**TEMA 3**

Tarea 1.1 : MULTISTART N- ARRANQUES COMENZANDO EN CADA N NODO ( PARTIENDO DEL ENTORNO\_1\_TSP\_EJCOORD.MOS DE FUNCIONES DE EXAMEN) (FUNCION ENTORNO\_CERCANO Y DECLARATIONS):

declarations

ini: integer

zmejor:=9999999

forall(k in nodos)do

ini:=k

zheur:=entorno\_cercano(ini)

writeln("Solucion entorno mas proximo en la iter ",k," :", zheur)

if(zheur<zmejor)then

zmejor:=zheur

end-if

end-do

writeln

writeln("Solucion entorno mas proximo con multistart: ", zmejor)

Tarea 1.2 : greedy aleatorizado

Grasp:

cx, cy, visitado, s,marc,ind,mejors:array(nodos)of integer

nvisit:integer

K=5

N=10

M=10

end-declaration

!---------------------------Greedy aleatorizado empezando en un mismo nodo---------------------!

!Greedy aleatorizado empezando en un mismo nodo.

zmax:=999999

forall(ab in 1..N)do

forall (aa in nodos)visitado(aa):=0

ini:=1

visitado(ini):=1

actual:=ini

nvisit:=1

zheur:=0

while(nvisit < n)do

forall(j in nodos|visitado(j)=0)marc(j):=0

forall (t in 1..K) do

!encontrar el entorno mas cercano

dmin:=999999

forall(i in nodos|visitado(i)=0 and marc(i)=0)do

if (dist(actual, i) < dmin) then

dmin:=dist(actual,i)

imin:=i

end-if

end-do

ind(t):=imin

marc(imin):=1

end-do

r:=ceil(random\*K)

imin:=ind(r)

visitado(imin):=1

s(actual):=imin

zheur:=zheur+dist(actual, imin)

nvisit:=nvisit+1

actual:=imin

end-do

!cerrar el ciclo

zheur:=zheur+dist(actual, ini)

s(actual):=ini

writeln("\niteracion ",ab," valor de zheur: ",zheur)

forall(i in nodos) write(" ",s(i))

if (zheur<zmax)then

zmax:=zheur

mejoriter:=ab

mejors:=s

end-if

end-do

writeln("\nSolucion entorno mas proximo greedy aleatorizado: ", zmax," en iteracion ",mejoriter)

!Hacemos la grafica

s:=mejors

forall(i in nodos) write("\n",i," -> ",s(i))

i1:=IVEaddplot("puntos", IVE\_BLACK)

i2:=IVEaddplot("lineas", IVE\_RED)

forall(i in nodos)IVEdrawpoint(i1,cx(i),cy(i))

forall(i in nodos)IVEdrawlabel(i1,cx(i), cy(i), ""+nodos(i))

forall(i in nodos)IVEdrawline(i2,cx(i),cy(i), cx(s(i)), cy(s(i)))

!--------------------------- greedy aleatorizado multistart ------------------------------!

writeln("\n\n\nEmpezamos el algoritmo de greedy aleatorizado pero cambiando el nodo base cada 10 iteraciones")

!greedy aletorizado empezando en distintos nodos

zmax:=999999

forall (nn in nodos) do

forall(ab in 1..M)do

forall (aa in nodos)visitado(aa):=0

ini:=nn

visitado(ini):=1

actual:=ini

nvisit:=1

zheur:=0

while(nvisit < n)do

forall(j in nodos|visitado(j)=0)marc(j):=0

forall (t in 1..K) do

!encontrar el entorno mas cercano

dmin:=999999

forall(i in nodos|visitado(i)=0 and marc(i)=0)do

if (dist(actual, i) < dmin) then

dmin:=dist(actual,i)

imin:=i

end-if

end-do

ind(t):=imin

marc(imin):=1

end-do

r:=ceil(random\*K)

imin:=ind(r)

visitado(imin):=1

s(actual):=imin

zheur:=zheur+dist(actual, imin)

nvisit:=nvisit+1

actual:=imin

end-do

!cerrar el ciclo

zheur:=zheur+dist(actual, ini)

s(actual):=ini

writeln("\niteracion ",ab," valor de zheur: ",zheur," nodo base: ",ini)

