Instituto Tecnológico de Costa Rica

Unidad de Computación

Ruta corta en estrategias, Genético, Probabilístico, Backtracking

Andres Fernandez

Jurguen Romero

22/11/2016

Introducción

En el siguiente trabajo se tratara sobre la creación y el análisis de un algoritmo que sea capaz de determinar la ruta corta de un grafo fuertemente conexo con tres diferentes tipos de estrategias de programación: genética, probabilística y backtraking ,también mostrar mediciones de los algoritmos realizados con el fin de determinar la complejidad de cada uno y lograr un análisis a fondo de cuál podría ser el mejor para solucionar el problema de la ruta corta con grafos de distintos tamaños: 10,100, 500 y 1000.

Con el algoritmo genético se busca sacar las mejores rutas a partir 8 padres los cuales generarán generaciones, este algoritmo se caracteriza por basarse en la teoría de la evolución de Charles Darwin la cual consiste que solo el ser más fuerte y con mejores cualidades sobrevive.

El segundo algoritmo que se mostrará es el backtracking ,también llamado vuelta atrás por su cualidad de buscar todas las soluciones y de al no encontrar una devolverse y continuar por otro camino, este algoritmo recursivo se analizará y se mostrará que tan eficiente y óptimo es encontrando la mejor ruta.

El último algoritmo que se analizará es el probabilístico, un algoritmo que no siempre muestra la respuesta correcta, este se implementara con la ayuda de una colonia de hormigas las cuales a partir de probabilidades toman sus caminos para llegar a sus destinos teniendo dejando un rastro de feromona para guiar a sus compañeras, con forme el camino sea más óptimo la feromona se notará más fuertemente.

El análisis de estos algoritmos será representado mediante datos sacados directamente de los algoritmos y los resultados que estos muestran.

Análisis del problema

Se requiere implementar tres diferentes estrategias para la creación de un programa que determine la ruta corta en un grafo fuertemente conexo, para esto se debe investigar como representar un grafo en el lenguaje de programación c#, los grafos pueden ser realizados mediante una clase que represente los vértices y contenga los arcos del grafo o mediante matrices, esta última siendo el método tomado para la realizar la solución del problema de ruta corta. También se tiene que investigar acerca de las diferentes estrategias, ya mencionadas en el inicio, para lograr su correcta implementación, además de ser uno de los requerimientos del programa.

La primer estrategia es el algoritmo genético es una técnica de programación con un procedimiento evolutivo comenzando de una población a la cual llamaremos padres estos son la raíz para una serie de cruces y mutaciones que van generando descendencias y consigo mejorando su solución. El algoritmo genético son mecanismos de búsqueda basados en las leyes de la selección natural y de la genética propuesta por Charles Darwin e implementado en algoritmo por John Henry Holland. Este combina la supervivencia de los individuos mejor adaptados junto con operadores de búsqueda genéticos como la mutación y el cruce, de ahí que sean comparables a una búsqueda biológica (Lopez, J. 2010).

El backtracking es una técnica de programación que realiza una búsqueda sistemática de las mejores soluciones con un recorrido a profundidad. Según el  Departamento de Lenguajes y Ciencias de la Computación, “En su forma básica la Vuelta Atrás se asemeja a un recorrido en profundidad dentro de un árbol cuya existencia sólo es implícita, y que denominaremos árbol de expansión. Este árbol es conceptual y sólo haremos uso de su organización como tal, en donde cada nodo de nivel k representa una parte de la solución y está formado por k etapas que se suponen ya realizadas. Sus hijos son las prolongaciones posibles al añadir una nueva etapa. Para examinar el conjunto de posibles soluciones es suficiente recorrer este árbol construyendo soluciones parciales a medida que se avanza en el recorrido” (2009).

La técnica probabilística se encarga de mediante datos de entrada tomar decisiones al azar y sacar un promedio de la solución más óptima. La ruta corta se saca de forma probabilística con un conjunto de hormigas que siguen una feromona de forma aleatoria se crea la probabilidad de que la hormiga pueda irse de nuevo por el mismo camino , tomando esta probabilidad y con esto generando una solución más óptima para toda la colonia.

