ALGORITMO PARA LA EXPLORACION DE TODOS LOS VALORES POSIBLES EN EL PROBLEMA DEL AGENTE VIAJERO (TPS)

Algorithms for the Exploration of all possible values on the problem of agents travellers

RESUMEN
El siguiente artículo muestra algunas aplicaciones empleadas para probar soluciones acerca del problema del agente viajero y busca mostrar ejemplos de algoritmos desarrollados para este fin, en esta primera parte el de fuerza bruta y en la siguiente los basados en algoritmos heurísticos.

PALABRAS CLAVES: Optimizaron, algoritmo, programación orientada a objetos, estructuras de datos, listas.

ABSTRACT
This paper show some techniques to create an algorithm capable to analyze all possible solutions in Travelling sales person problem.
.

KEYWORDS: *Optimization, algorithms, data structure, list.*

CARLOS ANDRES LOPEZ
Ingeniero de Sistemas
Profesor catedrático
Universidad Tecnológica de Pereira
asraset@yahoo.com.mx

JAIRO ALBERTO MENDOZA V.
Ingeniero Eléctrico, MsC
Profesor asistente
Universidad Tecnológica de Pereira
jam@utp.edu.co

ERNESTO CUARTAS
Ingeniero electrónico
Universidad Nacional de Colombia
Universidad Autónoma de Manizales

1. INTRODUCCIÓN

El problema del agente viajero o TPS por sus siglas en inglés (Travelling Salesperson Problem) es uno de los problemas mas famosos y complejos de las ciencias computacionales y ha sido abordado por varias ramas de la ingeniería y por distintas razones, su principal aplicación es la de rutear desde distintas perspectivas, ya sea un proceso que lleva una secuencia específica o una distribución de carácter logístico en la que intervienen elementos del transporte, buscando la mejor ruta posible con criterios de economía en distancia o en costo [1][2]. Proveer soluciones contribuye a mejorar tareas y procesos en distintos ámbitos, científicos e industriales, proponiendo alternativas para el mejor uso de los recursos. Disciplinas que abordan este tema son la investigación de operaciones y las ciencias informáticas como algoritmia y teoría de grafos.

2. DESCRIPCION DEL PROBLEMA

El problema del agente viajero es el siguiente: dado un numero n de ciudades, las cuales poseen características de costos o distancias, el vendedor partiendo de una ciudad debe recorrer todas las demás exactamente una vez hasta retornar a su punto de partida. Este tipo de recorridos bajo la teoría de grafos es lo que se conoce como un recorrido Hamiltoniano[3].

Fecha de Recepción: 5 de Junio de 2008.
Fecha de Aceptación: 24 de Julio de 2008.

Desde luego que por cuestiones aplicativas el problema puede diferir en su planteamiento en cuanto a que podría proponerse por ejemplo que el vendedor parta de un punto especifico o predeterminado, pensemos en que la solución es para una empresa que posee un punto central de distribución como una fabrica o una bodega, o todavía mas interesante si se tienen regiones claramente diferenciadas con varios puntos de distribución obligatoria, como ocurre con zonas de electrificación asociadas a una subestación.

En cuanto a la dificultad del algoritmo su dificultad esta ligada al tiempo de solución del problema, nos referimos al tiempo que emplea un computador en llegar a una solución. Un análisis a este punto es el siguiente: si se tienen N ciudades el número posible de recorridos es (n-1)!

Teniendo en cuenta que :

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$$

El tiempo de ejecución del algoritmo es grosso modo f(n)=(n-1)!, o en otros términos para encontrar una solución para 6 ciudades f(6)=5!=120, es decir que si empleo fuerza bruta tendrán que analizarse todas las rutas posibles analizando cada vez si el recorrido actual es el de menor costo, caso en el cual almacenaremos ese dato y el recorrido.

