Resumen

En esta sección se presenta la información complementaria del proyecto. Esta ha sido dividida en dos anexos:

- *Anexo A.* Representa un manual de usuario del programa implementado. En eso se explica como utilizar la *interface* y se detallan las características de los ficheros en entrada y salida.

- *Anexo B.* Contiene el código fuente del programa escrito en C# y la explicación de su estructura.

Los ficheros de los resultados de la experiencia computacional se adjuntan en el cd. Estos estan divididos por variante del programa, tipo de ejemplar y número de ejemplar. Estan disponibles dos ficheros de resultados: el primero, llamado *r.txt* es un fichero resumido donde sólo aparece el retraso total de la mejor solución encontrada y el tiempo de cálculo. El segundo llamado *result.txt* es un fichero detallado en que aparecen los datos de la programación de la solución de partida y los de la mejor solución encontrada. Cada ejemplar ha sido ejecutado cinco veces y para el análisis estadístico se ha considerado la media de los cinco resultados.

Sumario

Anexo A

En este anexo se describen las características esenciales para poder utilizar el programa *HFFS* propuesto.

El programa se inicia ejecutando su icono. Aparecerá entonces la ventana principal (fig. A.1).

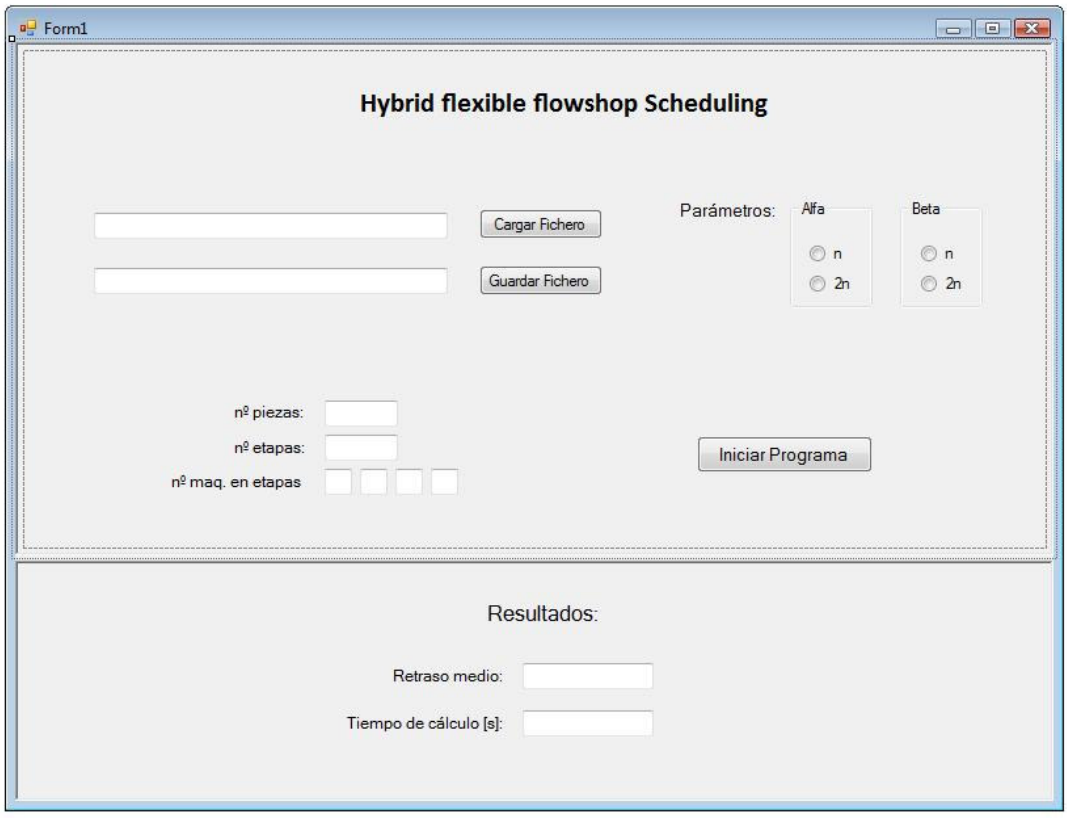


Figura A.1 – Ventana principal del programa *HFFS*.

Para obtener la programación de operaciones de un problema es necesario únicamente cargar el fichero con los datos del problema, indicar al programa un fichero donde guardar los resultados y decidir el valor de los parámetros α y β.

Cargar y guardar ficheros

Para cargar y guardar un fichero hay que inserir sus rutas en las respectivas casillas de texto y, entonces, apretar los botones *Cargar fichero* y *Guardar fichero*. Al hacerlo el programa indicará las características del ejemplar: número de piezas, número de etapas y máquinas por etapa (fig. A.2).