Según Frank-Neumann,”Ant Colony Optimization (ACO) se ha vuelto muy popular en los últimos años. En contraste con muchas aplicaciones exitosas, la base teórica de esta heurística de búsqueda aleatoria es bastante débil. La construcción de tal teoría se requiere para entender cómo funcionan estas heurísticas, así como para llegar a mejores algoritmos para ciertos problemas” (2006).

Solución del problema

El programa busca analizar el comportamiento de tres diferentes algoritmos (Algoritmo genético, BackTracking y probabilístico (Colonia de hormigas)). Para la implementación de estos algoritmos se necesita de la utilización de algunas estructuras las cuales se mencionaran conforme se vaya explicando los algoritmos o el problema que se pide solucionar.

**Backtracking**

Para lograr este algoritmo utilizó varias la matriz rutas la cual se genera de manera aleatoria y puede ser de 4 tamaños diferentes (10x10, 100x100, 500x500 y 1000x1000), conforme a los tamaños de la matriz rutas se genera otra matriz llamada “listaI” la cual contiene booleanos, esta matriz servirá para evitar que el backtracking regrese a rutas ya exploradas, por así decir esta matriz nos permite marcar la rutas exploradas, esta marca servirá como validación para no volver a recorrer un camino ya verificado. Este algoritmo contiene podas para las ramas con respuestas innecesarias, porque sean mayor al peso de la ruta más óptima o haya visitado más vértices de los que el límite permite. Se generan dos variable una llamada “distanciacorta”(int 32bits) donde se va guardando el peso de la los vértices esta se irá optimizando, y la segunda es “caminocorto”(string(largo \* 16) bits) la cual contendrá la concatenación de las rutas recorridas en el algoritmo. Este algoritmo se moverá por cada uno de los arcos del grafo hasta encontrar la ruta más corta, la forma de trabajar se verá muy similar al recorrido a profundidad de un árbol. Se mandarán por parámetros al método el vértice inicial para que conozca donde empezar y el vértice final para que saber dónde terminar el proceso.

Pseudocodigo:

if (distanciaDelLosCaminos < distanciaCorta Y verticesRecorridos(comb) <= saltos)

{

//condición de parada

if (CondicionDeParada == VerticeFinal)

{

Retorne el camino Mas optima

}

else

{

for (int i = k; i < n; i++)

{

if (Vertices no se repiten)

{

Lista de vértices repetidos[I,k] = true;//Marca rutas exploradas

for (int j = 0; j < n; j++)

{

if (i != j && i != VerticeFinal)

{

for (int h = 0; h < n; h++)

listaDeVérticesrepetidos[i,H] = true;

Backtraking(VerticeInicio, VerticeFinal, cooncatenacionDevertices + i + "," + j + " | ", j, listaDeverticesrepetidos, distancia + matrizRutas[i, j]);

}

}

lista[k, i] = false;

break;

}

}

}

**Algoritmo probabilístico**

La implementación de este algoritmo se logró gracias a una variable llamada “distanciaCorta” la cual guardará el peso de la mejor distancia encontrada por una hormiga, ”caminoCorto” la cual concatena los vértices de la mejor ruta encontrada, se utilizan una matriz rutas la cual puede tener 4 tamaños distintos (10x10, 100x100, 500x500 y 1000x1000). Para implementar este algoritmo se debe generar una feromona inicial esta será de 0.1 llamada “fI” con esta variable se me permite sacar la sumatoria a partir de esta fórmula:

FI\* (1/(peso del arco de la matriz))

Es decir las feromonas iniciales por el inverso del el peso en alguna posición de la matriz, se le saca a todos los arcos a partir del vértice inicio, y los resultados se suman dando la variable “sumatoria”

Se genera una lista probabilidadesXY la cual contiene la probabilidad de que la hormiga tome un camino, esta probabilidad se saca con:

(FI\* (1/(peso del arco de la matriz))) / Sumatoria

Esta probabilidad como antes ya dicho es para conocer por cual arco la hormiga se ira todos los valores que salen quedan entre un intervalo de [0,1] y con un random de 0 a 1 se consigue el arco por donde viaja la hormiga. Al final la feromona dejada por la hormiga se guarda en una matriz de feromonas para que se vea los cambios en los caminos.

