

UNA GENERALIZACION AL PROBLEMA DEL VIAJANTE GTSP

Kesner Delia

Yankelevich Daniel

ESLAI - Escuela Superior Latinoamericana de Informática - cc 3193 (1000) -
Buenos Aires - Argentina

RESUMEN

Se plantea una generalización al problema del viajante (GTSP), analizando su complejidad. Se proponen distintas soluciones aproximadas haciendo uso de la heurística de Christofides, la estrategia Greedy, una extensión al Tree-Algorithm y el método de Local Search.

RESUMEN

Se plantea una generalización al problema del viajante (GTSP), analizando su complejidad. Se proponen distintas soluciones aproximadas haciendo uso de la heurística de Christofides, la estrategia Greedy, una extensión al Tree-Algorithm y el método de Local Search.

1 - INTRODUCCION

Los problemas combinatorios pueden ser resueltos de una manera bastante sencilla con algoritmos exponenciales. En la práctica, la mayoría de esos algoritmos tardarían miles o millones de años en hallar la solución óptima, aun en las computadoras más poderosas.

Sería ideal encontrar algoritmos más veloces que resolvieran esos mismos problemas, pero en muchos casos se sospecha que dichos algoritmos no existen. Es por eso, que se intenta encontrar algoritmos que obtengan soluciones aproximadas que no disten demasiado de la óptima. Para esto existen diversos métodos y técnicas.

En este trabajo, se presenta una generalización al problema del viajante (GTSP), que posee las características mencionadas.

En el capítulo 2 se plantea el problema y se propone un modelo utilizando grafos. En el tercer capítulo se introducen algunos conceptos sobre complejidad y se estudia en particular la del GTSP. El siguiente capítulo muestra el algoritmo exponencial que resuelve el problema. En el capítulo 5 se presentan los tres algoritmos polinómicos desarrollados para hallar soluciones aproximadas al GTSP. Finalmente, se introduce la técnica de local search, aplicándola al GTSP, exponiendo además algunos resultados teóricos obtenidos.

2 - EL GTSP

2.1 - PLANEO DEL PROBLEMA

Una empresa debe vender sus productos en las K ciudades más importantes del país, y dispone para ello de N viajeros que viven en algunas de estas ciudades. No hay dos viajeros que vivan en la misma ciudad. A cada uno de ellos se le asigna la cantidad de ciudades a recorrer, sin determinar cuáles serán estas. Se desea vender el producto en todas las ciudades, pero un viajante no podrá pasar por una ciudad recorrida anteriormente por otro viajante o por el mismo.

Se quiere determinar cuáles serán las ciudades que visitará cada viajante, de tal manera que la suma de las distancias recorridas por todos los empleados de la empresa sea mínima.

2.2 - MODELO

En principio el modelo más natural del problema es un grafo simple no dirigido [Har 59], donde los nodos representan ciudades y los arcos las posibles rutas. Es posible acceder desde una ciudad a cualquier otra, por lo que el grafo será completo.

Cada arco tiene además asociado un costo, que representa la distancia de una ciudad a otra. Notar que la distancia de la ciudad A a la B es la misma que de B a A .

Al viajante i se le asigna un número K_i de ciudades a recorrer.

Resolver el GTSP es dar a cada viajante el conjunto de ciudades a recorrer y la forma de hacerlo.

Notación:

$G=(V,E)$ grafo completo con $\#V=K$

A en $\mathbb{R}^{K \times K}$ tal que $A_{ij} = A_{ji}$ y $A_{ii} = \text{infinito}$, para todo $i = 1..K$ (matriz de costos)

V' incluido en V y $\#V' = N$ (ciudades donde viven los viajeros)

los nodos de V' están numerados de 1 a N

sumatoria $K_i = K$; $K_i \geq 1$ para todo $i=1..N$

Una solución será una partición de V , $P_1, \dots, P_N$, cada P_i asociado a un A_i tal que:

i) A_i incluido en E para todo i

ii) $A_i \cap A_j = \emptyset \rightarrow i \neq j$

iii) $\#P_i = K_i$

iv) Para todo $e=(v_1, v_2)$ perteneciente a A_i , v_1, v_2 pertenecen a P_i y existen únicos u_1, u_2 en P_i ($u_1 \triangleleft v_2, u_2 \triangleleft v_1$), tales que (u_1, v_1) y (v_2, u_2) pertenezcan a A_i .

Tal vez resulte contraintuitivo que la diagonal de la matriz de costos contenga infinito, pero es la forma en la que se indica que un arco no puede utilizarse, además este planteo es útil para algunas de las soluciones que se presentan.

Se podría pensar que el problema se reduce a buscar el camino hamiltoniano de costo mínimo de longitud K_1 , luego el de longitud K_2 en el grafo resultante de G menos los nodos del primer ciclo, y así sucesivamente, es decir, a resolver N veces el problema del viajante.

Lamentablemente hay contraejemplos, es decir grafos en los cuales el algoritmo insinuado no da la solución óptima.

3 - COMPLEJIDAD

3.1 - ALGUNOS CONCEPTOS SOBRE COMPLEJIDAD

La clasificación de los problemas se ha deducido de las actuales conjeturas sobre la existencia o inexistencia de algoritmos polinómicos para resolverlos. Los problemas "demostrablemente irresolubles" son aquellos para los que no puede haber ningún tipo de algoritmos, los "demostrablemente difíciles" solamente tienen algoritmos de tiempo exponencial.

Los problemas para los que se conocen algoritmos de tiempo polinómico se asignan a la clase P . El carácter de los demás problemas es menos seguro solamente se conocen algoritmos suyos de tiempo exponencial, pero no está demostrada la inexistencia de algoritmos polinómicos.

Estos últimos problemas se asignan a las clases NP y co_NP . Los problemas de la clase NP se podrían resolver con suerte en tiempo polinómico por tanteo. La clase co_NP contiene todos los problemas cuya solución es la disyuntiva "sí" o "no", y cuyas versiones complementarias "no" o "sí" pertenecen a la clase NP . La clase NP completa es un subconjunto de la clase NP compuesto por los problemas que tienen la siguiente propiedad: si uno de ellos pudiera resolverse mediante un algoritmo de tiempo polinómico, entonces todos los demás problemas de la clase NP podrían resolverse en tiempo polinómico. El descubrimiento de un tal algoritmo sería demostración de que las clases P y NP son idénticas.

Para que un problema forme parte de la clase NP no es preciso que exista un método eficiente de responder las preguntas afirmativa o negativamente. Lo que se quiere es que cuando la respuesta sea afirmativa exista un método breve y convincente que así lo demuestre.

Al parecer todos los especialistas en ciencias de la computación están de acuerdo actualmente en que los algoritmos cuyo tiempo de ejecución crece exponencialmente en función del tamaño de la entrada carecen de valor práctico. Otro punto a tener en cuenta es que la rapidez de un algoritmo es propiedad intrínseca del mismo, y es independiente de la máquina que lo ejecute [LePa 78].

Puede parecer grotesco definir un método matemático en términos de felices ideas y tanteos afortunados, pero no obstante, es un método perfectamente legítimo de precisar cuáles son los problemas que pertenecen a la clase NP. En teoría se podría incluso mecanizar el procedimiento construyendo un artefacto denominado "máquina de Turing no determinística". Este dispositivo puede efectuar todas las funciones de una máquina de Turing determinística, pero además en algunos puntos, puede elegir de más de una manera cual va a ser la etapa siguiente. La clase NP de los problemas no determinísticamente polinómicos está formada precisamente por aquellos problemas cuyos casos particulares de solución afirmativa podrían ser identificados por máquinas efectuando una sucesión relativamente breve de cálculos por tanteo. Se considera que si la máquina no "adivina" correctamente, el tiempo invertido hasta ese momento no se toma en cuenta, y se vuelve hacia el punto de partida olvidándose de lo ocurrido. Esto también se podría explicar en los siguientes términos: cada vez que se llega a una solución no satisfactoria, se hace backtracking y se resta al tiempo de ejecución total aquel que fue ocupado en el camino hacia dicha solución.