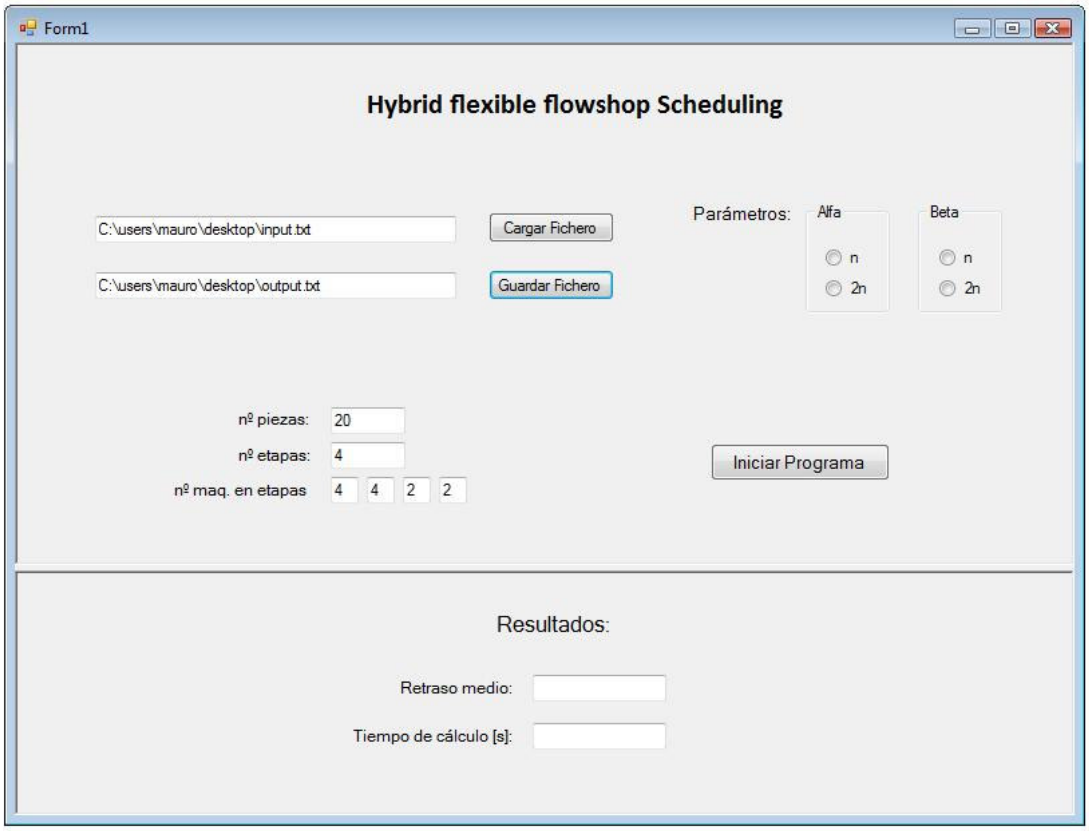


Figura A.2 – Cargar y guardar ficheos en *HFFS*.

Ajuste de los parámetros

El programa presenta la opción de ajustar los parámetros que establecen el número de iteraciones: α y β entre los valores ensayados. Para hacerlo es suficiente marcar el valor elegido en el campo *Parámetros* (fig. A.3).

Ejecutar el programa y sacar las soluciones

Una vez seguidos los puntos anteriores es suficiente apretar el boton *Iniciar programa*. La solución en forma resumida saldrá en las casillas de textos correspondientes (fig. A.3) mientras un fichero detallado será disponible en la dirección especificada por el usuario.

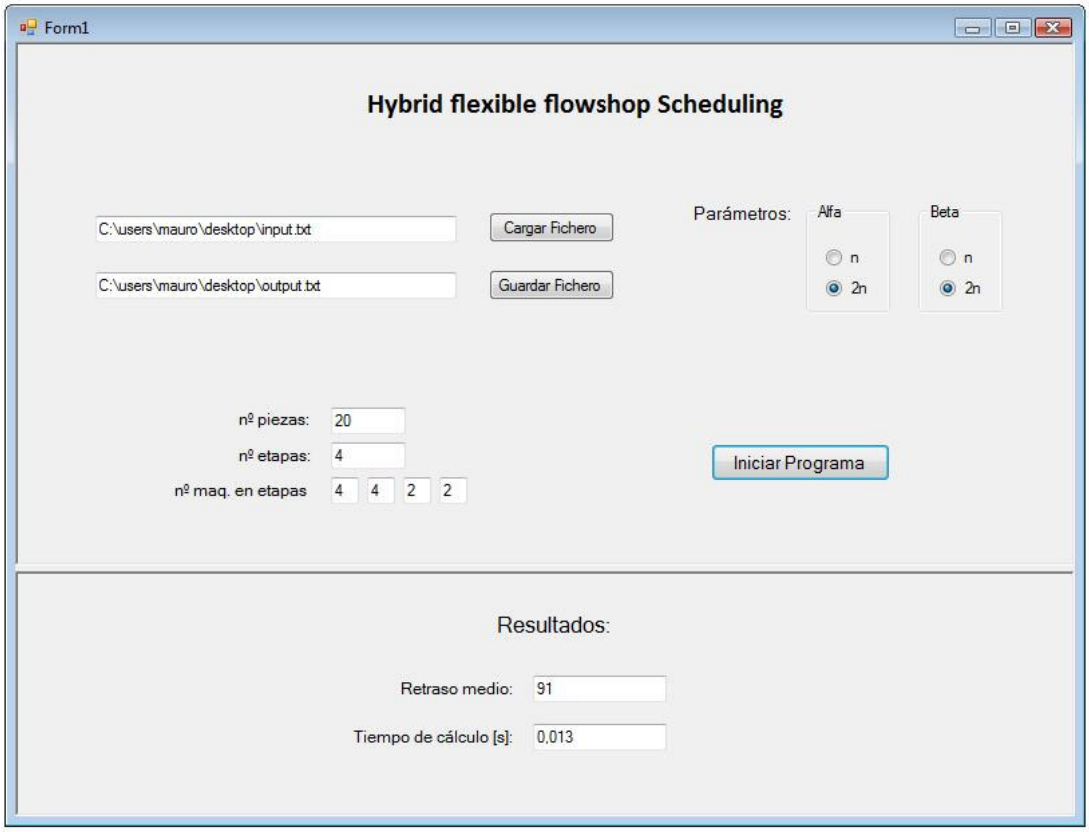


Figura A.2 – Pantalla de resultados de *HFFS*.