Seudocódigo de mejor camino de las hormigas:

Procedimiento AS ()

Inicializar parámetros

Mientras condicion do

Inicializar\_Nuevahormiga()

Mientras estado <> estado\_fin

Por cada\_arco\_posible\_movimiento

Calcular probalididad\_eleccion

Fin\_Por

Siguiente\_posicion = politica\_decision

Lista\_Posiciones =+ Siguiente\_posicion

Fin\_mientras

Realizar\_evaporacion

Depósito\_feromona(lista\_Posiciones)

Fin\_mientras

Fin\_procedimiento

**Algoritmo Genético**

Para la estrategia del genético se procede a utilizar una lista principal de objetos de tipo Ruta y listas temporales de enteros, contadores de tipo entero, algunas banderas de tipo boolean.

Pasos

1. Creación de la población inicial de forma aleatoria manteniendo el origen y el destino especificados y respetando la cantidad máxima de saltos ya establecida, esta es almacenada en un arreglo de rutas cada una con su respectivo peso.
2. Se crean los padres y los hijos de tipo ruta para usarlos en el proceso del algoritmo, se implementa el método que se encarga de empezar con el proceso evolutivo.
3. En este método se crea un while() el cual se ejecutara mientras que los hijos tengan un menor valor que el de alguno de los padre de lo contrario se sale del ciclo, esto con el fin de lograr una solución oprima y no caer en un ciclo sin mejora alguna.
4. Dentro del ciclo se crean las posibles combinaciones entre las rutas de la población inicial, luego se determina si existe algún tipo de cruce, si este no existe se crea de forma que puedan cruzar los padre creando sus progenitores.
5. En esta instancia se puede proceder a mutar, en el algoritmo existen tres tipos de mutación y las primeras dos son parte de la adaptabilidad de la ruta o cromosoma, con el cruce se modifica el tamaño de la ruta con esto si es mayor al número máximo de saltos se muta de forma que se eliminan los arcos con los mayores peso hasta que la cantidad de vértices deje de ser mayor a la definida por los saltos, cuando ya no es mayor termina la primera fase de la mutación.
6. La segunda fase de la mutación consiste en cambiar los vértices más pesados de forma que el peso de la ruta disminuya y se optimice la respuesta.
7. Se presenta la última mutación la cual se encarga de eliminar los vértices repetidos de esta forma no se generan ciclos en la ruta óptima.
8. Se vuelve a ejecutar el ciclo generando más cruces y mutaciones de las diferentes familias recorridas por el for ,esto se cumple hasta que ningún hijo es mejor que su padre a la hora de concluir cada mutación.

**Diagrama 1. Diagrama algoritmo genético**

**Análisis de resultados**

En la siguiente tabla se detallan los resultados obtenidos en el trabajo realizado.

**Tabla 1**

|  |  |  |
| --- | --- | --- |
| Proceso | Porcentaje | Comentarios |
| Algoritmo ruta estrategia probabilística | 100% | Completo |
| Algoritmo ruta corta estrategia backtraking | 100% | Completo |
| Algoritmo genético estrategia genetica | 100% | Funciona máximo con un grafo de 10 vertices |
| Documentación externa | 100% | Completo |
| Documentación interna | 100% | Completo |

Medición de algoritmos

Medición analítica genético

* 1. Código algoritmo con saltos\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

