Procedures** que en nuestra lengua quedaría así: Procedimientos de Búsqueda Basados en Funciones "Greedy" Aleatorizadas Adaptativas.

Los métodos GRASP fueron desarrollados al final de la década de los 80 con el objetivo inicial de resolver problemas de cubrimientos de conjuntos (Feo y Resende, 1989). El termino GRASP fue introducido por Feo y Resende como una nueva técnica mata heurística de propósito general.

GRASP es un procedimiento iterativo en donde cada paso consiste en una fase de construcción y una de mejora. En la fase de construcción se aplica un procedimiento heurístico constructivo para obtener una solución inicial. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local. La mejor de todas las soluciones examinadas se guarda como resultado final.

En la fase de construcción se construye interactivamente una solución posible, considerando un elemento en cada paso. En cada iteración la elección del próximo elemento para ser añadido a la solución parcial viene determinada por una función greedy. Esta función mide el beneficio de añadir cada uno de los elementos según la función objetivo y elegir la mejor.

En la fase de mejora se suele utilizar un procedimiento de intercambio simple con el objeto de no emplear mucho tiempo en esta mejora.

De esta forma funciona el algoritmo, atizando una condición de parada:

Fase constructiva

1. Seleccionar una lista de elementos candidatos
2. Considerar una lista restringida de los mejores candidatos
3. Seleccionar un elemento aleatoria mente de la lista restringida

Fase de mejora

1. Realizar un proceso de búsqueda local a partir de la solución construida hasta que no pueda mejorar mas
2. Actualización, es decir, si la solución obtenida mejora a la solución almacenada, actualizarla.

El realizar muchas iteraciones GRASP es una forma de realizar un muestreo del espacio de soluciones, sus implementaciones son generalmente robustas en el sentido que es difícil el encontrar ejemplos patológicos en donde el método funcione arbitrariamente mal.

Una de las características más relevantes de GRASP es su sencillez y facilidad de implementación. Basta con fijar el tamaño de la lista de candidatos y el número de iteraciones para determinar completamente el procedimiento. De esta forma se pueden concentrar los esfuerzos

en diseñar estructuras de datos para optimizar la eficiencia del código y proporcionar una gran rapidez del algoritmo; dado que este es uno de los objetivos principales del método.

Para mejor y mayor información puedes entrar en Internet y aclarar todas las dudas.

6.2 CAPITULO 2

Reseña Histórica

6.2.1 Objetivo del Capitulo

En este capitulo estudiaremos el origen y nacimiento de los algoritmos genéticos, cuando deben ser utilizados y en que problema nos pueden ayudar para buscar la mejor solución.

También conocer como el hombre a lo largo de su vida y trayectoria por el mundo va ideando la manera de como simular y crear de forma artificial los procesos que una vez le dieron vida a la humanidad.

6.2.2 Teoría de la Evolución

La teoría de la evolución, fue descrita por Charles Darwin. La hipótesis de Darwin, presentada junto con Wallace, que llegó a las mismas conclusiones independientemente, es que pequeños cambios heredables en los seres vivos y la selección son los dos hechos que provocan el cambio en la naturaleza y la generación de nuevas especies. Pero Darwin desconocía cual es la base de la herencia, pensaba que los rasgos de un ser vivo eran como un fluido, y que los "fluidos" de los dos padres se mezclaban en la descendencia; esta hipótesis tenía el problema de que al cabo de cierto tiempo, una población tendría los mismos rasgos intermedios.

Fue Mendel quien descubrió que los caracteres se heredaban de forma discreta, y que se tomaban del padre o de la madre,

UNIVERSIDAD SIMON BOLIVAR
HEMEROTECA
BARRANQUILLA

dependiendo de su carácter dominante o recesivo. A estos caracteres que podrían tomar diferentes valores se les llamaron genes, y a los valores que podrían tomar, alelos. En realidad, las teorías de Mendel, que trabajó en total aislamiento, se olvidaron y no se volvieron a redescubrir hasta principios del siglo XX.

Además, hasta 1930 el geneticista inglés Robert Aylmer no relacionó ambas teorías, demostrando que los genes mendelianos eran los que proporcionaban el mecanismo necesario para la evolución.

Más o menos por la misma época, el biólogo alemán Walther Flemming describió los cromosomas, como ciertos filamentos en los que se agregaba la cromatina del núcleo celular durante la división; poco mas adelante se descubrió que las células de cada especie viviente tenían un número fijo y característico de cromosomas. Y no fue hasta los años 50, cuando Watson y Crick descubrieron que la base molecular de los genes está en el ADN, ácido desoxirribonucleico. Los cromosomas están compuestos de ADN, y por tanto los genes están en los cromosomas.

Proyectos como el del Genoma Humano tratan de identificar cuales son estos genes, sus posiciones, y sus posibles alteraciones, que habitualmente conducen a enfermedades.

Todos estos hechos forman hoy en día la teoría del *neo-darwinismo*, que afirma que la historia de la mayoría de la vida está causada por una serie de procesos que actúan en y dentro de las poblaciones: reproducción, mutación, competición y selección.

La evolución se puede definir entonces como cambios en el *pool* o conjunto genético de una población.

Según los informáticos evolutivos, la evolución optimiza, puesto que va creando seres cada vez mas perfectos, cuya cumbre es el hombre; además, indicios de esta optimización se encuentran en el organismo de los animales, desde el tamaño y tasa de ramificación de las arterias, diseñada para maximizar flujo, hasta el metabolismo, que optimiza la cantidad de energía extraída de los alimentos.

Sin embargo, los geneticistas y biólogos evolutivos afirman que la evolución no optimiza, sino que adapta y optimiza localmente en el espacio y el tiempo; evolución no significa progreso. Un organismo más evolucionado puede estar en desventaja competitiva con uno de sus antepasados, si se colocan en el ambiente del último.

6.2.3 Nacimiento de los Algoritmos Genéticos

Los algoritmos genéticos son una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado un gran interés durante los últimos años por la comunidad científica alrededor del mundo.

La comunidad científica ha demostrado un gran interés en una nueva técnica de búsqueda basada en la teoría de la evolución y hoy se conoce como Algoritmos Genéticos, esta técnica se basa en los mecanismos de selección que utiliza la naturaleza, de acuerdo a los cuales los individuos más aptos de una población son los que sobreviven, al adaptarse mas fácilmente a los cambios que se

producen en su entorno. Hoy en día se sabe que estos cambios se efectúan en los genes de un individuo (unidad básica de codificación de cada uno de los atributos de un ser vivo), y que sus atributos mas deseables (los que le permiten adaptarse mejor a su entorno) se transmiten a sus descendientes cuando éste se reproduce sexualmente.

Los principios básicos de los Algoritmos Genéticos fueron establecidos por Holland (1975), y se encuentran bien descritos en varios textos Goldberg (1989), Davis (1991), Michalewicz (1992), Reeves (1993). Holland era consciente de la importancia de la selección natural, y a fines de los años 60 desarrolló una técnica que permitió incorporarla a un programa.

John Holland desde pequeño, se preguntaba como logra la naturaleza, crear seres cada vez mas perfectos. Lo curioso era que todo se lleva a cabo a base de interacciones locales entre individuos, y entre estos lo que les rodea. No sabía la respuesta, pero tenía una cierta idea de como hallarla: tratando de hacer pequeños modelos de la naturaleza, que tuvieran alguna de sus características, y ver como funcionaban, para luego extrapolar sus conclusiones a la totalidad.

En los años 50 entro en contacto con los primeros ordenadores, donde pudo llevar a cabo algunas de sus ideas, aunque no se encontró con un ambiente intelectual fértil para propagarlas.

Fue a principios de los 60, en la Universidad de Michigan en Ann Arbor, donde, dentro del grupo Logic of Computers, sus ideas comenzaron a desarrollarse y a dar frutos, lo ayudo, además, leer un

libro escrito por un biólogo evolucionista, R. A. Fisher, titulado "La teoría genética de la selección natural", y de ahí comenzó a descubrir los medios para llevar a cabo sus propósitos de comprensión de la naturaleza. De ese libro aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado.

En esa universidad, Holland impartía un curso titulado Teoría de sistemas adaptativos. Dentro de este curso, y con una participación activa por parte de sus estudiantes, fue donde se crearon las ideas que mas tarde se convertirían en los Algoritmos Genéticos.

Por tanto, cuando Holland se enfrentó a los algoritmos genéticos, los objetivos de su investigación fueron dos:

- Imitar los procesos adaptativos de los sistemas naturales

- Diseñar sistemas artificiales (normalmente programas) que retengan y modelen los mecanismos importantes de los sistemas naturales.

Unos 15 años más adelante, David Goldberg conoció a Holland, y se convirtió en su estudiante. Golberg era un ingeniero industrial que trabajaba en diseño de pipelines, y fue uno de los primeros que trató de aplicar los AGs a problemas industriales. Aunque Holland trató de disuadirle, porque pensaba que el problema era excesivamente

complicado como para aplicarle AGs, Goldberg consiguió lo que quería, escribiendo un AG en un ordenador personal Apple II.

Estas y otras aplicaciones creadas por estudiantes de Holland convirtieron a los AGs en un campo con base suficiente aceptado para celebrar la primera conferencia en 1985, ICGA'85. Tal conferencia se sigue celebrando bianualmente.

Su objetivo era lograr que las computadoras aprendieran por si mismas. A la técnica que inventó Holland se le llamo originalmente "planes reproductivos", pero se hizo popular bajo el nombre "algoritmo genético" tras la publicación de su libro en 1975.

6.2.4 Utilización de un Algoritmo Genético

La aplicación mas común de los AGs ha sido la solución de problemas de optimización, en donde han demostrado ser muy eficientes y confiables, pero sin embargo no todos los problemas pueden ser resueltos con esta técnica y se recomienda de forma general a todos tener en cuenta características tales como:

- Su especio de búsqueda (n,m,... posibles soluciones) debe estar limitado dentro de un rango.
- La función aptitud debe definirse de tal modo que indique que tan buena o mala es una cierta respuesta.
- La codificación de las soluciones posibles al problema deben codificarse de tal forma que se puedan implementar en la computadora.

El primer punto es muy importante y se recomienda intentar resolver problemas que tengan espacios de búsquedas discretos, aunque estos sean muy grandes. También se podría utilizar cuando el espacio de búsqueda sea continuo, pero se recomienda que el rango de soluciones sea relativamente pequeño.

Cuando hablamos de función aptitud estamos haciendo referencia a la función objetivo o al objetivo que debe cumplir un individuo para que pueda hallarse la solución al problema de optimización si es el caso.

Los algoritmos genéticos maximizan, pero también la minimización puede utilizarse fácilmente utilizando el reciproco de la función maximizante (teniendo en cuenta que la función no genere una división por cero) y debe tener como característica fundamental premiar y castigar a las soluciones.

Para la parte de la codificación, la más común es a través de cadenas binarias, aunque muchas veces se han utilizado letras y números reales y ha tenido gran popularidad el de codificación binaria ya que fue el que se propuso originalmente.

6.3 CAPITULO 3

Fundamentos Teóricos

6.3.1 Objetivo del Capitulo

En este capitulo se mostrarán las bases y fundamentos que son necesarios para saber lo que realmente son los AGs, como se codifican, que terminología utilizan y que se debe tener en cuenta para utilizarlos.

También se busca que las personas que accedan a este pueden entender y poner en uso y practica estos conceptos.

6.3.2 Estructura de un Algoritmo Genético

Un algoritmo genético simple o básico comprende cinco pasos principales a saber:

- Crear población inicial de cromosomas, ya sea en forma aleatoria o mediante cierto conocimiento inherente al problema en sí. Es muy importante que la población inicial tenga diversidad para que el algoritmo pueda explorar todo el espacio de soluciones, y se espera que se cumpla dadas las características de la inicialización. Además, si se tiene conocimiento del problema, éste puede ser insertado en la población inicial, con lo que se espera conseguir un mejor tiempo de convergencia del algoritmo.

- Evaluación, consiste en calcular la aptitud mediante una función definida para el problema en particular. El objetivo de esta función es proveer una medida numérica de la calidad de un cromosoma dado.

- Selección o explotación, se escogen de forma aleatoria los cromosomas con mejor función de aptitud unas varias veces y se insertan en un depósito de cruza.

- Exploración, lo que consiste en la aplicación de operadores de recombinación y mutación, con lo que se obtienen dos hijos los que según el tipo de algoritmo usado, reemplazarán a sus padres en la siguiente generación.

- La población es llenada con los nuevos cromosomas creados, se repiten estos pasos hasta que el criterio de parada se cumpla, con el un número de generaciones determinada.

También debemos tener en cuenta los siguientes términos pues nos serán de mucha ayuda a la hora de diseñar el algoritmo:

- **Genoma**: todos los parámetros que definen a todos los individuos de la población.

- **Genotipo:** descripción de un individuo, la secuencia de bits.

- **Fenotipo:** expresión de un genotipo, es decir la ejecución o conjunto de parámetros decodificados.

- **Gen:** cada uno de los parámetros del genotipo.

- **Cromosoma:** en los sistemas biológicos se combinan para formar la prescripción genética total de los nuevos organismos; y al paquete total se le llama genotipo. Están compuestos por genes, que pueden tomar un cierto número de valores llamados alelos.

6.3.3 Codificación de datos de un Algoritmo genético

Para conseguir que el ordenador procese los datos es necesario representarlos de una forma apropiada, en primer lugar para codificar los datos es necesario separar las posibles configuraciones del dominio del problema en un conjunto de estados.

Una ves obtenida la clasificación, el objetivo es representar cada estado de forma unívoca con una cadena de caracteres compuesta (0's y 1's), aunque cada estado puede codificarse con alfabetos de diferentes cardinalidades, sin embargo uno de los resultados fundamentales de la teoría de algoritmos genéticos (AGs) es el *Teorema Del Esquema*, que afirma que la codificación óptima es aquella en la que los algoritmos tienen un alfabeto de cardinalidad.

> ***Teorema del Esquema: Esquemas cortos, de bajo orden y aptitud superior al promedio reciben un incremento exponencial de representantes en generaciones subsecuentes de un AG. Ejemplo representativo seria:***

Atributo 1	Atributo 2	Atributo 3
101001	010100	000001

Ejemplo común de codificación y más utilizado para poder representar tal codificación es:

Si un atributo "tiempo" puede tomar tres valores posibles –despejado, nublado, lluvioso– una manera de representarlo es mediante tres bits de forma que: (Tiempo = Nublado ó Lluvioso) y (Viento = Fuerte) se representaría con la siguiente cadena: 01110

De esta forma podemos representar fácilmente conjunciones de varios atributos para expresar restricciones, precondiciones mediante la concatenación de dichas cadenas de bits.

Además si tenemos otro atributo "Viento" que puede ser Fuerte o Moderado, se representaría con la siguiente cadena: 01110

6.3.4 Procedimiento (seudo código) de un Algoritmo Genético Simple

```
BEGIN /* Algoritmo Genético Simple */
        Generar una población inicial.
        Computar la función de evaluación de cada individuo.
  WHILE NOT Terminado DO
  BEGIN /* Producir nueva generación */
  FOR Tamaño población/2 DO
  BEGIN /*Ciclo Reproductivo */
        Seleccionar dos individuos de la anterior generación, para el cruce
        (probabilidad de selección proporcional a la función de evaluación
        del individuo).
        Cruzar con cierta probabilidad los dos individuos obteniendo dos
        descendientes.
        Mutar los dos descendientes con cierta probabilidad.
```

UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA

```
            Computar la función de evaluación de los dos descendientes
            mutados.
            Insertar los dos descendientes mutados en la nueva generación.
        END
        IF la población ha convergido THEN
            Terminado:= TRUE
        END
    END
```

Se supone que los individuos (posibles soluciones del problema), pueden representarse como un conjunto de parámetros (que denominaremos genes), los cuales agrupados forman una ristra de valores (a menudo referida como cromosoma).

Si bien el alfabeto utilizado para representar los individuos no debe necesariamente estar constituido por el {0, 1}, buena parte de la teoría en la que se fundamentan los Algoritmos Genéticos utiliza dicho alfabeto.

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina *fenotipo.* El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como *genotipo.* Los mismos términos se utilizan en el campo de los Algoritmos Genéticos. La adaptación al problema de un individuo depende de la evaluación del *genotipo.* Esta última puede inferirse a partir del *fenotipo,* es decir puede ser computada a partir del cromosoma, usando la función de evaluación.

También podemos observar el siguiente grafico que muestra un bucle básico de un Algoritmo Genético a saber:

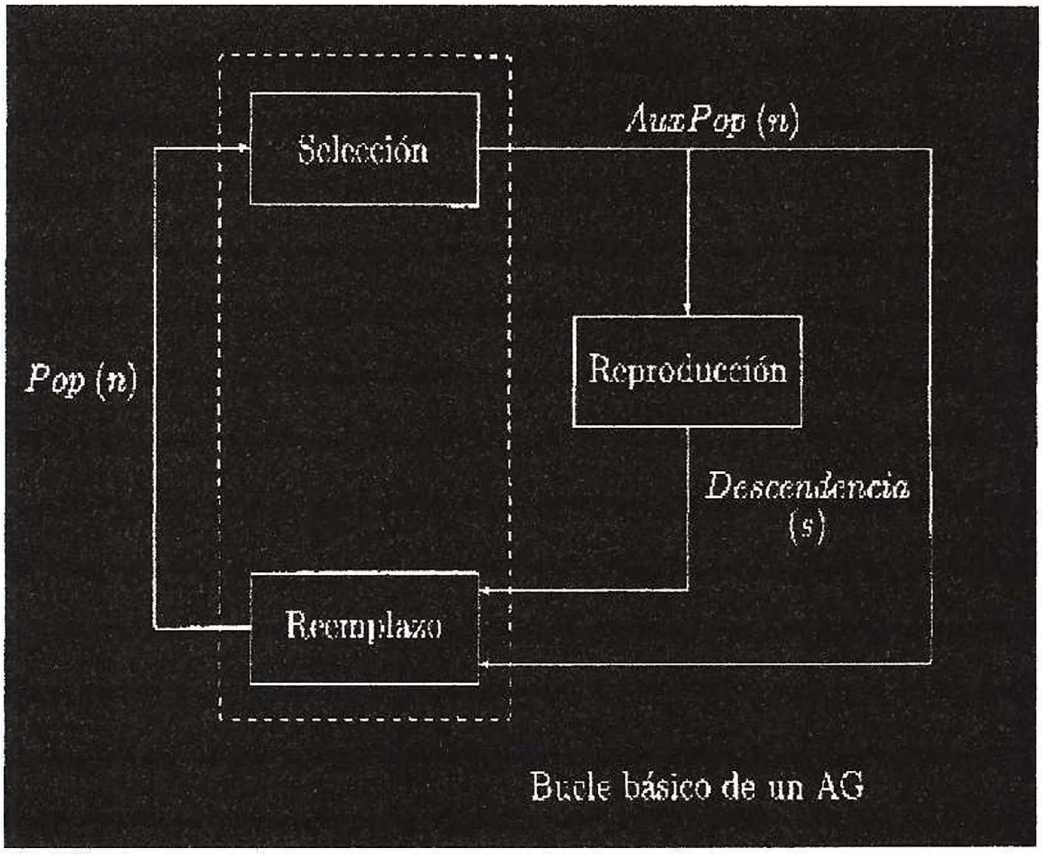


Figura 2. Bucle básico de un algoritmo genético

El proceso se describe como sigue: una población **Pop**, que consta de *n* miembros se somete a un proceso de **selección** para constituir una población intermedia **AuxPop** de *n* criadores. De dicha población intermedia se extrae un grupo reducido de individuos llamados **progenitores** que son los que efectivamente se van a reproducir. Sirviéndose de los operadores genéticos, los progenitores son sometidos a ciertas transformaciones de **alteración** y **recombinación** en la fase de reproducción, en virtud de las cuales se generan *s* nuevos individuos que constituyen la **Descendencia**.

Para formar la nueva **Pop [t+1]**, se deben seleccionar *n* supervivientes de entre los *n+s* de la población auxiliar y la descendencia, lo que ocurre en la fase de **reemplazo**. El cuadrado rayado hace referencia a la selección, la cual se realiza en dos etapas con la idea de emular las dos vertientes del Principio de Selección Natural: selección de craidores o selección a secas, y selección de supervivientes para la próxima generación o reemplazo.

El proceso descrito, puede ser expresado de forma algorítmica del siguiente modo:

t = 0
Inicializar Población (t)
Evaluar Población (t)
Mientras (nos se verifique la condición de parada) hacer

t = t +1
Seleccionar Población (t) a partir de Población (t-1)
Recombinar Población (t)
Evaluar Población (t)

FinMientras

Los pasos anteriores mostrados en el esquema pueden explicarse de la siguiente manera:

- **Inicialización de la Población**

  El primer paso es inicializar la población de origen. La mayoría de los casos la inicialización de hace de forma aleatoria para guardar una homogenización en los casos iniciales de prueba.

- **Evaluar Población**

  Durante cada iteración (generación) cada gen se decodifica convirtiéndose en un grupo de parámetros del problema y se evalúa el problema con esos datos.

  No en todas las implementaciones de AGs se realiza una fase de evaluación de población tal y como está descrita acá, en ciertas ocasiones se omite y no se genera ningún *fitness* asociado a cada estado evaluado.

- **Seleccionar la Población**

  La fase de selección elige los estados a reproducirse en la próxima generación, esta selección puede realizarse por distintos métodos (mas adelante serán ampliados los conceptos de acuerdo al los tipos de selección manejadas en los AGs).

- **Recombinar población (cross-over)**

  También llamado recombinación. El *Cross-over* es el operador genético mas utilizado y consiste en el intercambio de material genético entre dos cromosomas al azar (puede ser incluso entre el mismo elemento). Es el operador genético principal, hasta el punto que se puede decir que no es un AG sino tiene cross-over, y sin embargo puede serlo sin mutación, según descubrió Holland.

6.3.5 Parámetros utilizados para aplicar un Algoritmo Genético

- **Tamaño de la población:** Determina el tamaño máximo de la población a obtener. En la práctica debe ser de un valor lo suficientemente grande para permitir diversidad de soluciones e intentar llegar a una buena solución, pero siendo un número que sea computable en un tiempo razonable.

- **Condición de terminación:** Es la condición de parada del algoritmo. Habitualmente es la convergencia de la solución (si es que la hay), un número prefijado de generaciones o una aproximación a la solución con un cierto margen de error.

- **Individuos que intervienen en la reproducción de cada generación:** se especifica el porcentaje de individuos de la población total que formarían parte del acervo de padres de la siguiente generación. Esta proporción es denominada proporción de cruces.

- **Probabilidad de ocurrencia de una mutación:** En toda ejecución de un algoritmo genético hay que decidir con qué frecuencia se va a aplicar la mutación. Se debe de añadir algún parámetro adicional, que indique con qué frecuencia se va a aplicar dentro de cada gen del cromosoma. La frecuencia de aplicación de cada operador estaría en función del problema; teniendo en cuenta los efectos de cada operador, tendría que aplicarse con cierta frecuencia o no. Generalmente, la mutación y otros operadores que generen diversidad se suele aplicar con

poca frecuencia; la recombinación se suele aplicar con frecuencia alta.

6.3.6 Teorema del Esquema

Es interesante preguntarnos si podría caracterizarse de forma matemática la evolución de la población en un AG a lo largo del tiempo. El teorema de los esquemas (*Schema Theorem*) desarrollado por Holland (1975) nos proporciona esta caracterización. Se basa en el concepto de *esquemas*, o patrones que describen conjuntos de cadenas. De manera mas precisa, un esquema es cualquier cadena compuesta de 0's, 1's y *'s (suponiendo que estamos en el ámbito de un alfabeto de población binario). Cada esquema representa el conjunto de cadenas que contiene los 0's y 1's indicados (en las posiciones indicadas), interpretando cada * como un "no importa", es decir, un símbolo comodín.

Por ejemplo, el esquema dado por 0 * 1 1 0 * tiene las instancias siguientes:

0 0 1 1 0 0
0 0 1 1 0 1
0 1 1 1 0 0
0 1 1 1 0 1

Un esquema de longitud *l* tendría 2*l* realizaciones y un esquema con *k* comodines tendría 2 realizaciones. Cada cadena de bits puede verse como un representante de cada uno de los distintos esquemas que satisface. Por ejemplo, la cadena de bits 0010 podemos verla como

instancia de 24 esquemas distintos, entre los que se incluyen 00 * *, 0 * 10, * * * *, etc..

Con una población de *n* cadenas de longitud *l*, *n*2*l* esquemas pueden estar presentes en una población, y son transformados por los operadores genéticos del AG. Esto supone que una enorme cantidad de información es procesada en cada generación, incluso si el número de cadenas de la población es relativamente pequeño. Este aspecto de los AGs, el cual es frecuentemente visto como el secreto de su éxito, es denominado *paralelismo implícito*.

El teorema fundamental caracteriza la evolución de la población dentro de un AG con base al número de instancias que representan a cada esquema. La evolución de la población cuando se usa un AG depende de como se realicen los pasos de selección, cruce y mutación de los cromosomas.

La importancia de este teorema radica en que nos da la explicación matemática del hecho de que los algoritmos genéticos provocan la supervivencia de las hipótesis mas aptas a lo largo de las sucesivas generaciones de individuos, dando una cota del número esperado de instancias en una generación de un determinado esquema, en función de la aptitud de las hipótesis que lo representaban en la generación anterior.