Fichero de Input

El fichero de entrada está en formato de *texto delimitado por tabulación* como el siguiente (fig. A.4).

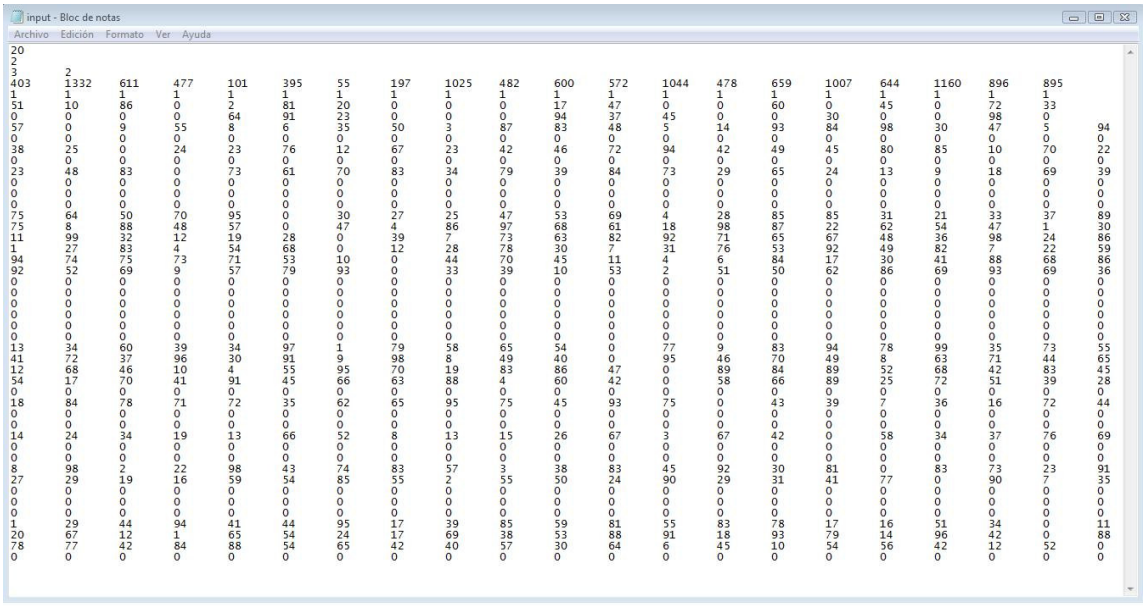


Figura A.4 – Ejemplo de fichero de INPUT en formato texto tabulado.

Este se puede obtener sencillamente de un fichero excel, guardandolo en extensión *.txt*. El fichero tiene una estructura que se detalla a continuación. En la explicación se hace referencia al ejemplo de figura A.5.

La primera fila, marcada en amarillo, está compuesta por una celda y contiene el número de piezas del ejemplar.

La segunda fila, marcada en azul claro, contiene el número de etapas (m), mientras la tercera, en maron claro, tiene tantas celdas cuantas etapas y contiene el número de máquinas que hay en cada etapa (mpe).

La cuarta fila, en verde claro, tiene tantas celdas cuantas piezas (n), y contiene la fecha de vencimiento de cada pieza.

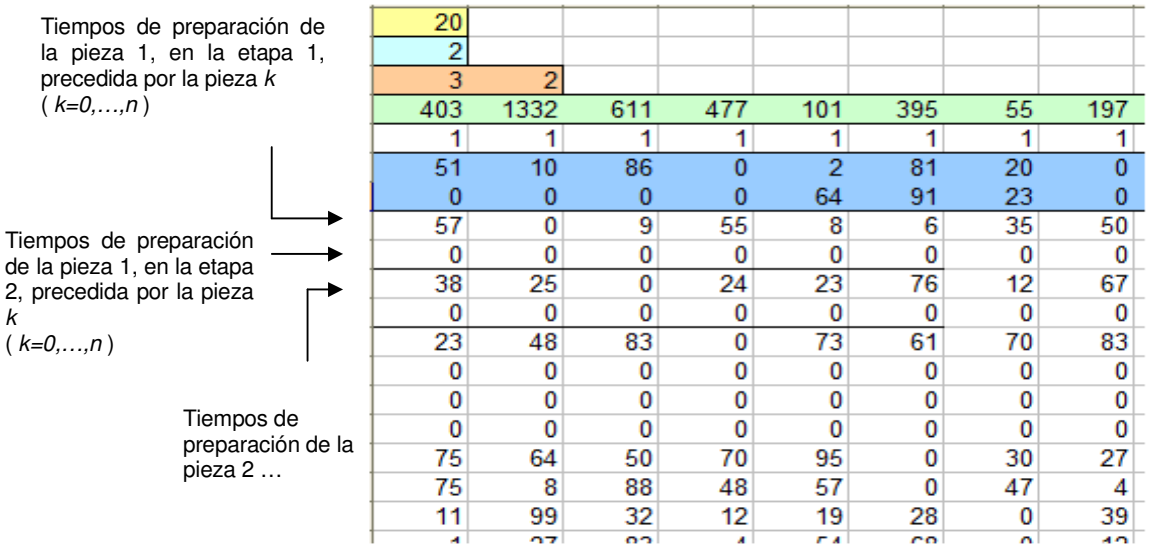
La quinta fila, en blanco, contiene los pesos de las piezas. Aunqué no han sido utilizados en nuestra exeriencia computacional, el programa, y por lo tanto los ficheros, estan diseñados considerando la posibilidad de procesar piezas con pesos distintos.