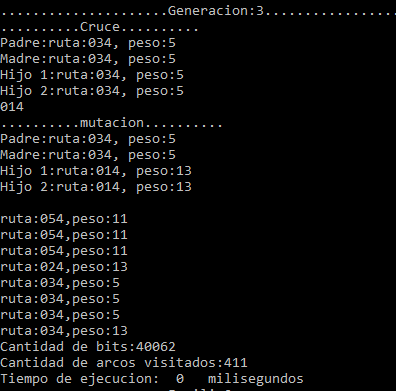
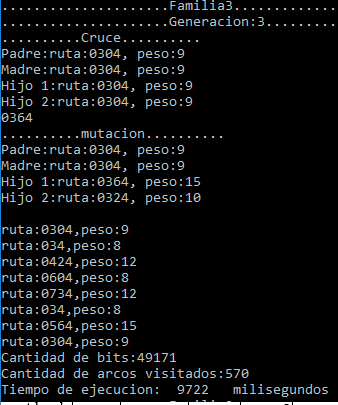
|  |  |
| --- | --- |
| Código fuente  Solo se analiza el código de ruta corta. | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| public static void generadorMutaciones()  {  generarPoblacionInicial(0, 4);  hu.peso = 0;  hd.peso = 0;  f.peso = 1;  mo.peso = 1;  counGeneraciones = 0;  while ((hd.peso < mo.peso || hd.peso < f.peso) || (hu.peso < mo.peso || hu.peso < f.peso))  {  //Instancias importantes para realizar cruces  for (int x = 0; x < 4; x++)  {  casados = new int[8];  councasa = 0;  parejas = new pareja[4];  countaparejas = 0;  padre = true;  bandera = true;  //Instancias importantes para mutar  f = new Ruta(0, "");  mo = new Ruta(0, "");  hu = new Ruta(0, "");  hd = new Ruta(0, "");  int[] v = new int[2];  salto = "";  bandera1 = false;  bits += v.Length\*32;  //............................  //Se encarga de generar nuevas combinaciones para que no se realicen los mismos cruces  generaCombinaciones();    generarCruses(x);    if (f.peso > mo.peso)  {  if (hu.peso < f.peso && existeEnrutasprincipales(hu.ruta).Equals(false))  {  cambioEvolutivo(f,hu);  }  else if (hd.peso < f.peso && existeEnrutasprincipales(hd.ruta).Equals(false))  {  cambioEvolutivo(f, hd);  }  else if (hu.peso < f.peso && hd.peso < mo.peso && existeEnrutasprincipales(hu.ruta).Equals(false) && existeEnrutasprincipales(hd.ruta).Equals(false))  {  cambioEvolutivo(f, hu);  cambioEvolutivo(mo, hd);  }  else if (hd.peso < f.peso && hu.peso < mo.peso && existeEnrutasprincipales(hu.ruta).Equals(false) && existeEnrutasprincipales(hd.ruta).Equals(false))  {  cambioEvolutivo(f, hd);  cambioEvolutivo(mo, hu);  }  }  else  {  if (hu.peso < mo.peso && existeEnrutasprincipales(hu.ruta).Equals(false))  {  cambioEvolutivo(mo, hu);  }  else if (hd.peso < mo.peso && existeEnrutasprincipales(hd.ruta).Equals(false))  {  cambioEvolutivo(mo, hd);  }  else if (hu.peso < f.peso && hd.peso < mo.peso &&existeEnrutasprincipales(hu.ruta).Equals(false) && existeEnrutasprincipales(hd.ruta).Equals(false))  {  cambioEvolutivo(f, hu);  cambioEvolutivo(mo, hd);  }  else if (hd.peso < f.peso && hu.peso < mo.peso && existeEnrutasprincipales(hu.ruta).Equals(false) && existeEnrutasprincipales(hd.ruta).Equals(false))  {  cambioEvolutivo(f, hd);  cambioEvolutivo(mo, hu);  }  }    }  }  } | A\*B\*C8\*D\*E+21  1  1  1  1  1  4  F  1  1  1  1  1  1  1  1  1  1  1  1  1  1  G\*H\*I\*J+15  11+k\*L+5\*n\*(M+4)+(M+4)+O+12+p  +3+3+(M+4)+(M+4)+(3+4+p)+(3+p+2)  +4+p+(2+3+q+4)+2\*(2+r+6+(A1+9)+3+(A2+9)+7+(A2+9)+10++(4+A4)+2\*(4+B1+B2+8+(B3+4))+C1\*C2\*C3+12)(10+D1)+(10+D2)  1  1  5+D3  1  5+D4  1  5+E1  5+E2  1  5+F1  5+F2  1  5+E3  1  5+G1  1  5+G2  5+G3  1  5+G4  5+G5 |
| Total | (A\*B\*C8\*D+E\*G\*H\*J+k\*L+n(2M)+2p+q+r+1+A2+A3+A4+2B1+2B2+2B3+2C1+2C2+2C3+D1+D2+D3+D4+E1+E2+F1+F2+E3+G1+G4+G5+G2+G3)+C |
| Clasificación en notación O | O(n) |