Debido a que no se encuentran fácilmente ejercicios resueltos y que además planteen el tiempo de ejecución de la maquina, entendiendo que este tiempo de ejecución es el periodo de tiempo requerido para que un computador con unas características determinadas tarde en analizar el modelo, como primera parte de la investigación se construyo un programa que permite adicionar n ciudades y configurar su costo, y para ese planteamiento encontrar la solución probando todos y cada uno de los caminos posibles teniendo en cuenta además el tiempo requerido para esta tarea[4][5].

Posteriormente se implemento algoritmo heurístico para la solución del TPS y se recolectaron datos para evaluar las soluciones encontradas y el tiempo necesario en comparación con un algoritmo de fuerza bruta[6].

3. ANALISIS DE LA SOLUCION

El primer algoritmo planteado se basa en árboles n-arios y la recursividad del proceso constructivo de las combinaciones generadas, puesto que de cada rama que se desprende es una posibilidad que excluye términos hasta un punto donde no es posible excluir mas, debe recorrerse cada rama del árbol hasta que no existan hojas, en ese momento se debe realizar una reconstrucción del camino recorrido.

Para realizar esta tarea, además debe crearse una interfaz que permita adicionar tantas ciudades se deseen y además un valor de costo o distancia que desee analizarse. Esta información por ser de carácter dinámico requiere que se empleen apuntadores o una interfaz a una base de datos configurada para tal fin.

Como lenguaje de desarrollo se empleo Visual Basic .Net del entorno de desarrollo Visual Studio .Net 2005, donde se creo como librería que posee diferentes objetos los cuales facilitan el manejo de los datos y el almacenamiento de las soluciones.

Inicialmente se creo la clase Ciudad y la clase Nodo (Figura 1), los métodos y atributos descritos deben ser implementados para un correcto funcionamiento del algoritmo, la siguiente es la descripción de métodos y parámetros.

Esta es la descripción de los atributos de la clase ciudad: publico nombre tipo cadena, público costo tipo entero, privado activo tipo booleado.

Los métodos son: activar que pone en verdadero el atributo activo, y desactivar que pone en falso el atributo activo.

En la clase nodo los atributos son los siguientes: privado nombre de tipo cadena, privado ciudades lista de tipo ciudad, privado anterior de tipo ciudad, privado siguiente de tipo ciudad.

Los métodos de la clase nodo son: ad_ciudad que permite agregar una ciudad a la lista, eliminar_ciudad que permite eliminar una ciudad de la lista, activar conexión activa las ciudades de la lista que están conectadas con el nodo, valor_conexion retorna el valor de una conexión de una ciudad con el nodo, ad_siguiente adiciona una ciudad como próximo destino del nodo, ad_anterior adiciona una ciudad que previamente esta conectada con el nodo, valor_siguiente retorna el valor de la conexión con la siguiente ciudad, valor_anterior retorna el valor de la conexión con la ciudad conectada previamente con el nodo.

La clase nodo se adapta a un modelo del elemento de una lista doblemente enlazada, esto debido a la naturaleza del problema que concuerda con un grafo conectado con un circuito Hamiltoniano y cuya representación computacional es la que estamos presentando.

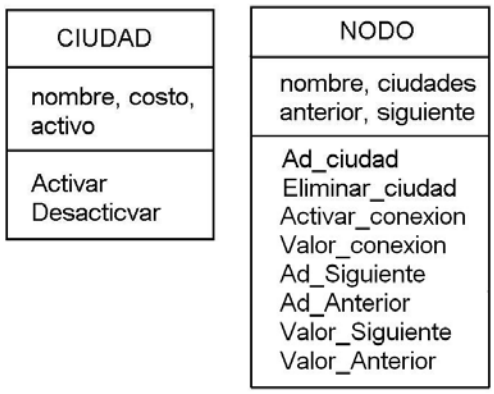

Figura 1: diagrama de clases ciudad y nodo

4. MODELO PROPUESTO

Sea L una lista, es decir una estructura dinámica de datos interconectados, dinámica debido a que puede cambiar de tamaño según las necesidades del usuario, e interconectadas porque cada dato, el cual es de naturaleza compuesta y que llamaremos nodo, esta ligado a otro puesto que de alguna forma conoce quien es el siguiente dato. Ahora, las razones por las cuales los nodos se interconectan obedecen a un propósito de representación de la conexión existente entre ellos.