Después de los pesos hay m filas, marcadas en azul marino. En el caso del ejemplo son dos. Estas contienen los tiempos de proceso de las n piezas respectivamente en la primera y en la segunda etapa. Para que una pieza no sea procesada en una etapa (*stage skipping*), es suficiente poner cero en el tiempo de proceso correspondiente.

Finalmente estan las $n \cdot m$ filas de los tiempos de preparación. En la figura A.6 se encuentra una explicación de cómo están organizadas.

20																				
2																				
3	2																			
403	1332	611	477	101	395	55	197	1025	482	600	572	1044	478	659	1007	644	1160	896	895	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
51	10	86	0	2	81	20	0	0	0	17	47	0	0	60	0	45	0	72	33	
0	0	0	0	64	91	23	0	0	0	94	37	45	0	0	30	0	0	98	0	
57	0	9	55	8	6	35	50	3	87	83	48	5	14	93	84	98	30	47	5	94
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	25	0	24	23	76	12	67	23	42	46	72	94	42	49	45	80	85	10	70	22
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	48	83	0	73	61	70	83	34	79	39	84	73	29	65	24	13	9	18	69	39
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
75	64	50	70	95	0	30	27	25	47	53	69	4	28	85	85	31	21	33	37	89
75	8	88	48	57	0	47	4	86	97	68	61	18	98	87	22	62	54	47	1	30
11	99	32	12	19	28	0	39	7	73	63	82	92	71	65	67	48	36	98	24	86
1	27	83	4	54	68	0	12	28	78	30	7	31	76	53	92	49	82	7	22	59
94	74	75	73	71	53	10	0	44	70	45	11	4	6	84	17	30	41	88	68	86
92	52	69	9	57	79	93	0	33	39	10	53	2	51	50	62	86	69	93	69	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	34	60	39	34	97	1	79	58	65	54	0	77	9	83	94	78	99	35	73	55
41	72	37	96	30	91	9	98	8	49	40	0	95	46	70	49	8	63	71	44	65
12	68	46	10	4	55	95	70	19	83	86	47	0	89	84	89	52	68	42	83	45
54	17	70	41	91	45	66	63	88	4	60	42	0	58	66	89	25	72	51	39	28
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	84	78	71	72	35	62	65	95	75	45	93	75	0	43	39	7	36	16	72	44
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	24	34	19	13	66	52	8	13	15	26	67	3	67	42	0	58	34	37	76	69
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	98	2	22	98	43	74	83	57	3	38	83	45	92	30	81	0	83	73	23	91
27	29	19	16	59	54	85	55	2	55	50	24	90	29	31	41	77	0	90	7	35
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	29	44	94	41	44	95	17	39	85	59	81	55	83	78	17	16	51	34	0	11
20	67	12	1	65	54	24	17	69	38	53	88	91	18	93	79	14	96	42	0	88
78	77	42	84	88	54	65	42	40	57	30	64	6	45	10	54	56	42	12	52	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura A.5 – Ejemplo de fichero de INPUT en formato excel.



20							
2							
3	2						
403	1332	611	477	101	395	55	197
1	1	1	1	1	1	1	1
51	10	86	0	2	81	20	0
0	0	0	0	64	91	23	0
57	0	9	55	8	6	35	50
0	0	0	0	0	0	0	0
38	25	0	24	23	76	12	67
0	0	0	0	0	0	0	0
23	48	83	0	73	61	70	83
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
75	64	50	70	95	0	30	27
75	8	88	48	57	0	47	4
11	99	32	12	19	28	0	39

Figura A.6 – Fichero de INPUT. Tiempos de preparación.

Fichero de Output

También el fichero de salida está en formato de *texto delimitado por tabulación*, y se puede abrir como una aplicación *excel.* Esto está compuesto por una primera parte donde se enseñan la fecha de vencimiento y el retraso de cada pieza para la mejor programación encontrada (fig. A.7), y la segunda donde se explicita dicha programación dividida por etapas (fig. A.8).

DATOS DEL PROBLEMA
número de piezas:20
número de stages:2

PIEZA	DUE DATE	COMPLETION TIME	TARDINESS
1	403	108	-295
2	1332	208	-1124
3	611	256	-355
4	477	0	-477
5	101	205	104
6	395	284	-111
7	55	137	82
8	197	0	-197
9	1025	0	-1025
10	482	0	-482
11	600	140	-460
12	572	489	-83
13	1044	63	-981
14	478	0	-478
15	659	302	-357
16	1007	38	-969
17	644	161	-483
18	1160	0	-1160
19	896	361	-535
20	895	294	-601

Figura A.x – Fichero de OUTPUT, primera parte.

*** PROGRAMACIÓN***

PIEZA	MÁQUINA	T_INICIO	T_FIN
1	2	0	108
2	1	186	208
3	3	161	256
5	1	0	77
6	1	77	186
7	3	30	61
11	3	0	30
12	2	294	386
15	1	208	302
17	3	61	161
19	2	108	209
20	2	209	294

5	1	137	205
6	2	186	284
7	1	63	137
11	2	38	140
12	2	386	489
13	1	0	63
16	2	0	38
19	1	209	361

Coste del retraso=186
TIEMPO CÁLCULO: 0 ore, 0 min, 0 sec, 46 ms.

Figura A.x – Fichero de OUTPUT, segunda parte.