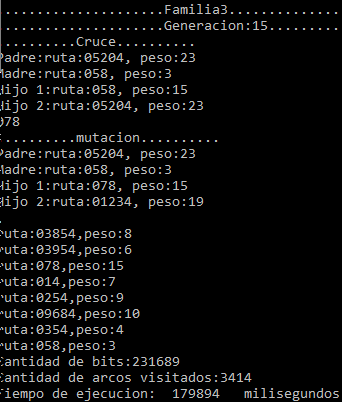
Medición empírica genético

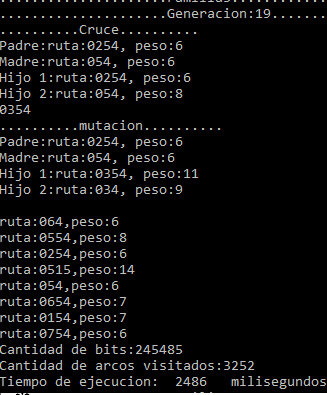
* 1. Código algoritmo #1 **acotado por saltos**:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Operaciones | Tamaños datos de entrada (parámetros del grafo y h) | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6  h=30% h=50% | | 8  h=30%.h=50% | | 9  h=30%.h=50% | | 10  h=30%.h=50% | |  |
| Arcos visitados | 769 | 411 | 480 | 3252 | 956 | 570 | 520 | 3414 |  |
| Memoria  utilizada | 74950 | 40622 | 47220 | 245485 | 91074 | 49171 | 51028 | 231689 |  |
| Tiempo de ejecución | 0 | 0 | 6368 | 2486 | 65175 | 9722 | 17394 | 179894 |  |
|  |  |  |  |  |  |  |  |  |  |

Screenshots de datos empíricos

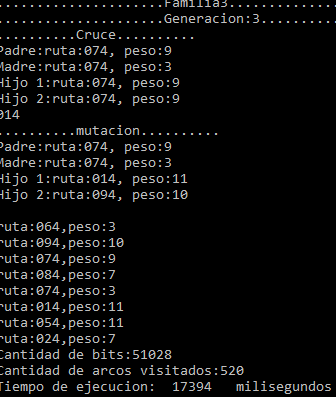
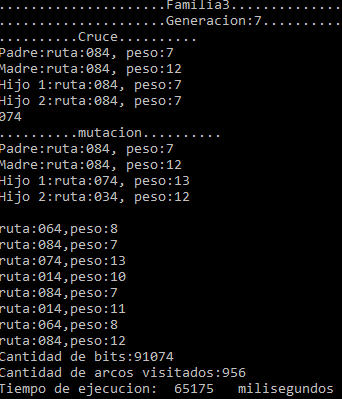
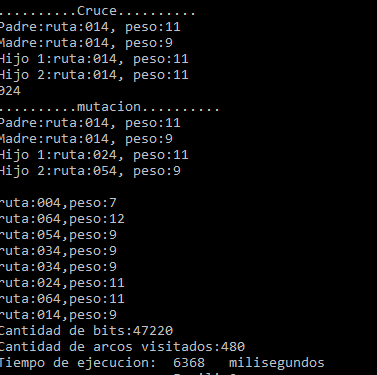
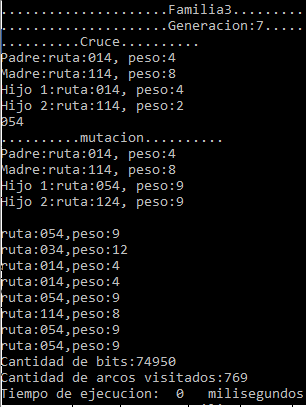