La frase "algoritmo no determinístico" fue propuesta por Floyd en 1967. En estos algoritmos Floyd alude al uso de la función "choice" para simplificar la descripción exhaustiva en estrategias de búsqueda. En el año 1970 Fikes describe un sistema completo de resolución de problemas en el cual estos son especificados en un lenguaje procedural que permite el uso de las funciones choice no determinísticas. Para que ello sea posible también se introdujeron los conceptos de "success" y "failure", donde "success" hace exitoso al algoritmo y "failure" indica un fracaso. De esta manera, un algoritmo no determinístico puede pensarse como un algoritmo que realiza copias de sí mismo al hacer un choice y una copia "muere" si ejecuta un failure, es decir, si falla [Nil 71].

Se dice que un problema pertenece a la clase NP_HARD si tiene la siguiente propiedad: no se sabe de su pertenencia a la clase NP, pero si fuera NP sería NP_completo. Esto quiere decir que es por lo menos tan complicado como un problema NP_completo.

3.2 - ALGUNOS CONCEPTOS SOBRE OPTIMIZACIÓN

DEFINICIÓN 1:

Un problema de optimización es un conjunto M de instancias. Cada instancia es un par (F, c) , donde F es un conjunto de soluciones factibles y c es una función de costo, $c: F \rightarrow \mathbb{R}$.

DEFINICIÓN 2:

Un problema de optimización combinatorio (POC) es un problema de optimización tal que para toda instancia (F, c) , F es finito.

De ahora en adelante se asume que F y c son dadas implícitamente en términos de dos algoritmos A_f y A_c , ya que la cantidad de soluciones factibles puede ser muy grande.

El algoritmo A_f , dado un objeto combinatorial f y un conjunto S de parámetros, decide cuando f es un elemento del conjunto F (el conjunto de soluciones factibles especificado por los parámetros dados).

El algoritmo A_c , dada una solución factible f y otro conjunto de parámetros Q , devuelve el valor de $c(f)$.

Una instancia de un POC puede ser definido entonces como la representación de los parámetros S y Q .

Ejemplo:

Una instancia del problema del viajante con N ciudades tiene como parametro S al entero N , y como parametro Q a la matriz simetrica de distancias entre las distintas ciudades. El algoritmo A_1 , dado un objeto f y el parametro N , examinara cuando f es un tour de N ciudades. El algoritmo A_2 , dado un tour f y la matriz simetrica de distancias, calcula el costo $c(f)$ sumando todas las distancias atravesadas por el tour.

DEFINICION 3:

La version de optimizacion de un POC es aquella que dados parametros S y Q para los algoritmos A_1 y A_2 , encuentra la solucion optima factible.

DEFINICION 4:

La version de evaluacion de un POC es aquella que dados S y Q , encuentra el costo de la solucion optima.

Bajo la hipotesis que A_2 es un algoritmo de tiempo polinomico (el costo c no es facil de computar), la version de evaluacion no puede ser mas facil que la version de optimizacion, pues dada la solucion optima, hallar su costo solo lleva tiempo polinomico.

DEFINICION 5:

La version de reconocimiento de un POC es aquella que dados S, Q y un entero L responde a la pregunta Existe una solucion factible f en F tal que $c(f) \leq L$?

La version de reconocimiento es una pregunta que puede ser respondida por SI o por NO. Responder esta pregunta no es mas facil que resolver la version de evaluacion, pues una vez resuelta esta, solo se debe comparar el costo optimo $c(f^*)$ con L , y responder SI si $c(f^*) \leq L$.

Por lo tanto cada una de las versiones de optimizacion, evaluacion y reconocimiento (en ese orden) no es mas facil que las anteriores asumiendo que c es computable en tiempo polinomico.

Cabe la pregunta: Son las tres versiones de la misma complejidad? Bajo la hipotesis de que el costo de una solucion optima es un entero cuyo logaritmo esta acotado por un polinomio en el tamaño de la entrada, se puede mostrar que la version de evaluacion puede ser resuelta eficientemente cuando se resuelve la version de reconocimiento. Para mostrar esto debemos evaluar el costo optimo $c(f^*)$ formulando la pregunta Es $c(f^*) \leq L$? para varios valores de L . Esto puede ser hecho mediante busqueda binaria y mediante la hipotesis que $\log(c(f^*))$ esta acotado por un polinomio en el tamaño de la entrada, podemos hacer eficiente uso del algoritmo que resuelve la version de reconocimiento para resolver la version de evaluacion.

No hay un metodo general para resolver la version de optimizacion haciendo un uso eficiente del algoritmo para la version de evaluacion, pero para algunos problemas esto se pudo hacer.

3.3 - COMPLEJIDAD DEL TSP

El GTSP es una generalizacion del problema del viajante (TSP), en el sentido que esta planteado para N viajeros y si $N=1$ es justamente el TSP. Por lo tanto como el problema del viajante es NP_completo [Lepa 81], el GTSP es por lo menos NP_completo.

Intuitivamente la complejidad del GTSP aparece por dos lados:

1) Hallar la particion optima que cumpla las restricciones. Cada conjunto de la particion representara las ciudades que debe visitar cada viajante.

2) Hallar dentro de cada conjunto de la particion, un camino hamiltoniano de costo minimo; el camino que cada viajante debiera recorrer.

TEOREMA 1:

Las versiones de evaluacion y de reconocimiento del GTSP son NP_COMPLETAS, la de optimizacion es NP_HARD.

Demostracion:

La version de evaluacion se puede formular del siguiente modo: Dados N nodos de un grafo de K nodos, y numeros $K_i > 0$, con la suma de los $K_i = K$, se quiere hallar una particion de los K nodos en N subconjuntos, tal que cada subconjunto contenga uno y solo uno de los N nodos, y tal que cada subconjunto contenga K_i nodos. Llamemos a esta condicion que ponemos a la particion condicion Cond_part.

El siguiente algoritmo nodeterministico resuelve el problema en tiempo polinomico. Este algoritmo falla si en el grafo dado no es posible hallar la particion que cumpla las condiciones o si no hay ninguna solucion de costo menor o igual que L , y tiene exito si existe una solucion de costo menor o igual a L .

Procedure VIAJANTE GENERALIZADO($G=(V,E)$:GRAFO, L:ENTERO);

var

{ V es el conjunto de los nodos del grafo }
conj, aux_conj:conjunto de conjuntos de nodos;
aux_V,vert:nodo;
bool:boolean;
costo:ENTERO;

```
begin
  aux_V:=choice(V);
  V:=V-aux;
  conj:={{aux}};
  while V  $\neq$  {} do
    begin
      vert:=choice(V);
      bool:=choice({true,false});
      if bool then
        begin
          conj:=conj U {vert};
        end
      else
        begin
          aux_conj:=choice(conj);
          conj:=conj-aux_conj;
          aux_conj:=aux_conj U {vert};
          conj:=conj U aux_conj;
        end;
      V:=V-vert;
    end;
```

(se obtuvo una particion del conjunto de nodos del grafo, en conj)

```
if Cond_part then
  begin
    costo:=0;
    while conj  $\neq$  {} do
      begin
        aux_conj:= choice(conj);
        costo:= costo + EVAL_TSP(SUBGRAFO_EXPANDIDO(aux_conj));
        conj:= conj-aux_conj;
      end;
    end;
    else failure;

    if costo  $\leq$  L then succes
      else failure
    end;
```

Es sencillo observar que este algoritmo es polinómico. La primer parte, que halla la particion, ejecuta K ciclos, ya que en cada ciclo retira un nodo del conjunto de vértices y termina cuando el conjunto esta vacío

La segunda parte es del orden de la cantidad de conjuntos de la partición, este número es igual al número de viajeros que está acotado por la cantidad de nodos del grafo, es decir por K , y, para cada conjunto, se realiza la llamada a *EVAL_TSP* que es polinómico. Luego, el algoritmo es polinómico.

EVAL_TSP es un algoritmo que dado un grafo (en este caso el subgrafo expandido del conjunto de nodos pasados como parámetros) devuelve el costo de la solución óptima. Este algoritmo es polinómico, ya que el problema del viajante pertenece a NP (polinómico no determinístico).