Anexo B

En este anexo se enseña el código del programa. Para facilitar la lectura se ha decidido seguir en la exposición la misma estructura del programa. Esto está formado por un programa principal, llamado *Main*, al final del cual se llaman a tres subprogramas: *Cuing*, que genera la solución inicial, *SR_file* que guarda un fichero con los datos de la solución inicial y *ILS*, que aplica la parte iterativa de la metaeurística *Iterated Local Search*.

Declaración de las variables globales

Antes de empezar el programa principal (Main) es necesario definir las variables globales, o sea las que vienen utilizadas en todas las funciones del programa.

```
static string direccion_load = "";
static string direccion_save = "";

static int alfa = 0;
static int beta = 0;
static int n = 0;
static int m = 0;
static float tau = 0;
static int[] w;
static int[] dd;
static int[,] p_ij;
static int[,] S;
static int[, ,] S_ijk;
static int[] mps;
static int type = 0;
static int coste_best = 0;

static float[] mu;
static float[] heta;
static float R = 0;
static float[] p_medio;
static float[] p_t;
static float[] S_medio;
static float Cmax = 0;
```

El programa principal Main

El programa principal *Main* realiza 6 operaciones:

- Lee y guarda los datos del fichero en entrada
- Calcula las características del ejemplar
- Llama el subprograma *Cuing*
- Llama el subprograma *SR_file*
- Llama el subprograma *ILS*
- Mide el tiempo de cálculo

Lectura del fichero de INPUT

```
StreamReader objReader = new StreamReader(direccion_load);
string sLine = "";
char[] charSeparators = new char[] { '\t' };
string[] result;
int linea = 0;


while (sLine != null)
{
  sLine = objReader.ReadLine();
  if (sLine != null)
     {
      result = sLine.Split(charSeparators,
               StringSplitOptions.RemoveEmptyEntries);

      if (linea == 0)
         {
          n = Int32.Parse(result[0]);
          dd = new int[n];
          w = new int[n];
         }

      if (linea == 1)
         {
           m = Int32.Parse(result[0]);
           mps = new int[m];
           p_ij = new int[n, m];
           S = new int[n + 1, n * m];
           S_ijk = new int[n, m, n + 1];
          }
```

```
      if (linea == 2)
         {
          for (int j = 0; j < m; j++)
               {
                mps[j] = Int32.Parse(result[j]);
                }
         }

      if (linea == 3)
         {
          for (int i = 0; i < n; i++)
               {
                dd[i] = Int32.Parse(result[i]);
               }
         }

      if (linea == 4)
         {
          for (int i = 0; i < n; i++)
               {
                w[i] = Int32.Parse(result[i]);
               }
         }

      if (linea > 4 && linea < m + 5)
         {
          for (int i = 0; i < n; i++)
               {
                p_ij[i, linea - 5] = Int32.Parse(result[i]);
               }
         }

      if (linea >= m + 5)
         {
          for (int i = 0; i < n + 1; i++)
            {
             S[i, linea - 5 - m] = Int32.Parse(result[i]);
            }
       }
    }
   linea = linea + 1;
  }
objReader.Close();


for (int j = 0; j < m * n; j++)
    {
     for (int k = 0; k < n + 1; k++)
          {
           S_ijk[j / m, j % m, k] = S[k, j];
           }
    }
```

Cálculo de las características del ejemplar

En este apartado se calculan los parámetros necesarios para el programa, como el tiempo medio de proceso y de preparación, las fechas de entrega mínima, máxima y media, los parámetros μ, η, τ y R y la estimación del *Makespan*.

```
DateTime t_start = DateTime.Now;     // Instante en que se empieza el
                                     algoritmo

int d_max = 0;           // Declaración variables locales
int d_min = 32000;
float d_medio = 0;
float mps_medio = 0;

p_medio = new float[m]; // Declaración arrays locales
p_t = new float[n];
S_medio = new float[m];
mu = new float[m];
heta = new float[m];

for (int j = 0; j < m; j++)
    {
     for (int i = 0; i < n; i++)
         {
          p_medio[j] = p_medio[j] + (float)p_ij[i, j];
          p_t[i] = p_t[i] + (float)p_ij[i, j];

          for (int k = 0; k < n + 1; k++)
              {
               if (i != (k - 1))
                  {
                   S_medio[j] = S_medio[j] + (float)S_ijk[i, j, k];
                   }
               }
         }
       p_medio[j] = p_medio[j] / n;
       S_medio[j] = S_medio[j] / (n * n);

       mu[j] = (float)n / (float)mps[j];
       heta[j] = S_medio[j] / p_medio[j];

       mps_medio = mps_medio + mps[j];
       Cmax = Cmax + (S_medio[j] + p_medio[j]) * mu[j];
   }
mps_medio = mps_medio / m;
Cmax = Cmax / mps_medio;

for (int i = 0; i < n; i++)
    {
     d_medio = d_medio + (float)dd[i];
     if (dd[i] < d_min)
         d_min = dd[i];
     if (dd[i] > d_max)
         d_max = dd[i];
    }
d_medio = (float)(d_medio / n);
tau = (float)(1 - (d_medio / Cmax));
R = (d_max - d_min) / Cmax;
```

Llamar subprogramas

```
// Se definen operadores utilizados en la heurística
int[,] tt_eu = new int[n, mps[0]];    // tabla con secuenciación 1ª etapa
int[, ,] tab_eu = new int[n, 4, m];   // tabla con programación completa
int[] Ctime_eu = new int[n];          // Cmax
int coste_eu = 0;

float[] part_ready = new float[n];    //array con tiempo de
                                      disponibilidad de las piezas

type = 0;          //flag que indica si realizar o no la
                   programación en la primera etapa en la función Cuing

Cuing(ref tab_eu, ref tt_eu, ref coste_eu, ref Ctime_eu, ref part_ready);

SR_file(tab_eu, Ctime_eu, coste_eu);

ILS(tt_eu, coste_eu);
```