50% 6x6 50% 9x9



50% 8x8 50% 10x10

30% 9x9 30% 8x8



30% 10x10 30% 6x6

Factor de crecimiento del Genético

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Tamaño1 a tamaño 2  6x6 | Tamaño2 a tamaño3  8x8 | Tamaño3 a tamaño4  9x9 | Tamaño1 a tamaño3  10x10 | | Tamaño1 a tamaño4 |
| Factor de talla | 8/6=1.333  h=30%,  h=50% | 9/8=1.125  h=30%,  h=50% | 10/9=1.111  h=30%,  h=50% | 9/6=1.5  h=30%,  h=50% | | 10/6=1.666  h=30%,  h=50% |
| Memo | 0.6300200133  0.604303087 | 1.928716645  0.2003014441 | 0.560291631  4.711903358 | 1.215130087  1.210452464 | | 0.6808272181  5.70353503 |
| Arcos | 0.6241872562  7.912408759 | 1.991666667  0.1752767528 | 0.5439330544  5.989473684 | 1.243172952  1.386861314 | | 0.6762028609  8.306569343 |
| Tiempo | 6368/0  2486/0 | 10.23476759  3.91069992 | 0.266881473  18.5038058 | 65175/0  9722/0 | | 17394/0  179894/0 |
|  |  |  |  |  | |  |
| Clasificación en notación O | | | | | O(n) | |

Se logra determinar que la complejidad del algoritmo es O(n) por la comparación de los datos con el factor de talla. El algoritmo es tratable debido a que tiene solución de una manera óptima.

Medición analítica del backtraking

|  |  |
| --- | --- |
|  | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| static void Backtraking(string VI, string VF, string comb, int k, bool[,] lista, int dist){  if (dist < distanciaCorta && verticesRecorridos(comb) <= saltos)  {  if (k == Int32.Parse(VF.ToString()))  {  Console.WriteLine(comb + dist);  distanciaCorta = dist  caminoCorto = comb;  }  else  {  for (int i = k; i < n; i++)  {  if (!lista[k, i])  {  lista[k, i] = true;  for (int j = 0; j < n; j++)  {  if (i != j && i != Int32.Parse(VF.ToString()))  {  bits =bits+ 32;  for (int h = 0; h < n; h++)  lista[i, h] = true;  Backtraking(VI, VF, comb + i + "," + j + " | ", j, lista, dist + matrizRutas[i, j]);  }  }  lista[k, i] = false;  break;  }  }  }  }  else  {  poda++;  } | 6  2  1  2  M  1  N  2  N  1  Nn  1 |
| Total | 15+2n+nn |
| Clasificación en notación O | O(nn) |

Medición empírica backtraking

* 1. Código algoritmo **acotado por saltos:**

| Operaciones | Backtracking | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tamaño 1 | | Tamaño 2 | | | Tamaño 3 | | Tamaño 4 | | | | |  |
| h = 5 h = 3 | | h = 50 h = 30 | | h =150 h =250 | | | h =300 h =500 | |  | |
| Memoria | 341328 | 400496 | 4083236608 | 5482361616 | 399327152080 | | 3511063093968 | 29572998338608 | 4220054286512 |  | |
| Podas | 218 | 258 | 50038 | 67295 | 199186 | | 1751432 | 3692576 | 526934 |  | |
| Tiempo de ejecución | 0 | 0 | 2 | 2 | 46 | | 20 | 3956 | 5259 |  | |

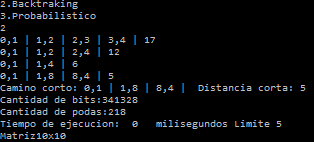
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Factor de crecimiento** | **Tamaño 1 a tamaño 2** | **Tamaño 2 a Tamaño 3** | **Tamaño 3 a tamaño 4** | **tamaño 4 a tamaño 1** |
| **Factor de talla** | 10 | 5 | 2 | 100 |
| **Memoria en bit** | 11962.79417 | 119.3523857 | 285.940724 | 12363633.47 |
| **Podas** | 229.5321101 | 35.00203845 | 0.3008589 | 2417.12844 |
| **Clasificación O** | O(nn ) | | | |

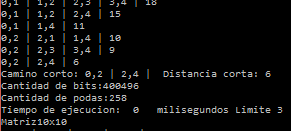
**Backtracking Factor de Talla saltos 50%:**

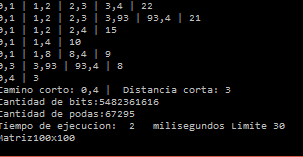
El algoritmo tiene una complejidad de O(nn) ,debido a que a que a partir de los datos se logra evidenciar que tiene un comportamiento exponencial conforme aumenta su talla. El algoritmo es intratable a que genera la solución, pero no lo hace de manera óptima.

