UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD CIENCIAS MATEMATICAS

E.A.P. DE..INVESTIGACIÓN OPERATIVA

Conceptos, algoritmo y aplicación al problema de las N – reinas

Anexos

MONOGRAFÍA

Para optar el Título de Licenciada de Investigación operativa

AUTOR

Alicia Cirila Riojas Cañari

LIMA – PERÚ
2005

ANEXO 3.1

Aplicaciones de Tabu search[11]	
Scheduling Flow-Time Cell Manufacturing Heterogeneous Processor Scheduling Workforce Planning Classroom Scheduling Machine Scheduling Flow Shop Scheduling Job Shop Scheduling Sequencing and Batching	**Telecommunications** Call Routing Bandwidth Packing Hub Facility Location Path Assignment Network Design for Services Customer Discount Planning Failure Immune Architecture Synchronous Optical Networks
Design Computer-Aided Design Fault Tolerant Networks Transport Network Design Architectural Space Planning Diagram Coherency Fixed Charge Network Design Irregular Cutting Problems	**Production, Inventory and Investment** Flexible Manufacturing Just-in-Time Production Capacitated MRP Part Selection Multi-item Inventory Planning Volume Discount Acquisition Fixed Mix Investment
Location and Allocation Multicommodity Location/Allocation Quadratic Assignment Quadratic Semi-Assignment Multilevel Generalized Assignment Lay-Out Planning Off-Shore Oil Exploration	**Routing** Vehicle Routing Capacitated Routing Time Window Routing Multi-Mode Routing Mixed Fleet Routing Traveling Salesman Traveling Purchaser
Logic and Artificial Intelligence Maximum Satisfiability Probabilistic Logic Clustering Pattern Recognition/Classification Data Integrity Neural Network |Training and Design	**Graph Optimization** Graph Partitioning Graph Coloring Clique Partitioning Maximum Clique Problems Maximum Planner Graphs P-Median Problems
Technology Seismic Inversion Electrical Power Distribution Engineering Structural Design Minimum Volume Ellipsoids Space Station Construction Circuit Cell Placement	**General Combinational Optimization** Zero-One Programming Fixed Charge Optimization Nonconvex Nonlinear Programming All-or-None Networks Bilevel Programming General Mixed Integer Optimization

[11] Tabu search Fred Glover & Manuel Laguna pp 2
(http://leeds-faculty.colorado.edu/laguna/articles/ts2.pdf)

ANEXO 4.1

Todas las soluciones posibles del problema de 4 reinas.

Alternativa #	Reinas				Cantidad de colisiones	
	1	2	3	4		
1	1	2	3	4	6	El óptimo local se encuentra en (1,3,4,2) y en (1,4,2,3), la FO = 1
2	1	2	4	3	2	
3	1	3	2	4	2	
4	1	3	4	2	1	
5	1	4	2	3	1	
6	1	4	3	2	4	
7	2	1	3	4	2	El óptimo local se encuentra en (2,4,1,3) y la FO = 0
8	2	1	4	3	4	
9	2	3	1	4	1	
10	2	3	4	1	4	
11	2	4	1	3	**0**	
12	2	4	3	1	1	
13	3	1	2	4	1	El óptimo local se encuentra en (3,1,4,2) y la FO = 0
14	3	1	4	2	**0**	
15	3	2	1	4	6	
16	3	2	4	1	1	
17	3	4	1	2	4	
18	3	4	2	1	2	
19	4	1	2	3	4	El óptimo local se encuentra en (4,1,3,2) y en (4,2,1,3), la FO = 1
20	4	1	3	2	1	
21	4	2	1	3	1	
22	4	2	3	1	2	
23	4	3	1	2	2	
24	4	3	2	1	6	