Las listas pueden ser lineales, las que su ultimo elemento apunta hacia un lugar vacío, circular, aquella cuyo último elemento apunta al primero o doblemente enlazada, es decir que cada nodo tiene la manera de apuntar hacia el

próximo elemento o elemento subsiguiente y a la vez apunta hacia el elemento anterior.

Por otro lado, otra cosa que necesitamos entender es que buscar las trayectorias pasando por todas las que son posibles de generar requiere que además necesitemos otro concepto, el de árbol. Un árbol es una estructura no lineal en el que cada nodo puede apuntar a uno o varios nodos, la caracterización de los nodos es la siguiente:

Nodo padre: nodo que apunta hacia otro.
Nodo hijo: nodo al que apunta otro nodo.
Nodo raíz: nodo que no tiene padre.
Nodo hoja: nodo que no tiene hijos.

La caracterización de los árboles en relación con su tamaño es la siguiente:

Orden: numero potencial de hijos que puede tener cada nodo del árbol, por ejemplo un nodo que a lo sumo puede apuntar a otros dos es de orden dos, si puede apuntar a tres es de orden tres, etc.
Grado: numero de hijos que tiene el elemento con más hijos dentro del árbol.
Nivel: distancia a la raíz medida en nodos.
Altura: se define como el nivel del nodo de mayor nivel.

Los árboles de uso mas frecuente son los árboles binarios o árboles de orden dos, sin embargo para nuestro caso de estudio veremos que se trata de un árbol n-ario, ya que en realidad no conocemos su orden hasta tanto el usuario no nos defina el número total de ciudades que vamos a analizar [9][10].

De lo anterior podemos decir que la clase ciudad realmente es un nodo, y que para el problema propuesto el análisis de todas las posibles ciudades se fundamenta en construir una estructura de tipo árbol n-ario, definir una trayectoria cerrada y evaluar el costo de esa trayectoria teniendo presente almacenar el mejor camino en términos de costo y los nodos que componen esa trayectoria mas económica.

Podremos considerar que cada trayectoria debe ser tratada como una lista cerrada, pues la definición se ajusta en todos los aspectos, sin embargo la búsqueda de la solución realmente obedece al proceso de recorrer un árbol cuyo inicio es la lista completa de ciudades a recorrer, de allí se desprende una lista donde debemos considerar todos los posibles caminos a seguir a excepción de la ciudad ya considerada y así sucesivamente (figura 2).

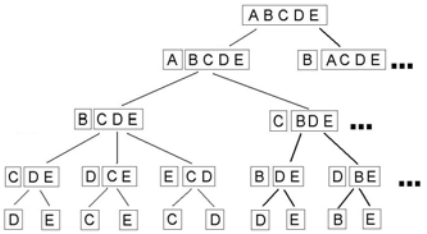

Figura 2: árbol n-ario que representa todas las posibles trayectorias.

Ahora bien, de este modelo podemos sacar todas las combinaciones de ciudades posibles que van desde la primera ciudad que consideremos hasta la última que es posible generar, pero debe tenerse en cuenta que una vez se genera una lista para que esta sea cerrada debe adicionársele el nodo inicial para que sea cerrada.

En la figura dos podemos apreciar como de una lista de cinco ciudades (A, B, C, D, y E) la primera ramificación toma A excluyéndola y considera las restantes, las cuales generan ramificaciones de las ciudades restantes (B, C, D y E), luego se toma B excluyéndola y repitiendo el proceso hasta que no existen más ciudades que excluir.

5. ALGORITMO DE FUERZA BRUTA

El algoritmo descrito a continuación fue implementado en C# y empleo una librería hecha específicamente para contener las clases necesarias para implementar tanto el algoritmo de fuerza bruta como la solución heurística que se describirá más adelante.