El teorema del esquema es con frecuencia interpretado como sigue: *(Los esquemas pequeños de bajo orden arriba del promedio reciben un incremento exponencial de representantes en las siguientes generaciones de un algoritmo genético).*

El Teorema de los Esquemas dado es un límite inferior debido a que sólo hemos contado con los efectos destructivos de los operadores. Sin embargo, el cruzamiento es la mayor fuente de poder de los AG, con la habilidad de recombinar instancias de buenos esquemas para formar instancias de esquemas igualmente buenos o mejores. En la evaluación de una población de n cadenas, el AG está implícitamente evaluando o estimando una media de la aptitud de todos los esquemas que están presentes en la población, e incrementando o decrementando su representación acorde con el Teorema de los Esquemas. Para *Holland*, el propósito de la mutación era evitar la pérdida de la diversidad en los bits de las cadenas.

El teorema de los esquemas es quizá la caracterización mas citada de la evolución de la población dentro de un AG. No obstante, un aspecto en el que es incompleto es el hecho de que no considera los (presumibles) efectos positivos del cruce genético y de la mutación. Han sido propuestos posteriormente numerosos análisis teóricos, incluyendo análisis basado en cadenas de Markov y análisis basado en modelos mecánicos estadísticos. Cosultar, por ejemplo, *Whitley y Vose (1995)* y *Mitchell (1996)*.

6.3.7 Convergencia de un Algoritmo Genético

Dado que el AG opera con una población en cada iteración, se espera que el método converja de modo que al final del proceso la población sea muy similar, y en el infinito se reduzca a una única instancia.

Se ha desarrollado toda una teoría para estudiar la convergencia de estos algoritmos en el caso de cadenas binarias. Esta teoría se basa

principalmente en considerar que una cadena es un representante de una clase de equivalencia o esquema, reinterpretando la búsqueda en lugar de entre cadenas, entre esquemas. De este modo se concluye lo que se conoce como paralelismo intrínseco:

> *En una población de m cadenas se están procesando implícitamente O (m3) esquemas.*

A partir de este resultado el teorema de esquemas (o teorema fundamental), prueba que la población converge a unos esquemas que cada vez son más parecidos, y en el límite a una única cadena.

En el caso de cadenas no binarias se introducen los conceptos de *forma* y *conjunto de similitud* que generalizan al de esquema. Se consideran una serie de condiciones sobre los operadores de manera que se garantice la convergencia. Básicamente se exige que al cruzar dos cadenas de la misma clase se obtenga otro dentro de ésta.

Además hay que respetar ciertas condiciones sobre selección de los progenitores. Bajo toda esta serie de hipótesis se prueba la convergencia del algoritmo.

En la práctica no se suelen respetar las condiciones vistas ya que son difíciles de seguir y probar, encontrándonos con que, en ocasiones los algoritmos genéticos resuelven satisfactoriamente un problema de dado y otras se quedan muy alejadas del óptimo. Los estudiosos del tema han tratado de caracterizar lo que han denominado problemas AG-fáciles (aquellos en los que los AG proporcionan buenos resultados) y AG-difíciles con el objetivo de saber de antemano, al

estudiar un nuevo problema, si los AG son una buena elección para su resolución.

Se han tratado de caracterizar estas clases mediante el concepto de *engaño* considerando que si el algoritmo converge al mejor esquema que aquel con mejor promedio del *fitness* de sus cadenas y en éste se encuentra el óptimo, entonces es fácil que se resuelva satisfactoriamente.

En caso de que el óptimo esté en un esquema con bajo promedio se denomina engaño y se pensaba que en estos casos es cuando el problema es AG-difícil. Sin embargo se ha visto que esta caracterización mediante el engaño no es siempre cierta y no constituye un criterio fiable.

Es importante citar que, a diferencia de otros meta heurísticos, los Algoritmos Genéticos han crecido de forma espectacular, hasta el punto de poder encontrar referencias sobre ellos en revistas de informática de carácter general. Además muchos de los investigadores de este campo están trabajando en desarrollar los aspectos teóricos de la materia incorporando algunas otras técnicas de búsqueda local en el esquema genético.

Especial mención merecen los Algoritmos Miméticos en donde podemos encontrar una clase amplia de Meta heurísticos basados en poblaciones. Estos métodos incorporan técnicas variadas desde "búsqueda local" hasta operadores de "cruzamiento" con el fin de introducir información y conocimiento en la búsqueda.

UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA
BARRANQUILLA

6.4 CAPITULO 4

Operadores Genéticos

6.4.1 Objetivo del Capitulo

Al desarrollar este capitulo se pretende mostrar y hacer que las personas que accedan a este material puedan entender y aplicar cada uno de los operadores que son necesarios para resolver un AG y como estos ayudan a encontrar la mejor solución al presentarse un problema.

6.4.2 Operador selección

La fase de selección elige los estados a reproducirse en la próxima generación, esta selección puede realizarse por distintos métodos.

6.4.2.1 Selección Directa

Toma los elementos de acuerdo a un criterio objetivo, como lo son <<los X mejores>>, <<los X peores>>, los del tipo << el cuarto individuo a partir del ultimo escogido>> son empleados con mucha frecuencia cuando se quiere seleccionar dos individuos distintos y se selecciona el primero por un método aleatorio o estocástico.

6.4.2.2 Selección Aleatoria

Puede ser realizada por selección equiparable o selección estocástica:

6.4.2.2.1 Selección Equiparable

Todos tienen la misma probabilidad de ser escogidos.

6.4.2.2.2 Selección Estocástica

La probabilidad que un individuo sea escogido depende de una heurística, los distintos métodos estocásticos serian:

6.4.2.2.2.1 Selección Elitista

Se llama elitismo al proceso por el cual se determinan elementos con una adaptación especialmente buena, tienen determinados privilegios, nunca mueren, proporción alta de pasos en que se produce uno de la elite con el otro al azar... sin embargo, en fases iniciales es peligroso, ya que pueden producirse que una elite de súper individuos acabe con la diversidad genética del problema.

Para ello lo que se puede hacer es escalar la función de adaptación en las primera fases del AG, de forma que las diferencias entre la elite y el pueblo sean menores y súper escalar la función de adaptación al final del AG para evitar un bloqueo de la convergencia.

Garantiza la selección de los miembros más aptos de cada generación. (La mayoría de los AGs no utilizan elitismo puro, sino que usan una forma modificada por la que el individuo mejor, o algunos de los mejores, son copiados hacia la siguiente generación en caso de que no surja nada mejor). La

forma de seleccionar los mejores individuos es mediante el *fitness*.

6.4.2.2.2.2 Selección Proporcional a la aptitud

Los individuos más aptos tienen más probabilidad de ser seleccionados, asignándoles una probabilidad de selección más alta. Una vez seleccionadas las probabilidades de selección a cada uno de los individuos se genera una nueva población teniendo en cuenta éstas.

6.4.2.2.2.3 Selección por Rueda de Ruleta

Es el usado por Goldberg en su libro "Genetic Algorithms in Search, Optimization and Machina Learning", este método es muy simple consiste en crear una ruleta en la que cada cromosoma tiene asignada una fracción proporcional a su aptitud.

Es un método conceptualmente similar al anterior. Se le asigna una probabilidad absoluta de aparición de cada individuo de acuerdo al *fitness* de forma que ocupe un tramo del tramo total de probabilidad (de 0 a 1) de forma acorde a su *fitness*. Una vez completado el tramo total se generan números aleatorios de 0 a 1 de forma que se seleccionen los individuos que serán base de cultivo de la siguiente generación.

6.4.2.2.2.4 Selección por Torneo

la idea de este método es muy simple, se baraja la población y después se hace competir a los cromosomas que la integran en grupos de tamaño predefinido (normalmente en parejas) en un torneo que resultaran ganadores aquellos que tengan valores de aptitud mas altos. Si se produce un torneo binario, entonces la población se debe barajar 2 veces.

6.4.2.2.2.5 Selección por Rango

A cada individuo de la población se le asigna un rango numérico basado en su *fitness*, y la selección se basa en este ranking, en lugar de las diferencias absolutas en el *fitness*. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.

6.4.2.2.2.6 Selección Generacional

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

6.4.2.2.2.7 Selección por Estado Estacionario

La descendencia de los individuos seleccionados en cada generación vuelve al acervo genético preexistente,

reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.

6.4.2.2.2.8 Selección de Búsqueda del Estado Estacionario

Se ordenan todos los genes por su *fitness* en orden decreciente y se eliminan los últimos *m* genes, que se sustituyen por otros *m* descendientes de los demás. Este método tiende a estabilizarse y converger.

6.4.2.2.2.9 Selección Jerárquica

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

6.4.3 Operador Cross-Over

También llamado recombinación. El *Cross-over* es el operador genético mas utilizado y consiste en el intercambio de material genético entre dos cromosomas al azar (puede ser incluso entre el

mismo elemento). Es el operador genético principal, hasta el punto que se puede decir que no es un AG sino tiene cross-over, y sin embargo puede serlo sin mutación, según descubrió Holland. La teoría de los esquemas confía en él para hallar la mejor solución a un problema combinando soluciones parciales.

Para aplicar cross-over, entrecruzamiento o recombinación, se escoge aleatoriamente dos miembros de la población, no pasa nada si se emparejan dos descendientes el mismo padre, pues eso indica perpetuación de un individuo con buena puntuación. El intercambio genético se puede llevar a cabo de muchas formas como:

6.4.3.1 Cross-Over Uniforme

Se genera un patrón aleatorio en binario (1s y 0s), y en los elementos que haya un 1 se realiza intercambio genético o bien se genera un número aleatorio para cada bit y si se supera una determinada probabilidad se intercambia ese bit entre los dos cromosomas.

Ejemplo:

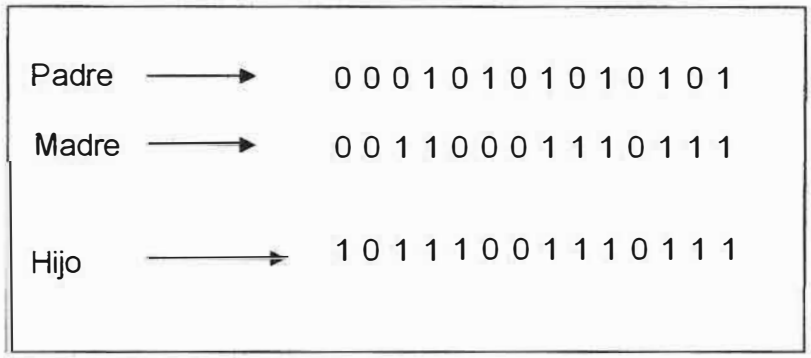


Figura 3. Proceso de Crossover

6.4.3.2 Cross-Over de n-puntos

Los cromosomas se cortan por n puntos y el material genético situado entre ellos se intercambia, ejemplo:

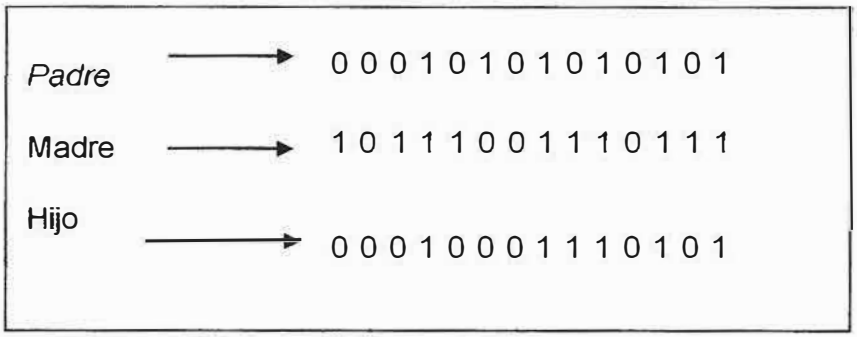


Figura 4. Crossover de n-puntos

6.4.3.3 Cross-Over Especializados

En ocasiones, el espacio de soluciones no es continuo y hay situaciones que a pesar de que sean factibles de producirse en el gen no lo son en la realidad, por lo que hay que incluir restricciones al realizar la recombinación que impidan la aparición de algunas combinaciones y aplicar un cross-over que genere siempre soluciones validas. Un ejemplo de estos son los operadores de cross-over utilizados en el problema del viajante.

6.4.4 Operador Mutación

En la evolución, la mutación es un proceso poco común (sucede aproximadamente una de cada mil replicaciones), en la mayoría de los casos las mutaciones son letales, pero en promedio, contribuyen a la

diversidad genética de la especie. En un AG tendrán el mismo papel y la misma frecuencia (muy baja).

Se modela de la siguiente forma cuando se genera un gen hijo se examinan uno a uno los bits del mismo y se genera un coefi-ciente aleatorio para cada uno. En el caso de que algún coeficiente supere un cierto umbral se modifica dicho bit. Modificando el umbral podemos variar la probabilidad de la mutación.

Las mutaciones son un mecanismo muy interesante por el cual es posible generar nuevos individuos con rasgos distintos a sus predecesores. Los tipos de mutación mas conocidos son:

6.4.4.1 Mutación de Bit

Existe una única probabilidad de que se produzca una mutación de algún bit. De producirse, el algoritmo toma aleatoriamente un bit, y lo invierte.

6.4.4.2 Mutación Multibit

Cada bit tiene una probabilidad de mutarse o no, que es calculada en cada pasada del operador de mutación multibit:

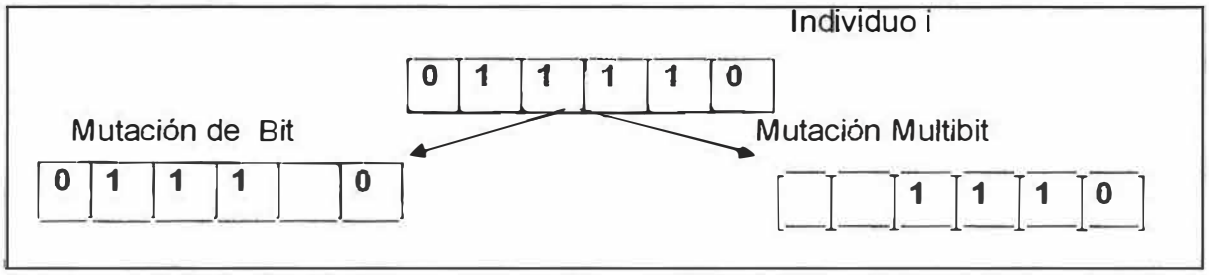


Figura 5. Mutación Multibit

6.4.4.3 Mutación de Gen

Igual que la mutación de bit, solamente que, en vez de cambiar un bit, cambia un gen completo. Puede sumar un valor aleatorio, un valor constante, o introducir gen aleatorio nuevo.

6.4.4.4 Mutación Multigen

Igual que la mutación de multibit, solamente que, en vez de cambiar un conjunto de bits, cambia un conjunto de genes. Puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo.

6.4.4.5 Mutación de Intercambio

Se intercambia el contenido de dos bits/genes aleatoriamente.

6.4.4.6 Mutación de Barajado

Existe una probabilidad de que se produzca una mutación. De producirse, toma dos bits o dos genes aleatoriamente y baraja

de forma aleatoria los bits -o genes, según se escoja comprendidos entre los dos.

A continuación un ejemplo grafico que representa dicha mutación.

Ejemplo de cruce y mutación

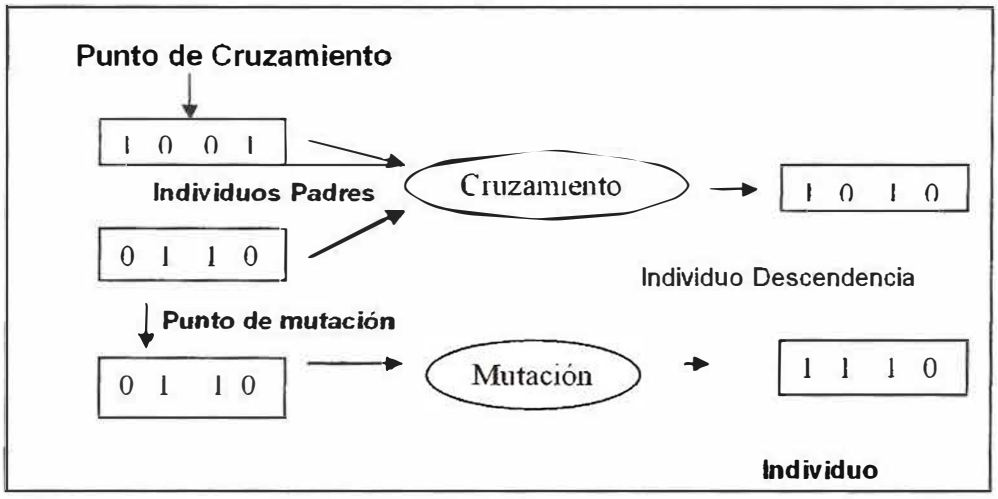


Figura 6. Ejemplo de Cruce y Mutación

6.4.4.7 Creep

Este operador aumenta o disminuye en 1 el valor de un gen; sirve para cambiar suavemente y de forma controlada los valores de los genes.

6.4.5 Otros Operadores

No se usan en todos los problemas, sino sólo en algunos, y en principio su variedad es infinita. Generalmente son operadores que

UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA
BARRANQUILLA

exploran el espacio de soluciones de una forma mas ordenadas y que actúan mas en las últimas fases de la búsqueda, en la cual se pasa de soluciones "casi buenas" a "buenas" soluciones. Ellos son:

6.4.5.1 Cromosoma de Longitud Variable

Hasta ahora se han descrito cromosomas de longitud fija, donde se conoce de antemano el número de parámetros de un problema. Pero hay problemas en los que esto no sucede. Por ejemplo, en un problema de clasificación, donde dado un vector de entrada, queremos agruparlo en una serie de clases, podemos no saber siquiera cuantas clases hay. O en diseño de redes neuronales, puede que no se sepa de hecho, nunca se sabe cuantas neuronas se van a necesitar. Por ejemplo, en un perceptrón hay reglas que dicen cuantas neuronas se deben de utilizar en la capa oculta; pero en un problema determinado puede que no haya ninguna regla heurística aplicable; tendremos que utilizar los AGs para hallar el número óptimo de neuronas.

En estos casos se necesita dos operadores más: añadir y eliminar. Estos operadores se utilizan para añadir un gen, o eliminar un gen del cromosoma. La forma mas habitual de añadir un locus es duplicar uno ya existente, el cual sufre mutación y se añade al lado del anterior.

En este caso, los operadores del algoritmo genético simple (selección, mutación, cross-over) funcionarían de la forma

habitual, salvo, claro está, que sólo se haría cross-over en la zona del cromosoma de menor longitud.

6.4.5.2 Operadores de Nicho Ecológico

Otros operadores importantes son los operadores de *nicho*. Estos operadores están encaminados a mantener la diversidad genética de la población, de forma que cromosomas similares sustituyan sólo a cromosomas similares, y son especialmente útiles en problemas con muchas soluciones; un algoritmo genético con estos operadores es capaz de hallar todos los máximos, dedicándose cada especie a un máximo.

Más que operadores genéticos, son formas de enfocar la selección y la evaluación de la población. Uno de las formas de llevar esto a cabo ya ha sido nombrada, la introducción del *crowding* (apiñamiento). Otra forma es introducir una función de comparición o *sharing*, que indica cuán similar es un cromosoma al resto de la población. La puntuación de cada individuo se dividirá por esta función de compartición, de forma que se facilita la diversidad genética y la aparición de individuos diferentes.

También se pueden restringir los emparejamientos, por ejemplo, a aquellos cromosomas que sean similares. Para evitar las malas consecuencias del *inbreeding* -destinado a potenciar ciertas características frente a otras- que suele aparecer en poblaciones pequeñas, estos periodos se

intercalan con otros periodos en los cuales el emparejamiento es libre.

6.4.5.3 Operadores Especializados

En una serie de problemas hay que restringir las nuevas soluciones generadas por los operadores genéticos, pues no todas las soluciones generadas van a hacer validas, sobre todo en los problemas con restricciones. Por ello, se aplican operadores que mantengan la estructura del problema. Otros operadores son simplemente generadores de diversidad, pero la genera de una forma determinada, ellos son:

6.4.5.3.1 Zap

En ves de cambiar un bit de un cromosoma, cambia un gen completo de un cromosoma.

6.4.5.3.2 Creep

Este operador aumenta o disminuye en 1 el valor de un gen; sirve para cambiar suavemente y de forma controlada los valores de los genes.

6.4.5.3.3 Transposición

Similar al cross-over y a la recombinación genética, pero dentro de un solo cromosoma; dos genes intercambian sus valores, sin afectar el resto del cromosoma. Similar a este es el operador de

eliminación-reinserción, en el que un gen cambia de posición con respecto a los demás.

En toda ejecución de un AG hay que decidir con que frecuencia se va aplicar cada uno de los AGs; en algunos casos como en la mutación o el cross-over uniforme, se debe añadir algún parámetro adicional, que indique con que frecuencia se va aplicar dentro de cada gen del cromosoma.

La frecuencia de aplicación de cada operador estará en función del problema; teniendo en cuenta los efectos de cada operador y ver si se pueden aplicar con cierta frecuencia o no. Generalmente, la mutación y otros operadores que generan diversidad se suelen aplicar con frecuencia alta.

En general la frecuencia de los operadores no varía durante la ejecución del algoritmo, pero hay que tener en cuenta que cada operador es más efectivo en un momento de la ejecución. Por ejemplo al principio, de la fase denominada de *exploración*, los mas eficaces son la mutación y la recombinación, posteriormente, cuando la población ha convergido en parte, la recombinación no es útil, pues se esta trabajando con individuos bastante similares, y es poca la información que se intercambia. Sin embargo, si se produce un estancamiento, la mutación tampoco es útil, pues esta reduciendo el AG a una búsqueda aleatoria; y ahí que aplicar otros operadores, en este caso se pueden utilizar operadores especializados.

6.5 CAPITULO 5

Ventajas y Limitaciones de los Algoritmos Genéticos

6.5.1 Objetivo del Capitulo

En este capitulo se pretende que las persona puedan establecer porque los AGs son mejores ante otros técnicas y aun así identificar las falencia que éstos tienen para poder entender de una forma mas clara lo que son y porque se están estableciendo en la actualidad como una buena técnica que ayuda a buscar los puntos mas óptimos.

6.5.2 Ventajas de los Algoritmos Genéticos

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria.

Veamos a continuación una serie de ventajas que hacen que los AGs sean considerados una buena técnica para la optimización:

- No necesitan conocimientos específicos sobre el problema que intentan resolver.

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

- Operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales.

- Debido al paralelismo que les permite evaluar implícitamente muchos esquemas a la vez, los AGs funcionan particularmente bien resolviendo problemas cuyo espacio de soluciones potenciales es realmente grande, demasiado amplio para hacer una búsqueda exhaustiva en un tiempo razonable. La mayoría de los problemas que caen en esta categoría se conocen como"no lineales".

- Cuando se usan para problemas de optimización (maximizar una función objetivo) resultan menos afectados por los

máximos locales (falsas soluciones) que las técnicas tradicionales.

- Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivamente paralelas.

- Usan operadores probabilísticos, en vez de los típicos operadores deterministicos de las otras técnicas.

- Pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen tamaño de la población, número de generaciones, etc.

- Pueden converger prematuramente debido a una serie de problemas de diversa índole.

- Finalmente, una de las cualidades de los algoritmos genéticos que, a primera vista, puede parecer un desastre, resulta ser una de sus ventajas.

6.5.3 Limitaciones de los Algoritmos Genéticos

Así como poseen ventajas sobre otras técnicas , también es sabido que todo innovación no es del todo excelente y puede tener fallas como todo, ahora bien los AGs no son la excepción y presenta desventajas, las cuales con el tiempo pueden ir convirtiéndose en algo que pueda estar a su favor.

- El poder de los AGs proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el AG encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al AG, tanto en rapidez como en eficacia.

- El problema de cómo escribir la función de aptitud debe considerarse cuidadosamente para que se pueda alcanzar una mayor aptitud que signifique una solución mejor para el problema dado. Si se elige mal una función de aptitud o se define de manera inexacta, puede que el algoritmo genético sea incapaz de encontrar una solución al problema, o puede acabar resolviendo el problema equivocadamente.

- Además de elegir bien la función de aptitud, también deben elegirse cuidadosamente los otros parámetros de un AG; el tamaño de la población, el ritmo de mutación y cruzamiento, el tipo y fuerza de la selección. Si el tamaño de la población es demasiado pequeño, puede que el algoritmo genético no explore suficientemente el espacio

de soluciones para encontrar buenas soluciones consistentemente.

- Varios investigadores (Holland, Forrest, Haupt y Haupt) aconsejan no utilizar algoritmos genéticos en problemas resolubles de manera analítica. No es que los algoritmos genéticos no puedan encontrar soluciones buenas para estos problemas; simplemente es que los métodos analíticos tradicionales consumen mucho menos tiempo y potencia computacional que los AGs y, a diferencia de los AGs, a menudo está demostrado matemáticamente que ofrecen la única solución exacta.