Hay dos asignaciones que producen cero colisiones

ANEXO 4.2
Programa en C++

```
// Programa   :        N REINAS que minimiza colisiones
// Descripción:        Usa el algoritmo de la búsqueda Tabú
para resolver el problema  de las n- reinas
// Autora     :        Alicia Riojas Cañari  Agosto 2005
/*****************************************************************/
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<MATH.H>
#include<string.h>
#include<PROCESS.H>
// variables globales
 int c=5;                // cantidad de candidatos
 int tabu_tenure=3+1;          // tiempo durante el cual un
movimiento es tabú (se agrega 1 pues se actualiza después
de asignarle la tenure)
 int n       ;             // cantidad de reinas    -  máximo 10
reinas !
 int RA[11]            ;      // solucion actual
 int RT[11]            ;      // solucion transitoria de prueba
 int M[11][11];        // matriz de ubicacion
 int Dnsup[11] ;       // diagonales negativas superior
 int Dninf[11] ;       // diagonales negativas inferior
 int Dpsup[11];        // diagonales positivas superior
 int Dpinf[11];        // diagonales positivas inferior
 int colisiones[11];   // como máximo pueden haber 10 reinas
en una diagonal
 int fo;               // función objetivo
 int vecinos[46][4]  ; // registra los intercambios posibles ,
soluciones vecinas(n tomadas de 2 en 2)
 int cant_de_vecinos ;
 int candidatos[10][7]          ;          // c mejores
candidatos c<10
 int tabu[11][11]     ;      // lista tabú
 int frecuentes[11][11]          ;      // frecuencia de
intercambios
 int freq              ;          // freq del candidato evaluado
 int corte1;                              // a partir de esta
iteración se considera el criterio de aspiraci¢n
 int corte2;                              // a partir de esta
iteración se chequea la memoria de largo plazo
 int i,j,fila,k        ;                  // índices de fila y
columna
 int iter, MAXITER;            // cantidad de iteraciones y
n£mero m ximo de iteraciones
 int transito,min,sw;
 int imp ;                                // para imprimir cada
iteracion o no no=0
 FILE *arch;                   // para grabar en disco
 char archivo[30];

//funciones

 void  inicio();
 void  coloca_reinas();
 int evalua_solucion();
 void imprime_solucion();
```

```
// PROGRAMA PRINCIPAL

  void main()
{
// rutina de inicio
 clrscr();
 inicio();
 coloca_reinas();
 fo=evalua_solucion();
 imprime_solucion();
 fprintf(arch,"%-24d%-3d%-3d%-3d%-3d%-3d%-3d%-3d%-
3d%-3d%-15d%-
7d\n",iter,RA[1],RA[2],RA[3],RA[4],RA[5],RA[6],RA[7],RA[8],
RA[9],RA[10],fo);
// fin de rutina de inicio

// iteraciones hasta que se llegue a MAXITER
 iter=1;
do
{
// se evaluan todos los vecinos
  clrscr();
  cout<<"       Evaluacion de todos los vecinos\n\n";
  cout<<"    # vecino    reina    reina       fo  \n";

  for (fila=1;fila<=cant_de_vecinos;fila++)
  {   for(j=1;j<=n;j++)
      {
            RT[j]=RA[j];  // RA = soluc actual se la preserva y
se trabaja en RT=solucion transitoria
      }
    transito=RT[vecinos[fila][1]];
    RT[vecinos[fila][1]]=RT[vecinos[fila][2]];
    RT[vecinos[fila][2]]=transito;
    coloca_reinas();
    fo= evalua_solucion();
    vecinos[fila][3]=fo;  // valor si se hace ese intercambio
    if(imp!=0) //se debe imprimir
    {
    cout<<"\n       "<<fila<<"        "<<vecinos[fila][1]<<"
"<<vecinos[fila][2]<<"         "<<vecinos[fila][3];
    }
  }

// se escogen los c mejores vecinos

   for (k=1;k<=c;k++)
  {    min=vecinos[1][3]; // se fija el primero como minimo
             for (fila=cant_de_vecinos;fila>=1;fila--) // se busca
de abajo hacia arriba de modo que si hay empates se
seleccione el primero
             {
              if(min>=vecinos[fila][3])
                {
                 min=vecinos[fila][3];