Como el problema del viajante es un caso particular de este, y el problema del viajante es NP_COMPLETO este problema es por lo tanto NP_COMPLETO (en la versión de reconocimiento).

Por lo argumentado más arriba, dado que la versión de reconocimiento del GTSP es NP, también lo es en su versión de evaluación. La idea es la misma, y pese a no ser una prueba formal lo expuesto es suficiente para decir que el GTSP es entonces NP_COMPLETO, aun en su versión de evaluación. Es sencillo hallar un algoritmo polinómico que dada una solución calcule su costo.

Para completar la demostración, por lo ya señalado, es conveniente además pedir las siguientes condiciones:

1) Los costos de las soluciones factibles al problema tienen una cota superior. Esta cota debe ser una función polinómica de la entrada del problema, en nuestro ejemplo, el número de nodos. Esto es posible teniendo, por ejemplo, una cota al costo de un arco. Si esta cota es H se puede acotar el costo de la solución por:

$$N \\ H * \sum_{i=1} K_i$$

Este número a su vez es menor que $H * N^2$, ya que el número de viajeros es menor que la cantidad de nodos del grafo ($\#V' < \#V$).

2) Los costos de las soluciones deben ser enteros. En realidad, alcanza con pedir que los costos sean discretizables, en la práctica esto siempre sucede, ya que jamás se trabaja con un número no acotado de decimales. Por lo tanto, siempre podemos pensar a los costos como enteros.

La versión de optimización es NP_HARD, ya que trivialmente, si fuera NP sería NP_COMPLETO, por ser el viajante un caso particular de ella.

4 - SOLUCION EXPONENCIAL AL GTSP

Como en todo POC existe una solución trivial, y consiste simplemente en explorar exhaustivamente todas las posibilidades.

Se hallan todas las particiones posibles del conjunto de ciudades tal que el cardinal de los conjuntos de la partición sean la cantidad de ciudades que debe recorrer cada viajante.

En cada una de estas particiones, y para cada uno de los conjuntos de ciudades de cada partición, se busca un camino hamiltoniano de costo mínimo. Así se obtiene para cada partición un costo que resultaría de sumar todos los costos de los caminos hamiltonianos mínimos hallados.

Dado un algoritmo que halla particiones distintas de un conjunto de cardinal K [NiWi 75], es muy fácil agregar restricciones tales como:

- la partición debe contener N conjuntos.
- cada subconjunto i de la partición debe tener cardinal $K_i - 1$.

De las K ciudades, solo interesan aquellas de las que no parte ningún viajante (estas son $K - N$). De ellas se debe ver todas las maneras de tomar $K_1 - 1$ ciudades para el primer viajante (ya que la ciudad de donde parte seguro la va a recorrer), $K_2 - 1$ para el segundo viajante, y así sucesivamente. Si llamamos $C(r, n)$ al número combinatorio "t tomados de n ", el número de particiones que nos interesan son:

$$C(K - N, K_1 - 1) * C(K - N - (K_1 - 1), K_2 - 1) * \dots * C(K_N - 1, K_N - 1)$$

este número es precisamente

$$\frac{(K - N)!}{((K_1 - 1)! * \dots * (K_N - 1)!)}$$

5 - SOLUCIONES NO OPTIMAS AL GTSP

5.1 - RAZONES PARA DISEÑAR ALGORITMOS POLINOMICOS

El GTSP dista mucho de ser solo un problema teórico. Este problema tiene un número muy grande de aplicaciones, desde recorridos y viajes hasta tareas de scheduling, y quizás aun mas que las del viajante al ser mas general que aquel. Por este motivo es importante y necesario plantear algoritmos que aunque no den soluciones optimas, hallen algunas "buenas"

Dos técnicas pueden ser usadas como modo de resolución de problemas para los cuales no se conocen algoritmos polinómicos:

1-Un problema Q para el que no se conoce algoritmo polinómico es reemplazado por un problema Q' el cual tiene un algoritmo polinómico conocido que lo resuelve cuya solución es aproximada a la de Q .

2-Un subconjunto relativamente pequeño de posibles soluciones de Q es examinado usando un criterio de selección razonable, intuitivo y generalmente pobre. La 'mejor' de estas soluciones potenciales es tomada como solución del problema Q .

[Ha 78].

5.2 - ALGORITMOS DE TIEMPO POLINOMICO PARA EL TSP

Para el TSP existen algoritmos que dan soluciones aproximadas en tiempo polinómico. Entre estos algoritmos se encuentran el GREEDY [AHU 83], el TREE - ALGORITHM, y el de CHRISTOFIDES [Pap 82].

DEFINICION 6:

Sea A un algoritmo de optimización con función de costo positiva C, sea A' otro algoritmo que da una solución factible F' y sea F la solución óptima que da el algoritmo A. Diremos que A' es K-aproximado para A ($K \geq 0$) si y solo si

$$\frac{c(F') - c(F)}{c(F)} \leq K$$

TEOREMA 2: El algoritmo TREE_ALGORITHM es 1-aproximado [Pap 82].

TEOREMA 3: El algoritmo de Christofides es 1/2-aproximado [Pap 82].

TEOREMA 4:

A menos que $P = NP$ no hay algoritmos polinómicos ϵ -aproximados para el problema del viajante para ningún $\epsilon > 0$.

Es necesario observar que en los algoritmos no óptimos citados para el TSP, se supone que los grafos sobre los que se van a aplicar los algoritmos cumplen la desigualdad triangular, dicho de otra forma, que los costos de los arcos tienen las propiedades de la distancia euclídea. La desigualdad triangular puede plantearse como una restricción a la matriz de costos A pidiendo que $A_{ij} + A_{jk} \geq A_{ik}$ para cualquier tripleta i, j, k . En algunos casos se pide la desigualdad estricta.

Este pedido, sin embargo, no se requiere para probar el algoritmo (es decir, para que funcione) sino para garantizar que la solución sea "buena" en algún sentido. Se pide que sea k-aproximada (k finito), o por lo menos, en caso de no ser posible acotar la solución en función del óptimo, que en la mayoría de los casos de un resultado no muchas veces mayor que el óptimo.

Sin la propiedad de la desigualdad triangular es posible hallar grafos para los cuales estos algoritmos encuentren una solución de costo tan grande como se quiera. Intuitivamente, bastaría con darle a alguno de los arcos que el algoritmo elige en forma obligada (sin tener en cuenta su costo), un costo tan grande como se quiera.

Es interesante mencionar que en los casos en los que se cumple esta propiedad, valiendo además el mayor estricto, cosa que sucede siempre con las distancias euclídeas, el circuito mínimo resulta hamiltoniano, aunque no se lo exija.

Agregar otras restricciones, como por ejemplo limitar el grado de los nodos del grafo no resulta demasiado útil:

TEOREMA 5:

El problema del circuito hamiltoniano es NP-COMPLETO aun si el grafo cumple la restricción de tener nodos solo de grado cuatro o menor.

TEOREMA 6:

El problema del circuito hamiltoniano para grafos con todos los nodos de grado 3 es NP-COMPLETO.

[Pap 82]

Pese a ser restricciones muy fuertes que eliminarían la mayoría de las aplicaciones, no ayudan tanto como para que valga la pena adoptarlas.

5.4 - ALGORITMOS DE TIEMPO POLINOMICO PARA EL GTSP

La primera decision tomada, es, entonces, restringirse a los grafos cuya matriz de costo respete la desigualdad triangular.

5.4.1 - PRIMERA SOLUCION

La primera de estas soluciones fue una generalizacion del algoritmo greedy para el TSP [AHU 83] que dio como resultado un algoritmo greedy para el GTSP. Este algoritmo tambien esta basado en los de Kruskal y Prim de busqueda de arboles expandidos de costo minimo [AHU 83].