Inicialmente se debe contar con un método que permita capturar la información de las ciudades, la cual debe ser adicionado a una estructura de tipo lista según el lenguaje empleado. De ahora en adelante esta lista la llamaremos Ciudades y supondremos una variable llamada Elementos que nos indicara cuantos elementos están contenidos en la lista Ciudades.

Definimos las siguientes variables de tipo entero: i, límite.
Definimos las siguientes variables que son listas de tipo nodo: Trayectoria, Ciudades, Tminima.
Definimos el siguiente atributo de tipo nodo: primera, mínimo.

Método que analiza las trayectorias:

```
Publico vacío Analisis ()
{
  Entero i, Limite;
  Lista Trayectoria, Ciudades;

  Limite=Ciudades.Elementos();
   Para i=0 hasta Limite
```

```
   {
      Trayectoria.Limpiar();
      primera= Ciudades(i).nombre;
      Construir(Ciudades,
Ciudades(i).nombre,Trayectoria);
    }//fin para
} //fin metodo Analisis
```

Suponemos que los elementos de tipo lista contienen los siguientes métodos:
Adicionar: adiciona un elemento a la lista.
Limpiar: elimina todos los elementos de una lista.

La lista debe ser considerada también de alguna forma como un vector y nos referimos a cada nodo por el orden que lleve en la lista como si fuese un vector.

Método Construir: construye las listas y las evalúa cuando se llega a una hoja.

Definimos las siguientes variables de tipo entero: i, j, limite, limite2.
Definimos las siguientes variables que son listas de tipo nodo: auxiliar.
Definimos la siguiente variable de tipo Ciudad: primera.

```
Privado vacío Construir (Lista C, cadena Cad, Lista T)
{
  Entero i, j, Límite, Limite2;
  Lista Auxiliar;

  J=0;
  Limite=C.Elementos();
  T.Adicionar(Cad);

  Para i=0 hasta Limite
  {
    Si (C(i).nombre <> Cad)
      Auxiliar.Adicionar(C(i).nombre);
   }//fin para

  Si (Limite>0)
  {
    Limite2=Auxiliar.Elementos();
     Para i=0 hasta Limite2
     {
       Construir(Auxiliar, Auxiliar(i).nombre,T);
       Eliminar (T, Auxiliar(i).nombre);
      }
  }
  Si no
  {
     Evaluar (T);
     Eliminar (T,Cad);
  }//fin si
}//fin método Construir
```

Método Eliminar: permite sacar de una lista un elemento específico.

Definimos las siguientes variables de tipo entero: i, Límite.

```
Privado vacío Eliminar (Lista L, Cadena Cad)
{
   Entero i, Limite;

   Limite=L.Elementos();
   Para i=0 hasta Limite
   {
     If (L(i).nombre == Cad)
     {
       L(i).Eliminar();
       //instrucción de eliminación de una lista
      }
    }//fin para
}//fin método
```

Método Evaluar: toma una lista ciudades y evalúa el costo de ese trayecto.

Definimos las siguientes variables de tipo entero: i, Límite, Valor.
Definimos las siguientes variables de tipo Ciudad: Ant, Sig.

```
Privado vació Evaluar (Lista L)
{
  Entero i, Limite;
  Ciudad Ant, Sig;

  Limite=L.elementos();

  Sig=nuevo Ciudad();
  Ant=nuevo Ciudad();

  Sig.nombre=L(1).nombre;
  Sig.costo=L(1).costo;
  Sig.Activar();

  Ant.nombre=L(0).nombre;
  Ant.costo=L(0).costo;
  Ant.Activar();.
  L(0).Ad_siguiente(Sig);
  i=1
  Repita
  {
     L(i).Ad_anterior(Ant);
     Sig.nombre=L(i+1).nombre;
     Sig.costo=L(i+1).costo;
     Sig.Activar();
     L(i).siguiente=Sig
     Ant.nombre=L(i).nombre;
     Ant.costo=L(i).costo;
     Ant.Activar();
     i=i+1;
   }hasta (i==Limite)
   L(i).Ad_anterior(Ant);
```

```
   L(i).Ad_siguiente(prim);

   Para i=0 hasta Limite
   {
      Valor=Valor+L(i).Valor_siguiente();
   }
    Si (minimo>Valor)
    {
        Minimo=Valor;
        Copiar (L,Tminima)
    }
}
```

El método copiar vacía el contenido de la primera lista en la segunda.