                 // min es la menor fo de todos los vecinos
```

```
                candidatos[k][1]=fila;                    // # de
orden s¢lo para comprobar calculos a mano
                candidatos[k][2]=vecinos[fila][1];  // reina que
intercambia con
                candidatos[k][3]=vecinos[fila][2];  // reina que
intercambia
                candidatos[k][4]=vecinos[fila][3];       // fo si se
realiza dicho intercambio
                candidatos[k][5]=0;                       // indica si
es tabu (0) o no (1)
                candidatos[k][6]=frecuentes[vecinos[fila][2]]
[vecinos[fila][1]];
                                                                  //
frecuencia de veces que se ha realizado dicho intercambio
               }
              }
             vecinos[candidatos[k][1]] [3]=999; // para que no
vuelva a ser min
  } // ya tenemos los c mejores candidatos

// chequear si el candidato es tabu o no
    for (k=1;k<=c;k++)
   { candidatos[k][5]=0;
     if(tabu[candidatos[k][2]] [candidatos[k][3]]>0 )
     {
               candidatos[k][5]=1;  // est  en la lista tabu
               if (candidatos[k][4]==0&&iter>=corte1) // la fo es
cero, no hay colisiones y debe considerar el criterio de a
                 {
                   candidatos[k][5]=0 ;        // criterio de
aspiración, se considera aun siendo tabu
                 }
     }
   }

// imprimir los c mejores candidatos
      cout<<"\n\n\n "<<c<<" mejores candidatos";
      if(imp!=0)  // se imprime cada paso solo si se estipuló así
en la entrada de datos
     {
              cout<<"\n\n       #      reina     reina    fo
tabu=1\n\n";
              for (k=1;k<=c;k++)  // c candidatos
              {
              for (j=1;j<=6;j++) // imprimir 6 columnas:
#,reina,reina,fo,status tabu, frecuencia
                {
                cout<<"       "<<candidatos[k][j];
                }
              cout<<"\n";
              }
              getch();
    }
// seleccionar la siguiente solucion
     sw=0;
     i=1;  // se escoger  el primero que no es tabu
   do
   {
    if(candidatos[i][5]==0)   // el primero no tabu que
encuentre
      {
      sw=1;  // el candidato i es no tabu
      // chequear si est  penalizado por ser muy frecuente
      if(iter>=corte2)
             {
              freq=candidatos[i][6];
              for(k=1;k<=c;k++)
                {

if((freq>candidatos[k][6])&&(candidatos[k][5]==0)) /* el actual
candidato tiene una frecuencia de  ocurrencias mayor que
otro candidato no tabu*/
                        {
                        sw=0;                    //  ya no es el elegido
(basta que encuentre uno)
                        }
                 }  // ya chequeé contra todos los demás
candidatos
             }
      if(sw==1)   // encontró un candidato no tabú no frecuente
              {       transito=RA[candidatos[i][2]];
                      RA[candidatos[i][2]]=RA[candidatos[i][3]];
                      RA[candidatos[i][3]]=transito;
                      fo=candidatos[i][4];
                      tabu[candidatos[i][2]]
[candidatos[i][3]]=tabu_tenure;   // 3 es la tabu tenure
                  frecuentes[candidatos[i][3]] [candidatos[i][2]]++;
/*se produjo un intercambio se registra en la estructura de
memoria a largo plazo*/
                }
     }
   i=i+1; //probara con el siguiente candidato si sw=0
   if(i>c&&sw==0)      // si todos los c candidatos son tabú o
son muy frecuentes
     {
     sw=2; // se han rechazado los c candidatos
     }
   }    // fin del while regresa a buscar otro solo si no encontró
una buena solucion
   while(sw==0);  //sw=1 cuando encuentra una soluc no tabu
sw=2 cuando los c candidatos son tabu

   if(sw==2)
     {   cout <<"\n\nmatriz de frecuencias en la iteraci¢n
"<<iter;
              for(k=2;k<=n;k++)