- 1-Crear para cada viajante una componente conexa que solo contenga a la ciudad donde el viva.
- 2-Ordenar los arcos del grato de menor a mayor.
- 3-Mientras exista algun viajante i tal que su componente tenga menos de K_i vertices hacer:
 - 3.1-Elegir la arista $L=(U,V)$ tal que:
 - U y V no esten en la misma componente, ni en distintas (sin perdida de generalidad U esta en una componente y V no)
 - Sea de costo minimo
 - No pertenezca a ninguna componente
 - $\text{grado}(U) < 2$ en el arbol expandido
 - La componente a la cual pertenece U aun admita vertices
 - 3.2-Agregar V a dicha componente
 - 3.3-Agregar L a dicha componente
- 4-Agregar el arco que forme un ciclo en cada una de las componentes. Notar que este queda univocamente determinado.

5.4.2 - SEGUNDA SOLUCION

La idea consiste en hallar una particion prometedora de alguna manera, y luego para cada elemento de la particion aplicar el TREE ALGORITHM.

- 1-Ordenar los arcos del grato de menor a mayor.
- 2-Colocar a cada nodo de V' en una nueva componente.
- 3-Ir agregando los arcos a las componentes hasta que todas tengan la cantidad de nodos deseada, es decir hasta que cada una tenga cardinal K_i , usar las siguientes restricciones:
 - 3.1-Si el arco une dos componentes distintas o
 - 3.2-Si el arco une dos nodos de una misma componente y forma ciclo dentro de esta o
 - 3.3-Si el arco incorpora a la componenete un nodo y entonces el cardinal de esa componente supera el numero deseado (K_i), no se agrega el arco
- 3.4-cualquier otro caso, se agrega el y se lo quita del conjunto de arcos a revisar.

(se obtuvo una particion del conjunto de vertices del grato, y para cada conjunto de la particion se obtuvo un arbol expandido T de costo minimo)

4-para cada conjunto de la particion hallada hacer:

- 4.1-crear un multigrafo G donde por cada arco $e=(x_1,x_2)$ de T crear los arcos $e'=(x_1,x_2)$ $e''=(x_2,x_1)$.
- 4.2-encontrar un camino euleriano de G.
- 4.3-encontrar un tour en dicho camino euleriano.

5.4.3 - TERCERA SOLUCION

Aqui la idea tambien consiste en hallar una particion prometedora, pero luego para cada elemento de la particion aplicamos el algoritmo de Christofides. Los puntos 1, 2 y 3 son los mismos que antes.

1-2-3

(se obtuvo una particion del conjunto de vertices del grafo, y para cada conjunto de la particion se obtuvo un arbol expandido T de costo minimo)

4-para cada conjunto de la particion hallada hacer:

- 4.1-sea T' =los vertices de T que tengan grafo impar.
- 4.2-encontrar el minimo matching completo M en el grafo completo compuesto solamente por los vertices T' .
- 4.3-sea G el multigrafo con vertices= $\{1,2,...,n\}$ y arcos=arcos de T y arcos de M. Notar que los vertices de G son los mismos que los del grafo original.
- 4.4-encontrar un camino euleriano en G.
- 4.5-encontrar en dicho camino euleriano un tour.

5.4.4 - TIEMPO DE EJECUCION

La tres soluciones son de orden $O(K^3)$.

El tree algorithm es de orden $O(K^2)$ y el de Christofides es de $O(K^3)$ [Pap 82].

La cantidad de arcos en un grafo compietto de K nodos es $K * (K - 1) / 2 = a$.

Usando estos resultados se hallan los ordenes de cada paso de cada uno de los algoritmos presentados.

primera solucion:

- 1- $O(N)$, ya que hay N viajeros
- 2- $O(2a \log a) = O(a * \log a)$, es el tiempo necesario para ordenar una lista de a elementos.
- 3- El cuerpo del Mientras es de orden $O(a)$, ya que cada vez se busca secuencialmente en la lista. En el peor caso, se debere recorrer toda la lista de a elementos. Este cuerpo se ejecuta K veces, ya que cada vez se agrega un arco y se termina cuando en ninguna componente se pueden agregar mas arcos, esto es, sumatoria (K_i) veces y sumatoria $(K_i) = K$. Por lo tanto el orden es $O(K * a)$.
- 4- $O(N)$, ya que hay N componentes.

El orden de este primer algoritmo es entonces $O(\max(a * \log a, K * a)) = O(\max(K^2 * \log K^2, K^3)) = O(K^3)$.

segunda y tercera solución:

1 - $O(a * \log a)$

2 - $O(N)$.

3 - $O(K * a)$, por lo mismo que en el punto 3 del algoritmo anterior.

Para cada elemento de la partición se aplica alguno de los dos algoritmos propuestos, por lo tanto si un elemento de la partición tiene K_i vértices y el orden del TREE ALGORITHM o del algoritmo de CHRISTOFIDES es $O(K_i)$, el orden total es la suma de los órdenes $O(K_i)$.

En la segunda solución el orden total es entonces $O(\max(K^3, \sum K_i^2)) = O(K^3)$, ya que $K_i < K$ y $N < K$.

En la tercera solución el orden total resulta $O(\max(K^3, \sum K_i^3)) = O(K^3)$, ya que los K_i son enteros positivos y por lo tanto la sumatoria $K_i^3 \leq K^3$.

6 - LOCAL SEARCH

Local Search es un método que sirve para hallar soluciones aproximadas a problemas de optimización. Consiste en hallar una solución inicial y a partir de ella buscar soluciones cercanas (en algún sentido) que permitan mejorarla. Al conjunto de soluciones cercanas a una solución dada se lo llamara vecindad.

La intención de definir óptimos locales en la vecindad de una solución es buscar entre las cercanas a ella la mejor.

Una posibilidad es definir la vecindad de una solución como todas las soluciones posibles. La apuesta es definirla como el conjunto formado por ella misma. Entre estos dos extremos pueden elegirse muchísimas vecindades, estas deben ser sencillas de explorar pero no demasiado pequeñas, de manera que el óptimo local sea lo bastante bueno.

Puede ser conveniente elegir una vecindad pequeña, aun a riesgo de revisar varias vecindades antes de hallar una solución buena, o trabajar con vecindades grandes, pagando el costo de un mayor tiempo de búsqueda.

Todos estos conceptos de LOCAL SEARCH se apoyan en la práctica. Solo mediante experimentación puede resolverse si una vecindad es mejor que otra o si una solución inicial o conjunto de soluciones iniciales es mejor que otro.

6.1 - CONCEPTOS GENERALES

DEFINICION 7:

Un óptimo global de una instancia (F, c) es un f perteneciente a F tal que $c(f) \leq c(y)$ para todo y perteneciente a F .

Si (F_i, c_i) es una instancia del GTSP, F_i es el conjunto de todos los ciclos que cumplen las restricciones para un grafo dado y la función c_i de costo esta dada por la suma de los costos de los arcos de la solución factible. El GTSP es el conjunto de todas las instancias correspondientes a grafos completos con funciones de costo simétricas que cumplen la desigualdad triangular.

DEFINICION 8:

Dado un problema de optimización Z con instancias (F_z, c_z) una vecindad es una función $N: F_z \rightarrow \text{partes}(F_z)$, definida para cada instancia.

DEFINICION 9.

Dada una instancia (F_Z, c_Z) de un problema de optimización Z y una vecindad N para dicha instancia, una solución factible f de F_Z será un óptimo local con respecto a N si $c(f) \leq c(y)$ para todo y perteneciente a $N(f)$.

Ejemplo:

Para el problema del TSP se define la vecindad 2-change como $N_2(f) = \{ g / g \text{ pertenece a } F \text{ y } g \text{ puede obtenerse de } f \text{ removiendo dos arcos del ciclo y reemplazándolos por dos arcos} \}$

Se podría generalizar esta idea para una vecindad R-change.

TEOREMA 7:

Para el TSP el óptimo local de N_2 no es necesariamente el óptimo global pero el óptimo local de N_K (K número de vértices) es el óptimo global.

[Pap 82]

DEFINICION 10:

Una solución local óptima para el TSP en una vecindad R-change se llama R-opt.