!forall(i in nodos) write(" ",s(i))

if (zheur<zmax)then

zmax:=zheur

mejoriter:=ab

mejors:=s

mejorbase:=ini

end-if

end-do

end-do

writeln("\nSolucion entorno mas proximo greedy aleatorizado multistart: ", zmax," en iteracion ",mejoriter," empezando en nodo: ",mejorbase)

i1:=IVEaddplot("puntos", IVE\_BLACK)

i3:=IVEaddplot("lineas", IVE\_BLUE)

forall(i in nodos)IVEdrawpoint(i1,cx(i),cy(i))

forall(i in nodos)IVEdrawlabel(i1,cx(i), cy(i), ""+i)

forall(i in nodos)IVEdrawline(i3,cx(i),cy(i), cx(s(i)), cy(s(i)))

Tarea 1.2 : búsqueda local 2-opt a heurística Clarke-wright (programa en funciones de exámenes y el procedure de entorno1\_heuristicaIntercambios2opt\_TSP)

(en la función da error mejora y fin)-> declarar mejora como integer dentro de la función y fin a fin2

zheur:=sum(i in nodos)dist(i,s(i))

write("\nbase: ",base,", zheur = ",zheur)

!!!!!!!!!!!!!!!!!!Solucion heuristica 2-opt=======================

heur\_2opt

writeln("Solucion despues de 2-opt: ", zheur)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

dist\_mejorada:=sum(i in nodos)dist(i,s(i))

write(" --> ",dist\_mejorada) !vemos la distancia mejorada

if(dist\_mejorada < zmejor)then

!if(zheur < zmejor)then

zmejor:=dist\_mejorada

smejor:=s

basemejor:=base

end-if

**TEMA 4**

**PRÁCTICA 1 – MODELOS EXACTOS :**

* SABER CONCEPTOS DE HUECO, COTA RELATIVA, TIEMPO, NODOS VISTOS ,VARIABLES, RESTRICCIONES, TIEMPO DE EJECUCION, Nº DE RUTAS:

La cota es mucho mayor en el modelo con red auxiliar que en Tucker-Miller-Zremlin.

En cuanto al hueco relativo, es mucho mayor en Tucker-Miller-Zremlin, luego la solución se acerca más a la óptima en red auxiliar ya que el hueco mide la optimalidad relativa de la solución.

También el número de variables y restricciones en el modelo de red auxiliar es mayor para lograr bajar ese hueco

* REVISAR COMO SON LOS DATASETS Y SU LECTURA. MODELO DE 2 INDICES Y FLUJO DE REDES-> FORMULACIÓN. TIEMPOS DE EJECUCION CAMBIAR
* SABER PONER LAS RESTRICCIONES SI TE DAN UN NÚMERO DE VEHÍCULOS PREFIJADO K -> RESTRICCIONES APUNTES INDICES MAL:

! Restricciones apartado 2

sum(i in puntos) x(i,1) = 5

sum(j in puntos) x(1,j) = 5

* MODELO DE 3 INDICES BASADO EN TMZ:

coches = 1..m

x:array(puntos,puntos, coches) of mpvar

y:array(puntos, coches) of mpvar

u: array(puntos, coches) of mpvar

writeln("\nFormulacion de Tucker-Miller-Zemlin")

objetivo:=sum(i,j in puntos, k in coches)dist(i,j)\*x(i,j,k)

forall(i in puntos|i<>base)sum(k in coches)y(i,k)=1

sum(k in coches)y(1,k)=5

forall(i in puntos, k in coches) sum(j in puntos)x(i,j,k) = sum(j in puntos)x(j,i,k)

forall(i in puntos, k in coches) sum(j in puntos)x(i,j,k) = y(i,k)

forall(i,j in puntos, k in coches | i <> base and j <> base and i <> j and dem(i)+dem(j)<=cap) restmz(i,j,k):=u(i,k)-u(j,k)+cap\*x(i,j,k)<=cap-dem(j)

forall(i in puntos, k in coches|i<>base)dem(i)<=u(i,k)

forall(i in puntos, k in coches|i<>base)u(i,k)<=cap

forall(i,j in puntos, k in coches)x(i,j,k) is\_binary

forall(i in puntos, k in coches)y(i,k) is\_binary

forall(i in puntos, k in coches|i <> base)u(i,k)>=0

nr:=0

forall(i in puntos, k in coches | x(base,i,k).sol >=0.9) do

nr+=1

inicio\_ruta(nr):=i

end-do

dist\_total:=0.

forall(k in 1..nr)do

carga\_ruta(k):=0

dist\_ruta(k):=0.