- Un problema muy conocido que puede surgir con un AG se conoce como convergencia prematura. Si un individuo que es más apto que la mayoría de sus competidores emerge muy pronto en el curso de la ejecución, se puede reproducir tan abundantemente que merme la diversidad de la población demasiado pronto, provocando que el algoritmo converja hacia el óptimo local que representa ese individuo, en lugar de rastrear el paisaje adaptativo lo bastante a fondo para encontrar el óptimo global. Esto es un problema especialmente común en las poblaciones pequeñas, donde incluso una variación aleatoria en el ritmo de reproducción puede provocar que un genotipo se haga dominante sobre los otros.

- El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridandolas con los Algoritmos Genéticos.

6.5.4 Robustez de los Métodos de Búsqueda y Optimización tradicionales

Podemos encontrar tres métodos de búsqueda: los basados en el cálculo, los enumerativos, y los aleatorios. Los métodos basados en el cálculo se subdividen en dos clase principales: directos e indirectos.

Los métodos indirectos buscan extremos locales resolviendo el conjunto de ecuaciones que resultan de calcular el gradiente de la función objetivo e igualarlo a cero; esta es la generalización multidimensional del cálculo elemental de puntos extremos.

Es así también que los métodos directos buscan soluciones locales óptimas evaluando la función en direcciones cercanas a la gradiente local. Ambos métodos son de alcance local; el óptimo que buscan está en un entorno del punto actual.

Como es sabido los métodos basados en el cálculo dependen sobre todo de la existencia de las derivadas y esta restricción no es adecuada para la mayoría de los problemas de optimización, por tal

UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA
BARRANQUILLA

razón estos métodos no son lo suficientemente robustos para dominios que no se hayan escogido o evaluado específicamente.

Ahora bien los esquemas enumerativos trabajan a partir de un conjunto de puntos "espacio de búsqueda" que puede ser finito o infinito discreto. En cada iteración, el algoritmo de búsqueda evalúa la función objetivo en un punto aleatorio del espacio.

Los Algoritmos de búsqueda aleatoria usan elecciones al azar como herramienta que los guie en la exploración de un espacio previamente codificado con parámetros. Recuerda que la búsqueda aleatoria no precisamente implica una búsqueda sin dirección.

De lo anterior podemos decir que los métodos de búsqueda tradicionales no son robustos, pero tal conclusión no implica que no sean útiles.

6.5.5 Diferencia entre los algoritmos Genéticos y los Métodos tradicionales

Los algoritmos genéticos se desarrollaron para abordar problemas de optimización de funciones (funciones objetivos, (fitness)) y difieren de otros procedimientos de búsqueda en lo siguiente:

- Trabajan con una codificación del conjunto de parámetros, no con los parámetros en si mismo.
- Buscan a partir de un conjunto de puntos (población) y no desde uno solo.

- Usan como información la función objetivo en vez de otros conocimientos auxiliares.
- Utilizan reglas de transición probabilística, no reglas deterministas.

Los AGs necesitan que el conjunto de parámetros naturales del problema de optimización se codifique como una secuencia de medida finita sobre algún alfabeto finito.

Los AGs trabajan a partir de una serie de puntos simultáneos (población de secuencias), escalando muchos picos en paralelo; por consiguiente la probabilidad de encontrar un pico falso es inferior a la de otros métodos que trabajan punto -- a – punto.

Son muchas la técnicas de búsqueda que requieren gran cantidad de información auxiliar para trabajar apropiadamente, entre ellas se encuentran las técnicas de gradiente que necesitan derivadas, calculadas analíticamente para poder escalar el pico actual y otros procedimientos de búsqueda local (técnicas de optimización combinatoria). Sin embargo los AGs no necesitan esta información auxiliar para realiza la búsqueda efectiva, solo requieren de valores de la función objetivo asociados con consecuencia individuales y esto hace que los AGs sean un método mas canónico que los demás esquemas de búsqueda.

6.6 CAPITULO 6
Áreas De Aplicación

6.6.1 Objetivo del Capitulo

Conocer las áreas donde son útiles y utilizados los AGs y los adelantos que éstos han brindando a la ciencia.

6.6.2 Acústica

Ejemplos de la aplicación de AGs en acústica puede ser el uso de los mismos para el diseño de una sala de conciertos, procurando maximizar la calidad del sonido para la audiencia, la atenuación de ruidos mediante la generación de ondas artificiales destructoras del ruido etc.

6.6.3 Ingeniería Aeroespacial

Diseño de alas de aviones supersónicos, obteniendo una solución de compromiso entre los parámetros fundamentales tales como minimizar la resistencia aerodinámica a velocidades de vuelo supersónicas, minimizar la resistencia a velocidades subsónicas y minimizar la carga aerodinámica (la fuerza que tiende a doblar el ala). Otros ejemplos del uso de AGs en Ingeniería Aeroespacial es el uso de éstos para la construcción de brazos mecánicos para operar con satélites en el espacio, o el problema de maximizar las rutas de los satélites de comunicaciones teniendo como objetivo minimizar las

situaciones de perdida de cobertura con los centros de comunicación.

6.6.4 Astronomía y Astrofísica

Ejemplos de la aplicación de AGs en la Astronomía y Astrofísica pueden ser problemas tales como obtener la curva de rotación de una galaxia basándose en las velocidades rotacionales observadas de sus componentes, determinar el periodo de pulsación de una estrella variable basándose en series de datos temporales, y sacar los valores de los parámetros críticos de un modelo magnetohidrodinámico del viento solar. Son tres difíciles problemas no lineales y multidimensionales.

6.6.5 Química

Cálculo de pulsos de láser para la destrucción de moléculas y formación de elementos simples. Desarrollo de nuevos aminoácidos y proteínas mediante la llamada *química combinatoria*.

6.6.6 Ingeniería Eléctrica

Un ejemplo clásico y el primer caso de que registrara una patente de un invento desarrollado por una máquina fue el desarrollo de una antena mediante Algoritmos Genéticos para su aplicación en un ámbito muy concreto. El desarrollo de antenas eficientes no es en absoluto una tarea fácil y requiere no sólo aplicación de cálculos muy complicados sino que

además requiere de varios intentos para aproximar una antena adecuada. La resolución del problema (que sería encontrar la antena óptima), actualmente es mucho menos que imposible.

6.6.7 Mercados Financieros

Un problema recurrente en cualquier técnica de inteligencia artíllales la modelización de mercados financieros para realizar previsiones y evaluaciones. Cálculos de los tipos de cambio y de acciones bursátiles mediante algoritmos genéticos son ejemplos claros de la aplicación de los AGs a la industria.

6.6.8 Juegos

Los juegos siempre han sido la base de experimentación de muchos de los métodos en inteligencia artificial. En los algoritmos genéticos esta política se sigue cumpliendo. Se han aplicado AGs en la resolución de juegos tales como las damas.

6.6.9 Geofísica

Búsqueda de hipocentros a partir de las mediciones de un terremoto.

6.6.10 Ingeniería de Materiales

Diseño de nuevos polímeros útiles para el tránsito de la corriente eléctrica basados en el carbono (mas conocidos como *poli anilinas*). Otro ejemplo de AGs es su uso para el diseño de

patrones de exposición para un haz de electrones de litografía para la construcción de estructuras micrométricas en circuitos integrados.

6.6.11 Matemáticas y Algoritmia

Uso de AGs para resolver ecuaciones de derivadas parciales no lineales de alto orden, resolución de problemas de ordenación etc.

6.6.12 Ingeniaría de Sistemas

Diseño de circuitos electrónicos, control de de un sistema de frenos de un coche, control de horarios, etc.

6.6.13 Aplicaciones de Búsqueda y Optimización

Desde aplicaciones evidentes, como la biología o la medicina, hasta otros campos como la industria (clasificación de piezas en cadenas de montaje). Los algoritmos genéticos poseen un importante papel en este ámbito.

6.6.14 Aprendizaje Automático

La capacidad que poseen para favorecer a los individuos que se acercan al objetivo, a costa de los que no lo hacen, consigue una nueva generación con mejores reglas y, por lo tanto el sistema será capaz de ir aprendiendo a conseguir mejores

resultados. Es aquí donde encuentran un estupendo marco de trabajo los AGs.

La forma normal de utilización de los AGs es conjuntamente con otro algoritmo existente aplicado al problema de estudios (*algoritmos genéticos híbridos*), aprovechando las ventajas de ambos. La metodología habitual es:

- Utilizar la codificaron del algoritmo existente.

- Incorporar del algoritmo existente:

    1. La elección de las mejores soluciones como punto de partida
    2. Los operadores
    3. La evaluación de las soluciones

- Adaptar los operadores genéticos y las demás técnicas al AG

Los AGs se aplican más eficientemente a problemas de optimización combinatoria sin restricciones. Los problemas de programación en maquinas pertenecen a esta clase y se demuestra que son NP completos para tamaños de maquinas y piezas no muy elevados, es por ello que la utilización de los AGs parece una herramienta viable para estos problemas.

También encontramos que los AGs son utilizados en campos como:

- **Optimización** (estructural, de topologías, numérica y combinatoria): se trata de un campo especial se trata de un campo especialmente abonado para el uso de los AGs, por las características intrínsecas de estos problemas. No en vano fueron la fuente de inspiración para los creadores estos algoritmos. Los AGs se han utilizado en numerosas tareas de optimización, incluyendo la optimización numérica, los problemas de optimización combinatoria, optimización en bases de datos.

- **Programación automática**: los AGs se han empleado para desarrolla programas para tareas específicas, y para diseñar otras estructura computacionales tales como el autómata celular, y las redes de clasificación.

- **Aprendizaje máquina**: los AGs se han utilizado también en muchas de estas aplicaciones, tales como la predicción del tiempo o la estructura de una proteína.

Han servido asimismo para desarrollar determinados aspectos de sistemas particulares de aprendizaje, como pueda ser el de los pesos en una red neuronal, las reglas para sistemas de clasificación de aprendizaje o sistemas de producción simbólica, y los sensores para robots.

- **Economía**: en este caso, se ha hecho uso de estos algoritmos para modelizar procesos de innovación, el desarrollo estrategias de puja, y la aparición de mercados económicos.

- **Sistemas inmunes**: a la hora de modelizar varios aspectos de los sistemas inmunes naturales, incluyendo la mutación somática durante la vida de un individuo y el descubrimiento de familias de genes múltiples en tiempo evolutivo, ha resultado útil el empleo de esta técnica.

- **Ecología**, en la modelización de fenómenos ecológicos tales como las carreras de armamento biológico, la coevolución de parásito-huésped, la simbiosis, y el flujo de recursos.

- **Genética de poblaciones**, en el estudio de preguntas del tipo: *"¿Bajo qué condiciones será viable evolutivamente un gen para la recombinación?"*

- **Evolución y aprendizaje:** los AGs se han utilizado en el estudio de las relaciones entre el aprendizaje individual y la evolución de la especie.

- **Sistemas sociales**, en el estudio de aspectos evolutivos de los sistemas sociales, tales como la evolución del comportamiento social en colonias de insectos, y la evolución de la cooperación y la

comunicación en sistemas multi-agentes. Aunque esta lista no es, en modo alguno, exhaustiva, sí transmite la idea de la variedad de aplicaciones que tienen los AGs. Gracias al éxito en estas y otras áreas, los AGs han llegado a ser un campo puntero en la investigación actual.

6.6.15 Aplicación De los AGs En La Computación Evolutiva

A continuación, pasaremos a comentar algunas aplicaciones exitosas de los Algoritmos Genéticos con la Computación Evolutiva:

- **Diseño de pépticos:** un equipo de Unilever Research ha usado algoritmos genéticos combinados con redes neuronales para diseñar nuevos pépticos bactericidas para usarse en limpiadores anti-bacterianos y preservativos de alimentos.
  Las redes neuronales se utilizaron para predecir la actividad bactericida en los pépticos, y posteriormente se combinaron con algoritmos genéticos para optimizar pépticos virtuales. El resultado fue la generación demás de 400 bactericidas virtuales potencialmente activos, de los cuales 5 fueron sintetizados.

- **Optimización de estrategias de producción:** la empresa de software escocesa Quadstone, uso algoritmos genéticos para resolver un problema de optimización de estrategias de producción de British Petrol. El objetivo era maximizar el retomo financiero de un grupo de campos petrolíferos y de gas interdependientes.

UNIVERSIDAD SIMÓN BOLÍVAR
HEMEROTECA
BARRANQUILLA

El problema es bastante complejo debido a los muchos compromisos posibles entre beneficios y penalizaciones. Sin embargo, aun una mejora relativamente pequeña puede traer enormes ahorros a la larga, ya que su impacto es acumulativo.

El uso de algoritmos genéticos en este problema produjo retornos netos substancialmente mejores que los producidos previamente por cualquier planeador humano o por cualquier otra técnica de optimización.

- **Predicción:** la empresa holandesa Cap Gemini y la empresa británica KiQ Ltd han desarrollado de forma conjunta un sistema llamado Omega, el cual usa algoritmos genéticos para resolver problemas de mercadotecnia, crédito y aplicaciones financieras relacionadas. Omega usa como punto de partida un portafolio de comportamiento previo de un cliente, ya partir de el genera un modelo matemático que puede usarse posteriormente para predecir comportamientos de clientes que se encuentren fuera de los portafolios conocidos.

  Este software puede utilizarse para evaluar solicitudes de crédito, generar listas de correo (para publicidad), modelar lealtad de los clientes a un producto, y para detectar fraudes. Recientemente, un banco holandés comparo Omega contra su sistema de asignación de crédito tradicional (basado en sistemas expertos), encontrando que Omega era substancialmente superior tanto en cantidad (ofrecía mas prestamos) como en calidad (se reducía el riesgo en los créditos) de los prestamos evaluados.

- **Diseño de un sistema de suspensión:** investigadores del KanGAL, el "Laboratorio de Algoritmos genéticos de Kanpur", en la India, han utilizado esta técnica para diseñar un sistema de suspensión para un automóvil que es más cómodo que el previamente utilizado por una empresa automotriz reconocida mundialmente. Utilizando un modelo tridimensional de un automóvil, estos investigadores optimizaron el diseño de los amortiguadores y los resortes de rigidez del vehículo. Las simulaciones efectuadas muestran que el sistema generado por el algoritmo genético hace que los pasajeros sufran menor aceleración vertical, de manera que se disfruta de un mayor confort que con los sistemas utilizados previamente por el fabricante de automóviles en cuestión.

- **Programación de horarios:** Un grupo de investigadores del Joszef Stefan Institute, en Eslovenia, desarrollaron un sistema de programación de horarios basado en técnicas evolutivas. El sistema ha reducido sustancialmente los costos de energía en la planta de prensado de una fábrica de automóviles. El sistema utiliza una heurística "avariciosa" (greedy, en ingles) para diseñar horarios iniciales que luego son utilizados como punto de partida por una técnica evolutiva que minimiza el consumo de energía durante las horas pico.

- **Optimización de losas de concreto prefabricadas:** el siguiente es un problema que fue abordado en el Engineering Design Centre de la Universidad de Plymouth, en Inglaterra. La empresa británica Redlands produce losas prefabricadas de concreto en grandes volúmenes. El diseño de la topología (o

sea la forma) de dichas losas ha sido optimizado por matemáticos usando técnicas tradicionales de optimización. Desde el punto de vista de ingeniería civil, la losa se representa como una placa sujeta a diversos tipos de cargas, comenzando con una puntual (el caso más simple) hasta varias cargas distribuidas en diferentes partes de su superficie.

El problema tiene una alta dimensionalidad (unas 400 variables de decisión, las cuales son todas continuas). El costo computacional asociado con la solución de este problema también es muy alto. Una sola evaluación de la función de aptitud toma alrededor de 10 minutos en una estación de trabajo Sun Sparc con procesadores. Para la solución del problema se recurrió al uso de ANSYS, que es un programa comercial para realizar análisis por medio del elemento finito. Debido al alto costo computacional asociado con el problema, resulto muy obvia la necesidad de desarrollar un esquema que redujera al mínimo posible la cantidad de evaluaciones de la función de aptitud.

La pregunta fundamental era si se podrá diseñar un algoritmo genético el cual, con un número relativamente bajo de evaluaciones de la función de aptitud, pudiese mejorar las soluciones producidas por expertos humanos. Para utilizar computación evolutiva se recurrió a un algoritmo genético con representación real implementado en FORTRAN. A fin de mantener dentro de un límite razonable el costo computacional del algoritmo, se recurrió a un esquema distribuido para evaluar la función de aptitud. La clave fundamental de la propuesta fue

una representación con cambio de granularidad, en la cual se empezaba con una granularidad gruesa (es decir, un intervalo grande entre cada valor de las variables) y se hacia correr al algoritmo genético durante un cierto numero de generaciones (pre-convenido, o hasta alcanzar la convergencia nominal). Una vez hecho eso, se cambiaba la granularidad a otra mas fina, y se procedía a continuar con las iteraciones del algoritmo genético. Esto permitió un acercamiento progresivo hacia la zona factible, manteniendo un costo computacional razonable para el algoritmo genético.

6.7 CAPITULO 7

Paquetes para Resolver Algoritmos Genéticos

6.7.1 Objetivo del Capitulo

Brindar a quien acceda a esta información las herramientas que hoy día se están desarrollando para trabajar con AGs para que se dirijan a ellas y puedan con su ayuda desarrollar AGs para determinados problemas que se les puedan presentar.

6.7.2 Galopps

Bastante bueno, es muy flexible y además nos permite con facilidad puede ser encontrado en *GARAGe*. Su dirección primaria en Internet es: GARAGe.cps.msu.edu/software/software-index.html, y su dirección para descargarlo vía FTP es garage.cps.msu.edu/pub/GA/galopps/.

6.7.3 Gags

Generador de aplicaciones basadas en algoritmos genéticos, escrito en C++. Desarrollado por el grupo de J.J. Melero. Excelente. Su dirección Web es kal-el.ugr.es/gags.html, y su dirección para descargarlo vía FTP es kal-el.ugr.es/GAGS/.

6.7.4 FORTRAN GA

Desarrollo de algoritmos genéticos para Fortran. Su dirección Web es www.staff.uiuc.edu/~ carroll/ga.html.

6.7.5 Galib

Biblioteca de algoritmos genéticos de Matthew. Conjunto de clases en C++ de algoritmos genéticos. Su dirección Web es lancet.mit.edu/ga/, y su dirección para descargarlo vía FTP es lancet.mit.edu/pub/ga/. Podemos registrarlo en http://lancet.mit.edu/ga/Register.html.

6.7.6 Gas

Paquete para desarrollar aplicaciones de algoritmos genéticos en Python. Su dirección Web es starship.skyport.net/crew/gandalf, y su dirección para descargarlo vía FTP es ftp.coe.uga.edu/users/jae/ai.

6.7.7 Geco

Conjunto de herramientas para Lisp. Su dirección para descargarlo vía FTP es ftp://ftp.aic.nrl.navy.mil/pub/galist/src/.

6.7.8 GPdata

Para desarrollar algoritmos genéticos en C++. Su dirección para descargarlo vía FTP es: ftp.cs.bham.ac.uk/pub/authors/W.B.Langdon/gp-code/, y su documentación -GPdata-icga-95.ps- la podemos encontrar en el *site* de Internet cs.ucl.ac.uk/genetic/papers/.

6.7.9 GPJPP

Bibliotecas de clases para desarrollar algoritmos genéticos en Java Su dirección Web es www.turbopower.com/~ kimk/gpjpp.asp.

6.7.10 GP Kernel

Biblioteca de clases para programación genética en C++. Su dirección Web es www.emk.e-technik.th-darmstadt.de/~ thomasw/gp.html.

6.7.11 LIL-GP

Herramientas para programación genética en C. Su dirección Web es isl.msu.edu/GA/software/lil-gp/index.html, y su dirección para descargarlo vía FTP es isl.cps.msu.edu/pub/GA/lilgp/. Podemos encontrar los parches para Linux en www.cs.umd.edu/users/seanl/patched-gp.

6.7.12 PGAPack

Parallel Genetic Algorithm Library. Biblioteca de algoritmos genéticos paralelos. Podemos encontrarlo en la dirección de Internet con un navegador en: www.mcs.anl.gov/home/levine/PGAPACK/index.html, y su dirección para descargarlo vía FTP es ftp.mcs.anl.gov/pub/pgapack/.

6.7.13 SUGAL

SUnderland Genetic ALgorithm system. Para hacer experimentos con algoritmos genéticos. Podemos encontrarlo en la dirección de Internet con el navegador en www.trajan-software.demon.co.uk/sugal.htm.

6.7.14 ADATE

Automatic Design of Algorithms Through Evolution. Programación evolutiva. Su dirección Web es www-ia.hiof.no/~rolando/adate_intro.html.

6.7.15 GPsys

Sistema de programación genética en Java. Podemos encontrarlo en la dirección de Internet www.cs.ucl.ac.uk/staff/A.Qureshi/gpsys.html.

6.8 CAPITULO 8

Colonias De Hormigas (A*nt Colony Optimization* U O*ptimización Por Colonia De Hormigas).*

6.8.1 Objetivo del Capitulo

En este capitulo se mostraran las bases y fundamentos que son necesarios para utilizar los algoritmos de colonias de hormigas, como se codifican, que terminología se utiliza y que se debe tener en cuenta para utilizarlos.

También se busca que las personas que accedan a este pueden entender, poner en uso y practica estos conceptos.

6.8.2 Fundamentos Teóricos

Se basa en el comportamiento colectivo de las hormigas en la búsqueda de alimentos para su subsistencia. Resulta fascinante entender como animales casi ciegos, moviéndose aproximadamente al azar, pueden encontrar el camino más corto desde su nido hasta la fuente de alimentos y regresar. Para esto, cuando una hormiga se mueve, deja una señal odorífera, depositando una sustancia denominada feromona, para que las demás puedan seguirla. En principio, una hormiga aislada se mueve esencialmente al azar, pero las siguientes deciden con una buena probabilidad seguir el camino con mayor cantidad de feromonas.

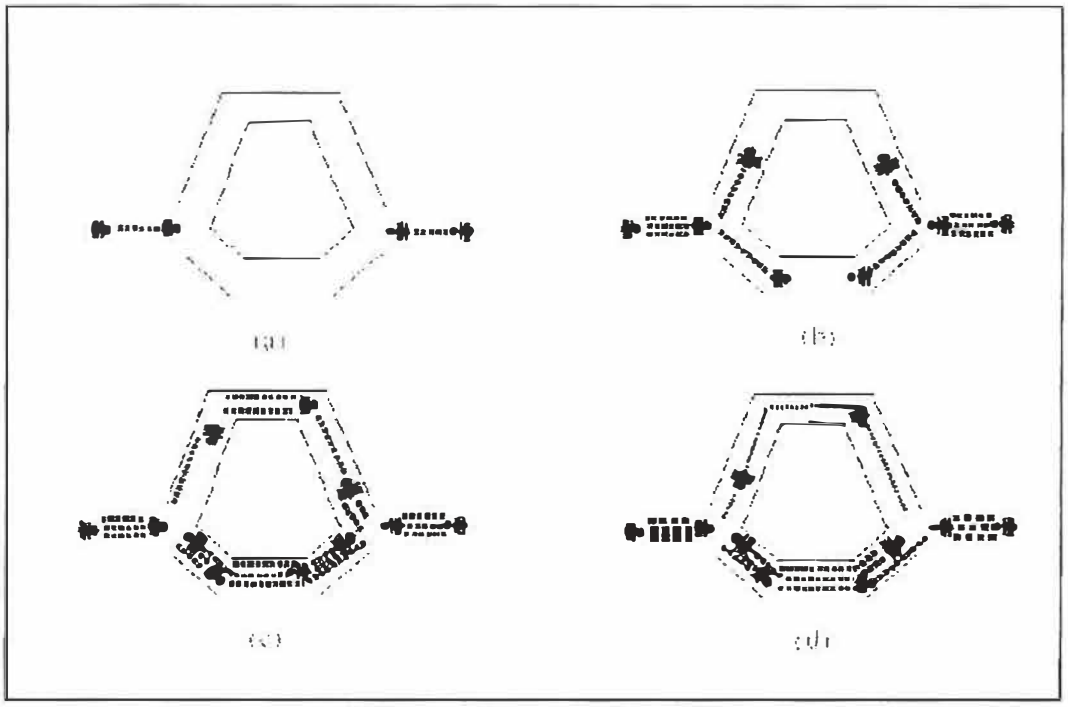


Figura 7. Comportamiento de las Hormigas Reales.

Considere la Figura 1 en donde se observa como las hormigas establecen el camino más corto. En el figura (a) las hormigas llegan al punto en que tienen que decidir por uno de los caminos que se les presenta; en (b) realizan la elección de manera aleatoria, algunas hormigas eligen el camino hacia arriba y otras hacia abajo; en (c) como las hormigas se mueven aproximadamente a una velocidad constante, las que eligieron el camino más corto alcanzarán el otro extremo más rápido que las otras que tomaron el camino más largo, depositando mayor cantidad de feromona por unidad de longitud; en (d) la cantidad de feromona depositada en el trayecto más corto hace que la mayoría de las hormigas elijan este camino, por realimentación positiva.

Esta innovadora técnica basada en agentes muy simples llamados hormigas, nació con la tesis doctoral de Marco Dorigo (1992) quien en

Puesto que las hormigas depositan feromona en el camino que siguen, este comportamiento lleva a un proceso de autorefuerzo que concluye con la formación de rastros señalados por una concentración de feromona elevada. Este comportamiento permite además a las hormigas encontrar los caminos más cortos entre su hormiguero y la fuente del alimento.