Para hallar una solución R-opt se define la función MEJORAR:

MEJORAR(f) = si existe s perteneciente a $N_R(f)$ tal que $c(s) < c(f)$ entonces s
sino 'no'

Un algoritmo para hallar un ciclo R-opt es entonces:

```
procedure R-opt (var f)
  t = alguna solución inicial;
  while MEJORAR(t) <> 'no' do
    t = MEJORAR(t);
```

Este algoritmo busca el óptimo en una vecindad.
El problema ahora es

- a) encontrar una vecindad adecuada
- b) dar una solución inicial (o mas)

Por otro lado, se tiene la posibilidad de buscar pocos óptimos locales en vecindades muy grandes, o buscar muchos en varias vecindades pequeñas.

Las vecindades R-change fueron estudiadas para el TSP, haciendo énfasis en N_2 y N_3 [Bo 58, Cr 58, Li 65, HK 62], con las siguientes conclusiones empíricas:

a) Las vecindades 2-change y 3-change resultaron muy buenas y relativamente sencillas de utilizar.

b) La vecindad 4-change resultó mejor que la 3-change, pero las mejoras son tan pequeñas que no conviene enfrentar el costo mayor de búsqueda.

c) La vecindad 3-change resulta tan poderosa, que en la práctica es casi lo mismo comenzar con una solución 'buena' (aproximada) que con una hallada al azar. Esto es completamente contraintuitivo, en general se esperaría que empezando con soluciones buenas se conseguirían aproximaciones mejores.

d) La probabilidad (hallada empíricamente) de que una solución 3-opt para un problema de 48 ciudades dado sea óptimo global, es aproximadamente de 0.04. Luego, para 100 corridas, comenzando de soluciones 'al azar' la probabilidad de hallar el óptimo global es de 0.99.

e) La probabilidad (hallada empíricamente) de que una solución 3-opt sea el óptimo global es aproximadamente de $2^{-K/10}$.

Como conclusión de lo anterior, se ve que para el TSP es efectivo utilizar soluciones iniciales elegidas al azar, que 3-change es una vecindad buena y que muchas corridas ayudan mucho para encontrar una solución aproximada. El éxito obtenido con el TSP impulsó a utilizar esta técnica con otros problemas. Sin embargo, en otros problemas en los que se deben definir vecindades más débiles, puede resultar conveniente comenzar con una solución buena, tal vez hallada con un algoritmo de aproximación. Lamentablemente, todos los resultados en este área no están apoyados sobre una base teórica, por lo que toda hipótesis debe justificarse empíricamente.

El parecido entre el problema del viajante y el GTSP y la posibilidad de encontrar con relativa facilidad una buena solución inicial con los algoritmos propuestos anteriormente, llevan a contemplar la posibilidad de utilizar este método para el GTSP.

6.2 - SOLUCIONES INICIALES AL GTSP

a-Para el problema del TSP es efectivo usar soluciones iniciales elegidas al azar [Pap 82]. Otra posibilidad sería entonces generar varias soluciones al azar para el GTSP y tomarlas como soluciones iniciales.

b-Una posibilidad es tomar como soluciones iniciales aquellas que se obtienen por los algoritmos no óptimos ya propuestos. Esto daría tres soluciones iniciales (Greedy, tree algorithm y Christofides generalizado), o menos pues podrían coincidir algunas.

c-Otra posibilidad es generar las x primeras soluciones con el algoritmo exponencial. Pero las obtenidas en a serán en la mayoría de los casos mejores con respecto al costo. Por otro lado esta idea no da soluciones totalmente al azar, ni soluciones 'buenas', ni soluciones demasiado lejanas, por lo tanto esta técnica no se tomara en cuenta.

6.3 - VECINDADES PARA EL GTSP

a-La primera propuesta es hallar la vecindad 2-change o 3-change para cada subconjunto de una partición.
El cardinal de 2-change es:

$$\# N_2(f) = \sum_{i=1}^N [K_i * (K_i - 3)] / 2$$

con $K_i \geq 3$. Es importante que este numero sea polinómico con relación al numero de nodos, ya que da una cota polinómica del tiempo que se requiere para realizar una búsqueda en la vecindad.

Esto se debe a que elegido un arco, no puedo elegir ninguno de los adyacentes a el (ni el mismo arco) pues el resultado de remover esos dos arcos y agregar otros dos (resultarian ser los mismos) daría el mismo grafo. La división por dos es porque a los efectos del problema resulta lo mismo elegir el arco (u,v) que el arco (v,u). Además se pide $K_i \geq 3$ porque sino el numero $[K_i * (K_i - 3)] / 2$ sería negativo y no tendría sentido. Esto es porque remover dos arcos en un ciclo de dos o tres arcos obligaría a elegir dos arcos-adyacentes y eso no puede pasar por lo dicho anteriormente.

b-Llamemos 1-part a la vecindad que se obtiene al aplicar una función N_{1p} tal que: $N_{1p}(f) = \{g / g \text{ pertenece a } F \text{ y } g \text{ puede obtenerse de } f \text{ intercambiando 2 nodos del grafo de manera tal que cada nodo pertenezca a un subconjunto de la partición distinto}\}$

De igual manera se puede definir una vecindad m-part, obtenida a partir de una solución factible aplicando la función N_{mp} tal que: $N_{mp}(f) = \{g / g \text{ pertenece a } F \text{ y } g \text{ puede obtenerse de } f \text{ intercambiando } 2*m \text{ nodos del grafo de manera tal que } m \text{ pertenezcan a un subconjunto de la partición, y los otros } m \text{ nodos a otra distinta}\}$

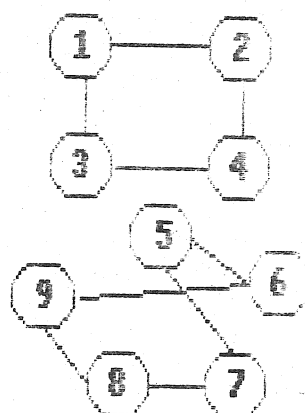
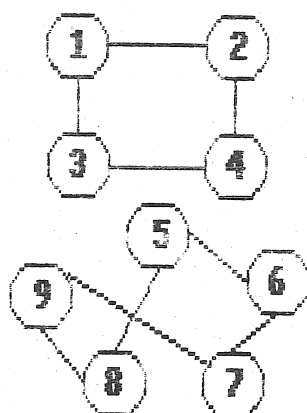
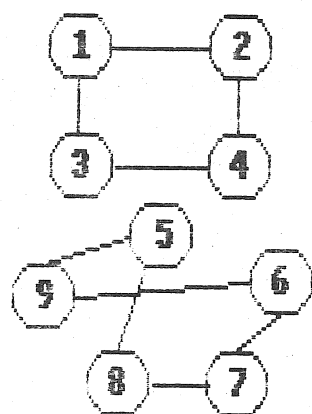
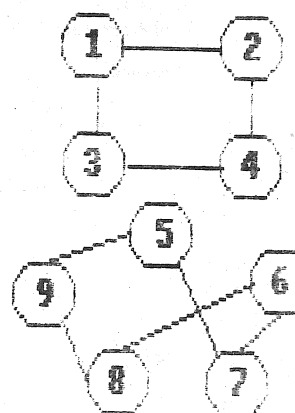
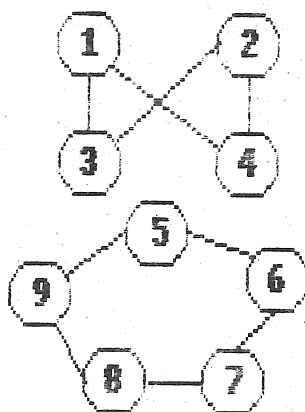
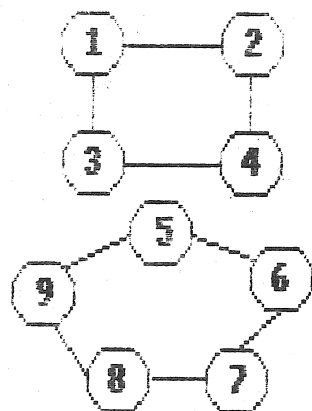
Para ello se pide que los dos subconjuntos tengan cardinal estrictamente mayor que m.

El cardinal de la vecindad 1-part está dado por:

$$\# N_{1p}(f) = \frac{\sum_{i=1}^N [K_i * (K_i - K_i)]}{2}$$

Este numero se debe a que cada ciudad del grafo puede ser intercambiada con cualquiera que no este en su mismo subconjunto de la partición. Se divide por 2 para no contar arcos repetidos.