Cálculo del tiempo empleado

```
DateTime t_end = DateTime.Now;        // Instante en que se acaba el
                                          algoritmo

TimeSpan elapsed = (t_end - t_start);
double t_empleado = 0;

t_empleado = elapsed.TotalSeconds;
```

Subfunción Cuing

La subfunción cuing calcula y dispone los trabajos en orden decreciente del índice ATCS modificado. Por lo tanto se utiliza en la generación de la solución inicial y para terminar las soluciones modificadas por las fases de búsqueda local y perturbación. Un *flag*, llamada *type*, indica al subprograma en cuál de las dos fases se encuentre.

```
static void Cuing(ref int[, ,] tab, ref int[,] tt, ref int coste, ref
                  int[] Ctime, ref float[] part_ready)
{
 int part = 1;            //número piezas a procesar, inicializamos a 1
 int indr = 0, indi = 0;  // guardan la máquina y la pieza escogida

 int cont = 0;            //marca si el programa ha entrado en el ciclo
```

```
                                principal
int cambio = 0;                //marca si ha sido elegida una pieza en el
                                ciclo principal

float time = 0;          //reloj
float ti = 0;            // tiempo inicio trabajo de la pareja escogida
float factor = 0;
double II_max = 0;
double a = 5 * (Math.Pow(10, -323));      // valor mínimo representable

int[] flag = new int[n];              // valor 1 -> pieza a procesar

float[] k1 = new float[m];            // factores de escala del ATCS
float[] k2 = new float[m];

float prima_macc = 0;                 // variables usadas para adelantar el
float primo_pezzo = 0;                   reloj

float[] p_totale = new float[n];  // tiempo de proceso remanente

      for (int i = 0; i < n; i++)
          {
           if (type == 0)
              {
               p_totale[i] = p_t[i];
              }
           if (type == 1)
              {
               p_totale[i] = p_t[i] - p_ij[i, 0];
              }
          }
int[] cont2 = new int[mps[0]];

// El ciclo principal empieza de la primera(solución inicial) o de la
segunda etapa (perturbación y b.l.)

for (int j = type; j < m; j++)
    {
    for (int i = 0; i < n; i++)      // el tiempo vuelve al instante de
                                        disponibilidad de la primera pieza
        {
         if (part_ready[i] < time)
            {
             time = part_ready[i];
             }
         if (p_ij[i, j] == 0)
             flag[i] = 0;        // marca las piezas que no se procesan
         else
             flag[i] = 1;
         }

    k1[j] = (float)1.2 * (float)(Math.Log(mu[j])) - (float)R;
    k2[j] = (float)tau / (float)(1.9 * (Math.Sqrt(heta[j])));

    int[] m_prec = new int[mps[j]];  // guarda la última pieza procesada

    float[] mach_ready = new float[mps[j]];  // instante disponibilidad
maq.

    double[,] II = new double[mps[j], n];    // matriz en que se guardan
los
                                             ATCS
```

```
    part = 1;

    while (part > 0)
        {
    cambio = 0;
    cont = 0;
    II_max = 0;    //inicializa el máximo ATCS a cero

    for (int r = 0; r < mps[j]; r++) //por cada máquina de la etapa
        {
         for (int i = 0; i < n; i++) //por cada pieza
             {
              if (mach_ready[r] <= time && part_ready[i] <= time)
                 {
                 if (flag[i] != 0)
                    {
                     cont = cont + 1;
                     factor = Math.Max(dd[i] - p_totale[i] - time, 0);

                     II[r, i] = (double)w[i] / p_totale[i] *
                     (double)(Math.Exp(-(factor) / (k1[j] * p_medio[j]))) *
                     (double)(Math.Exp(-(S_ijk[i, j, m_prec[r]]) / (k2[j] *
                     S_medio[j])));


                         if (II[r, i] < a)
                            {
                             II[r, i] = a;
                             }
                        }
                    else
                        {
                         II[r, i] = 0;
                        }

                    if (II[r, i] > II_max)
                        {
                         II_max = II[r, i];
                         indr = r;
                         indi = i;
                         ti = time;
                         cambio = 1;
                        }
                    }
                }
            }
//si ha sido elegida una combinación pieza-maquina, se actualiza la
programación
if (cambio == 1)
   {
    flag[indi] = 0;
    part_ready[indi] = ti + p_ij[indi, j] + S_ijk[indi, j, m_prec[indr]];
    mach_ready[indr] = ti + p_ij[indi, j] + S_ijk[indi, j,m_prec[indr]];
    m_prec[indr] = indi + 1;

    tab[indi, 0, j] = (int)indi;
    tab[indi, 1, j] = (int)indr;
    tab[indi, 2, j] = (int)ti;
    tab[indi, 3, j] = (int)(part_ready[indi]);

//si está en la generación de la solución inicial guarda la secuenciacion
en la primera etapa que utilizará sucesivamente
```

```
    if (j == 0)
        {
         tt[cont2[indr], indr] = indi + 1;
            cont2[indr] = cont2[indr] + 1;
            }
       }

      part = 0;    //calcula piezas que quedan por procesar

      for (int i = 0; i < n && part == 0; i++)
            {
            part = part + flag[i];
            }

     //Se adelanta el tiempo al instane de disponibilidad de la 1ª pareja
      if (cont == 0)
          {
            prima_macc = 1000000;
            primo_pezzo = 1000000;

            for (int r = 0; r < mps[j]; r++)
                  {
                   if (mach_ready[r] < prima_macc)
                       prima_macc = mach_ready[r];
                   }
            for (int i = 0; i < n; i++)
                   {
                    if (part_ready[i] < primo_pezzo && flag[i] != 0)
                       primo_pezzo = part_ready[i];
                   }

            time = Math.Max(primo_pezzo, prima_macc);
          }
      }

//Calcula tiempo de proceso remenente

     for (int i = 0; i < n; i++)
         {
           p_totale[i] = p_totale[i] - p_ij[i, j];
          }
}

//Calcola Cmax

int[] retraso = new int[n];

for (int i = 0; i < n; i++)
    {
     for (int j = 0; j < m; j++)
         {
          if (tab[i, 3, j] > Ctime[i])
              Ctime[i] = tab[i, 3, j];
          }
      retraso[i] = (int)Math.Max(Ctime[i] - dd[i], 0);
      coste = coste + (w[i] * retraso[i]);
     }
}
```