                {   cout<<"\n";
                        for(j=1;j<=k-1;j++)
                        {
                         cout<<frecuentes[k][j]<<"   ";
                        }
                }
              cout<<"\n\nlista de los ultimos candidatos\n";
              for(k=1;k<=c;k++)
                {   cout<<"\n\n";
                        for(j=1;j<=6;j++)
                        {
                         cout<<"   "<<candidatos[k][j]<<"   ";
                        }
                }
              getch();
```

```
            exit(0);

    }  //  si sale es porque ya se encontr¢ una nueva soluci¢n

// actualizar lista tabú    //s¢lo se usa la diagonal superior
    for(k=1;k<=n-1;k++)
    {
                for(j=k+1;j<=n;j++)
                {
                  if(tabu[k][j]>0)
                  {
                        tabu[k][j]=tabu[k][j]-1;
                  }
                }
     }
//imprimir nueva solucion (si se seleccionó imprimir en
pantalla)
   if(imp!=0)
    {
            cout<<"\nsolucion "<<iter<<") = ";
            for(k=1;k<=n;k++)
            {
            cout<<RA[k]<<" ,   ";  //imprime nueva solucion
            }
            cout<<"funcion objetivo= "<<fo;
            getch();
            imprime_solucion();
            getch();
     }
// grabar en archivo de texto
   fprintf(arch,"%-10d%-3d%-11d%-3d%-3d%-3d%-3d%-
3d%-3d%-3d%-3d%-3d%-3d%-15d%-7d\n",iter,candidatos[i-1][2],
candidatos[i-
1][3],RA[1],RA[2],RA[3],RA[4],RA[5],RA[6],RA[7],RA[8],RA[9]
,RA[10],fo);

   iter=iter+1;   // realizar otra iteración

}  // cierra el while
while (iter <= MAXITER);
// el proceso se detiene cuando se ha superado MAXITER

// RUTINA FINAL
   fclose(arch);
   clrscr();
   cout<<"\n\n\n\nLos resultados de cada iteraci¢n se
encuentran en el archivo " <<archivo;
   cout<<"\n\n\n envie sus comentarios a
aliriojasc@hotmail.com";
   cout<<"\n\n\n presione cualquier tecla para salir del
programa";
   getch();
} // cierra PROGRAMA PRINCIPAL

//_____

void  inicio()
{
// ingresa datos
 gotoxy(25,2);cout<<"BUSQUEDA TABU PARA RESOLVER
EL PROBLEMA DE N REINAS "<<endl;
 gotoxy(25,4);cout<<"Programa elaborado por Alicia Riojas
Cañari 2005\n"<<endl;

 gotoxy(8,6);cout<<"Nombre del archivo donde se guardaran
las iteraciones";
 gotoxy(8,7);cout<<"nombre.txt :   ";
 cin>>archivo;
 arch=fopen(archivo,"w+");
 fprintf(arch,"Iter     intercambio      Solucion i         func
objetivo  \n");

 gotoxy(8,10);cout<<"Ingrese en que iteracion comienza a
usar el criterio de aspiraci¢n ";
 cin>>corte1;
 gotoxy(8,12);cout<<"Ingrese en que iteracion comienza a
usar la memoria de largo plazo ";
 cin>>corte2;

 gotoxy(8,14);cout<<"Ingrese el maximo n£mero de
iteraciones permitido ";
 cin>>MAXITER;
 gotoxy(8,16);cout<<"¨Desea imprimir paso a paso? (si= 1 /
no=0)";
 cin>>imp;

 gotoxy(8,18);cout<<"Ingrese la cantidad de REINAS:  ";
cin>>n;

 gotoxy(8,19);cout<<"Ingrese una SOLUCION INICIAL:  ";
 for(j=1;j<=n;j++)
  {
  gotoxy(15,19+2*j);cout<<"["<<j<<"]:";  cin>> RA[j];
  }
  cout<<"\n";
  cout<<"solucion inicial = ";
  for(i=1;i<=n;i++)
  {   RT[i]=RA[i] ;    // se pasa a una solucion transitoria para
evaluar
      cout <<RA[i]<<" , ";
  }
  // calcula y llena la matriz de vecinos este proceso se
realiza una s¢la vez
  cant_de_vecinos=(n*(n-1))/2   ;
  cout<<"\n\n cantidad de vecinos   "<<cant_de_vecinos;
  getch();
  clrscr();
  // construye todos los posibles intercambios entre dos
reinas
  //la primera siempre es menor que la segunda, i.e. no hay
4,2 sino 2,4
  k=1;
  for(i=1;i<=n-1;i++)
  {
     for(j=i+1;j<=n;j++)
     {
      vecinos[k][1]=i;
      vecinos[k][2]=j;
      k=k+1;
      }
  }
// inicializa lista tabú           (memoria de corto plazo)
```

```
// inicializa lista de frecuentes   (memoria de largo plazo)

  for(i=1;i<=10;i++)
  {
     for(j=1;j<=10;j++)