indr(1,k):=base

sig:=inicio\_ruta(k)

nnr(k):=1

dist\_ruta(k)+=dist(base,sig)

while(sig <> base)do

carga\_ruta(k)+=dem(sig)

nnr(k)+=1

forall(j in puntos, h in coches | x(sig,j, h).sol>=0.9)do

jj:=j

end-do

indr(nnr(k),k):=sig

dist\_ruta(k)+=dist(sig,jj)

sig:=jj

end-do

dist\_total+=dist\_ruta(k)

end-do

dist\_total:=sum(r in 1..nr)dist\_ruta(r)

writeln("\nDistancia total = ",dist\_total)

writeln("Numero de rutas: ",nr,". Rutas:")

writeln("\nr\tdist\tcarga\tpuntos\n")

forall(r in 1..nr)do

write("\n",r,"\t",strfmt(dist\_ruta(r),6,2),"\t",carga\_ruta(r),"\t")

forall(i in 1..nnr(r))write(indr(i,r),"-")

write("1")

end-do

if(formato\_datos = 2)then

dibujar\_rutas(nr, nnr, indr)

end-if

* MODELO DE 3 INDICES FLOTA HETEROGENEA BASADO EN TMZ:

declarations

puntos = 1..n

coches = 1..m

cap:array(coches) of integer

objetivo:=sum(i,j in puntos, k in coches)dist(i,j)\*x(i,j,k)

forall(i in puntos|i<>base)sum(k in coches)y(i,k)=1

sum(k in coches)y(1,k)=m

forall(i in puntos, k in coches)sum(j in puntos)x(i,j,k) = y(i,k)

forall(i in puntos, k in coches)sum(j in puntos)x(j,i,k) = y(i,k)

forall(i,j in puntos, k in coches | i <> base and j <> base and i <> j and dem(i)+dem(j)<=cap(k)) restmz(i,j,k):= u(i,k)-u(j,k)+cap(k)\*x(i,j,k)<=cap(k)-dem(j)

forall(i in puntos, k in coches|i<>base)dem(i)<=u(i,k)

forall(i in puntos, k in coches|i<>base)u(i,k)<=cap(k)

forall(k in vehiculos) sum(i in puntos) dem(i) \*y(i,k) <=cap(k)

forall(i,j in puntos, k in coches)x(i,j,k) is\_binary

forall(i in puntos, k in coches)y(i,k) is\_binary

forall(i in puntos, k in coches|i <> base)u(i,k)>=0

nnr:array(rutas)of integer!numero nodos en la ruta

nr:integer

indr:array(range,rutas)of integer!indices de los nodos en las rutas

! ============================ MEJORA 2-OPT EN CADA RUTA (INTRARUTA) ============================

writeln("\n================ MEJORA 2-OPT EN CADA RUTA (INTRA-RUTA) ==================")

dist\_mejorada:=0.

forall(k in 1..nr)do

nn:=nnr(k)

forall(j in 1..nnr(k))p(j):=indr(j,k)

dist\_ruta(k):=heur\_2opt(nn,p)

forall(i in 1..nn)indr(i,k):=p(i)

dist\_mejorada+=dist\_ruta(k)

end-do

writeln("\nRutas despues de 2-opt")

writeln("\nr\tnn\tdist\tcarga\tpuntos\n")

forall(k in 1..nr)do

write("\n",k,"\t",nnr(k),"\t",strfmt(dist\_ruta(k),6,2),"\t",carga\_ruta(k),"\t")

forall(i in 1..nnr(k))write(indr(i,k),"-")

write("1")

end-do

writeln("\n\nDistancia final mejorada = ",dist\_mejorada)

**PRÁCTICA 2 – HEURÍSTICAS DE CONSTRUCCIÓN:**

* SABER HACER DIBUJOS DEL CASO 1 Y 3 Y DAR UNA VUELTA A LOS OTROS DOS POR SI ACASO:
* REVISAR LOS GRAFICOS (DIBUJAR RUTA EN 2-OPT SE HACE EN EL NORMAL Y EN 2-OPT CUIDADO A VER QUE NOS PIDE. Variable grafico) Y COMO SE SACAN LOS VALORES QUE PIDEN. QUE, EN ALGUNOS, SOBRE TODO 2-OPT, HAY QUE SACARLOS Q NO ESTAN PARA LA TABLA.
* HEUR\_CLARKE\_WRIGHT\_SIN\_ORDENACIÓN (NORMAL Y 2-OPT)
* HEUR\_CLARKE\_WRIGHT\_CON\_ORDENACIÓN (2-OPT) (EN 2-OPT TB SE ENCUENTRA EL NORMAL)
* HEUR\_CLARKE\_WRIGHT\_CON\_ORDENACIÓN\_ALEATORIZADO (NORMAL)