Esta convergencia se complementa con la acción del entorno natural que provoca que la feromona se evapore transcurrido un cierto tiempo.

Así, los caminos menos prometedores pierden progresivamente feromona porque son visitados cada vez por menos hormigas. Sin embargo, algunos estudios biológicos han demostrado que los rastros de feromona son muy persistentes la feromona puede permanecer desde unas pocas horas hasta varios meses dependiendo de varios aspectos distintos, como la especie de las hormigas, el tipo de suelo, lo que provoca una menor influencia del efecto de la evaporación en el proceso de búsqueda del camino más corto.

En numerosos experimentos muestran que, debido a la gran persistencia de feromona, es difícil que las hormigas "olviden" un camino que tiene un alto nivel de feromona aunque hayan encontrado un camino aún más corto. Hay que tener en cuenta que si se traslada este comportamiento directamente al ordenador para diseñar un algoritmo de búsqueda podemos encontramos con que se quede rápidamente estancado en un óptimo local.

6.8.4 Algoritmos de optimización basados en Colonias de hormigas

Antes de entrar a lo que son el desarrollo de lo algoritmos de Colonias de Hormigas es necesario conocer le evolución de los algoritmos hasta lo que se conoce como Colonia de Hormigas.

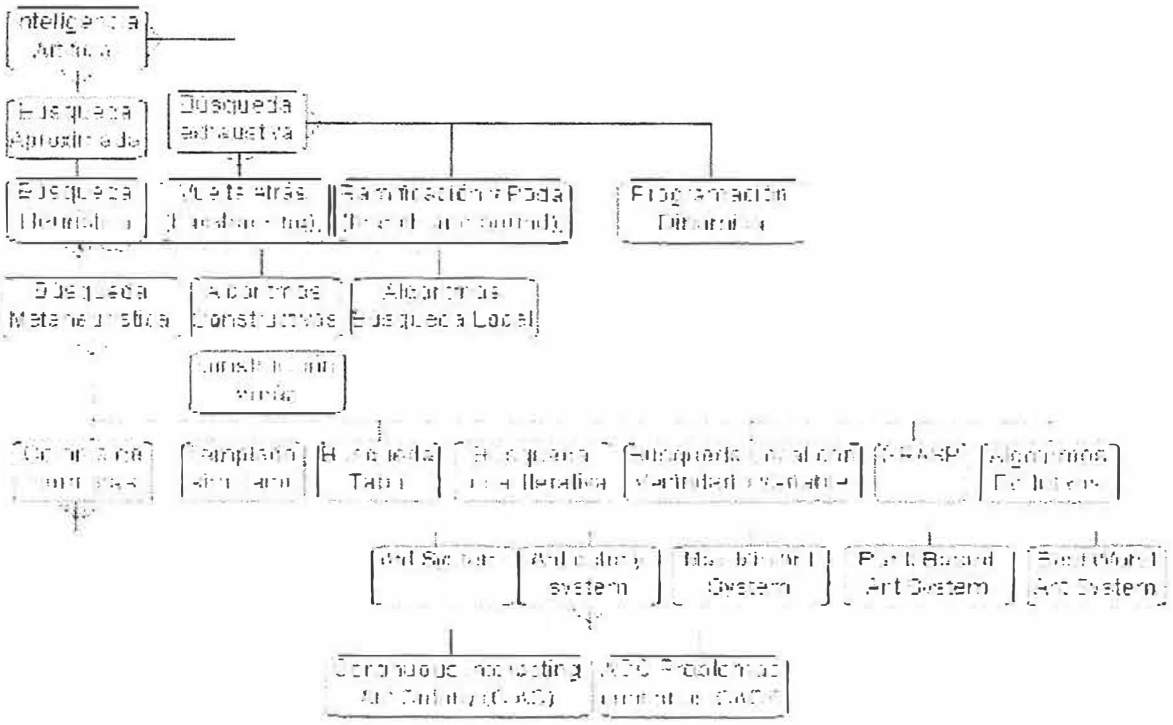


Figura 8. Evolución de los Algoritmo vía Colonia de Hormigas

6.8.4.1 De las hormigas naturales a la OCH

Los algoritmos de OCH se inspiran directamente en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización combinatoria.

1996 publicó tres variantes del algoritmo para la resolución del problema del cajero viajante (Traveling Salesman Problem TSP).

6.8.3 Colonias de Hormigas Naturales

Las hormigas son insectos sociales que viven en colonias y que, debido a su colaboración mutua, son capaces de mostrar comportamientos complejos y realizar tareas difíciles desde el punto de vista de una hormiga individual. Un aspecto interesante del comportamiento de muchas especies de hormigas es su habilidad para encontrar los caminos más cortos entre su hormiguero y las fuentes de alimento. Este hecho es especialmente interesante si se tiene en cuenta que muchas de las especies de hormigas son casi ciegas, lo que evita el uso de pistas visuales.

Mientras que se mueven entre el hormiguero y la fuente de alimento, algunas especies de hormigas depositan una sustancia química denominada *feromona* (una sustancia que puede "olerse"). Si no se encuentra ningún rastro de feromona, las hormigas se mueven de manera básicamente aleatoria, pero cuando existe feromona depositada, tienen mayor tendencia a seguir el rastro. De hecho, los experimentos realizados por biólogos han demostrado que las hormigas prefieren de manera probabilística los caminos marcados con una concentración superior de feromona. En la práctica, la elección entre distintos caminos toma lugar cuando varios caminos se cruzan. Entonces, las hormigas eligen el camino a seguir con una decisión probabilística sesgada por la cantidad de feromona: cuanto más fuerte es el rastro de feromona, mayor es la probabilidad de elegirlo.

Se basan en una colonia de hormigas artificiales, estos son, unos agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales.

Los algoritmos de OCH son esencialmente *algoritmos constructivos*: en cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción. Cada arista del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociada dos tipos de información que guían el movimiento de la hormiga:

- *Información heurística*, que mide la preferencia heurística de moverse desde el nodo *r* hasta el nodo *s*, o sea, de recorrer la arista *ars*. Se nota por *hrs*. Las hormigas no modifican esta información durante la ejecución del algoritmo.
- *Información de los rastros de feromona artificiales*, que mide la "deseabilidad aprendida" del movimiento de *r* a *s*. Imita a la feromona real que depositan las hormigas naturales. Esta información se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas. Se nota por *trs*.

En esta sección se presentan los pasos que llevan desde las hormigas reales a la OCH. A partir de ahora debe tenerse en cuenta que los algoritmos OCH presentan una doble perspectiva:

- Por un lado, son una abstracción de algunos patrones de comportamiento naturales relacionados con el comportamiento que permite encontrar el camino más corto.
- Por otro lado, incluyen algunas características que no tienen una contrapartida natural, pero que permiten que se desarrollen algoritmos para obtener buenas soluciones al problema que se pretende resolver

(por ejemplo, el uso de información heurística que guíe el movimiento de las hormigas).

6.8.4.2 La Hormiga Artificial

La hormiga artificial es un agente La hormiga artificial es un agente computacional simple que intenta construir soluciones posibles al problema explotando los rastros de feromona disponibles y la información heurística. Sin embargo, en algunos problemas, puede también construir soluciones no válidas que podrán ser penalizadas dependiendo de lo inadecuado de la solución. Tiene las siguientes propiedades:

- Recuerda los nodos que ha recorrido, utilizando para ello una lista de nodos visitados (L).
- Tras cada iteración, esta lista contiene la solución construida por la hormiga.
- En cada paso, elige hacia qué nodo s moverse (qué arista seguir) de entre los alcanzables desde el actual r que no hayan sido visitados aún (J(r) = {s | $ ars y s Ï L}), según una regla probabilística de transición.
- Una vez construida su solución, vuelve atrás en el camino recorrido, actualiza la feromona trs (en una cantidad que depende de la bondad de la solución generada) en cada arco (*actualización de feromona en línea a posteriori*) y vacía L
- Opcionalmente, puede también depositar feromona en cada arco que recorre mientras construye la solución (*actualización de feromona en línea paso a paso*).

- El único mecanismo de comunicación entre las hormigas es la estructura de datos que almacena los rastros de feromona de cada arco (memoria compartida).

6.8.4.3 Similitudes y Diferencias Entre las Hormigas Naturales y Artificiales

Las colonias de hormigas naturales y artificiales comparten una serie de características. Las más importantes pueden resumirse a continuación:

- Uso de una colonia de individuos que interaccionan y colaboran para solucionar una tarea dada.
- Tanto las hormigas naturales como las artificiales modifican su "entorno" a través de una comunicación estimérgica basada en la feromona. En el caso de las hormigas artificiales, los rastros de feromona artificiales son valores numéricos que están disponibles únicamente de manera local.
- Ambas, las hormigas naturales y las artificiales, comparten una tarea común: la búsqueda del camino más corto (construcción iterativa del una solución de costo mínimo) desde un origen, el hormiguero (decisión inicial), hasta un estado final, la comida (última decisión).
- Las hormigas artificiales construyen las soluciones iterativamente aplicando una estrategia de transición local estocástica para moverse entre estados adyacentes, tal como hacen las hormigas naturales. Sin embargo, estas características por sí solas no permiten desarrollar algoritmos eficientes para problemas combinatorios difíciles. De hecho,

las hormigas artificiales viven en un mundo discreto y tienen algunas capacidades adicionales:

- Las hormigas artificiales pueden hacer uso de la información heurística (no solo de los rastros locales de feromona) en la política estocástica de transición que apliquen.
- Tienen una memoria que almacena el camino seguido por la hormiga.
- La cantidad de feromona depositada por la hormiga artificial es función de la calidad de la solución encontrada. Otra gran diferencia se refiere al momento de depositar la feromona. Las hormigas artificiales normalmente sólo deposita feromona después de generar una solución completa.
- La evaporación de feromona en los algoritmos de OCH es diferente a como se presenta en la naturaleza, ya que la inclusión del mecanismo de evaporación es una cuestión fundamental para evitar que el algoritmo se quede estancado en óptimos locales. La evaporación de feromona permite a la colonia de hormigas artificiales olvidar lentamente su historia pasada para redireccionar su búsqueda hacia nuevas regiones del espacio. Esto evita una convergencia prematura del algoritmo hacia óptimos locales.
- Para mejorar la eficiencia y eficacia del sistema, los algoritmos OCH pueden enriquecerse con habilidades adicionales.

6.8.4.4 Modo de Funcionamiento y Estructura Genérica de un Algoritmo de OCH

El modo de operación básico de un algoritmo de OCH es como sigue: las *m* hormigas (artificiales) de la colonia se mueven, concurrentemente y de manera asíncrona, a través de los estados adyacentes del problema (que puede representarse en forma de grafo con pesos). Este movimiento se realiza siguiendo una regla de transición que está basada en la información local disponible en las componentes (nodos).

Esta información local incluye la información heurística y memorística (rastros de feromona) para guiar la búsqueda. Al moverse por el grafo de construcción, las hormigas construyen incrementalmente soluciones. Opcionalmente, las hormigas pueden depositar feromona cada vez que crucen un arco (conexión) mientras que construyen la solución (*actualización en línea paso a paso de los rastros de feromona*). Una vez que cada hormiga ha generado una solución se evalúa ésta y puede depositar una cantidad de feromona que es función de la calidad de su solución (*actualización enlínea a de los rastros de feromona*). Esta información guiará la búsqueda de las otras hormigas de la colonia en el futuro.

Además, el modo de operación genérico de un algoritmo de OCH incluye dos procedimientos adicionales, la evaporación de los rastros de feromona y las accione del demonio. La evaporación de feromona la lleva a cabo el entorno y se usa como un mecanismo que evita el estancamiento en la búsqueda y permite que las hormigas busquen y exploren nuevas regiones del espacio.

Las acciones del demonio son acciones opcionales -que no tienen un contrapunto natural para implementar tareas desde una perspectiva global que no pueden llevar a cabo las hormigas por la perspectiva local que ofrecen.

La estructura de un algoritmo de OCH genérico es como sigue:

```
1 Procedimiento Metaheuristica_OCH()
2 Inicializacion_de_parámetros
3 mientras (criterio_de_terminación_no_satisfecho)
4 Programación_de_actividades
5 Generación_de_Hormigas_y_actividad()
6 Evaporacion_de_Feromona()
7 Acciones_del_demonio() {opcional}
8 fin Programación_de_actividades
9 fin mientras
10 fin Procedimiento
1 Procedimiento Generación_de_Hormigas_y_actividad()
2 repetir en paralelo desde k=1 hasta m (numero_hormigas)
3 Nueva_Hormiga(k)
4 fin repetir en paralelo
5 fin Procedimiento
1 Procedimiento Nueva_Hormiga(id_Hormiga)
2 inicializa_hormiga(id_Hormiga)
3 L = actualiza_memoria_hormiga()
4 mientras (estado_actual ¹ estado_objetivo)
5 P = calcular_probabilidades_de_transición(A,L,W)
6 siguiente_estado = aplicar_política_decisión(P,W)
7 mover_al_siguiente_estado(siguiente_estado)
si (actualizacion_feromona_en_linea_paso_a_paso)
8 depositar_feromona_en_el_arco_vistado()
fin si
9 L = actualizar_estado_interno()
```

```
10 fin mientras
si (actualizacion_feromona_en_linea_a_posteriori)
11 para cada arco visitado
12 depositar_feromona_en_el_arco_visitado()
13 fin para
fin si
14 liberar_recursos_hormiga(id_Hormiga)
15 fin Procedimiento
```

El primer paso incluye la inicialización del los valores de los parámetros que se tienen en consideración en el algoritmo. Entre otros, se deben fijar el rastro inicial de feromona asociado a cada transición, *t0*, que es un valor positivo pequeño, normalmente el mismo para todas las componentes/conexiones, el número de hormigas en la colonia, *m*, y los pesos que definen la proporción en la que afectarán la información heurística y memorística en la regla de transición probabilística. El procedimiento principal de metaheurística OCH controla, mediante el constructor Programación_de_Actividades, la planificación de las tres componentes: 1) la generación y puesta en funcionamiento de las hormigas artificiales, 2) la evaporación de feromona, y 3) las acciones del demonio.

La implementación de este constructor determinará la sincronía existente entre cada una de las tres componentes. Mientras que la aplicación a problemas "clásicos" NP-duros (no distribuidos), normalmente usa una planificación secuencial, en problemas distribuidos como el enrutamiento en redes, el paralelismo puede ser explotado de manera sencilla y eficiente.

Como se ha comentado antes, varias componentes son o bien opcionales, como las acciones del demonio, o bien dependientes estrictamente del algoritmo de OCH especifico, por ejemplo cuándo y cómo se deposita la feromona. Generalmente, la actualización en línea paso a paso de los rastros de feromona y la actualización en línea a posteriori de los rastros de feromona son mutuamente excluyentes y no suelen estar presentes a la vez ni faltar ambas al mismo tiempo (si las dos faltan, el demonio suele actualizar los rastros de feromona).

Por otro lado, hay que remarcar que el procedimiento actualiza_memoria_hormiga() se encarga de especificar el estado inicial desde el que la hormiga comienza su camino y, además almacenar la componente correspondiente en la memoria de la hormiga *L*. La decisión sobre cuál será dicho nodo depende del algoritmo específico (puede ser una elección aleatoria o una fija para toda la colonia, o una elección aleatoria o fija para cada hormiga, etc.).

Por último, comentar que los procedimientos calcular_probabilidades_de_transición y aplicar_política_decisión tienen en consideración el estado actual de la hormiga, los valores actuales de la feromona visibles en dicho nodo y las restricciones del problema W para establecer el proceso de transición probabilístico hacia otros estados válidos.

6.8.4.5 Pasos a seguir para Resolver Problemas mediante OCH

Observando las aplicaciones actuales de la OCH, podemos identificar algunas directivas sobre como atacar problemas utilizando esta metaheurística.

Estas directivas se pueden resumir en las seis tareas de diseño que se presentaremos a continuación:

1. Representar el problema como un conjunto de componentes y transiciones o a través de un grafo ponderado que será recorrido por las hormigas para construir soluciones.
2. Definir de manera apropiada el significado de los rastros de feromona *trs*, esto es, el tipo de decisión que inducen. Éste es un paso crucial en la implementación de un algoritmo de OCH y, a menudo, una buena definición de los rastros de feromona no es una tarea trivial. De hecho, normalmente implica tener un buen conocimiento del problema que se quiere solucionar.
3. Definir de manera apropiada la preferencia heurística de cada decisión que debe tomar una hormiga mientras que construye una solución, es decir, definir la información heurística *hrs* asociada a cada componente o transición. Hay que remarcar que la información heurística es crucial para un buen rendimiento si no existen o no pueden ser aplicados algoritmos de búsqueda local.
4. Si es posible, implementar una búsqueda local eficiente para el problema que se esea solucionar, porque los resultados de muchas aplicaciones de la OCH a problemas de optimización combinatoria NP-duros demuestran que el mejor rendimiento se alcanza cuando se complementa con optimizaciones.

5. Escoger un algoritmo de OCH especifico (algunos de los cuales serán descritos en la siguiente sección) y aplicarlo al problema que hay que solucionar, teniendo en cuenta, obviamente, todos los aspectos previamente comentados.
6. Refinar los parámetros del algoritmo de OCH. Un buen punto de partida para la especificación de los valores de los parámetros es usar configuraciones de parámetros que han demostrado ser buenas cuando se aplicaban en el algoritmo de OCH a problemas similares o a una gran variedad de problemas distintos. Otra posible alternativa para evitar el esfuerzo y tiempo necesarios para esta tarea es utilizar procedimientos automáticos de refinamiento de parámetros.

Debe quedar claro que los pasos arriba mencionados sólo dan una idea a grosso modo sobre la implementación del algoritmo de OCH. Además, en muchas ocasiones la implementación es un proceso iterativo, donde se obtiene una visión más en profundidad del problema y del comportamiento del algoritmo, deben revisarse algunas de las decisiones iniciales. Los pasos más importantes son los cuatro primeros, ya que una elección poco acertada en esos puntos no suele poder arreglarse con un buen refinamiento de parámetros.

6.8.4.6 Sistemas de Hormigas

El SH, desarrollado por Dorigo, Maniezzo y Colorni en 1991, fue el primer algoritmo de OCH. Inicialmente, se presentaron 3 variantes distintas: *SH-densidad*, *SH-cantidad* y *SH-ciclo*, que se diferenciaban en la manera en que se actualizaban los rastros de feromona. En los dos primeros, las hormigas depositaban feromona mientras que construían

sus soluciones (esto es, aplicaban una *actualización en-línea paso a paso de feromona*), con la diferencia de que la cantidad de feromona depositada en el SH-densidad es constante, mientras que la depositada en SH-cantidad dependía directamente de la deseabilidad heurística de la transición *hrs*. Por último, en SH-ciclo, la deposición de feromona se lleva a cabo una vez que la solución está completa (actualización en línea a posteriori de feromona). Esta última variante era la que obtenía unos mejores resultados y es por tanto la que se conoce como SH en la literatura (y en el resto de este trabajo).

El SH se caracteriza por el hecho de que la actualización de feromona se realiza una vez que todas las hormigas han completado sus soluciones, y se lleva a cabo como sigue: primero, todos los rastros de feromona se reducen en un factor constante, implementándose de esta manera la evaporación de feromona. A continuación cada hormiga de la colonia deposita una cantidad de feromona que es función de la calidad de su solución. Inicialmente, el SH no usaba ninguna acción del demonio, pero es relativamente fácil, por ejemplo, añadir un procedimiento de búsqueda local para refinar las soluciones generadas por las hormigas.

6.8.4.7 Sistema de Colonias de Hormigas

El SCH es uno de los primeros sucesores del SH que introduce tres modificaciones importantes con respecto a dicho algoritmo de OCH:

- La regla de transición establece un equilibrio entre la exploración de nuevos arcos y la explotación de la información acumulada.

- La actualización de feromona la efectúa el demonio (fuera de línea), considerando sólo la hormiga que generó la mejor solución. Sólo se evapora feromona en los arcos que componen ésta.
- Las hormigas modifican automáticamente la feromona de cada arco que visitan mediante una actualización en línea paso a paso.

6.8.4.8 Funciones adicionales de los algoritmos de OCH

Como algoritmos de optimización y búsqueda, los algoritmos de OCH deben guardar un equilibrio adecuado entre eficiencia y la eficacia

- El demonio puede realizar una serie de funciones adicionales que permiten mejorar ambos aspectos:
  - Eficiencia: Uso de listas de candidatos para reducir el tiempo de ejecución de las reglas de transición y actualización.
  - Eficacia Þ Uso de algoritmos de búsqueda local para refinar las soluciones obtenidas por cada hormiga en cada iteración.

USO DE LISTAS DE CANDIDATOS

- Cuando los casos de los problemas son de gran tamaño, los algoritmos de OCH pueden requerir un tiempo de ejecución alto. Por ejemplo, cuando el grafo es totalmente conexo, el SH es de orden O (t·m·n2) (t = nº iteraciones, m = nº hormigas, n= nº nodos).

- La mayor parte del costo computacional de la OCH proviene de las reglas de transición de estados y actualización de feromona.
- Para acelerar ambas, en algunos problemas como el Viajante de Comercio, se usan las Listas de Candidatos (LCs).

La LC almacena los cl nodos preferidos para visitar desde cada nodo r ordenados de mejor a peor valor heurístico. Cuando la hormiga está en el nodo r, la regla de transición sólo considera los nodos incluidos en LC(r) no visitados aún. Sólo se usa el resto cuando todos los nodos de LC(r) han sido visitados. En ese caso, existen dos posibilidades:

- Escoger directamente el nodo con mejor valor heurístico de los no incluidos en la lista.
- Aplicar la regla de transición considerando los nodos restantes.

Con la primera opción, el tiempo de ejecución necesario para que una hormiga construya una solución es lineal (O (cl·n)).

Además, en este caso, es posible acelerar la ejecución de la regla de actualización de feromona en línea a posteriori en algoritmos de OCH en los que todas las hormigas actualizan (como el SH).

Para ello, sólo se modifica la feromona de los arcos que conectan nodos incluidos en la LC. Esto no es válido en la segunda opción que considera los demás rastros de feromona.

USO DE ALGORITMOS DE BÚSQUEDA LOCAL

- Existe la posibilidad de hibridar los algoritmos de OCH co técnicas de búsqueda por entornos para mejorar su eficacia.
- La hibridación consiste en que el demonio aplica una búsqueda por entornos sobre las soluciones construidas por todas las hormigas en cada iteración antes de actualizar la feromona.
- El aumento en la eficacia se consigue a costa de una disminución en la eficiencia, por lo que es habitual emplear a la vez versiones restringidas de técnicas de búsqueda local y LCs.
- El algoritmo básico de OCH con búsqueda local es:
  Mientras (no se dé la condición de parada) *hacer*
  - Construcción probabilística de las soluciones preliminares mediante la colonia de hormigas.
  - Refinamiento de dichas soluciones mediante la búsqueda local.
  - Actualización de feromona fuera de línea.

De este modo, Los algoritmos de OCH con búsqueda local son algoritmos híbridos con una generación probabilística de soluciones iniciales para una técnica clásica de búsqueda local.

Así, pueden encuadrarse en los algoritmos de Búsqueda Local Múltiarranque, aunque éstos generan aleatoria mente las soluciones iniciales sin usar información heurística. También son similares a los algoritmos GRASP, aunque con otra filosofía de generación

probabilística de soluciones. La principal diferencia con los dos algoritmos anteriores es el mecanismo de cooperación global de las soluciones generadas que hace que las ejecuciones no sean independientes entre sí.

Es así como las Colonias Hormigas se aplican para resolver problemas como:

1. Enrutamiento de Paquetes en redes de Telecomunicaciones.
2. Secuencias de tareas.
3. Coloreo de grafos.
4. encaminamiento de Vehículos.
5. Ordenación secuencial.
6. "Pooling " de Vehículos.
7. Líneas de producción de coches.
8. Problemas de agrupamiento (Clustering).
9. Aprendizaje de Reglas Clásicas y Difusas.

Podemos decir que las Colonias de Hormigas es una meta heurística Bioinspirada que permite desarrollar algoritmos sencillos de entender, rápidos y con buen rendimiento para problemas que de puedan representar en forma de grafos con pesos.

6.9 CAPITULO 9

Técnica Backtracking (Fuerza Bruta)

6.9.1 Objetivo del capitulo

En este capitulo se mostraran las bases y fundamentos que son necesarios para saber lo que realmente es la técnica de fuerza bruta, que terminología se utilizan, que problemas se han desarrollado utilizándola, que resultados se han obtenido y como hacer para poder utilizarla en futuros problemas.

También se busca que las personas que accedan a este pueden entender y poner en uso y practica estos conceptos.

6.9.2 Fundamentos Teóricos

Los algoritmos de vuelta atrás se utilizan para encontrar soluciones a un problema. No siguen unas reglas para la búsqueda de la solución, simplemente una búsqueda sistemática, que más o menos viene a significar que hay que probar todo lo posible hasta encontrar la solución o encontrar que no existe solución al problema. Para conseguir este propósito, se separa la búsqueda en varias búsquedas parciales o subtareas. Asimismo, estas subtareas suelen incluir más subtareas, por lo que el tratamiento general de estos algoritmos es de naturaleza recursiva.