*solucion
inicial*



Algunos de los vecinos N2 de la solucion inicial

Para formalizar el concepto de vecindad 1-part se introducen los siguientes conceptos:

DEFINICION 11:

Dada una particion uniforme (A,B) y elementos a en A y b en B , la operacion definida por:

$$\begin{aligned} A' &= (A - \{a\}) \cup \{b\} \\ B' &= (B - \{b\}) \cup \{a\} \end{aligned} \text{ se llama swap.}$$

Una particion uniforme (A,B) es tal que $\#A = \#B$

Consideremos el efecto que tiene el swap sobre el costo de la particion A,B .

DEFINICION 12:

Llamaremos costo externo asociado a un elemento a perteneciente a A

$$E(a) = \sum_{i \text{ en } B} d_{ai} \quad (d_{xy} \text{ es el costo del arco } (x,y))$$

Llamaremos costo interno asociado a un elemento a

$$I(a) = \sum_{j \text{ en } A} d_{aj}$$

$D(a)$ sera la suma $E(a) + I(a)$, el costo de un nodo.

LEMA 1:

Dada una solucion factible f , el swap de a y b resulta en una reduccion de costo de:

$$g(a,b) = c(f) - D(a) - D(b) + d_{av_3} + d_{bv_4} + d_{bv_1} + d_{bv_2} \text{ , donde } (a,v_3), (v_4,a), (b,v_1) \text{ y } (v_2,b) \text{ son los nuevos arcos incorporados.}$$

DEFINICION 13:

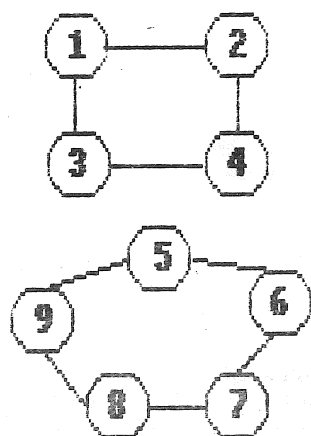
La VECINDAD SWAP (Ns) para un problema de particiones uniformes es $Ns(a,b) = \{ \text{todas las particiones uniformes } A',B' \text{ que pueden ser obtenidas de la particion uniforme } A,B \text{ por un simple swap} \}$

Se puede encontrar un swap favorable en $O(N^4)$, examinando $g(a,b)$ para todo par a en A , b en B . Se puede investigar la vecindad definida por el intercambio de dos elementos de A con dos elementos de B en $O(N^6)$. Esta es la idea de N_{20} , se podria formalizar del siguiente modo:

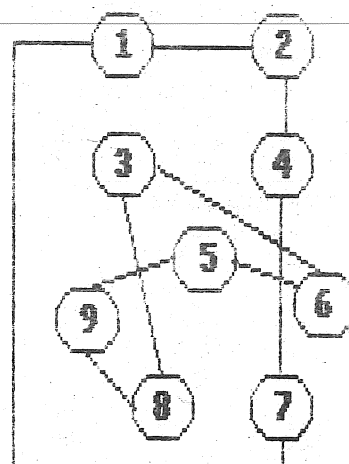
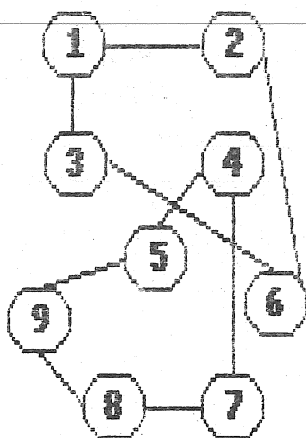
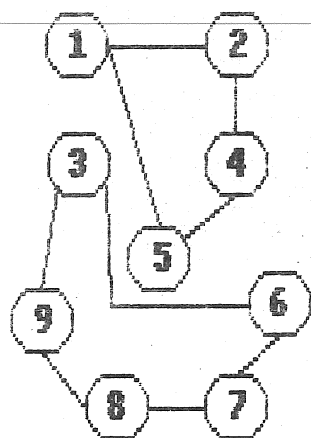
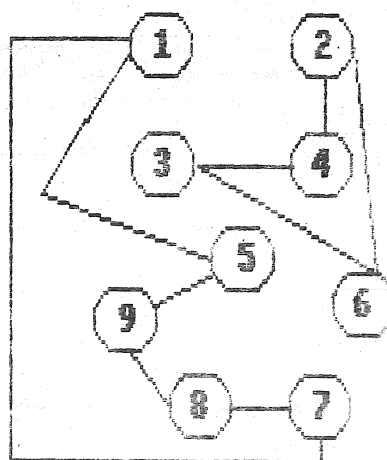
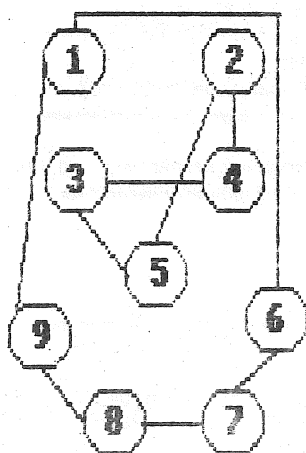
Dada una particion $A1=A11 \cup A12, A2=A21 \cup A22, A3, \dots, AN$, un elemento de la vecindad N_{mp} seria:

$$A1'=A11 \cup A22, A2'=A21 \cup A12, A3, \dots, AN$$

donde $\#A12=\#A22=m$



solucion inicial



Algunos de los vecinos Nip de la solucion inicial

La idea es reemplazar la búsqueda de un swap favorable por una secuencia favorable de swaps, usando el costo de la instancia del problema. Así una secuencia favorable de k swaps es obtenida:

- 1-calcular $D(v)$ para todos los elementos v en $V=A \cup B$.
 - 2-buscar el par a_1', b_1' tal que $g_1 = g(a_1', b_1')$ sea lo mas chico posible.
 - 3-intercambiar a_1' y b_1' y recalcular D .
 - 4-repetir los pasos 2 y 3, obteniendo una secuencia de pares $a_2', b_2'; \dots; a_k', b_k'$.
- Una vez que un par fue elegido no se puede elegir nuevamente.

De esta forma se obtiene una secuencia $g_1, \dots, g_k$ que corresponde a los pares elegidos y tal que el resultado de intercambiar el conjunto $X=\{a_1', \dots, a_k'\}$ con $Y=\{b_1', \dots, b_k'\}$ es

$$G(k) = \sum_{i=1}^k g_i$$

Cuando $k = \#A = \#B$ se estan intercambiando todos los nodos de A con los de B por lo tanto $G(k)=D$. La idea es buscar k tal que $G(k)$ sea minimo

En el GTSP de todos los subconjuntos de la particion solo se trabaja con dos y estos pueden tener cardinales diferentes i_1 y i_2 , por lo tanto se pide $k \leq i_1, k \leq i_2$.

c-A partir de la idea sugerida en b se propone una vecindad con la funcion definida por el siguiente algoritmo:

1-Sea A la particion dada por la solucion factible. Se pide que A tenga al menos dos miembros (por lo menos dos viajeros).

2-Mientras haya mas de un miembro en A hacer

2.1-Elegir dos miembros A_1 y A_2 de A y aplicar N_1 .

2.2- $A=A-(A_1 \cup A_2)$; volver a 2.

A lo sumo quedo un miembro de A igual que antes para cada elemento de la vecindad (ya que N puede ser impar)

e-La ultima posibilidad seria combinar todas estas ideas. Pero aqui tambien se presentan varias maneras posibles de hacerlo.

Una de ellas seria dada una solucion factible hallar la vecindad con la funcion definida en c y para cada elemento de la vecindad aplicar la funcion N_1 .

Otra forma seria hallar la vecindad 1-part mediante la funcion N_{1p} y para cada elemento de la vecindad aplicar N_1 . Tambien se podria hallar primero 2-change y luego 1-part para cada elemento de 2-change. Este metodo podria generalizarse:

$N_p(N_{m-p}(f))$ para P -change de m -part

$N_{m-p}(N_p(f))$ para m -part de P -change

(admitiendo el abuso de notacion, pues N_p y N_{m-p} estan definidas de F en partes(F)).