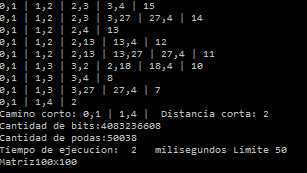
**Pruebas:**

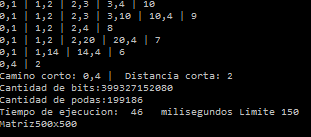
**Backtracking:**

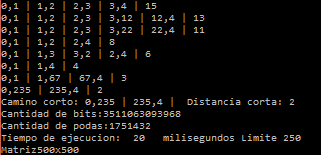
**Matriz 10x10 lim -> 5 Matriz 10x10 lim -> 3**

****

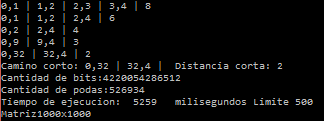
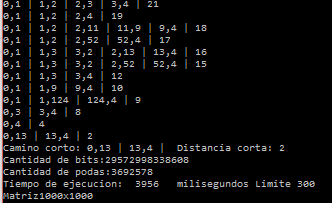
**Matriz 100x100 lim -> 30 Matriz 100x100 lim -> 50**

****

**Matriz 500x500 lim -> 150 Matriz 500x500 lim -> 250**

** Matriz 500x500 lim -> 150**

**Matriz 1000x1000 lim -> 500 Matriz 1000x1000 lim -> 300**

****

* 1. Código algoritmo :\_\_\_Probabilístico\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  | Medición de líneas ejecutadas en el peor de los casos  (línea por línea) |
| for (int h = 0; h <100; h++)  {  int i;  while (VF != VI)  {  double sumatoria = calcularSumatoria(VI);  calcularVisibilidad(VI, sumatoria);  Random rnd = new Random();  double numA = rnd.NextDouble();    //  for (i = 0; i < n; i++)  {  if (numA <= listaProbabilidadXY[i] & listaProbabilidadXY[i] != 0 & limite >= verticesRecorridos(combRutas))  {  if (!matrizVisitados[i])  {  matrizVisitados[i] = true;  costo =costo+ matrizRutas[VI, i];  combRutas = combRutas + VI + "," + i + "|";  VI = i;  break;  }  else  {  numA = rnd.NextDouble();  i = 0;  }  }  }  if(!(limite >= verticesRecorridos(combRutas))& VI != VF)  {  i = 0;  combRutas = "";  }  }  if (distanciaCorta >= costo & limite >= verticesRecorridos(combRutas))  {  distanciaCorta = costo;  caminoCorto = "";  caminoCorto =combRutas;  Console.WriteLine("Hormiga " + h+"Costo"+costo);  }  for (int j = 0; j < n; j++)  {  matrizVisitados[j] = false;  }  //agregarEnMatriz(combRutas, costo);  //imprimirMatrizFeromonas();  combRutas = "";  costo = 0;  VI = vAux; | 100  1  m  2n2+1  2  n  3  1  1  1  1  1  1  2  1  1  2  3  N2  1  3 |
| Total | 124+3n2+2n3 |
| Clasificación en notación O | O(n3) |

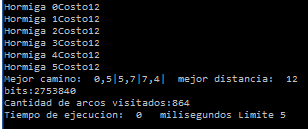
| Operaciones | Probabilístico | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tamaño 1 | | Tamaño 2 | | | Tamaño 3 | | Tamaño 4 | | |
| h = 5 h = 3 | | h = 50 h = 30 | | h =150 h =250 | | | h =300 h =500 | | |
| Memoria | 2753840 | 1908976 | 33607856 | 384465792 | 136834366080 | | 6699806272 | 35579827744 | 191576359664 | |
| Arcos visitados | 824 | 738 | 2217 | 2041 | 4076 | | 1919 | 3402 | 8220 | |
| Tiempo de ejecución | 0 | 0 | 6 | 30 | 654 | | 26 | 3402 | 9654 | |