¿Por qué se llaman algoritmos de vuelta atrás?. Porque en el caso de no encontrar una solución en una subtarea se retrocede a la subtarea

original y se prueba otra cosa distinta (una nueva subtarea distinta a las probadas anteriormente).

Puesto que a veces nos interesa conocer múltiples soluciones de un problema, estos algoritmos se pueden modificar fácilmente para obtener una única solución (si existe) o todas las soluciones posibles (si existe más de una) al problema dado.

Estos algoritmos se asemejan al recorrido en profundidad dentro de unos grafos, siendo cada subtarea un nodo del grafo. El caso es que el grafo no está definido de forma explícita (como lista o matriz de adyacencia), sino de forma implícita, es decir, que se irá creando según avance el recorrido. A menudo dicho grafo es un árbol, o no contiene ciclos, es decir, al buscar una solución es, en general, imposible llegar a una misma solución **x** partiendo de dos subtareas distintas **a** y **b**; o de la subtarea **a** es imposible llegar a la subtaréa **b** y viceversa.

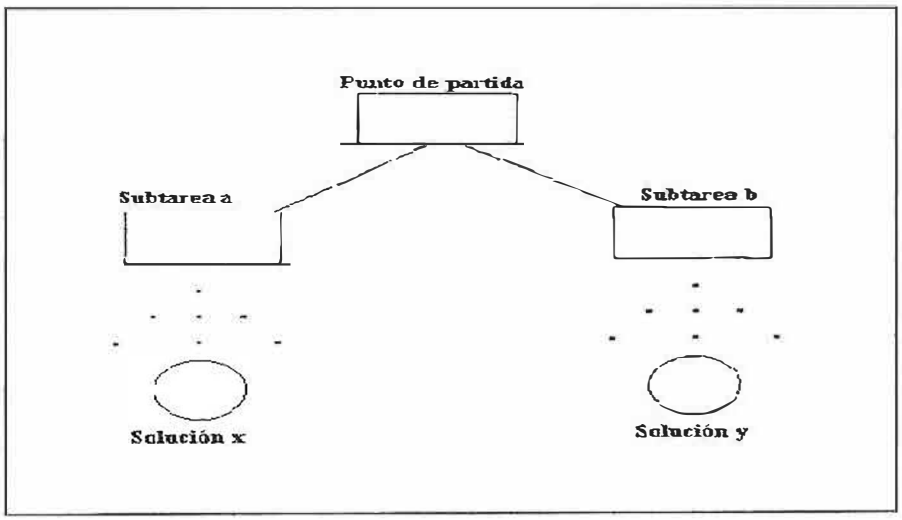


Figura 9. Recorrido semejante utilizado en el Backtracking

A menudo ocurre que el árbol o grafo que se genera es tan grande que encontrar una solución o encontrar la mejor solución entre varias posibles es computacionalmente muy costoso. En estos casos suelen aplicarse una serie de restricciones, de tal forma que se puedan *podar* algunas de las ramas, es decir, no recorrer ciertas subtareas.

Esto es posible si llegado a un punto se puede demostrar que la solución que se obtendrá a partir de ese punto no será mejor que la mejor solución obtenida hasta el momento. Si se hace correctamente, la poda no impide encontrar la mejor solución.

A veces, es imposible demostrar que al hacer una poda no se esté ocultando una buena solución. Sin embargo, el problema quizás no pida la mejor solución, sino una que sea razonablemente buena y cuyo coste computacional sea bastante reducido. Esa es una buena razón para aumentar las restricciones a la hora de recorrer un nodo. Tal vez se pierda la mejor solución, pero se encontrará una aceptable en un tiempo reducido.

Los algoritmos de vuelta atrás tienen un esquema genérico, según se busque una o todas las soluciones, y puede adaptarse fácilmente según las necesidades de cada problema. A continuación se exponen estos esquemas, extraídos de Wirth. Los bloques se agrupan con **begin** y **end**, equivalentes a los corchetes de C, además están tabulados.

- **esquema para una solución:**

```
procedimiento ensayar (paso : TipoPaso)
 repetir
 | seleccionar_candidato
 | if aceptable then
 | begin
 |   anotar_candidato
 |   if solucion_incompleta then
 |   begin
 |     ensayar(paso_siguiente)
 |     if no acertado then borrar_candidato
 |   end
 |   else begin
 |     anotar_solucion
 |     acertado <- cierto;
 | end
 hasta que (acertado = cierto) o (candidatos_agotados)
fin procedimiento
```

- esquema para todas las soluciones:

```
procedimiento ensayar (paso : TipoPaso)
 para cada candidato hacer
 | seleccionar candidato
 | if aceptable then
 | begin
 |   anotar_candidato
 |   if solucion_incompleta then
```

```
|    ensayar(paso_siguiente)
|  else
|    almacenar_solucion
|  borrar_candidato
| end
 hasta que candidatos_agotados
fin procedimiento
```

Por último, se exponen una serie de problemas típicos que se pueden resolver fácilmente con las técnicas de vuelta atrás. El primero que se expone es muy conocido. Se trata de la vuelta del caballo. Muchos problemas de los pasatiempos de los periódicos pueden resolverse con la ayuda de un ordenador.

6.9.2.1 La vuelta del caballo

Se dispone de un tablero rectangular, por ejemplo el tablero de ajedrez, y de un caballo, que se mueve según las reglas de este juego. El objetivo es encontrar una manera de recorrer todo el tablero partiendo de una casilla determinada, de tal forma que el caballo pase una sola vez por cada casilla. Una variante es obligar al caballo a volver a la posición de partida en el último movimiento.

Por último se estudiará como encontrar todas las soluciones posibles partiendo de una misma casilla.

Para resolver el problema hay que realizar todos los movimientos posibles hasta que ya no se pueda avanzar, en cuyo caso hay que dar marcha atrás, o bien hasta que se cubra el tablero. Además, es

necesario determinar la organización de los datos para implementar el algoritmo.

- ¿Cómo se mueve un caballo?. Para aquellos que no sepan jugar al ajedrez se muestra un gráfico con los ocho movimientos que puede realizar. Estos movimientos serán los ocho candidatos.

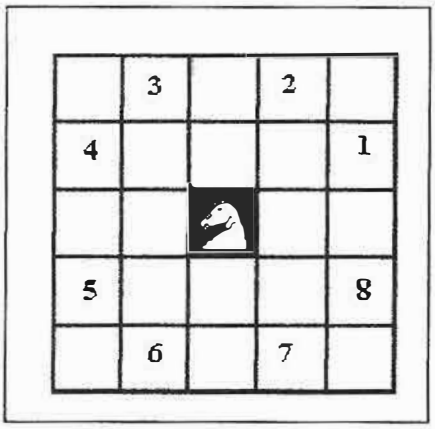
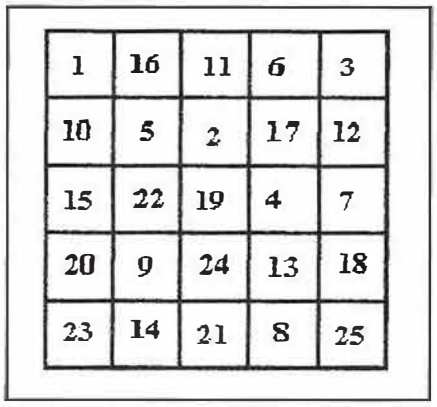


Figura 10. Grafico del tablero con movimientos del caballo

Con las coordenadas en las que se encuentre el caballo y las ocho coordenadas relativas se determina el siguiente movimiento. Las coordenas relativas se guardan en dos arrays:

ejex = [2, 1, -1, -2, -2, -1, 1, 2]
ejey = [1, 2, 2, 1, -1, -2, -2, -1]

El tablero, del tamaño que sea, se representará mediante un array bidimensional de números enteros. A continuación se muestra un gráfico con un tablero de tamaño 5x5 con todo el recorrido partiendo de la esquina superior izquierda.



1	16	11	6	3
10	5	2	17	12
15	22	19	4	7
20	9	24	13	18
23	14	21	8	25

Figura 11. Tablero de tamaño5*5 con todo el recorrido

Cuando se encuentra una solución, una variable que se pasa por referencia es puesta a 1 (cierto). Puede hacerse una variable de alcance global y simplificar un poco el código, pero esto no siempre es recomendable.

Para codificar el programa, es necesario considerar algunos aspectos más, entre otras cosas no salirse de los límites del tablero y no pisar una casilla ya cubierta (selección del candidato). Se determina que hay solución cuando ya no hay más casillas que recorrer.

A continuación se expone un código completo en C, que recubre un tablero cuadrado de lado N partiendo de la posición (0,0).

```
#include <stdio.h>
#define N 5
#define ncuad N*N
void mover(int tablero[][N], int i, int pos_x, int pos_y, int *q);
const int ejex[8] = { -1,-2,-2,-1, 1, 2, 2, 1 },
          ejey[8] = { -2,-1, 1, 2, 2, 1,-1,-2 };
```

```
int main(void)
{
  int tablero[N][N]; /* tablero del caballo. */
  int i,j,q;

   /* inicializa el tablero a cero */
  for (i = 0; i < N; i++)
   for (j = 0; j < N; j++)
    tablero[i][j] = 0;

   /* pone el primer movimiento */
  tablero[0][0] = 1;
  mover(tablero,2,0,0,&q);

  if (q) { /* hay solucion: la muestra. */
    for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++)
        printf("%3d ", tablero[i][j]);
      putchar('\n');
    }
  }
  else
    printf("\nNo existe solucion\n");

  return 0;
}

void mover(int tablero[][N],int i, int pos_x, int pos_y, int *q)
{
```

```
  int k, u, v;

  k = 0;
  *q = 0;
  do {
    u = pos_x + ejex[k]; v = pos_y + ejey[k]; /* seleccionar candidato */
    if (u >= 0 && u < N && v >= 0 && v < N) { /* esta dentro de los
limites? */
      if (tablero[u][v] == 0) { /* es valido? */
        tablero[u][v] = i;  /* anota el candidato */
        if (i < ncuad) { /* llega al final del recorrido? */
          mover(tablero,i+1,u,v,q);
          if (!*q) tablero[u][v] = 0; /* borra el candidato */
        }
        else *q = 1; /* hay solucion */
      }
    }
    k++;
  } while (!*q && k < 8);
}
```

Cambiando el valor de *N* puede obtenerse una solución para un tablero cuadrado de tamaño N.

A continuación, se muestra una adaptación del procedimiento que muestra todas las soluciones. Si se ejecuta para N = 5 se encuentra que hay 304 soluciones partiendo de la esquina superior izquierda.

Cuando se encuentra una solución se llama a un procedimiento (no se ha codificado aquí) que imprime todo el tablero.

```
void mover(int tablero[][N],int i, int pos_x, int pos_y)
{
  int k, u, v;

  for (k = 0; k < 8; k++) {
    u = pos_x + ejex[k]; v = pos_y + ejey[k];
    if (u >= 0 && u < N && v >= 0 && v < N) { /* esta dentro de los
limites */
      if (tablero[u][v] == 0) {
        tablero[u][v] = i;
        if (i < ncuad)
          mover(tablero,i+1,u,v);
        else imprimir_solucion(tablero);
        tablero[u][v] = 0;
      }
    }
  }
}
```

6.9.2.2 El problema de las ocho reinas

Continuamos con problemas relacionados con el ajedrez. El problema que ahora se plantea es claramente, como se verá, de vuelta atrás. Se recomienda intentar resolverlo *a mano*.

Se trata de colocar ocho reinas sobre un tablero de ajedrez, de tal forma que ninguna amenace (pueda comerse) a otra. Para los que no

sepan ajedrez deben saber que una reina amenaza a otra pieza que esté en la misma columna, fila o cualquiera de las cuatro diagonales.

La dificultad que plantea este problema es la representación de los datos. Se puede utilizar un array bidimensional de tamaño 8x8, pero las operaciones para encontrar una reina que amenace a otra son algo engorrosas y hay un truco para evitarlas. La solución aquí expuesta vuelve a ser tomada de Wirth .

Es lógico que cada reina debe ir en una fila distinta. Por tanto, en un array se guarda la posición de cada reina en la columna que se encuentre. Ejemplo: si en la tercera fila hay una reina situada en la quinta columna, entonces la tercera posición del array guardará un 5. A este array se le llamará **col.**
Hace falta otro array que determine si hay puesta una reina en la fila j-ésima. A este array se le llamará **fila**.
Por último se utilizan dos arrays más para determinar las diagonales libres, y se llamarán **diagb** y **diagc.**

Para poner una reina se utiliza esta instrucción:
col[i] = j ; fila[j] = diagb[i+j] = diagc[7+i-j] = FALSE;

Para quitar una reina esta otra:
fila[j] = diagb[i+j] = diagc[7+i-j] = TRUE;

Se considera válida la posición para este caso:
if (fila[j] && diagb[i+j] && diagc[7+i-j]) entonces proceder ...

A continuación se expone el código completo en C. Se han utilizado tipos enumerados para representar los valores booleanos.

```
#include <stdio.h>
enum bool {FALSE, TRUE};
typedef enum bool boolean;

void ensayar(int i, boolean *q, int col[], boolean fila[], boolean diagb[],
boolean diagc[]);

int main(void)
{
  int i;
  boolean q;

  int col[8];
  boolean fila[8],diagb[15], diagc[15];

  for (i = 0; i < 8; i++) fila[i] = TRUE;
  for (i = 0; i < 15; i++) diagb[i] = diagc[i] = TRUE;

  ensayar(0,&q,col,fila,diagb,diagc);
  if (q) {
    printf("\nSolucion:");
    for (i = 0; i < 8; i++) printf(" %d", col[i]);
  } else printf("\nNo hay solucion");

  return 0;
}

void ensayar(int i, boolean *q, int col[], boolean fila[], boolean diagb[],
boolean diagc[])
```

```
{
  int j;

  j = 0;
  *q = FALSE;
  do {
    if (fila[j] && diagb[i+j] && diagc[7+i-j]) {
      col[i] = j; fila[j] = diagb[i+j] = diagc[7+i-j] = FALSE;
      if (i < 7) { /* encuentra solucion? */
        ensayar(i+1,q,col,fila,diagb,diagc);
        if (!*q)
          fila[j] = diagb[i+j] = diagc[7+i-j] = TRUE;
      } else *q = TRUE; /* encuentra la solucion */
    }
    j++;
  } while (!*q && j < 8);
}
```

Por último, se deja al lector que implemente un procedimiento que encuentre todas las soluciones. Si se desea complicar más entonces se puede pedir que encuentre todas las soluciones distintas, es decir, aquellas que no sean rotaciones o inversiones de otras soluciones.

Ahora que se conoce el método general, puede hacerse extensible a múltiples piezas simultáneamente.

6.9.2.3 El problema de la mochila (Selección Optima)

Con anterioridad se ha estudiado la posibilidad de encontrar una única solución a un problema y la posibilidad de encontrarlas todas. Pues bien, ahora se trata de encontrar la mejor solución, la solución óptima, de entre todas las soluciones.

Partiendo del esquema que genera todas las soluciones expuesto anteriormente se puede obtener la mejor solución (la solución óptima, seleccionada entre todas las soluciones) si se modifica la instrucción almacenar solución por esta otra:
si f(solucion) > f(optimo) **entonces** optimo <- solucion
siendo *f(s)* función positiva, *optimo* es la mejor solucion encontrada hasta el momento, y *solucion* es una solucion que se está probando.

El problema de la mochila consiste en llenar una mochila con una serie de objetos que tienen una serie de pesos con un valor asociado. Es decir, se dispone de *n tipos* de objetos y que no hay un número limitado de cada tipo de objeto (si fuera limitado no cambia mucho el problema). Cada tipo *i* de objeto tiene un peso w_i positivo y un valor v_i positivo asociados. La mochila tiene una capacidad de peso igual a *W*. Se trata de llenar la mochila de tal manera que se **maximice** el **valor** de los objetos incluidos pero respetando al mismo tiempo la restricción de capacidad. Notar que no es obligatorio que una solución óptima llegue al límite de capacidad de la mochila.

Ejemplo: se supondrá:
n = 4
W = 8

w() = 2, 3, 4, 5
v() = 3, 5, 6, 10

Es decir, hay 4 tipos de objetos y la mochila tiene una capacidad de 8. Los pesos varían entre 2 y 5, y los valores relacionados varían entre 3 y 10.

Una solución no óptima de valor 12 se obtiene introduciendo cuatro objetos de peso 2, o 2 de peso 4. Otra solución no óptima de valor 13 se obtiene introduciendo 2 objetos de peso 3 y 1 objeto de peso 2. ¿Cuál es la solución óptima?

A continuación se muestra una solución al problema, variante del esquema para obtener todas las soluciones.

```
void mochila(int i, int r, int solucion, int *optimo)
{
 int k;

 for (k = i; k < n; k++) {
  if (peso[k] <= r) {
   mochila(k, r - peso[k], solucion + valor[k], optimo);
   if (solucion + valor[k] > *optimo) *optimo = solucion+valor[k];
  }
 }
}
```

Dicho procedimiento puede ser ejecutado de esta manera, siendo `n`, `W`, `peso` y `valor` variables globales para simplificar el programa:

```
n = 4,
W = 8,
peso[] = {2,3,4,5},
valor[] = {3,5,6,10},
optimo = 0;
mochila(0, W, 0, &optimo);
```

Observar que la solución óptima se obtiene independientemente de la forma en que se ordenen los objetos.

Muchos otros problemas requieren vuelta atrás, en general se trata de algoritmos auxiliares para resolver una pequeña tarea.

También los algoritmos de vuelta atrás permiten la resolución del problema del *laberinto*, puesto que la vuelta atrás no es más que el recorrido sobre un grafo implícito finito o infinito. De todas formas, los problemas de laberintos se resuelven mucho mejor mediante exploración en anchura.

Cabe anotar que los códigos de algoritmos acá presentados han sido tomados de la web para mejor entendimiento y posterior puesta en práctica.

6.9.2.4 El Problema del Laberinto

El problema del laberinto plantea que, dado un laberinto representado con una matriz de booleanos, encontrar un camino para salir del laberinto. Se pueden realizar movimientos desde cada casilla del laberinto a otras cuatro posiciones. Para esto, las casillas a las que se accedan deben ser siempre transitables. El problema consiste en encontrar un camino desde una casilla origen a una casilla destino si es que el camino existe.

- ***Solución***

El laberinto estará representado por la matriz L[1..n,1..n] de booleanos donde L[i, j]= cierto si se puede pasar por la casilla (i, j) y L[i, j]= falso si no se puede pasar por la casilla (i, j) (1 <= i <= n).

Los posibles movimientos desde una casilla serán a las casillas adyacentes vertical y horizontalmente. Es decir, arriba, abajo, izquierda y derecha.

Suponemos que la posición inicial es L[1,1]=cierto y la posición final es L[n, n]=cierto. Por tanto se tratará de obtener el camino de (1,1) a (n, n).

Las soluciones son listas de posiciones del laberinto de longitud n^2 como máximo, en el caso de que no haya obstáculos y se recorran todas las casillas. Dichas soluciones se representarán con el vector sol[1..n^2] de posición.

Respecto a las posiciones que almacenamos en la lista deben cumplir 3 características:

1º) Válidas: no salirse de los límites del laberinto.

2º) Transitables: se puede pasar por ellas (L[i, j]=cierto).

3º) Adyacentes: las posiciones deben ser adyacentes ya que sólo se tienen 4 tipos de movimientos posibles en los que se avanza de 1 en 1.

- **Árbol de Exploración**

Desde cada casilla se pueden realizar 4 movimientos, por lo que el árbol de exploración tendrá n^2 niveles con 4 hijos por nodo (uno por cada movimiento).

- **Implementación en Pseudocódigo**

Para implementar el algoritmo se necesitarán 2 funciones auxiliares. Una para comprobar si una posición del laberinto es correcta, transitable y no visitada. Otra para que nos devuelva la posición a la que se llega desde una casilla aplicando un determinado movimiento.

- **Representación de la posición**

Se usa un registro que indica la fila y la columna de una posición en el laberinto.

```
tipos
  posicion = reg
               fila: 1..n;
               columna: 1..n;
             freg
ftipos
```

El algoritmo de vuelta_atrás del laberinto no desmarca las posiciones ya usadas para que las sucesivas llamadas recursivas no la consideren. Esto se debe a que al volver de la llamada recursiva o la

última posición por la que se ha pasado no lleva a la salida o bien ya se ha terminado.

La búsqueda consiste en probar si es posible obtener un camino usando todos los movimientos posibles. En la búsqueda de un movimiento podemos tener éxito, en cuyo caso se termina. Dentro del procedimiento se llevará una variable auxiliar para controlar el número de movimientos realizados.

```
proc laberinto-va(L[1..n,1..n] de bool, sol[1..n²] de posición, e
k:1..n2, usada[1..n,1..n] de bool, éxito: bool, núm-movimientos:
1..n²)
 m:=1;  //genera movimientos
 mientras m<=4 AND NOT éxito hacer
    Pos:=siguiente(m,sol[k-1])
    si casilla-valida?(L,usada, pos) entonces
      sol[k]:=pos;
      Usada[pos.fila, pos.columna]:=cierto;
      si pos.fila = n AND pos.columna=n entonces
          éxito:=cierto;
          núm-movimientos:=k;
      sino
          laberinto-va(L, sol, k+1,usada,éxito, núm-movimientos);
      fsi
    fsi
    m:= m+1; //siguiente movimiento
 fmientras
fproc
```

- **Llamada inicial**

La llamada inicial llama al procedimiento de vuelta atrás con los parámetros inicializados de manera adecuada.

```
fun laberinto (L[1..n,1..n] de bool) dev <éxito: bool, sol[1..n²] de
posición, núm-movimientos:1..n²>
   var usada [1..n, 1..n] de bool;
   <sol[1].fila, sol[1].columna>:= <1,1>;
   usada[1..n,1..n]:=[falso]; usada [1,1]:=cierto;
```

```
    éxito:=falso;
    laberinto-va(L,sol,2,usada,éxito,núm-movimientos);
  ffun
```

- **Función Auxiliar1**

Función que valida una posición. Para ello debe comprobar las características que se han mencionado anteriormente.

```
  fun casilla-valida?(L[1..n],usada[1..n,1..n] de bool, pos:posicion) dev
respuesta: bool
    (i,j):= <pos.fila,pos.columna>;
    respuesta:= (1<=i) AND (i<=n) AND (1<=j) AND (j<=n) AND (L[i,j]
AND NOT usada[i,j];
  ffun
```

- **Función Auxiliar2**

La función auxiliar siguiente devuelve la casilla a la que se debe pasar según el número de movimiento.

```
  fun siguiente(m:1..4, act: posicion) dev sig: posicion
    (i,j):= (act.fila,act.columna);
    casos
       m=1 --> (i',j')=(i+1,j);
       m=2 --> (i',j')=(i,j+1);
       m=3 --> (i',j')=(i-1,j);
       m=4 --> (i',j')=(i,j-1);
    fcasos
    (sig.fila,sig.columna)=(i',j');
    dev sig;
  ffun
```

6.9.3 Algoritmo Genérico de Backtracking

```
Bt(A, k)
1 if Solucion(A, k)
2 then procesarSolucion(A, k)
3 else for each c 2 Sucesores(A, k)
4 do A[k] = c
5 Bt(A, k + 1)
```

6 if terminar
7 then return

Donde:

• Solución: es una función que retorna verdadero si su argumento es una solución.

• ProcesarSolucion: depende del problema y que maneja una solución.

• Sucesores (·) es una función que dado un candidato, genera todos los candidatos que son extensiones de éste.

• Terminar. es una variable global booleana inicialmente es falsa, pero que puede ser hecha verdadera por procesar Solución, en caso que sólo interesa encontrar.

6.10 CAPITULO 10

Aplicación a desarrollar

Con la investigación se prentente:

- Objetivo principal: comparar métodos tradicionales y técnicas heurísticas - meta heurísticas, teniendo en cuenta la calidad de la solución, tiempo para la obtención de la solución, números de soluciones encontradas, etc.
- Técnica heurística a utilizar: Backtracking
- Técnica meta heurística escogida para la comparación: Ant-Q (colonias de Hormigas), exploratoria.
- Problema a resolver salida de un lugar mediante el uso de laberintos.
- Definición del problema
  - Laberintos: presenta condiciones tales como: no se reconfiguran dinámicamente, no tiene trampas, tiene una y solo una salida que es alcanzable desde cualquier parte del laberinto.

Laberinto utilizando Backtracking

1. El laberinto se toma como una matriz.

2. El tamaño de la matriz es suministrado por el usuario al igual el lugar donde habrá caminos y obstáculos respectivamente, ubicación de la salida, ubicación de las hormigas para iniciar su recorrido.

Se tuvo en cuenta:

1. El individuo arranca desde la posición asignada por el usuario.

2. Comienza a desplazarse por todo el laberinto pegado a la pared y en dirección derecha.

3. se trabaja con cuatro posiciones tales como: derecha, izquierda, arriba, abajo.

4. La hormiga solo se moverán si hay camino, en caso de que no hayan camino se devolverá hasta que encuentre la salida.

5. Una ves hecho los recorridos el algoritmo arrojará datos de cuantos pasos fueron dados por la hormiga para encontrar la salida, asi como el tiempo de ejecución que tarda el algoritmo e imprimirá una matriz con los pasos dados.