Es claro que la mejor opcion es una estrategia combinada. Este concepto intuitivo se justifica de la siguiente manera

TEOREMA 8:

$$\#N_2(N_{1D}(f)) = \#N_{1D}(N_2(f)) = \#N_{1D}(f) * \#N_2(f)$$

Sea f una solución inicial, y sean $g1 = \#N_{1D}(f)$, $G1 = N_{1D}(f)$, $g2 = \#N_2(f)$, $G2 = N_2(f)$.

$(N_2 N_{1D})(f) = (N_2(N_{1D}(f)))$; esto es aplicar la función N_{1D} a la solución inicial f , y luego aplicar N_2 a cada elemento del conjunto $G1$.

$$\#(N_2(N_{1D}(f))) = \sum_{g \text{ en } N_{1D}(f)} \#N_2(g)$$

Pero como $\#N_{1D}(f) = g1$ para cualquier f , entonces $\#(N_2(N_{1D}(f))) = g1 * g2$

$(N_{1D} N_2)(f) = (N_{1D}(N_2(f)))$; esto es aplicar la función N_2 a la solución inicial f , y luego aplicar N_{1D} a cada elemento del conjunto $G2$.

$$\#(N_{1D}(N_2(f))) = \sum_{g \text{ en } N_2(f)} \#N_{1D}(g)$$

Nuevamente como $\#N_2(f) = g2$ para cualquier f , entonces $\#(N_{1D}(N_2(f))) = g2 * g1$.

TEOREMA 9:

R-change esta incluida en R-change de m-part (dualmente m-part esta incluida en m-part de R-change) y m-part esta incluida en R-change de m-part (dualmente R-change esta incluida en m-part de R-change).

La demostración es trivial. Dada la función N_R y N_{mp} se quiere ver que $N_R(f)$ esta incluida en $N_R(N_{mp}(f))$, f pertenece a $N_R(f)$ y a $N_{mp}(f)$. Por otro lado, $N_R(N_{mp}(f))$ es un abuso de notación que representa en realidad

$$N_R(N_{mp}(f)) = \bigcup \{ N_R(g) / g \text{ pertenece a } N_{mp}(f) \}$$

En particular, como f pertenece a $N_{mp}(f)$, $N_R(f)$ esta incluido en la gran union luego $N_R(f)$ esta incluido en $N_R(N_{mp}(f))$.

Como h pertenece a $N_R(h)$, para cada g perteneciente a $N_{mp}(f)$, g pertenece a $N_R(g)$. Luego para todo g en $N_{mp}(f)$, g esta en la union definida mas arriba. Luego, $N_{mp}(f)$ esta incluido en $N_R(N_{mp}(f))$.

Dualmente se demuestra la segunda afirmación.

Quedaría ahora por definir que estrategia combinada de las posibles elegimos.

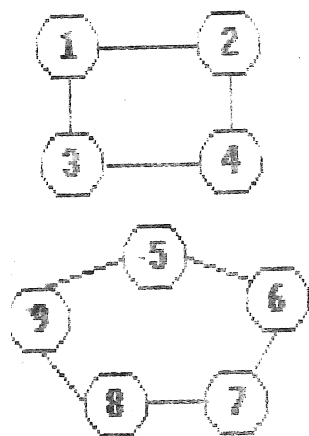
LEMA 2:

En la vecindad 2-change de una solución factible f , dados los arcos (s_1, s_2) y (s_2, s_4) de un ciclo C de la solución f , queda determinado un unico vecino de f sacando esos arcos; y los arcos que los substituyen son (s_2, s_4) y (s_1, s_3) .

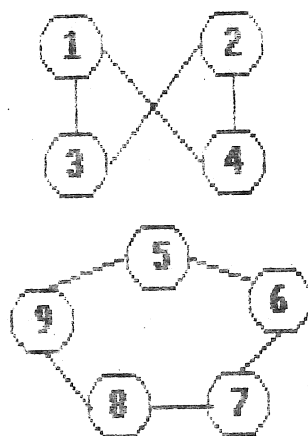
Este lema se observa graficamente. Deben verificarse por separado los casos (a) los arcos son adyacentes y (b) los arcos no son adyacentes.

TEOREMA 10:

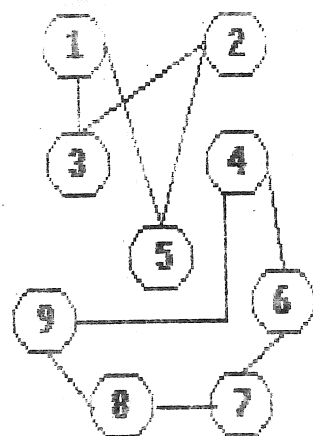
$$N_2(N_{1D}) = N_{1D}(N_2)$$



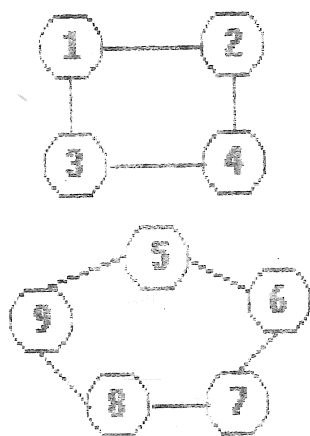
solucion inicial



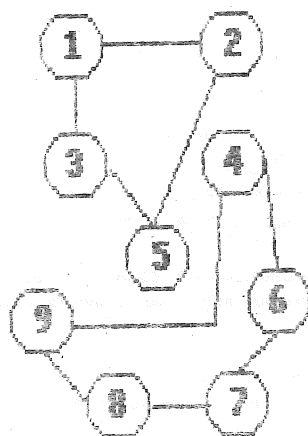
vecino N2



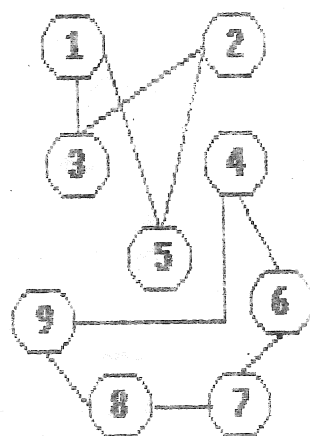
vecino N1p.N2



solucion inicial



vecino N1p



vecino N2.N1p

Demostración.

$$N_{1p}(N_2(f)) = \text{Union } N_{1p}(g) \\ g \text{ en } N_2(f)$$

Dado r en $N_{1p}(N_2(f))$ existe h en $N_2(f)$ tal que r pertenece a $N_{1p}(h)$.

Por el lema anterior, como h pertenece a $N_2(f)$, h está unívocamente determinado por los arcos elegidos.

Sean (s_1, s_2) y (s_3, s_4) los arcos elegidos. Por el lema anterior, los cambios fueron:

$$(s_1, s_2) \rightarrow (s_1, s_3)$$

$$(s_3, s_4) \rightarrow (s_2, s_4)$$

Si $\{(s_1, s_2), (s_3, s_4)\} = \{(s_1, s_3), (s_2, s_4)\}$ (en este caso los arcos serían adyacentes) se tiene $h=i$, luego $N_{1p}(h) = N_{1p}(f)$ por un teorema anterior está incluido en $N_2(N_{1p}(f))$ luego $N_{1p}(N_2(f))$ está incluido en $N_2(N_{1p}(f))$.

Es sencillo ver que si r está en $N_{1p}(h)$, r está unívocamente determinado por los vértices v_1, v_2 elegidos para el cambio.

Análisis por casos:

a) $v_1 = v_2$, luego $r = h$, luego r está en $N_2(f)$ por el teorema anterior está en $N_2(N_{1p}(f))$

b) $v_1 \neq v_2 \Rightarrow v_1$ está en $P1$ y v_2 está en $P2$ con $P1 \neq P2$ elementos de la partición inducida por la solución factible f .

Hay dos posibilidades:

b1) sin pérdida de generalidad, $v_1 = s_1$.

b2) $v_1 \neq s_j$ para $j = 1..4; i = 1, 2$.