     {
              tabu[i][j]=0;       //s¢lo se usa la diagonal superior
pero se limpia todo
              frecuentes[i][j]=0;  //s¢lo se usa la diagonal inferior
pero se limpia todo
     }
  }
} // fin del ingreso de datos , ojo falta consistenciar
entrada

//_____________________________________________

 void  coloca_reinas()
{
// inicializa la matriz de posiciones de las reinas
  for(i=1;i<=10;i++)
  {
     for(j=1;j<=10;j++)
     {M[i][j]=0;}
  }

// coloca las reinas
  for(i=1;i<=n;i++)
   {
     M[i][RT[i]]=1;
    }
}// fin de colocar las reinas

//_____________________________________________

// evalúa la solución

 int evalua_solucion()
{
 int sum, tot_colisiones ;
 fo=0;
 tot_colisiones=0;
   for(k=1;k<=n;k++) // se blanquean los contadores de
diagonales porsia
   {
   Dnsup[k]=0;
   Dninf[k]=0;
   Dpsup[k]=0;
   Dpinf[k]=0;
   colisiones[k]=0;
   }
// se suman los valores de las diagonales para detectar
colisiones
   for(k=1;k<=n;k++)   //diagonal negativa superior (son n
diagonales)
   {    sum=0;
              i=k;
              for(j=1;j<=k;j++)
              {
              sum = sum+M[i][j];
              i=i-1;
              }
     Dnsup[k]= sum;
   }
   for(k=2;k<=n;k++)   //diagonal negativa inferior (son n-1
diagonales)
   {    sum=0;
              i=n;
              for(j=k;j<=n;j++)
              {
              sum = sum+M[i][j];
              i=i-1;
              }
     Dninf[k]= sum;
   }

   for(k=1;k<=n;k++)   //diagonal positiva inferior (son n
diagonales)
   {    sum=0;
              i=n;
              for(j=k;j>=1;j--)
              {
              sum = sum+M[i][j];
              i=i-1;
              }
     Dpinf[k]= sum;
   }
   for(k=2;k<=n;k++)   //diagonal positiva superior (son n-1
diagonales)
   {    sum=0;
              i=1;
              for(j=k;j<=n;j++)
              {
              sum = sum+M[i][j];
              i=i+1;
              }
     Dpsup[k]= sum;
   }

  for(i=2;i<=n;i++)
  {
              for(k=1;k<=n;k++)
              {
              if(Dnsup[k]==i) {colisiones [i]=colisiones[i]+1;}
              if(Dninf[k]==i) {colisiones [i]=colisiones[i]+1;}
              if(Dpsup[k]==i) {colisiones [i]=colisiones[i]+1;}
              if(Dpinf[k]==i) {colisiones [i]=colisiones[i]+1;}
              }
  }
// se calculan las colisiones
  tot_colisiones=0;   //para calcular cuantas reinas colisionan
  for(k=2;k<=n;k++)
  {
  i=(k*(k-1))/2;
  tot_colisiones = tot_colisiones+(colisiones[k]*i);
  }
  fo=tot_colisiones;
  return (fo);

}  // fin de evalua solucion
```

```
//_____________________________________________

// imprime la ubicaci¢n de las reinas
 void imprime_solucion()
{

  clrscr();
  gotoxy(15,2);cout<<"MATRIZ DE UBICACION";
  gotoxy(5,4);cout<<"La fila indica a la reina y la columna su posici¢n";
  for(j=1;j<=n;j++)
    {
      gotoxy(6+6*j,6);cout<<j;                //  imprime cabecera
      gotoxy(6+6*j,7);cout<<"-";
    }

  for(k=1;k<=n;k++)
    {
            for(j=1;j<=n;j++)
              {
               gotoxy(6,7+2*k);cout<<k<<") " ;
               gotoxy(6+6*j,7+2*k);cout<<M[k][j]; // imprime matriz
              }
     }

// imprime diagonales

  cout<<"\n\n\n\n Diagonales negativas superiores   ";
  for(k=1;k<=n;k++)
    {        cout <<Dnsup[k]<<"  " ;    }
  cout<<"\n\n Diagonales negativas inferiores   ";
  for(k=2;k<=n;k++)
    {  cout <<Dninf[k]<<"  " ;    }
  cout<<"\n\n Diagonales positivas inferiores   ";
  for(k=1;k<=n;k++)
    {  cout <<Dpinf[k]<<"  " ;    }
  cout<<"\n\n Diagonales positivas superiores   ";
  for(k=2;k<=n;k++)
    {  cout <<Dpsup[k]<<"  " ;    }

// imprime la funcion objetivo

  cout<<"\n\n\n la funcion objetivo = "<<fo;
  cout<<"\n\n   presione una tecla para continuar";
  getch();
}
// fin de imprime solucion
```

-- . -- . --