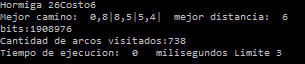
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Factor de crecimiento** | **Tamaño 1 a tamaño 2** | **Tamaño 2 a Tamaño 3** | **Tamaño 3 a tamaño 4** | **tamaño 4 a tamaño 1** |
| **Factor de talla** | **10** | **5** | **2** | **100** |
| **Memoria en bit** | **12.20399733** | **119.3523857** | **1.201930633** | **69566.98997** |
| **Vértices recorridos** | **2.690533981** | **0.8655841227** | **4.28348098** | **9.975728155** |
| **Clasificación O** | **O(n3 )** | | | |

Factor de crecimiento

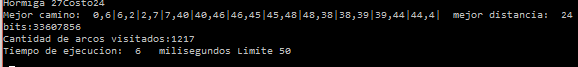
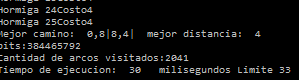
El algoritmo probabilístico es un algoritmo tratable no es el más eficiente ni optimo, pero realiza su trabajo de manera muy rápida. Es una posible solución para los algoritmos con valores de tiempo intratables y que sean insolucionables por esta característica.

**Probabilístico:**

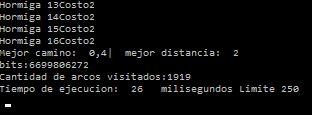
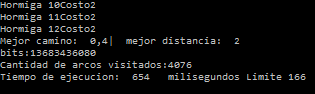
**Matriz 10x10 lim -> 5 Matriz 10x10 lim -> 3**

****

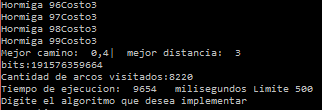
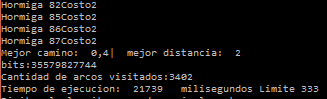
**Matriz 100x100 lim -> 30 Matriz 100x100 lim -> 50**

****

**Matriz 500x500 lim -> 150 Matriz 500x500 lim -> 250**

****

**Matriz 1000x1000 lim -> 500 Matriz 1000x1000 lim -> 300**

****

**Conclusiones**

Para las estrategias de diseño de algoritmos analizadas con un problema de encontrar la ruta corta, se determina que el backtraking es más eficiente a la hora de encontrar la solución, debido a que explora un árbol con cada una de las soluciones posibles de forma recursiva dejándose con él la mejor. Este tipo de técnica al ser exponencial se convierte en un algoritmo intratable, por lo tanto de sacrifica la optimización con tal de obtener el resultado más acertado.

El algoritmo probabilístico conforme aumente su factor de talla el seguirá realizando su búsqueda de manera aleatoria y de igual manera sacando en algunos casos la mejor ruta. Este algoritmo sacrifica una cierta parte del 100% de efectividad en la solución con tal de conseguir una respuesta más rápida, este porcentaje de desacierto puede disminuirse volviendo a mandar el proceso repetidas veces.

**Bibliografía**

Lopez, J. C. (2010).Adictos al trabajo. Recuperado de <https://www.adictosaltrabajo.com/tutoriales/jgap/#07>

Departamento de Lenguajes y Ciencias de la Computación.(2009).Recuperado de <http://www.lcc.uma.es/~av/Libro/CAP6.pdf>

Newmann ,F.(2006). Runtime Analysis of a Simple Ant Colony Optimization Algorithm.

Metodos Optimizacion(1/Jun/2009).¨Colonia de Hormigas¨ Recuperado de:  
[https://www.youtube.com/watch?v=7adfoRX1JFU](https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3D7adfoRX1JFU&h=BAQGS0diR)  
  
Metodos Optimizacion(1/Jun/2009).¨Colonia de Hormigas¨ Recuperado de:  
[https://www.youtube.com/watch?v=N\_rkfK5QRR8](https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DN_rkfK5QRR8&h=BAQGS0diR)  
  
Metodos Optimizacion(1/Jun/2009).¨Colonia de Hormigas¨ Recuperado de:  
[https://www.youtube.com/watch?v=eJ5XGrrSKIw&t=19s](https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DeJ5XGrrSKIw%26t%3D19s&h=BAQGS0diR)