6. EXPLICACIÓN DEL ALGORITMO

Antes que nada se debe tener en cuenta que su implementación esta pensada para lenguajes orientados a objetos y eventos, por lo tanto el método Análisis debe ser activado por el usuario bajo condición que la lista Ciudades que es de tipo nodo, posea valores.

La variable trayectoria es una lista que debe ser general con respecto a la aplicación, es decir, que se trata de una variable global que almacena datos de tipo cadena, así como también lo es la lista ciudades que almacena datos de tipo nodo.

En este método que funciona de manera recursiva, se toma una lista vacía que va a contener los nombres de las trayectorias de las ciudades, todos y cada uno que es posible generar, de la siguiente manera: toma de toda la lista de los nodos la primera ciudad y la almacena en trayectoria y llama al ciclo construir que excluye el que esta en trayectoria y continua con el resto de la lista, esto ocurre hasta que no quedan mas elementos y es en ese caso cuando se ha generado un camino cerrado, aquel donde se regresa al punto de partida.

Luego se evalúa retomando la información de los pesos de los caminos tomados y sumándolos en una variable llamada valor, el cual se compara con una llamada mínimo que debe inicializarse al inicio de la aplicación y que debe ser igual a la suma de los valores más altos de todos los nodos.

Si el valor es inferior al mínimo hallado, se considera que se ha encontrado un nuevo valor mínimo y además se almacena en una lista llamada Tminima, nombre dado por trayectoria mínima, y se almacena la lista de nodos en el orden en que se da esta condición, la de ser mínimo.

Finalmente puede mostrarse la mejor trayectoria y el costo de esa trayectoria.

7. CONCLUSIONES

Es importante y necesario párale estudio de cualquier problema proveerse hermanitas que amplíen la perspectiva del asusto tratado.

Se requieren conocimientos sólidos sobre estructura de datos para poder comprender y abordar el problema del TSP.

La programación orientada a objetos es un poderoso paradigma que al combinarlo con estructuras de datos nos acercan a soluciones eficientes y económicas en términos de memoria, lo que se hace importante para este tipo de problemas.

El programa expuesto en seudo código permite que investigador plasme este algoritmo en el lenguaje de su preferencia.

8. BIBLIOGRAFIA

[1]http://es.wikipedia.org/wiki/Problema_de_rutas_de_veh%C3%ADculos

[2]http://es.wikipedia.org/wiki/Problema_del_viajante

[3]http://catarina.udlap.mx/u_dl_a/tales/documentos/msp/castaneda_r_cy/indice.html

[4]http://ccc.inaoep.mx/~emorales/Cursos/NvoAprend/node84.html

[5]http://personal.telefonica.terra.es/web/criptologia/recerca13.html

[6]http://www.matcom.uh.cu/eda/Descargas.aspx

[7]http://www.programacion.net/tutorial/jap_data_alg/

[8]Investigacion de Operaciones en accion:Aplicacion del TSP en problemas de manufactura y logistica. Jose Luis Gonzales Velarde. Roger Z. Rios Mercado.Revista Ingenierias Mayo-agosto 1999 VolII No 4.

[9]Apuntes para el curso de estructura de datos en C/C++. Abdiel E. Caceres Gonzalez.ITESM-CCM. Junio 2, 2005.

[10]Algoritmos y estructuras de datos. Mario Stori. Jorge D'Elia, otros. Universidad Nacional del Litoral. Argentina.