Laberinto utilizando Colonias de hormigas

1. El laberinto se toma como una matriz.

2. El tamaño de la matriz es suministrado por el usuario al igual el lugar donde habrá caminos y obstáculos respectivamente, ubicación de la salida, ubicación de las hormigas para iniciar su recorrido.

Se tuvo en cuenta:

1. El individuo arranca desde la posición asignada por el usuario.

2. Comienza a desplazarse por todo el laberinto haciendo movimientos como arriba, abajo, derecha, izquierda.

3. Se trabaja con cuatro posiciones tales como: derecha, izquierda, arriba, abajo.

4. La hormiga solo se moverán si hay camino, en caso de que no hayan camino morirá, si encuentra camino se guiará por el rastro de feromona trabajando con probabilidades, en caso de que no haya pasado hormiga alguna por el camino esta escogerá el camino que desee de acuerdo a las probabilidades de escogencia del camino (arriba, abajo, izquierda, derecha).

5. Se trabajaran diferentes laberintos y se guardaran los recorridos para luego comparar y ver que tan bueno es el rendimiento de los algoritmos puestos en práctica.

6. Una ves hecho los recorridos el algoritmo arrojará datos de cuantos pasos fueron dados por la hormiga que logró encontrar la salida, asi como el tiempo de ejecución que tarda el algoritmo e imprimirá una matriz con los pasos dados.

Pruebas utilizando matrices de 20*20 utilizando los dos algoritmos Backtracking y Colonia de Hormigas, los resultados fueron los siguientes:

Matriz de 20 X 20

Backtracking

Numero de obstáculos: 86
Cantidad de pasos: 22
Pasos:[1,1][2,1][3,1][4,1][5,1][6,1][7,1][8,1][9,1][10,1][10,2][9,2][9,3][9,4][9,5][9,6][9,7][10,7][10,8][10,9][10,10][9,10]
Tiempo: 1.093 ms

Colonia de hormiga

Numero de obstáculos: 45
Numero de individuos: 20
Cantidad de pasos: 19
Pasos: [1,1] [2,1] [2,2] [2,3] [2,4] [2,5] [2,6] [2,7] [2,8] [2,9] [2,10] [3,9] [4,9] [5,9] [6,9] [7,9] [8,9] [9,9] [9,10]
Cantidad de pasos: 19
Tiempo: 20.136 ms

Backtracking

Numero de obstáculos: 72
Cantidad de pasos: 58
Pasos:[1,1][2,1][3,1][4,1][5,1][6,1][7,1][8,1][9,1][10,1] [11,1] [12,1] [13,1] [14,1] [15,1] [16,1] [17,1] [18,1] [19,1] [20,1] [20,2] [19,2] [19,3] [19,4] [19,5] [19,6] [19,7] [19,8] [19,9] [19,10] [19,11] [19,12] [19,13] [19,14] [19,15] [20,15] [20,16] [20,17] [20,18] [20,19] [20,20] [19,20] [18,20] [18,19] [18,18] [18,17] [18,16] [18,15] [18,14] [18,13] [17,13] [16,13] [16,14] [16,15] [16,16] [16,17] [16,18] [16,19] [16,20]
Tiempo: 32.259 ms

Colonia de hormiga

Numero de obstáculos: 72
Numero de individuos: 25
Pasos: [1,1] [2,1] [3,1] [4,1] [5,1] [6,1] [7,1] [8,1] [9,1] [9,2] [9,3] [9,4] [9,5] [9,6] [9,7] [9,8] [9,9] [10,9] [11,11] [12,11] [13,11] [14,11] [15,11] [15,12] [16,12] [16,13] [16,14] [16,15] [16,16] [16,17] [16,18] [16,19] [16,20]
Cantidad de pasos: 33
Tiempo: 62.351 ms

Backtracking

Numero de obstáculos: 95
Cantidad de pasos: 65
Pasos: [1,1][2,1][2,2] [2,3] [2,4] [2,5] [2,6] [2,7] [3,7] [4,7] [5,7] [6,7] [7,7] [8,7] [9,7] [9,6] [9,5] [8,5] [7,5] [6,5] [5,5] [4,5] [4,4] [4,3] [4,2] [4,1] [5,1] [6,1] [7,1] [8,1] [9,1] [10,1] [11,1] [12,1] [13,1] [14,1] [15,1] [16,1] [17,1] [18,1] [19,1] [20,1] [20,2] [19,2] [19,3] [19,4] [19,5] [19,6] [19,7] [19,8] [19,9] [19,10] [19,11] [19,12] [19,13] [18,13] [17,13] [16,13] [16,14] [16,15] [16,16] [16,17] [16,18] [16,19] [16,20]
Tiempo: 26.851 ms

Colonia de Hormiga

Numero de individuos: 40
Numero de obstáculos: 95
Pasos: 35
Pasos: [1,1] [2,1][2,2] [2,3] [2,4] [2,5] [2,6] [2,7][2,8][3,8] [4,8][4,9][4,10][4,11] [4,12] [4,13] [4,14] [4,15] [4,16] [516] [6,16] [7,16] [8,16] [9,16] [10,16] [11,16] [12,16] [12,17] [12,18] [12,19] [13,19] [14,19] [15,19] [16,19] [16,20]
Tiempo: 48.542 ms

Backtracking

Numero de Obstaculos: 100
Cantidad de pasos: 39
Pasos: [1,1] [2,1] [2,2] [2,3] [2,4] [2,5] [2,6] [2,7] [3,7] [4,7] [5,7] [6,7] [8,7] [9,7] [10,7] [11,7] [12,7] [13,7] [14,7] [15,7] [16,7] [17,7] [17,8]

[17,9] [17,10] [17,11] [17,12] [17,13] [17,14] [17,15] [16,15] [16,16] [16,17] [16,18] [16,19] [16,20]
[15,20] [14,20] [13,20]
Tiempo: 21.345 ms

Colonia de Hormiga

Numero de Individuos: 80
Numero de Obstaculos: 100
Cantidad Pasos: 32
Pasos: [1,1] [1,2] [1,3] [1,4] [1,5] [1,6] [1,7] [1,8] [1,9] [1,10] [1,11] [1,12] [1,13] [1,14] [1,15] [1,16] [1,17] [1,18] [1,19] [1,20] [2,20] [3,20] [4,20] [5,20] [6,20] [7,20] [8,20] [9,20] [10,20] [11,20] [12,20] [13,20]
Tiempo: 75.324 ms

Backtracking

Numero de Obstaculos: 66
Cantidad de pasos: 64
Pasos: [1,1][1,2] [2,2] [3,2] [4,2] [5,2] [6,2] [6,3] [6,4] [6,5] [6,6] [6,7] [6,8] [6,9] [6,10] [6,11] [5,11] [4,11] [3,11] [3,12] [3,13] [3,14] [4,14] [5,14] [6,14] [7,14] [8,14] [8,13] [8,12] [8,11] [8,10] [8,9] [8,8] [8,7] [8,6] [8,5] [9,5] [10,5] [11,5] [12,5] [13,5] [14,5] [15,5] [15,6] [15,7] [15,8] [15,9] [15,10] [15,11] [15,12] [14,12] [13,12] [12,12] [11,12] [11,14] [11,15] [11,16] [11,17] [12,17] [14,17] [15,17] [16,17]
Tiempo: 27.947

Colonia de Hormiga

Numero de Individuos: 100
Numero de Obstaculos: 66
Cantidad Pasos: 34
Pasos: [1,1] [2,1] [3,1] [4,1] [5,1] [6,1] [7,1] [8,1] [9,1] [10,1] [11,1] [12,1] [13,1] [13,2] [14,2] [15,2] [16,2] [17,2] [17,3] [17,4] [17,5] [17,6] [17,7] [17,8] [17,9] [17,10] [17,11] [17,12] [17,13] [17,14] [17,15] [17,16] [17,17] [16,17]
Tiempo: 56.154

Backtracking

Numero de Obstaculos: 64
Cantidad de pasos: 40
Pasos: [1,1] [2,1] [2,2] [3,2] [4,2] [5,2] [6,2] [7,2] [7,1] [8,1] [9,1] [10,1] [10,2] [11,2] [12,2] [13,2] [13,3] [13,4] [13,5] [13,6] [12,6] [11,6] [10,6] [9,6] [9,7] [9,8] [9,9] [9,10] [9,11] [9,12] [10,12] [11,12] [12,12] [13,13] [14,12] [14,13] [14,14] [14,15] [14,16] [14,17]
Tiempo: 10.214

Colonia de Hormiga

Numero de Individuos: 30
Numero de Obstaculos: 64
Cantidad Pasos: 32
Pasos: [1,1] [1,2] [1,3] [1,4] [2,4] [3,4] [3,5] [3,6] [3,7] [3,8] [3,9] [3,10] [3,11] [3,12] [3,13] [3,14] [3,15] [3,16] [4,16] [5,16] [6,16] [7,16] [7,17] [7,18] [8,18] [9,18] [10,18] [11,18] [12,18] [13,18] [14,18] [14,17]
Tiempo: 51.268

Backtracking

Numero de Obstaculos: 49
Cantidad de pasos: 56
Pasos: [1,1] [1,2] [1,3] [1,4] [1,5] [2,5] [3,5] [4,5] [5,5] [6,5] [7,5] [7,4] [7,3] [7,2] [7,1] [8,1] [9,1] [10,1] [11,1] [12,1] [12,1] [12,3] [12,4] [12,5] [12,6] [12,7] [12,8] [12,9] [11,9] [10,9] [9,9] [8,9] [7,9] [7,10] [7,11] [7,12] [7,13] [7,14] [7,15] [7,16] [7,17] [8,17] [9,17] [10,17] [11,17] [12,17] [12,16] [12,15] [12,14] [13,14] [14,14] [15,14] [16,14] [16,15] [16,16] [16,17]
Tiempo: 12.521

Colonia de Hormiga

Numero de Individuos: 90
Numero de Obstaculos: 49
Cantidad Pasos: 32
Pasos: [1,1] [2,1] [2,2] [3,2] [2,4] [2,5] [2,6] [2,7] [2,8] [2,9] [2,10] [2,11] [2,12] [2,13] [2,14] [2,15] [2,16] [2,17] [3,17] [4,17] [5,17] [6,17] [7,17] [8,17] [9,17] [10,17] [11,17] [12,17] [13,17] [14,17] [15,17] [16,17]
Tiempo: 55.214

Matriz con variaciones de tamaño

Backtracking

Numero De Obstaculos: 20
Cantidad De Pasos: 23
Pasos: 1 1, 1 2, 1 3, 1 4, 2 4, 2 5, 3 5, 4 5, 4 4, 4 3, 4 2, 4 1, 4 2, 5 2, 6 2, 6 1, 6 2, 6 3, 6 4, 6 5, 6 6, 5 6, 4 6
Tiempo: 7.157

Colonia de Hormiga

Numero De Individuos: 50
Cantidad De Obstaulos: 20
Cantidad Pasos: 9
Pasos: 1 1, 1 2, 1 3, 1 4, 2 4, 2 5, 3 5, 4 5, 4 6
Tiempo: 36.541

Backtracking

Numero de Obstaculos: 28
Cantidad Pasos: 20
Pasos: 8 8, 8 7, 7 7, 7 6, 6 6, 6 5, 7 5, 7 4, 8 4, 8 5, 9 5, 10 4, 10 5, 10 6, 9 6, 9 7, 9 8, 10 8, 10 9, 10 10
Tiempo: 8.245

Colonia de Hormiga
Numero De Individuos: 50
Cantidad De Obstaulos: 28
Cantidad Pasos: 9
Pasos: 8 8, 8 9, 7 9, 7 10, 7 11, 8 11, 9 11, 10 11, 10 10
Tiempo: 38.542

Comparativo entre Algoritmos

Laberinto	Backtracking	ANTQ	Ctdad de pasos	T. Ejecución	Obstáculos	Media	Varianza
1	X		22	1.093 ms	86	36,364	1301.9519
1		X	19	20.136	86	41.712	953.721
2	X		58	32.259	72	54.086	211.957
2		X	33	62.139	72	55.713	240.366
3	X		65	26.851	93	61.617	689.372
3		X	35	48.542	93	58.847	573.078
4	X		39	21.345	100	53.448	3923.71
4		X	32	75.324	100	69.108	734.326
5	X		64	27.947	66	52.649	19.820
5		X	34	56.154	66	52.051	150.233
6	X		40	10.214	64	38.071	464.705
6		X	32	51.268	64	49.089	147.244
7	X		56	12.521	49	39.173	1972.168
7		X	32	55.214	49	45.404	1103.066
8	X		23	7.157	20	16.719	306.754
8		X	9	36.541	20	21.847	577.513
9	X		20	8.245	28	18.748	394.405
9		X	9	38.542	28	25.180	750.480