Subfunción SR_file

```
static void SR_file(int[, ,] tab, int[] Ctime, int coste)
{
StreamWriter sw = new StreamWriter(direccion_save, true);

sw.WriteLine("\n");
sw.WriteLine("DATOS DEL PROBLEMA");
sw.WriteLine();
sw.WriteLine("número de piezas:{0}", n);
sw.WriteLine();
sw.WriteLine("número de stages:{0}", m);
sw.WriteLine();
sw.WriteLine("PIEZAS\tDUE DATE\tCOMPLETION TIME\tTARDINESS");

for (int i = 0; i < n; i++)
    {
     sw.WriteLine("{0}\t{1}\t{2}\t{3}", i + 1, dd[i], Ctime[i], Ctime[i]
                  - dd[i]);
     }
sw.WriteLine();
sw.WriteLine("*** PROGRAMACIÓN ***\n");
sw.WriteLine();
sw.WriteLine("PIEZA\tMÁQUINA\tT_INICIO\tT_FIN");
sw.WriteLine();

for (int j = 0; j < m; j++)
    {
     for (int i = 0; i < n; i++)
          {
           if (p_ij[i, j] != 0)
               {
                sw.WriteLine("{0}\t{1}\t{2}\t{3}", tab[i, 0, j] + 1,
                        tab[i, 1, j] + 1, tab[i, 2, j], tab[i, 3, j]);
               }
          }
      sw.WriteLine("\n");
      }
sw.WriteLine("\n");

sw.WriteLine("coste del retraso={0}", coste);
sw.WriteLine("\n");

sw.WriteLine("datos estadísticos del problema");
sw.WriteLine();
for (int j = 0; j < m; j++)
     {
      sw.WriteLine("tiempo medio de proceso en la etapa {0}={1}", j + 1,
                  p_medio[j]);
      }
sw.WriteLine();
for (int j = 0; j < m; j++)
    {
     sw.WriteLine("tiempo medio de preparación en la etapa {0}={1}", j +
                  1, S_medio[j]);
     }
sw.WriteLine();
sw.WriteLine("Cmax estimado={0}", Cmax);

sw.Close();
```

Subfunción ILS

En este apartado se presenta el código de la heurística ILS, la explicación detallada de su funcionamiento se encuentra en la memoria del proyecto.