Por el lema anterior, toda otra posibilidad es equivalente (s.p.g.) a alguna de estas dos.

En el caso b2) la aplicación de las funciones es totalmente conmutativa, es decir, si se llega de f a h eligiendo (s_1, s_2) y (s_3, s_4) y de h a r eligiendo v_1, v_2 , se puede llegar de f a h' eligiendo primero v_1 y v_2 y luego de h' a r eligiendo (s_1, s_2) y (s_3, s_4) . La verificación de que ambas transformaciones son válidas (llevan de una solución factible a otra solución factible) es muy sencilla.

En el caso b1) tenemos que:

$$v_1 = s_1, \text{ luego como } P1 \neq P2, v_2 \neq s_j; i = 1..4$$

de f se llega a h por una transformación asociada a N_2 y de h a r por una asociada a N_{1p} .

$$f \rightarrow h \rightarrow r$$

Esta transformación, con la misma notación que antes, es:

$$\begin{array}{lcl}
 N_1 & & N_2 \\
 (v_1, s_2) & \rightarrow & (v_1, s_2) \rightarrow (v_1, s_3) \\
 (s_2, s_4) & \rightarrow & (s_2, s_4) \rightarrow (s_2, s_4) \\
 & & (u_1, v_1) \rightarrow (u_1, v_2) \\
 & & (u_2, v_2) \rightarrow (u_2, v_1) \\
 & & (v_2, u_2) \rightarrow (v_1, u_2) \\
 P1 & \rightarrow & P1 - \{v_1\} + \{v_2\} \\
 P2 & \rightarrow & P2 - \{v_2\} + \{v_1\}
 \end{array}$$

Donde u_1 es el vertice adyacente a v_1 tal que $u_1 \triangleleft s_2$ y u_2, u_3 son los vertices adyacentes a v_2 .

Definidas las siguientes transformaciones:

$$\begin{array}{lcl}
 N_1 & & N_2 \\
 (v_1, s_2) & \rightarrow & (v_2, s_2) \rightarrow (v_2, s_3) \\
 (s_2, s_4) & \rightarrow & (s_2, s_4) \rightarrow (s_2, s_4) \\
 P1 & \rightarrow & P1 - \{v_1\} + \{v_2\} \\
 P2 & \rightarrow & P2 - \{v_2\} + \{v_1\} \\
 (u_1, v_1) & \rightarrow & (u_1, v_2) \\
 (u_2, v_2) & \rightarrow & (u_2, v_1) \\
 (v_2, u_2) & \rightarrow & (v_1, u_2)
 \end{array}$$

Es sencillo verificar que son validas. Por lo visto mas arriba y el lema anterior, estas transformaciones definen univocamente un vecino que cumple las transformaciones en las vecindades dadas. El vecino hallado es precisamente r .

Luego, se ha probado que $N_{1D}(N_2(t))$ esta incluido en $N_2(N_{1D}(t))$ para toda t solucion factible. Como $\#N_{1D}(N_2(t)) = \#N_2(N_{1D}(t))$ para todo t se deduce que $N_{1D}(N_2) = N_2(N_{1D})$.

Posiblemente este resultado pueda generalizarse para m y R arbitrarios aunque no sera demostrado.

CONJETURA:

$$N_{mR}(N_R) = N_R(N_{mR}) \text{ para } R, m \text{ naturales.}$$

Esto demuestra que al siegir una estrategia combinada no interesa el orden en que se aplican las funciones para encontrar vecindades.

6.4 - METODO DE BUSQUEDA

Pueden aplicarse varios metodos de busqueda. Uno de ellos seria adoptar la primer solucion que se encuentre en la vecindad que sea mejor que la que se tiene. El otro extremo seria buscar en toda la vecindad hasta encontrar la mejor solucion y adoptarla.

Se debe tener en cuenta, tambien, la idea de *REDUCCION*. Un ejemplo de la aplicacion de esta idea en el TSP es: al obtener varios optimos locales, se observa que tienen arcos en comun. Una estrategia es, entonces, fijar esos arcos y seguir buscando pero ahora con menos arcos: se redujo el espacio de busqueda y se trabaja en subconjuntos de las vecindades.

Otra idea, llamada *DENIAL*, consiste en hacer exactamente lo opuesto. Estos arcos que aparecen repetidos se prohíben, ya que se piensa que buscar siempre "por ese lado" puede estar engañando a la estrategia. Se fuerza a respetar la heurística pero para un conjunto de soluciones alejado al que se estuvo revisando hasta el momento. Esto sirve sobre todo si la solución obtenida se aleja mucho del óptimo.

En el momento de implementar el algoritmo, tambien debe tenerse en cuenta el orden en el que se va a realizar la busqueda. Este orden puede tener o no importancia, dependiendo del problema.

7 - CONCLUSIONES

En este trabajo se presenta una generalización de un problema muy conocido. A través del GTSP se intenta ejemplificar una metodología de trabajo para enfrentar y resolver problemas muy complejos, ya sea usando soluciones a problemas similares o técnicas modernas como local search.

Al haber contado con una formalización del concepto de vecindad, se pudieron obtener valiosos resultados teóricos.

Las conclusiones halladas acerca de la complejidad del GTSP sirvieron de impulso para buscar distintas soluciones aproximadas de tiempo polinómico.

Es importante tener en cuenta que el GTSP es una posible generalización al problema del viajante, otras generalizaciones podrían aportar distintas ideas a los temas planteados.

En el trabajo se presentaron varios algoritmos. Para poder compararlos y obtener conclusiones más precisas sería interesante implementarlos y hacer estadísticas sobre ellos.

3 - BIBLIOGRAFIA

- [AHU 83] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, "Data structures and algorithms", editorial Addison-Wesley, 1983
- [Bo 58] F. Bock, "An algorithm for solving TS and related network optimization problems", 14 National Meeting de la ORS de America, St Louis, Missouri, 1958
- [Cr 58] G. A. Croes, "A method for solving TSP", OR, 6, numero 6, paginas 791-812, 1958
- [Eve 79] Shimon Even, "Graph algorithms", Technion Institute, editorial Computer Science Press, 1979
- [Fl 59] Merrill M. Flood, "Operations research for management", editores Joseph Mc Closkey y John M. Coppingier, capitulo 19, "The traveling salesman problem", 1959
- [Fri 83] A. M. Frieze, "An extension of Christofides heuristic to the K-person travelling salesman problem", University of London, Discrete Applied Mathematics, numero 6, paginas 79-83, 1983
- [Ha 78] John E. Hayes, "Computer architecture and organization", University of Southern California, editorial McGraw-Hill, 1978
- [Har 69] Frank Harary, "Graph theory", University of Michigan, editorial Adison-Wesley Publishing Company, 1969
- [Hk 62] M. Held, "A dinamic programming approach to sequencing problems", J. SIAM, 10, numero 1, paginas 196-210, 1962
- [LePa 78] Harry R. Lewis, Christos H. Papadimitrou, "La eficiencia de los algoritmos", revista Investigacion y Ciencia, numero 18, paginas 78 a 91, 1978
- [LePa 81] Harry R. Lewis, Christos H. Papadimitrou, "Elements of the theory of computation", editorial Prentice-Hall, 1981
- [Li 65] S. Lin, "Computer solution of the TSP", BSTJ, 44, numero 10, paginas 2245-2269, 1965
- [Nil 71] Nils J. Nilsson, "Problem-solving methods in artificial intelligence", editorial McGraw-Hill, 1971
- [NiWi 75] Albert Nijenhuis, Herbert S. Wiff, "Combinatorial algorithms", editorial Academic Press, 1975
- [Pap 82] Christos H. Papadimitrou, Kenneth Steiglitz, "Combinatorial optimization, algorithms and complexity", editorial Prentice-Hall, 1982
- [Pap 84] Christos H. Papadimitrou, "On the complexity of unique solutions", Stanford University and National Technical University of Athens, Journal of the Association for Computing Machinery vol. 31, numero 2, paginas 392 a 400, 1984
- [RND 77] Edward M. Reingold, Jurg Nievergelt, Narsingh Deo, "Combinatorial algorithms, theory and practice", editorial Prentice-Hall, 1977