Código del los laberintos Backtracking y Colonia de Hormigas

```
#include<iostream.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
#include<stdio.h>
#include<math.h>
#include<time.h>
#include"puperidad4.h"
struct nodo3//
{
int persona;
int iii;
int jjj;
nodo3 *sig2;
nodo3 *abajo2;
};
struct nodo//
{
int persona;
int ii;
int jj;
nodo *sig;
nodo *abajo;
};
struct nodo2//
{
int iii;
int jjj;
int porcentaje;
nodo2 *sig;
nodo2 *abajo;
};
void eliminar(nodo *&newpasos,nodo *&hormi,nodo *&cab);
void laberinto1(int M[50][50],int x1,int x2,int i,int j,int fil,int col,int fil2,int
col2);//int &cont,nodo3 *&cab,double &tiempo1);
void laberinto2(int M[50][50],int x1,int x2,int i,int j,int fil,int col,int fil2,int
col2);//nodo *&asd,int &cont,int &ite,double &tiempo2);
void inicio();
void about();
void descarga()
```

```
{
int swt=1;
cout<<endl;
printf("\tCARGANDO ");
printf("[°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°]\r");
printf("\tCARGANDO ");
for(int i=1; i<=44; i++)
{
	if(swt==0)
	{
		printf("Û");
	}
	else{
		printf("[");
		swt=0;
	}
	for(int j=1; j<=33333333; j++)
	{
	}
}
cout<<endl;
}
void opcion(int &k)
{
int x,cont=3;
cout<<endl;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  º SELECCIONE SU OPCION: º"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl<<endl;
printf("\t\t   ÉÍ» ÉÍ» ÉÍ»\n");
printf("\t\t   º1º º2º º3º\n");
printf("\t\t   ÈÍ¼ ÈÍ¼ ÈÍ¼\n");
printf("\t\t            ");
for(;;)
{
	x=getch();
	if((x==75)&&(cont>=2))
	{
		for(int i=1; i<=4; i++)
		{
			printf("\b");
		}
```

```
                cont--;k=cont;
        }else{
                if((x==77)&&(cont<=2))
                {
                        for(int j=1; j<=4; j++)
                        {
                                printf(" ");
                        }
                        cont++;k=cont;
                }else{
                        if(x==13)
                        {
                                cout<<cont;
                                break;
                        }
                }
        }
}
}
void main()
{
int op=3;
system("cls");
   cout<<endl<<endl<<endl<<endl<<endl<<endl<<endl;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  º              ÉÍÍÍÍÍÍÍ»             ºÛ"<<endl;
cout<<"\t\t  º              º AVISO ºÛ             ºÛ"<<endl;
cout<<"\t\t  º              ÈÍÍÍÍÍÍÍ¼Û            ºÛ"<<endl;
cout<<"\t\t  º               ÛÛÛÛÛÛÛÛÛÛ             ºÛ"<<endl;
cout<<"\t\t  ÌÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹Û"<<endl;
cout<<"\t\t  º                                     ºÛ"<<endl;
cout<<"\t\t  º  OPRIMA ALT+ENTER SIMULTANEAMENTE PARA
ºÛ"<<endl;
cout<<"\t\t  º                                     ºÛ"<<endl;
cout<<"\t\t  º    APRECIAR MEJOR EL PROGRAMA QUE
ºÛ"<<endl;
cout<<"\t\t  º                                     ºÛ"<<endl;
cout<<"\t\t  º  A CONTINUACION OBSERVARA Y OPRIMA UNA
ºÛ"<<endl;
cout<<"\t\t  º                                     ºÛ"<<endl;
cout<<"\t\t  º        TECLA PARA CONTINUAR .        ºÛ"<<endl;
cout<<"\t\t  º                                     ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
```

```
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"
<<endl;
cout<<endl<<endl<<endl<<endl;
op=getch();
inicio();
}
void inicio()
{
char hora[50];
char f[50];
_strtime(hora);
_strdate(f);
int op=2;
int op2=3;
while(op!=3)
{
	op=3;
	system("cls");
	cout<<endl<<endl<<endl<<endl<<endl;
	cout<<"\t\t\t\t\tÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»";   cout<<"  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
	cout<<"\t\t\t\t\tº FECHA: "<<f<<"ºÛ";   cout<<" º HORA:
"<<hora<<"ºÛ"<<endl;
	cout<<"\t\t\t\t\tÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û";  cout<<" ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
	cout<<"\t\t\t\t\t ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ";  cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
	cout<<endl;
	printf("                              ÉÍÍÍ»
");cout<<endl;
	printf("                              ºMENUºÛ
");cout<<endl;
	printf("                              ÈÍÍÍ¼Û
");cout<<endl;
	printf("                               ÛÛÛÛÛÛ
");cout<<endl;cout<<endl;
	printf("\t\t   ÉÍÍÍÍÍÍËÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»");cout<<endl;
	printf("\t\t   º ÉÍÍÍ»  º ÉÍÍÍÍÍÍÍÍÍÍÍ»    ºÛ");cout<<endl;
	printf("\t\t   º º 1 ºÛ º º ALGORITMOS ºÛ   ºÛ");cout<<endl;
	printf("\t\t   º ÈÍÍÍ¼Û º ÈÍÍÍÍÍÍÍÍÍÍÍ¼Û   ºÛ");cout<<endl;
	printf("\t\t   º  ÛÛÛÛÛ º  ÛÛÛÛÛÛÛÛÛÛÛÛÛ
ºÛ");cout<<endl;
	printf("\t\t   ÌÍÍÍÍÍÍÎÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹Û");cout<<endl;
	printf("\t\t   º ÉÍÍÍ»  º ÉÍÍÍÍÍÍÍÍÍÍÍ»  ºÛ");cout<<endl;
```

```
        printf("\t\t  º º 2 ºÛ º º PRESENTACION ºÛ ºÛ");cout<<endl;
        printf("\t\t  º ÈÍÍͼÛ º ÈÍÍÍÍÍÍÍÍÍÍÍÍͼÛ ºÛ");cout<<endl;
        printf("\t\t  º  ÛÛÛÛÛ º  ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ºÛ");cout<<endl;
        printf("\t\t  ÌÍÍÍÍÍÍÍÎÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¹Û");cout<<endl;
        printf("\t\t  º ÉÍÍÍ»  º ÉÍÍÍÍÍÍ»        ºÛ");cout<<endl;
        printf("\t\t  º º 3 ºÛ º º SALIR ºÛ        ºÛ");cout<<endl;
        printf("\t\t  º ÈÍÍͼÛ º ÈÍÍÍÍÍͼÛ        ºÛ");cout<<endl;
        printf("\t\t  º  ÛÛÛÛÛ º  ÛÛÛÛÛÛÛÛÛ        ºÛ");cout<<endl;
        printf("\t\t  ÈÍÍÍÍÍÍÍÊÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍͼÛ");cout<<endl;
        printf("\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ");cout<<endl;
        opcion(op);
        switch (op)
        {
        case 1:
                        system("cls");
                        op2=1;

                        int x1,x2,i,j,fil,col,fil2,col2;
                        char res,respuesta;
                        int M[50][50];

                        while(op2!=3)
                        {
                                system("cls");
                                op2=3;
cout<<endl<<endl<<endl<<endl<<endl;
cout<<"\t\t\t\t\tÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»";   cout<<"  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t\t\t\tº FECHA: "<<f<<"ºÛ";   cout<<" º HORA:
"<<hora<<"ºÛ"<<endl;
cout<<"\t\t\t\t\tÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍͼÛ";  cout<<" ÈÍÍÍÍÍÍÍÍÍÍÍÍͼÛ"<<endl;
cout<<"\t\t\t\t\t ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ";  cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<endl;

printf("                        ÉÍÍÍ»                    ");cout<<endl;
printf("                        ºMENUºÛ
");cout<<endl;
printf("                        ÈÍÍͼÛ                    ");cout<<endl;
printf("                         ÛÛÛÛÛ                    ");
cout<<endl;cout<<endl;
printf("\t\t  ÉÍÍÍÍÍÍÍËÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»");cout<<endl;
```

```
printf("\t\t  º Èííí»  º Èíííííííííííííí»        ºÛ");cout<<endl ;
printf("\t\t  º º 1 ºÛ º º Backtracking ºÛ       ºÛ");cout<<endl;
printf("\t\t  º Èííí¼Û º Èíííííííííííííí¼Û      ºÛ");cout<<endl;
printf("\t\t  º  ÛÛÛÛÛ º  ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ      ºÛ");cout<<endl;
printf("\t\t  ÌííííííííÎíííííííííííííííííííííííííí¹Û");cout<<endl;
printf("\t\t  º Èííí»  º Èííííííííííííííííííí»  ºÛ");cout<<endl;
printf("\t\t  º º 2 ºÛ º º Colonia de Hormiga ºÛ ºÛ");cout<<endl;
printf("\t\t  º Èííí¼Û º Èííííííííííííííííííí¼Û ºÛ");cout<<endl;
printf("\t\t  º  ÛÛÛÛÛ º  ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ºÛ");cout<<endl;
printf("\t\t  ÌííííííííÎíííííííííííííííííííííííííí¹Û");cout<<endl;
printf("\t\t  º Èííí»  º Èííííííí»              ºÛ");cout<<endl;
printf("\t\t  º º 3 ºÛ º º SALIR ºÛ            ºÛ");cout<<endl;
printf("\t\t  º Èííí¼Û º Èííííííí¼Û            ºÛ");cout<<endl;
printf("\t\t  º  ÛÛÛÛÛ º  ÛÛÛÛÛÛÛÛÛÛ            ºÛ");cout<<endl;
printf("\t\t  ÈííííííííÊíííííííííííííííííííííííííí¼Û");cout<<endl;
printf("\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ");cout<<en
dl;
        opcion(op2);
        switch (op2)
        {
        case 1:
        system("cls");
        cout<<"\t\t  Èííííííííííííííííííííííííííí»"<<endl;
        cout<<"\t\t  ºDESA LLENAR LA MATRIZ: S/NºÛ"<<endl;
        cout<<"\t\t  Èííííííííííííííííííííííííííí¼Û"<<endl;
        cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
        cout<<"\t\t\t\t ";cin>>respuesta;
                if((respuesta=='s')||(respuesta=='S'))
                {
                cout<<"\t\t  Èíííííííííííííííííííííííííííííííííííííííííí»"<<endl;
cout<<"\t\t  ºDIGITE LA LONGITUD DE LA FILA DE LA
MATRIZ:ºÛ"<<endl;
cout<<"\t\t  Èíííííííííííííííííííííííííííííííííííííííííí¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>x1;
cout<<"\t\t  Èíííííííííííííííííííííííííííííííííííííííííííí»"<<endl;
cout<<"\t\t  ºDIGITE LA LONGITUD DE LA COLUMNA DE LA
MATRIZ:ºÛ"<<endl;
```

```
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>x2;
system("cls");
int res1;
for(i=1; i<=x1; i++)
        {
        for(j=1; j<=x2; j++)
        {
        cout<<"  FILA NUMERO "<<i<<":"<<endl;
        cout<<"  DESDE LA POSICION "<<i<<" CON COLUMNA
"<<j<<" HAY CAMINO O NO s/n :"<<endl;
        cout<<"\t\t ";cin>>res;
        int sw;
        if((res=='s')||(res=='S'))
        {
        sw=0;

while(sw==0)
{
cout<<"  DESDE LA POSICION "<<i<<" CON COLUMNA "<<j<<"
HASTA QUE COLUMNA HAY CAMINO :"<<endl;
cout<<"\t\t ";cin>>res1;
if((res1<=x2)&&(res1>=j))
{
for(int j2=j; j2<=res1; j2++)
{
M[i][j2]=2;
system("cls")
}
sw=1;
}else{
cout<<"EL VALOR DIGITADO DEBE SER MENOR O IGUAL QUE LA
COLUMNA";
}
}
}else{
sw=0;
while(sw==0)
                                                  {
```

```
cout<<"  DESDE LA POSICION "<<i<<" CON COLUMNA "<<j<<"
HASTA QUE COLUMNA NO HAY CAMINO :"<<endl;
cout<<"\t\t ";cin>>res1;
if((res1<=x2)&&(res1>=j))
{
for(int j2=j; j2<=res1; j2++)
{
M[i][j2]=1;
system("cls");
}
sw=1;
}else{
cout<<"EL VALOR DIGITADO DEBE SER MENOR O IGUAL QUE LA
COLUMNA";
                                }
                        }
                }
                j=res1;
        }
}

cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<endl;cout<<"É";
for(i=1; i<=x1+4; i++)//
{
cout<<"Í";
}
cout<<"»"<<endl;
for(i=1; i<=x1; i++)
{
cout<<"º ";
for(j=1; j<=x2; j++)
{
if(M[i][j]==2)
{
cout<<" "<<" ";
}
if(M[i][j]==9)
{
```

```
cout<<"X"<<" ";
}
if(M[i][j]==1)
{
cout<<"Û"<<" ";
}
if(M[i][j]==3)
{
cout<<"@"<<" ";
}
}cout<<"º"<<endl;
}
cout<<"È";
for(i=1; i<=x1+4; i++)
{
cout<<"Í";
}
cout<<"¼";
cout<<endl;
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<"\t\t ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t ºDIGITE LA POSICION DE LA SALIDAºÛ"<<endl;
cout<<"\t\t ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t ºDIGITE LA FILA:ºÛ"<<endl;
cout<<"\t\t ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t  ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>fil;
cout<<"\t\t ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t ºDIGITE LA COLUMNA:ºÛ"<<endl;
cout<<"\t\t ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t  ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>col;
        M[fil][col]=9;
        system("cls");
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
```

```
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<endl;cout<<"\t\t\t\t\tÉ";
        for(i=1; i<=x1+4; i++)
        {
        cout<<"Í";
        }
        cout<<"»"<<endl;
        for(i=1; i<=x1; i++)
        {
        cout<<"º ";
        for(j=1; j<=x2; j++)
        {
        if(M[i][j]==2)
        {
        cout<<" "<<" ";
        }
        if(M[i][j]==9)
        {
        cout<<"X"<<" ";
        }
        if(M[i][j]==1)
        {
        cout<<"Û"<<" ";
        }
        if(M[i][j]==3)
        {
        cout<<"@"<<" ";
        }
        }cout<<"º"<<endl;
        }
        cout<<"\t\t\t\t\tÈ";
        for(i=1; i<=x1+4; i++)
        {
                cout<<"Í";
                }
                cout<<"¼";
                cout<<endl;
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
int sw;
```

```
sw=0;
while(sw==0)
{
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA POSICION DE LA PARTIDAºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA FILA:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>fil2;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA COLUMNA:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>col2;
if((fil!=fil2)||(col!=col2))
        sw=1;
        else

cout<<"ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"ºLAS COORDENADAS NO PUEDEN SER LAS MISMAS QUE
DIGITO EN LA SALIDA!!!ºÛ"<<endl;
cout<<"ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
        }
M[fil2][col2]=3;
system("cls");
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<endl;cout<<"É";
        for(i=1; i<=x1+4; i++)
        {
        cout<<"Í";
        }
        cout<<"»"<<endl;
```

```
for(i=1; i<=x1; i++)
{
cout<<"º ";
for(j=1; j<=x2; j++)
{
if(M[i][j]==2)
{
cout<<" "<<" ";
}
if(M[i][j]==9)
{
cout<<"X"<<" ";
}
if(M[i][j]==1)
{
cout<<"Û"<<" ";
}
if(M[i][j]==3)
{
cout<<"@"<<" ";
}
}cout<<"º"<<endl;
}
cout<<"È";
for(i=1; i<=x1+4; i++)
{
cout<<"Í";
}
cout<<"¼";
cout<<endl;
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<
endl;
}else{
        for(i=1; i<=x1; i++)
        {
        for(j=1; j<=x2; j++)
        {
        if(M[i][j]==3)
        {
        M[i][j]=2;
```

```
					}
				}
			}
				for(i=1; i<=x1; i++)
				{
					for(j=1; j<=x2; j++)
					{
					if((col2==j)&&(fil2==i))
					{
					M[i][j]=3;
					}
					if((col==j)&&(fil==i))
					{
						M[i][j]=9;
							}
						}
					}
				}

laberinto1(M,x1,x2,i,j,fil,col,fil2,col2);//cont,cab,tiempo1);
break;
case 2:
system("cls");
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDESA LLENAR LA MATRIZ: S/NºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>respuesta;
if((respuesta=='s')||(respuesta=='S'))
{
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA LONGITUD DE LA FILA DE LA
MATRIZ:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>x1;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA LONGITUD DE LA COLUMNA DE LA
MATRIZ:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
```

```
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>x2;
system("cls");
int res1;
for(i=1; i<=x1; i++)
        {
        for(j=1; j<=x2; j++)
        {
        cout<<"  FILA NUMERO "<<i<<":"<<endl;
        cout<<"  DESDE LA POSICION "<<i<<" CON COLUMNA
"<<j<<" HAY CAMINO O NO s/n :"<<endl;
        cout<<"\t\t ";cin>>res;
        int sw;
        if((res=='s')||(res=='S'))
        {
        sw=0;
        while(sw==0)
        {
        cout<<"  DESDE LA POSICION "<<i<<" CON COLUMNA
"<<j<<" HASTA QUE COLUMNA HAY CAMINO :"<<endl;
        cout<<"\t\t ";cin>>res1;
        if((res1<=x2)&&(res1>=j))
        {
        for(int j2=j; j2<=res1; j2++)
        {
        M[i][j2]=2;
        system("cls");
        }
        sw=1;
        }else{
        cout<<"EL VALOR DIGITADO DEBE SER MENOR O IGUAL
QUE LA COLUMNA";
                }
        }
        }else{
                sw=0;
                while(sw==0)
                {
        cout<<"  DESDE LA POSICION "<<i<<" CON COLUMNA
"<<j<<" HASTA QUE COLUMNA NO HAY CAMINO :"<<endl;
```

```
cout<<"\t\t ";cin>>res1;
        if((res1<=x2)&&(res1>=j))
        {
        for(int j2=j; j2<=res1; j2++)
        {
        M[i][j2]=1;
        system("cls");
        }
        sw=1;
        }else{
        cout<<"EL VALOR DIGITADO DEBE SER MENOR O IGUAL
QUE LA COLUMNA";
                }
        }
}
j=res1;
}
}
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<endl;cout<<"\t\t\t\t\tÉ";
for(i=1; i<=x1+4; i++)//imprime la matriz en pantalla
        {
        cout<<"Í";
        }
        cout<<"»"<<endl;
        for(i=1; i<=x1; i++)
        {
        cout<<"\t\t\t\t ";cout<<"\tº ";
        for(j=1; j<=x2; j++)
        {
        if(M[i][j]==2)
                {
                cout<<" "<<" ";
                }
                if(M[i][j]==9)
                {
                cout<<"X"<<" ";
                }
                if(M[i][j]==1)
```

```
{
cout<<"Û"<<" ";
}
if(M[i][j]==3)
{
cout<<"@"<<" ";
}
}cout<<"º"<<endl;
}
cout<<"\t\t\t\t\tÈ";
for(i=1; i<=x1+4; i++)
{
cout<<"Í";
}
cout<<"¼";
cout<<endl;
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA POSICION DE LA SALIDAºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA FILA:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>fil;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA COLUMNA:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>col;
        M[fil][col]=9;
        system("cls");

cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<endl;cout<<"\t\t\t\t\tÉ";
```

```
for(i=1; i<=x1+4; i++)
{
cout<<"Í";
}
cout<<"»"<<endl;
for(i=1; i<=x1; i++)
{
cout<<"\t\t\t\t ";cout<<"\tº ";
for(j=1; j<=x2; j++)
{
if(M[i][j]==2)
{
cout<<" "<<" ";
}
if(M[i][j]==9)
{
cout<<"X"<<" ";
}
if(M[i][j]==1)
{
cout<<"Û"<<" ";
}
if(M[i][j]==3)
{
cout<<"@"<<" ";
        }
}cout<<"º"<<endl;
}
cout<<"\t\t\t\t\tÈ";
    for(i=1; i<=x1+4; i++)
    {
    cout<<"Í";
    }
    cout<<"¼";
    cout<<endl;
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
  int sw;
  sw=0;
  while(sw==0)
  {
```

```
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA POSICION DE LA PARTIDAºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA FILA:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>fil2;
cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"\t\t  ºDIGITE LA COLUMNA:ºÛ"<<endl;
cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t   ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>col2;
        if((fil!=fil2)||(col!=col2))
                sw=1;
                else
cout<<"ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"ºLAS COORDENADAS NO PUEDEN SER LAS MISMAS QUE
DIGITO EN LA SALIDA!!!ºÛ"<<endl;
cout<<"ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
        }
        M[fil2][col2]=3;
        system("cls");
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;

        cout<<endl;cout<<"\t\t\t\t\tÉ";
        for(i=1; i<=x1+4; i++)
        {
                cout<<"Í";
        }
        cout<<"»"<<endl;
        for(i=1; i<=x1; i++)
        {
                cout<<"\t\t\t\t ";cout<<"\tº ";
                for(j=1; j<=x2; j++)
```

```
{
        if(M[i][j]==2)
{
cout<<" "<<" ";
}
        if(M[i][j]==9)
{
cout<<"X"<<" ";
}
if(M[i][j]==1)
{
cout<<"Û"<<" ";
}
if(M[i][j]==3)
{
cout<<"@"<<" ";
        }
}cout<<"º"<<endl;
        }
        cout<<"\t\t\t\tÈ";
        for(i=1; i<=x1+4; i++)
        {
                cout<<"Í";
        }
        cout<<"¼";
        cout<<endl;
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;

        }else{
                for(i=1; i<=x1; i++)
                {
                for(j=1; j<=x2; j++)
                {
                if(M[i][j]==3)
                {
                M[i][j]=2;
                }
        }
        }
        for(i=1; i<=x1; i++)
```

```
                    {
                    for(j=1; j<=x2; j++)
                    {
                    if((col2==j)&&(fil2==i))
                    {
                    M[i][j]=3;
                    }
                            if((col==j)&&(fil==i))
                            {
                            M[i][j]=9;
                                    }
                            }
                    }
            }

laberinto2(M,x1,x2,i,j,fil,col,fil2,col2);//asd,cont2,ite,tiempo2);
break;
case 3:
op2=3;
break;
            }
        }
        break;
        case 2:
        system("cls");
        about();
        break;
        case 3:
                cout<<endl;
                cout<<"\t\t ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
                cout<<"\t\t º
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿ ºÛ"<<endl;

                cout<<"\t\t º ³ GRACIAS POR UTILIZAR ESTE
SOFTWARE ³ ºÛ"<<endl;
                cout<<"\t\t º
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ ºÛ"<<endl;
                cout<<"\t\t º                              ºÛ"<<endl;
                cout<<"\t\t º                  ÉÍÍÍÍÍÍÍÍÍÍÍÍ»  ºÛ"<<endl;
                cout<<"\t\t º                  º   ENTER   ºÛ ºÛ"<<endl;
                cout<<"\t\t º                  ÈÍÍÍÍÍÍÍÍÍÍÍÍ¼Û ºÛ"<<endl;
                cout<<"\t\t º                   ÛÛÛÛÛÛÛÛÛÛÛÛÛ
ºÛ"<<endl;
```

```
                cout<<"\t\t ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
                cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<
<endl;
                op=3;
                getch();
                break;

        }
}

}
void about()
{
char op;
        system("cls");
        cout<<endl<<endl<<endl<<endl<<endl<<endl;
        descarga();
        system("cls");
        puperidad4();
        system("cls");
        cout<<"\t\tÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
   cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº    COMPARACION Y APLICACION DE TECNICAS
METAHEURISTICAS   ºÛ"<<endl;
        cout<<"\t\tº            Y HEURISTICAS           ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº           FREDDY A. CELIN T.         ºÛ"<<endl;
        cout<<"\t\tº          INMACULADA CONSUEGRA
ºÛ"<<endl;
        cout<<"\t\tº            BEIRO ESCOBAR           ºÛ"<<endl;
        cout<<"\t\tº            ADOLFO JIMENEZ          ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
        cout<<"\t\tº                                   ºÛ"<<endl;
```

```
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº        U N I V E R S I D A D        ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº          SIMON BOLIVAR              ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº       INGENIERIA DE SISTEMAS          ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº          DECIMO SEMESTRE            ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº          BARRANQUILLA               ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº             2006                  ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tº                                   ºÛ"<<endl;
cout<<"\t\tÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛÛ"<<endl;

cout<<"ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿"<<endl;

cout<<"³ OPRIMA UNA TECLA PARA CONTINUAR³Û"<<endl;

cout<<"ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙÛ"<<endl;
cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
op=getch();
system("cls");
cout<<endl<<endl;

cout<<"ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
cout<<"º
ºÛ"<<endl;
```

```
        cout<<"º
ºÛ"<<endl;
        cout<<"º
ºÛ"<<endl;
        cout<<"º            ÉÍÍÍÍÍÍÍÍÍÍÍÍÍ»          ÉÍÍÍÍÍÍÍÍÍÍÍÍÍ»
ºÛ"<<endl;
        cout<<"º            ºÛÛÛÛÛÛÛÛÛÛÛÛÛº
ºÛÛÛÛÛÛÛÛÛÛÛÛÛº                ºÛ"<<endl;
        cout<<"º            ÈÍÍ»ÛÛÛÛÛÛÛÉÍÍ¼
ÈÍÍ»ÛÛÛÛÛÛÛÉÍÍ¼              ºÛ"<<endl;
        cout<<"º              ºÛÛÛÛÛÛÛº              ºÛÛÛÛÛÛÛº
ºÛ"<<endl;
        cout<<"º              ºÛÛÛÛÛÛÛº              ºÛÛÛÛÛÛÛº
ºÛ"<<endl;
        cout<<"ºÉÍÍÍÍÍ»        ºÛÛÛÛÛÛÛº  ÈÍÍÍÍÍÍÍÍÍ»  ºÛÛÛÛÛÛÛº
ÉÍÍÍÍÍ»ºÛ"<<endl;
        cout<<"ººÛÛÛÛÛº        ºÛÛÛÛÛÛÛº  ºÛÛÛÛÛÛÛÛÛº
ºÛÛÛÛÛÛÛº          ºÛÛÛÛÛººÛ"<<endl;
        cout<<"ººÉÍÍÍ¼         ºÛÛÛÛÛÛÛº  È͹ÛÛÉÍÍ»ÛÛº
ºÛÛÛÛÛÛÛº          ºÛÛÉ»Ûºº Û"<<endl;
        cout<<"ººÛº            ºÛÛÛÛÛÛÛº  ºÛÛÈÍÍ¼ÛÛº
ºÛÛÛÛÛÛÛº          ºÛÛÈ¼Ûºº Û"<<endl;
        cout<<"ººÛÈÍÍÍ»        ºÛÛÛÛÛÛÛº  ºÛÛÛÛÛÍÍÍ<
ºÛÛÛÛÛÛÛº          ºÛÛÛÛÛÛºº Û"<<endl;
        cout<<"ººÛÛÛÛÛº        ºÛÛÛÛÛÛÛº  ºÛÛÛÛÛÛÛº
ºÛÛÛÛÛÛÛº          ºÛÛÛÈÍ¼ºÛ"<<endl;
        cout<<"ºÈÍÍ»ÛÛº        ÈÍÍÍÍÍÍÍ¼  ºÛÛÉÍÍ»ÛÛº  ÈÍÍÍÍÍÍÍ¼
ºÛÛÛÈÍ»ºÛ"<<endl;
        cout<<"º  ºÛÛº É» ÉÍ»ÉÍ» ÉÍÍ» ÉÍ» É»  ºÛÛº  ºÛÛº ÉÍÍ» É»  É»
É» É»  ÉÍÍ»  ºÛÛÛÛÛººÛ"<<endl;
        cout<<"ºÉÍÍ¼ÛÛº ºº º È¼ º º É»º º È»ºº ÉÍ¼ÛÛÈÍÍ¼ÛÛº ºÉ»º ºº
ºº ºº ºº É¼É»È» ºÛÉÍ»ÛººÛ"<<endl;
        cout<<"ººÛÛÛÛÛÛº ºº º    º ºººº ºÉ»È¼º ºÛÛÛÛÛÛÛÛÛÛº ºººº ºÈ»
ºº ºº ºº º È¼ º ºÛº ºÛÈ¼Û"<<endl;
        cout<<"ººÛÛÛÛÛº ºº ºÉ»È»º ºÈ¼º ººº  º ºÛÛÛÛÛÛÛÛÛÛº ºÈ¼º
º º ºº ºÈÍ¼º º É» º ºÛº ºÛÛÛÛ"<<endl;
        cout<<"ºÉÍÍÍÍ¼ È¼ È¼È¼È¼ ÈÍÍ¼ È¼ÈÍÍ¼ ÈÍÍÍÍÍÍÍÍÍ¼ ÈÍÍ¼ ÈÍ¼
È¼ ÈÍÍÍ¼ ÈÍ¼ÈÍ¼ ÈÍ¼ ÈÍÍ»Û"<<endl;
        cout<<"º              ÉÍÍÍÍÍÍ»              ÉÍÍÍÍÍÍ»
ºÛ"<<endl;
        cout<<"º              ºÛÛÛÛÛÛÛÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼ÛÛÛÛÛÛÛº
ºÛ"<<endl;
```

```
        cout<<"º
ºÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛº
ºÛ"<<endl;
        cout<<"º
ºÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛº
ºÛ"<<endl;
        cout<<"º                ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
ºÛ"<<endl;
        cout<<"º
ºÛ"<<endl;
        cout<<"º
ºÛ"<<endl;

cout<<"ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
        cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
        cout<<endl<<endl;

cout<<"ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿"<<endl;

        cout<<"³ OPRIMA UNA TECLA PARA CONTINUAR³Û"<<endl;

cout<<"ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙÛ"<<endl;
        cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ"<<endl;
        op=getch();
}
//**********************************************************************************
****************************************
//**********************************************************************************
****************************************
void laberinto1(int M[50][50],int x1,int x2,int i,int j,int fil,int col,int fil2,int
col2)
{
clock_t comienzo;
comienzo=clock();

int x,xx;
x=fil2;//filas
xx=col2;//columnas
int cont;
cont=0;
```

```
nodo3 *cab;
cab=new nodo3;
cab=NULL;
nodo3 *ant;


for(i=1; i<=x1*x2; i++)
{
        nodo3 *pasos;//

        pasos=new nodo3;
        pasos->iii=fil2;
        pasos->jjj=col2;
        if(cab==NULL)
        {
                cab=pasos;
        }else{
                ant->sig2=pasos;
        }
        ant=pasos;
        pasos->sig2=NULL;
        pasos->abajo2=NULL;

        cont++;
        if(((x==fil2)&&(xx==col2+1)&&(M[fil2][col2-1]>=2)&&(M[fil2-
1][col2]<=1))||((x==fil2-1)&&(xx==col2)&&(M[fil2][col2-
1]>=2))||((M[fil2][col2-1]>=2)&&(M[fil2-
1][col2]<=1)&&(M[fil2+1][col2]<=1)&&(M[fil2][col2+1]<=1)))
        {
                cout<<"\t\t\t\t\t\t";cout<<"izquierda"<<endl;//

                M[fil2][col2]=2;
                x=fil2;
                xx=col2;
                col2--;
                M[fil2][col2]=3;
                if((fil2==fil)&&(col2==col))
                {
                        i=50000;
                }
        }
        else{
```

```
                if(((M[fil2][col2+1]>=2)&&(M[fil2+1][col2]>=2)&&(M[fil2-
1][col2]<=1)&&(M[fil2][col2-1]<=1))||((x==fil2-
1)&&(col2==xx)&&(M[fil2+1][col2]>=2))||((M[fil2+1][col2]>=2)&&(M[fil2-
1][col2]<=1)&&(M[fil2][col2-
1]<=1)&&(M[fil2][col2+1]<=1))||((x==fil2)&&(col2-
1==xx)&&(M[fil2+1][col2]>=2)))
                {
                        cout<<"\t\t\t\t\t\t";//
                        cout<<"abajo"<<endl;
                        M[fil2][col2]=2;
                        x=fil2;
                        xx=col2;
                        fil2++;
                        M[fil2][col2]=3;
                        if((fil2==fil)&&(col2==col))
                        {
                                i=50000;
                        }
                }
                else{
                        if(((xx==col2-
1)&&(x==fil2)&&(M[fil2][col2+1]>=2))||((x==fil2+1)&&(xx==col2)&&(M[fil
2][col2+1]>=2))||((M[fil2][col2+1]>=2)&&(M[fil2][col2-1]<=1)&&(M[fil2-
1][col2]<=1)&&(M[fil2+1][col2]<=1))||((x==fil2-
1)&&(xx==col2)&&(M[fil2][col2+1]>=2)))
                        {
                                cout<<"\t\t\t\t\t";cout<<"derecha"<<endl;//

                                M[fil2][col2]=2;
                                x=fil2;
                                xx=col2;
                                col2++;
                                M[fil2][col2]=3;
                                if((fil2==fil)&&(col2==col))
                                {
                                        i=50000;
                                }
                        }else{
                                if(((x==fil2+1)&&(xx==col2)&&(M[fil2-
1][col2]>=2))||((M[fil2-
1][col2]>=2)&&(M[fil2+1][col2]<=1)&&(M[fil2][col2+1]<=1)&&(M[fil2][col
2-
```

```
1<=1]))||((x==fil2)&&(xx==col2+1)&&(M[fil2][col2+1]>=2))||((x==fil2)&&(
xx==col2-1)&&(M[fil2-1][col2]>=2)))
                                {

cout<<"\t\t\t\t\t\t";cout<<"arriba"<<endl;//se mueve hacia arriba
                                        M[fil2][col2]=2;
                                        x=fil2;
                                        xx=col2;
                                        fil2--;
                                        M[fil2][col2]=3;
                                        if((fil2==fil)&&(col2==col))
                                        {
                                                i=50000;
                                        }
                                }
                        }
                }
        }

cout<<endl;
cout<<"\t\t\t\t\tÉ";
for(int ii=1; ii<=x1+4; ii++)
{
        cout<<"Í";
}
cout<<"»"<<endl;

for(int i2=1; i2<=x1; i2++)
{
        cout<<"\t\t\t\t ";cout<<"\tº ";
        for(j=1; j<=x2; j++)
        {
                if(M[i2][j]==2)
                {
                        cout<<" "<<" ";
                }
                if(M[i2][j]==9)
                {
                        cout<<"X"<<" ";
                }
                if(M[i2][j]==1)
                {
                        cout<<"Û"<<" ";
```

```
                }
                if(M[i2][j]==3)
                {
                        cout<<"@"<<" ";
                }
        }cout<<"º"<<endl;
}
cout<<"\t\t\t\t\tÈ";
for(ii=1; ii<=x1+4; ii++)
{
        cout<<"Í";
}
cout<<"¼";
cout<<endl;

}
if(i>50000)
{
        cout<<"SALIDA ENCONTRADA!!!"<<endl;
        cout<<endl;
        cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
        cout<<"\t\t  ºCARACTERISTICAS DE LA PERSONAº"<<endl;
        cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
        cout<<endl;
        cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
        cout<<"\t\t  ºCANTIDAD DE PASOS E ITERACIONES QUE
DIO: º"<<cont+1<<endl;
        cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
        cout<<endl;
        cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
        cout<<"\t\t  ºPASOS QUE DIO LA PERSONA: º"<<endl;
        cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
        ant=cab;
        while(ant!=NULL)
        {
                cout<<"                         "<<ant->iii<<ant->jjj<<endl;
                ant=ant->sig2;
        }
        cout<<"                         "<<fil2<<col2<<endl;
        double tiempo1;
        tiempo1=(clock()-comienzo)/(double)CLOCKS_PER_SEC;
        cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ» "<<endl;
        cout<<"\t\t  ºTIEMPO TRANSCURRIDO: º "<<tiempo1<<endl;
```

```
	cout<<"\t\t ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍͼ "<<endl;
}

cout<<"ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄ¿"<<endl;
	cout<<"³ OPRIMA UNA TECLA PARA SALIR DE ESTA
OPCION³Û"<<endl;

cout<<"ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÙÛ"<<endl;
	cout<<"
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛ"<<endl;
getch();
}//********************************************************************************
****************************************
//*********************************************************************************
****************************************

void laberinto2(int M[50][50],int x1,int x2,int i,int j,int fil,int col,int fil2,int
col2)
{
int cont,ite;
double tiempo2;
tiempo2=0;
ite=0;
cont=0;

int z;
ite=0;
clock_t comienzo;
comienzo=clock();

cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;

int sw;
sw=0;

int indi;
cout<<"\t ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
```

```
cout<<"\t  ºDIGITE CUANTOS INDIVIDUOS ESTARAN EN EL
LABERINTO:ºÛ"<<endl;
cout<<"\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼Û"<<endl;
cout<<"\t
ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ
ÛÛÛÛÛÛÛÛ"<<endl;
cout<<"\t\t\t\t ";cin>>indi;


system("cls");
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<endl;cout<<"\t\t\t\t\t";
for(i=1; i<=x1+6; i++)//

{
        cout<<"Ä";
}
cout<<endl;
for(i=1; i<=x1; i++)
{
        cout<<"\t\t\t\t ";cout<<"\t| ";
        for(j=1; j<=x2; j++)
        {
                if(M[i][j]==2)
                {
                        cout<<" "<<" ";
                }
                if(M[i][j]==9)
                {
                        cout<<"X"<<" ";
                }
                if(M[i][j]==1)
                {
                        cout<<"Û"<<" ";
                }
                if(M[i][j]==3)
                {
                        cout<<"@"<<" ";
                }
        }cout<<"|"<<endl;
```

```
}
cout<<"\t\t\t\t\t";
for(i=1; i<=x1+6; i++)
{
        cout<<"Ä";
}
cout<<endl;
cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
getch();

        nodo *cab;
        cab=NULL;
        nodo *ant;


        for(i=1; i<=indi; i++)
        {
                nodo *q;
                q=new nodo;
                if(cab==NULL)
                {
                        cab=q;
                }else{
                        ant->sig=q;
                }
                ant=q;
                q->persona=i;
                q->ii=fil2;
                q->jj=col2;
                q->sig=NULL;

                nodo *q2;
                q2=new nodo;
                q->abajo=q2;
                q2->abajo=NULL;
                q2->ii=fil2;
                q2->jj=col2;

        }
        nodo2 *cab2;
```

```
cab2=NULL;
nodo2 *ant2;
nodo2 *ant3;
//




//
for(i=1; i<=x1; i++)
{
        for(j=1; j<=x2; j++)
        {
                if((M[i][j]==2)||(M[i][j]==3)||(M[i][j]==9))
                {
                        nodo2 *camino;
                        camino=new nodo2;
                        camino->iii=i;
                        camino->jjj=j;
                        cont=0;
                        if(cab2==NULL)
                        {
                                cab2=camino;
                        }else{
                                ant2->sig=camino;
                        }
                        if((M[i-1][j]==2)||(M[i-1][j]==3)||(M[i-1][j]==9))
                        {
                                nodo2 *camino2;
        camino2=new nodo2;
        camino->abajo=camino2;
        camino2->iii=i-1;
        camino2->jjj=j;
        ant3=camino2;
        camino2->abajo=NULL;
        cont++;
        }

if((M[i][j+1]==2)||(M[i][j+1]==3)||(M[i][j+1]==9))
        {
        nodo2 *camino2;
        camino2=new nodo2;
```

```
        if((M[i-1][j]==2)||(M[i-1][j]==3)||(M[i-1][j]==9))
        {
        ant3->abajo=camino2;
        }else{
        camino->abajo=camino2;
        }
        camino2->iii=i;
        camino2->jjj=j+1;
        camino2->abajo=NULL;
        ant3=camino2;
        cont++;
        }

if((M[i+1][j]==2)||(M[i+1][j]==3)||(M[i+1][j]==9))
{
        nodo2 *camino2;
        camino2=new nodo2;
if((M[i-1][j]==2)||(M[i][j+1]==2)||(M[i-1][j]==3)||(M[i-
1][j]==9)||(M[i][j+1]==3)||(M[i][j+1]==9))
                        {
                        ant3->abajo=camino2;
                        }else{
                        camino->abajo=camino2;
                        }
                        camino2->iii=i+1;
                        camino2->jjj=j;
                        camino2->abajo=NULL;
                        ant3=camino2;
                        cont++;
                }
if((M[i][j-1]==2)||(M[i][j-1]==3)||(M[i][j-1]==9))
                {
        nodo2 *camino2;
        camino2=new nodo2;
if((M[i-1][j]==2)||(M[i][j+1]==2)||(M[i+1][j]==2)||(M[i-1][j]==3)||(M[i-
1][j]==9)||(M[i][j+1]==3)||(M[i][j+1]==9))
                        {
                        ant3->abajo=camino2;
                        }else{
                        camino->abajo=camino2;
                        }
                        camino2->iii=i;
                        camino2->jjj=j-1;
```

```
                camino2->abajo=NULL;
                cont++;
                }

                camino->porcentaje=cont;

                ant2=camino;
                camino->sig=NULL;
                }
        }
}
nodo2 *x;
nodo2 *y;
x=cab2;


//
while(x!=NULL)
{
y=x->abajo;
y->porcentaje=0;
ant2=y;
while(y!=NULL)
{

if(x->porcentaje==1)
{
y->porcentaje=100;
        }
        if(x->porcentaje==2)
        {
        y->porcentaje=ant2->porcentaje+50;
        ant2=y;
        }
        if(x->porcentaje==3)
        {
        y->porcentaje=ant2->porcentaje+33;
        ant2=y;
        }
        if(x->porcentaje==4)
        {
        y->porcentaje=ant2->porcentaje+25;
                ant2=y;
```

```
                }
                y=y->abajo;
        }
        x=x->sig;
}

int f;
int vec[5];
for(j=1; j<=x1*x2; j++)
{
for(i=1; i<=indi; i++)
{
f=1+(int) (100.0*rand()/(RAND_MAX+90.0));//

int sw;
nodo *hormi;
hormi=cab;
sw=0;
while((hormi!=NULL)&&(sw==0))//

        {
        if(hormi->persona==i)
        {
        sw=1;
        }else{
        hormi=hormi->sig;
        }
        }
        if(sw==1)
        {

cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
cout<<"numero aleatorio:"<<f<<endl;
cout<<"Persona Numero: "<<hormi->persona<<endl;

nodo *pasos;//
pasos=hormi;
while(pasos->abajo!=NULL)//

{
pasos=pasos->abajo;
}
```

```
int cont;
cont=0;
if(M[pasos->ii-1][pasos->jj]>=2)//

        {
        vec[1]=0;
        cont++;
        }else{
vec[1]=1;//
                }
                if(M[pasos->ii][pasos->jj+1]>=2)
                {
                vec[2]=0;
                cont++;
                }else{
                vec[2]=1;
                }
                if(M[pasos->ii+1][pasos->jj]>=2)
                {
                vec[3]=0;
                cont++;
                }else{
                vec[3]=1;
                }
                if(M[pasos->ii][pasos->jj-1]>=2)
                {
                vec[4]=0;
                cont++;
                }else{
                vec[4]=1;
                }
                nodo *newpasos;
                newpasos=new nodo;
                if(cont==1)//
                {
                if(vec[1]==0)//ir hacia arriba
                {
                newpasos->ii=pasos->ii-1;
                newpasos->jj=pasos->jj;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }
                if(vec[2]==0)//
```

```
                {
                newpasos->ii=pasos->ii;
                newpasos->jj=pasos->jj+1;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }
                if(vec[3]==0)//
                {
                newpasos->ii=pasos->ii+1;
                newpasos->jj=pasos->jj;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }
                if(vec[4]==0)//
                {
                newpasos->ii=pasos->ii;
                newpasos->jj=pasos->jj-1;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }
eliminar(newpasos,hormi,cab);//


                        }
                        if(cont==2)//si hay 2 caminos
                        {
                        nodo2 *ncaminos;
                        ncaminos=cab2;
                        sw=1;
                        while((ncaminos!=NULL)&&(sw==1))//


                {
        if((ncaminos->iii==pasos->ii)&&(ncaminos->jjj==pasos->jj))
                        {
                        ncaminos=ncaminos->abajo;
                        sw=0;
                        }else{
                        ncaminos=ncaminos->sig;
                        }
                }
```

```
                if(f<=ncaminos->porcentaje)
                {
                ncaminos->porcentaje=ncaminos-
>porcentaje+1;//
                ncaminos=ncaminos->abajo;
                ncaminos->porcentaje=ncaminos->porcentaje-1;

                if(vec[1]==0)
                {
                newpasos->ii=pasos->ii-1;
                newpasos->jj=pasos->jj;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }else{

                if(vec[2]==0)
                {
                newpasos->ii=pasos->ii;
                newpasos->jj=pasos->jj+1;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }else{

                if(vec[3]==0)
                {
                newpasos->ii=pasos->ii+1;
                newpasos->jj=pasos->jj;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                                        }
                                }
                        }
                }else{



                ncaminos->porcentaje=ncaminos->porcentaje-1;
                ncaminos=ncaminos->abajo;
```

```
                ncaminos->porcentaje=ncaminos->porcentaje+1;

                if(vec[4]==0)
                {
                newpasos->ii=pasos->ii;
                newpasos->jj=pasos->jj-1;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }else{

                if(vec[3]==0)
                {
                newpasos->ii=pasos->ii+1;
                newpasos->jj=pasos->jj;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                }else{

                if(vec[2]==0)
                {
                newpasos->ii=pasos->ii;
                newpasos->jj=pasos->jj+1;
                pasos->abajo=newpasos;
                newpasos->abajo=NULL;
                                                }
                                        }
                                }
                        }

eliminar(newpasos,hormi,cab);//


                        }
                        if(cont==3)// si hay 3 caminos
                        {
                                nodo2 *ncaminos;
                                ncaminos=cab2;
                                sw=1;


                                while((ncaminos!=NULL)&&(sw==1))
                                {
        if((ncaminos->iii==pasos->ii)&&(ncaminos->jjj==pasos->jj))
```

```
{
ncaminos=ncaminos->abajo;
sw=0;
}else{
ncaminos=ncaminos->sig;
                    }
                }



                    if(f<=ncaminos->porcentaje)
                    {
            ncaminos->porcentaje=ncaminos->porcentaje+2;
            ncaminos=ncaminos->abajo;
            ncaminos->porcentaje=ncaminos->porcentaje+1;

                        if(vec[1]==0)
                        {
                            newpasos->ii=pasos->ii-1;
                            newpasos->jj=pasos->jj;
                            pasos->abajo=newpasos;
                            newpasos->abajo=NULL;
                        }else{

                    if(vec[2]==0)
                    {
        newpasos->ii=pasos->ii;
        newpasos->jj=pasos->jj+1;
        pasos->abajo=newpasos;
        newpasos->abajo=NULL;
                    }
                }
        }else{
        nodo2 *ncaminos2;
        ncaminos2=ncaminos->abajo;
//

    if((f>ncaminos->porcentaje)&&(f<=ncaminos2->porcentaje))
                    {
        ncaminos2->porcentaje=ncaminos2->porcentaje+1;
        ncaminos->porcentaje=ncaminos->porcentaje-1;
        //
            if(vec[2]==0)
```

```
                    {
                    newpasos->ii=pasos->ii;
                    newpasos->jj=pasos->jj+1;
                    pasos->abajo=newpasos;
                    newpasos->abajo=NULL;
                    }else{

                                                    if(vec[3]==0)
                                                    {
                                newpasos->ii=pasos->ii+1;
                                newpasos->jj=pasos->jj;
                                pasos->abajo=newpasos;
                                newpasos->abajo=NULL;
                                                    }
                                            }
                                    }else{

        ncaminos2->porcentaje=ncaminos2->porcentaje-2;
        ncaminos->porcentaje=ncaminos->porcentaje-1;
        if(vec[3]==0)
                {
        newpasos->ii=pasos->ii+1;
        newpasos->jj=pasos->jj;
        pasos->abajo=newpasos;
        newpasos->abajo=NULL;
        }else{
                if(vec[4]==0)
                        {
newpasos->ii=pasos->ii;
newpasos->jj=pasos->jj-1;
pasos->abajo=newpasos;
newpasos->abajo=NULL;
                                }
                        }
                }
        }
        eliminar(newpasos,hormi,cab);
        }
if(cont==4
{
nodo2 *ncaminos;
nodo2 *ncaminos3;
ncaminos=cab2;
```

```
sw=1;


while((ncaminos!=NULL)&&(sw==1))
{
if((ncaminos->iii==pasos->ii)&&(ncaminos->jjj==pasos->jj))
{
ncaminos=ncaminos->abajo;
ncaminos3=ncaminos;
sw=0;
}else{
ncaminos=ncaminos->sig;
        }
}
//f=Random

if(f<=ncaminos->porcentaje)
{
ncaminos->porcentaje=ncaminos->porcentaje+3;
ncaminos=ncaminos->abajo;
ncaminos->porcentaje=ncaminos->porcentaje+2;
ncaminos=ncaminos->abajo;
ncaminos->porcentaje=ncaminos->porcentaje+1;
if(vec[1]==0)
{
newpasos->ii=pasos->ii-1;
newpasos->jj=pasos->jj;
pasos->abajo=newpasos;
newpasos->abajo=NULL;
}
}else{
nodo2 *ncaminos2;
ncaminos2=ncaminos->abajo;

if((f>ncaminos->porcentaje)&&(f<=ncaminos2->porcentaje))
{
ncaminos2->porcentaje=ncaminos2->porcentaje+2;
ncaminos->porcentaje=ncaminos->porcentaje-1;
ncaminos2=ncaminos2->abajo;
ncaminos2->porcentaje=ncaminos2->porcentaje+1;
if(vec[2]==0)
{
newpasos->ii=pasos->ii;
```

```
newpasos->jj=pasos->jj+1;
pasos->abajo=newpasos;
newpasos->abajo=NULL;
}
}else{
ncaminos=ncaminos2->abajo;

if((f>ncaminos2->porcentaje)&&(f<=ncaminos->porcentaje))
{
ncaminos->porcentaje=ncaminos->porcentaje+1;
ncaminos2->porcentaje=ncaminos2->porcentaje-2;
ncaminos3->porcentaje=ncaminos->porcentaje-1;
if(vec[3]==0)
{
newpasos->ii=pasos->ii+1;
newpasos->jj=pasos->jj;
pasos->abajo=newpasos;
newpasos->abajo=NULL;
}
}else{

ncaminos->porcentaje=ncaminos->porcentaje-3;
ncaminos2->porcentaje=ncaminos2->porcentaje-2;
ncaminos3->porcentaje=ncaminos->porcentaje-1;
if(vec[4]==0)
{
newpasos->ii=pasos->ii;
newpasos->jj=pasos->jj-1;
pasos->abajo=newpasos;
newpasos->abajo=NULL;
                        }
                }
        }
}
                                eliminar(newpasos,homi,cab);
                        }
                        int tt,uu

                        for(tt=1; tt<=x1; tt++)
                        {
                                for(uu=1; uu<=x2; uu++)
                                {
```

```
                                        if((tt==newpasos-
>ii)&&(uu==newpasos->jj))
                                        {
                                                M[tt][uu]=3;
                                        }
                                                //cout<<M[tt][uu];
                        }cout<<endl;
                }
                cout<<endl;
                cout<<"\t\t\t\t\tÉ";
                for(int ii=1; ii<=x1+4; ii++)
                {
                        cout<<"Í";
                }
                cout<<"»"<<endl;
                for(tt=1; tt<=x1; tt++)
                {
                        cout<<'\t\t\t\t ";cout<<"\tº ";
                        for(uu=1; uu<=x2; uu++)
                        {
                                if(M[tt][uu]==2)
                                {
                                        cout<<" "<<" ";
                                }
                                if(M[tt][uu]==9)
                                {
                                        cout<<"X"<<" ";
                                }
                                if(M[tt][uu]==1)
                                {
                                        cout<<"Û"<<" ";
                                }
                                if(M[tt][uu]==3)
                                {
                                        cout<<"@"<<" ";
                                }
                        }cout<<"º"<<endl;
                }
                cout<<"\t\t\t\t\tÈ";
                for(ii=1; ii<=x1+4; ii++)
                {
                        cout<<"Í";
                }
```

```
					cout<<"¼";
					cout<<endl;
					if((newpasos->ii==fil)&&(newpasos->jj==col))
					{
						z=i;
						j=50000;
						i=50000;
						cout<<"Salida Encontrada!!!"<<endl;
					}
					ite++;
				}
			}
		}

		cout<<endl;
		cout<<"ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ"<<endl;
		if(j==50001)
		{
			nodo *asd;
			asd=cab
			int cont;
			cont=0;
			cout<<endl;
			cout<<"\t\t ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
			cout<<"\t\t ºCARACTERISTICAS DE LA PERSONA "<<z<<"
ES: º"<<endl;
			cout<<"\t\t ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
			cout<<endl;
			cout<<"\t\t ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
			cout<<"\t\t ºPASOS QUE DIO LA PERSONA: º"<<endl;
			cout<<"\t\t ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
			cout<<endl;
			while(asd!=NULL)
			{
				if(asd->persona==z)
				{
					nodo *y;
					y=asd->abajo;
					while(y!=NULL)
					{
```

```
	cout<<"                                    "<<y->ii<<y->jj<<endl;
		cont++;
		y=y->abajo;
			}
		}
		asd=asd->sig;
	}
	cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
	cout<<"\t\t  ºCANTIDAD DE PASOS DEL GANADOR ES:
º"<<cont<<endl;
	cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
}else{
	cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
	cout<<"\t\t  ºNO LLEGO NINGUNA PERSONA º"<<endl;
	cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
	cout<<endl;
}
	cout<<endl;
	cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»"<<endl;
	cout<<"\t\t  ºEL NUMERO DE ITERACIONES ES: º"<<ite<<endl;
	cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼"<<endl;
	tiempo2=(clock()-comienzo)/(double)CLOCKS_PER_SEC;
	cout<<"\t\t  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ» "<<endl;
	cout<<"\t\t  ºTIEMPO TRANSCURRIDO: º "<<tiempo2<<endl;
	cout<<"\t\t  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼ "<<endl;
	cout<<endl;

getch();

}
void eliminar(nodo *&newpasos,nodo *&hormi,nodo
*&cab

{
nodo *busc;
busc=hormi;
int sw=0;
while(busc->abajo!=NULL)
{
	if((newpasos->ii==busc->ii)&&(newpasos->jj==busc->jj))
	{
		sw=1;
	}
```

```
        busc=busc->abajo;
}
if(sw==1)
{
        nodo *elim;
        elim=cab;
        if(cab->persona==hormi->persona)
        {
                hormi=hormi->sig;
                cab=hormi;
        }else{
                nodo *ant;
                ant=elim;
                while(elim->persona==hormi->persona)
                {
                        ant=elim;
                        elim=elim->sig;
                }
                if(hormi->sig==NULL)
                {
                        ant->sig=NULL;
                }else{
                        ant->sig=hormi->sig;
                }
        }
        cout<<"persona eliminada!!!"<<endl;
}
}
```