PARA METER 2-OPT EN EL NORMAL:

writeln("\n\nIndices de cada ruta:") ! Escribimos los índices de cada ruta

forall(k in 1..nr)do

write("\nruta : ",k,", nnr = ",nnr(k)," => ")

forall(t in 1..nnr(k))write(indr(t,k),",")

write(" dist = ",dist\_ruta(k))

writeln

end-do

**AQUÍ EL 2-OPT (COGER EL PRIMER FOR DE CLARKE-WRIGHT CON ORDENACION) (METER LA FUNCIÓN AL FINAL Y AL PRINCIPIO LA DECLARACION)**

if(busqueda\_local = 1)then

dist\_total:=0.

forall(k in 1..nr)do

nn:=nnr(k)

forall(j in 1..nnr(k))p(j):=indr(j,k)

dist\_mej(k):=heur\_2opt(nn,p)

dist\_total+=dist\_mej(k)

forall(j in 1..nnr(k))indr(j,k):=p(j)

end-do

writeln("\n\nMejora 2-opt:") ! Escribimos los �ndices de cada ruta

writeln("\n\nIndices de cada ruta:") ! Escribimos los �ndices de cada ruta

forall(k in 1..nr)do

write("\nruta : ",k,", nnr = ",nnr(k)," => ")

forall(t in 1..nnr(k))write(indr(t,k),",")

write(" dist = ",dist\_ruta(k))

writeln

end-do

writeln("\ndistancia total: " ,dist\_total)

end-if

if(dist\_total < mejor\_dist)then

mejor\_dist:=dist\_total

writeln("iteracion ",itergrasp,", dist = ",mejor\_dist)

nrf:=nr

forall(k in 1..nr)do

nnrf(k):=nnr(k)

forall(j in 1..nnr(k))indrf(j,k):=indr(j,k)

end-do

end-if

end-do !fin aleatorizacion

* INSERCIÓN SECUENCIAL: MOLE-JAMENSON (NORMAL Y 2-OPT)
* INSERCIÓN SECUENCIAL: MOLE-JAMENSON MULTISTART **(ENTREGA PRÁCTICA 2)**

nnr,nnrf:array(rutas)of integer !AÑADIDO nnrf para quedarme con la mejor de multistart

indr,indrf:array(range,rutas)of integer ! AÑADIDO indrf para indices nodos mejor solución

!========= Algoritmo de Mole-Jamenson ===================================

! Inicializaciones:

dist\_mejor:=999999.9

forall(iter in 1..N)do

!Inicializaciones

forall(j in puntos)visitado(j):=0

dist\_total:=0.

nvis:=1

visitado(1):=1

r:=0

!ACTUALIZAR CARGA

while(nvis < n)do

….

end-do ! final de while(nvis < n)

nr:=r

writeln("\n\nIndices de cada ruta:") ! Escribimos los ?ndices de cada ruta sin optimizar 2-opt

forall(k in 1..nr)do

write("\n",k,"\t",nnr(k),"\t",strfmt(dist\_ruta(k),6,2),"\t",carga(k),"\t")

forall(i in 1..nnr(k))write(indr(i,k),"-")

write("1")

end-do

writeln("\n")

if(busqueda\_local = 1)then

dist\_total:=0.

forall(k in 1..nr)do

nn:=nnr(k)

forall(j in 1..nnr(k))p(j):=indr(j,k)

dist\_mej(k):=heur\_2opt(nn,p)

dist\_total+=dist\_mej(k)

forall(j in 1..nnr(k))indr(j,k):=p(j)

end-do

writeln("\n\nMejora 2-opt:") ! Escribimos los ?ndices de cada ruta

writeln("\n\nIndices de cada ruta:") ! Escribimos los ?ndices de cada ruta

forall(k in 1..nr)do

write("\nruta : ",k,", nnr = ",nnr(k)," => ")

forall(t in 1..nnr(k))write(indr(t,k),",")

write(" dist = ",dist\_ruta(k))

writeln

end-do

writeln("\ndistancia total: " ,dist\_total)

end-if

writeln("\n")

!si es la mejor de todas las que he construido

if(dist\_total<dist\_mejor)then

write("\niter ",iter," ,dist = ",dist\_mejor," -> ",dist\_total)

dist\_mejor:=dist\_total

nrf:=nr

forall(k in 1..nr)do

nnrf(k):=nnr(k)

forall(j in 1..nnr(k))indrf(j,k):=indr(j,k)

end-do

end-if

end-do

if(formato\_datos = 2)then

dibujar\_rutas(nrf, nnrf, indrf)

end-if

writeln("\n")

writeln("\nSolucion final multistart insercción secuencias:")

writeln("\n\nDistancia mejor = ",dist\_mejor)

writeln("\nTiempo = ",gettime-t1)

* INSERCIÓN PARALELO: CAMPBELL-SAVELSBERGH (NORMAL Y 2-OPT)