7. Conclusión

Al realizar varias pruebas con los algoritmos de Colonia de Hormigas y Backtracking con matrices de diferentes tamaños, hemos concluido que las tecnicas que se han utilizado, ambas son buenas para resolver problemas de busquedas.

Las diferencias entre los metodos son el tiempo de ejecución y el consumo de memoria que ambos toman.

En el caso del Backtracking es rápido en el sentido de tiempo de ejecución pero a la hora de encontrar la respuesta optima debe recorrer y analizar todas las posibles soluciones en cambio Colonia de Hormigas se guian por el rastro de feromona y toman probabilisticamente una ruta que en la mayoria de los casos es mas corta pero consume mas tiempo de ejecucion y memoria.

Los resultados obtenidos son muy alentadores y deja abierto muchas espectativas para el uso de esta tecnica en cualquier problema de optimizacion.

8. ANEXOS

Tabla 1. Comparación de las técnicas en intensificación y diversificación

Figura 1. Algoritmo de Tabú para realizar la búsqueda

Figura 2. Bucle básico de un algoritmo genético

Figura 3. Proceso de Crossover

Figura 4. Crossover de n-puntos

Figura 5. Mutación Multibit

Figura 6. Ejemplo de Cruce y Mutación

Figura 7. Comportamiento de las Hormigas Reales

Figura 8. Evolución de los Algoritmo vía Colonia de Hormigas

Figura 9. Recorrido semejante utilizado en el Backtracking

Figura 10. Grafico del tablero con movimientos del caballo

Figura 11. Tablero de tamaño 5*5 con todo el recorrido

9. CRONOGRAMAS

CRONOGRAMA DE ACTIVIDADES FORMATIVA I

AÑO	2005															
MES	MARZO				ABRIL				MAYO				JUNIO			
SEMANAS ACTIVIDADES	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Escogencia del tema	****	****														
Presentación del tema			****													
Presentación de los asesores				****												
Clasificación e interpretación de la información					****	****	****	****								
Escogencia de Titulo									****	****	****					
Planteamiento de Objetivos												****	****	****		
Presentación de Planteamiento del Problema a resolver														***	****	
Revisión Proyecto															****	****
Sustentación del Proyecto																****

CRONOGRAMA DE ACTIVIDADES FORMATIVA II

AÑO	2005															
MES	AGOSTO				SEPTIEMBRE				OCTUBRE				NOVIEMBRE			
SEMANAS ACTIVIDADES	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Cambio nombre del proyecto	***															
Revisión de Planteamientos		****														
Presentación de ejemplos y aplicaciones			****													
Planteamiento del problema a dar solución				****												
Escogencia de algoritmo para aplicar AGs					****	****	****	****								
Escogencia del lenguaje para desarrollar la aplicación									****	****	****					
Organización de Capítulos												****	****	****		
Revisión Proyecto															****	****
Sustentación del Proyecto																****

CRONOGRAMA DE ACTIVIDADES FORMATIVA III

AÑO	2006															
MES	FEBRERO				MARZO				ABRIL				JUNIO			
SEMANAS / ACTIVIDADES	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Presentación de ejemplos y aplicaciones	****	****	****													
Desarrollar la Hipótesis a comprobar				****	****											
Presentación de AGs como SPEA, SPEAII, NSGA						****	****	****								
Desarrollo de las funciones de selección, cruce y mutación									****	****						
Desarrollo prototipo											****	****	****			
Entrega del proyecto														***		
Sustentación del Proyecto															****	****

NOTA

Corrección de actividades debido a cambio de la técnica meta heurística a utilizar, el proyecto seguirá la investigación de los algoritmos genéticos como parte de estudio y conocimiento, pero se trabajará la parte de la aplicación con la técnica meta heurística conocida como Colonias de Hormigas o ANT Q, respectivamente se desarrollará la documentación necesaria y pertinente para conocer esta técnica y como se trabaja con esta heurística como es el Backtracking.

Se utilizara para tal comparación el desarrollo de laberintos.

CRONOGRAMA DE ACTIVIDADES FORMATIVA CON CORRECION III

AÑO	2006																ADICIONAL			
MES	FEBRERO				MARZO				ABRIL				JUNIO				JULIO			
SEMANAS ACTIVIDADES	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Utilización de la Técnica De Colonia de Hormigas y Backtracking para realizar la comparación																	****			
Recapitulación de objetivos y marco teórico																	****	**	**	
Utilización de la Técnica De Colonia de Hormigas y Backtracking para																		**	**	

realizar la comparación																				
Desarrollo de especificaciones para la construcción del laberinto																			**	**
Desarrollo del los Algoritmos utilizando Backtracking y ANT Q																			**	**
Entrega del proyecto																				**

CRONOGRAMA DE ACTIVIDADES FORMATIVA IV

AÑO	2005															
MES	AGOSTO				SEPTIEMBRE				OCTUBRE				NOVIEMBRE			
SEMANAS ACTIVIDADES	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Perfeccionamiento de los algoritmos: Backtracking y ANT Q	***	***	***	***												
Pruebas de los algoritmos y respectiva comparación de éstas					***	***	***	***								
Realización de cuadros comparativos (métricas escogidas para evaluar los algoritmos)									****	***	***	***				
Desarrollo del capitulo de conclusiones												***	***			
Revisión Proyecto													***	***		
Sustentación del Proyecto															****	

9. BIBLIOGRAFIA

MERELO CUERVOS, J.J. Informática Evolutiva. Tutorial.jmerelo@jeneura.ugr.es.

Información tomada de Trabajos de grados sobre Algoritmos Genéticos realizados por:

1. Fogel David B., "Evolutionary Computation". IEEE Press. Estados Unidos. 1995.
2. Goldberg David E., "Genetic Algorithms in Search, Optimization, and Machine Leamig". Addison-Wesley Publishing Company. Inc. Estados Unidos. 1989.
3. Algoritmos Genéticos y Computación Evolutiva de Adam Marczyk (seminario).
4. http://www.orcero.org/irbis/disertacion/node189.html "Algoritmos Genéticos", David Santo Orcero.
5. http://www.uv.es/~rmarti/genet.html

   "Algoritmos genéticos", Rafael Martí. Profesor Titular de Universidad.

6. Departamento de Estadística e Investigación Operativa Universidad de Valencia.
7. www.redcientifica.com/doc/doc199904260011.html
8. http://www.aircenter.net/gaia.
9. http://dns1.mor.itesm.mx/~emorales/Cursos/KDD/node87.html.
10. http://apuntes.rincondelvago.com/algoritmos-geneticos.html
11. http://www.redcientifica.com/doc/doc199904260011.html

12. http://www.lsi.upc.es/~iea/transpas/9_geneticos.
13. http://scalab.uc3m.es/~docweb/doctorado/aa/descripcion.html.
14. trabajo de Algoritmos Genéticos _ inteligencia Artificial I 2004 – 2005
15. http://www.algoritmia.net/articles.php?id=33

Trabajos De Grados Fundamentados En Técnica De Búsqueda Atrás Backtracking

- http://www2.ing.puc.cl/~jabaier/iic2552/backtracking.pdf
- Introducción a las Meta heurísticas, Pablo Rodríguez Bocca
- www2.ing.puc.cl/~jabaier/iic2552/backtracking.pdf
- http://www.algoritmia.net/articles.php?id=33
- http://www.eps.uam.es/~swerc/ole/ejemplos/backtracking.pdf

Trabajos De Grados Fundamentados En Técnica Colonia de Hormigas

- http://neo.lcc.uma.es/radi-aeb/html/ant.html

- http://fisica.usac.edu.gt/~jccordova/Hormigas/Optimizacion_via_Colonias_de_Hormigas.htm.

  Optimización a través de colonias de hormigas
  Juan Carlos Córdova Zeceña
  Universidad de San Carlos de Guatemala Facultad de Ingeniería Departamento de Física

➢ http://www.cnc.una.py/cms/invest/download.php?id=250344,48,
1

Colonia de Hormigas en un Ambiente Paralelo Asíncrono
Benjamín Baran y Martha Almiron
Universidad Nacional de Asunción
Centro Nacional de Computación
San Lorenzo, Paraguay

UNIVERSIDAD SIMÓN BOLÍVAR
BIBLIOTECA