En el código transcrito a continuación, se puede notar como las fases de búsqueda local y perturbación estan enlazadas. Además se ve cómo el programa se sirve de la misma función *Cuing*, utilizada en la generación la solución inicial, para completar las soluciones de búsqueda local y perturbación.

```
// tt y coste son la secuenciación y el retraso total en curso

static void ILS(int[,] tt, int coste)
{
  int iter_ls = 0;      //contador iteraciones en la búsqueda local
  int iter_tot = 0;     //contador iteraciones en el algoritmo
  int num_cas1 = 0, num_cas2 = 0, num_cas_macch = 0;
  int err1 = 0, err2 = 0, err3 = 0, err4 = 0;
  int tempor = 0;
  int prec = 0;
  int n_per_m1 = 0;
  int n_per_m2 = 0;
  int n_per_p1a = 0, n_per_p1b = 0, n_per_p2a = 0, n_per_p2b = 0;
  int tp1 = 0, tp2 = 0, tp3 = 0, tp4 = 0, tp5 = 0, tp6 = 0;
  int prec1 = 0, prec2 = 0, prec3 = 0;
  int perm = 0;

  float time = 0;
  float time1 = 0, time2 = 0, time3 = 0;

  int[,] tt_best = new int[n, mps[0]];    //mejor secuenciación
  int[, ,] tab_best = new int[n, 4, m];
  int[] Ctime_best = new int[n];

  int[,] tt_ls = new int[n, mps[0]]  //secuenciación de la búsqueda local
  int[, ,] tab_ls = new int[n, 4, m];    //programación búsqueda local

  Random rnd = new Random();

  for (int i = 0; i < n; i++)
      {
       for (int s = 0; s < mps[0]; s++)
           {
//Inicializa la mejor secuenciación con la encontrada por la heurística
            tt_best[i, s] = tt[i, s];
           }
      }

coste_best = coste;

while (iter_tot < beta)       // Empieza la metaheurística
      {
```

```
        for (int i = 0; i < n; i++)
            {
             for (int s = 0; s < mps[0]; s++)
                 {
                  tt_ls[i, s] = tt[i, s];
                 }

             for (int v = 0; v < 4; v++)
                 {
                  for (int j = 0; j < m; j++)
               {
                tab_ls[i, v, j] = 0;
               }
           }
      }

perm = 0;

while (perm == 0)   // elige la máquina y las piezas a intercambiar
       {
        num_cas_macch = Convert.ToInt32(rnd.Next(0, mps[0]));
        num_cas1 = Convert.ToInt32(rnd.Next(0, n));
        num_cas2 = Convert.ToInt32(rnd.Next(0, n));
        if (tt_ls[num_cas1, num_cas_macch] == 0 || tt_ls[num_cas2,
             num_cas_macch] == 0 || num_cas2 == num_cas1)
            {
             perm = 0;
            }
        else
             perm = 1;
       }

// se intercambian las posiciones en la secuencia

tempor = tt_ls[num_cas1, num_cas_macch];
tt_ls[num_cas1, num_cas_macch] = tt[num_cas2, num_cas_macch];
tt_ls[num_cas2, num_cas_macch] = tempor;

// se guardan la programación (1ª etapa) que se obtiene

for (int j = 0; j < mps[0]; j++)
     {
      prec = 0;
      time = 0;
      for (int i = 0; i < n; i++)
          {
           if (tt_ls[i, j] != 0)
               {
                tab_ls[tt_ls[i, j] - 1, 0, 0] = tt_ls[i, j] - 1;
                tab_ls[tt_ls[i, j] - 1, 1, 0] = j;
                tab_ls[tt_ls[i, j] - 1, 2, 0] = (int)time;
                time = time + p_ij[tt_ls[i, j] - 1, 0] + S_ijk[tt_ls[i, j]
                       - 1, 0, prec];
                tab_ls[tt_ls[i, j] - 1, 3, 0] = (int)time;
                 prec = tt_ls[i, j];
               }
          }
      }

// Calcula los instantes de disponibilidad de las piezas en la 2ª etapa

float[] part_ready = new float[n];
```

```
for (int i = 0; i < n; i++)
    {
     part_ready[i] = tab_ls[i, 3, 0];
    }

//envía la solución modificada a Cuing para que se complete en las otras
etapas

int coste_ls = 0;
int[] Ctime_ls = new int[n];
type = 1;


Cuing(ref tab_ls, ref tt_ls, ref coste_ls, ref Ctime_ls, ref part_ready);

if (coste_ls < coste || coste_ls==0)
   {

    iter_ls = 0;
    for (int i = 0; i < n; i++)
        {
         for (int r = 0; r < mps[0]; r++)
              {
                tt[i, r] = tt_ls[i, r];
              }
        }

        if (coste_ls < coste_best || coste_ls==0)
           {
            coste_best = coste_ls;

                   for (int i = 0; i < n; i++)
                       {
                        Ctime_best[i] = Ctime_ls[i];

                         for (int r = 0; r < mps[0]; r++)
                             {
                               tt_best[i, r] = tt_ls[i, r];
                             }
                        }

                   for (int j = 0; j < m; j++)
                       {
                        for (int i = 0; i < n; i++)
                            {
                             tab_best[i, 0, j] = tab_ls[i, 0, j];
                             tab_best[i, 1, j] = tab_ls[i, 1, j];
                             tab_best[i, 2, j] = tab_ls[i, 2, j];
                             tab_best[i, 3, j] = tab_ls[i, 3, j];
                            }
                        }
                  if (coste_best == 0)
                     {
                      iter_ls = 2 * n;
                      iter_tot = 2 * n;
                     }
             }
    }

iter_ls = iter_ls + 1;
```

```
//cuando se han efectuado las iteraciones de búsqueda local se entra en
la perturbación
if (iter_ls > alfa)
   {
    iter_ls = 0;
    iter_tot = iter_tot + 1;

// se eligen las máquinas para efectuar la perturbación

n_per_m1 = Convert.ToInt32(rnd.Next(0, mps[0]));
n_per_m2 = Convert.ToInt32(rnd.Next(0, mps[0]));

err1 = 0;
err2 = 0;
err3 = 0;
err4 = 0;


//se modifica la secuenciación tt[i,r], o sea la solución en curso, aquí
está el criterio de aceptación

if (n_per_m1 == n_per_m2)
   {
    do
      {
       n_per_p1a = Convert.ToInt32(rnd.Next(0, n));
      }
    while (tt[n_per_p1a, n_per_m1] == 0);

    do
      {
       n_per_p1b = Convert.ToInt32(rnd.Next(0, n));
      }
    while (n_per_p1b == n_per_p1a || tt[n_per_p1b, n_per_m1] == 0);

    do
      {
       n_per_p2a = Convert.ToInt32(rnd.Next(0, n));
       err1 = err1 + 1;
       if (err1 > 100)
           goto Salto;
      }
    while (n_per_p2a == n_per_p1a || n_per_p2a == n_per_p1b ||
            tt[n_per_p2a, n_per_m2] == 0);

    do
      {
       n_per_p2b = Convert.ToInt32(rnd.Next(0, n));
       err2 = err2 + 1;
       if (err2 > 100)
          goto Salto;
      }
    while (n_per_p2b == n_per_p1a || n_per_p2b == n_per_p1b || n_per_p2b
            == n_per_p2a || tt[n_per_p2b, n_per_m2] == 0);

   }
else
   {
    do
      {
       n_per_p1a = Convert.ToInt32(rnd.Next(0, n));
      }
```

```
    while (tt[n_per_p1a, n_per_m1] == 0);

    do
      {
       n_per_p1b = Convert.ToInt32(rnd.Next(0, n));
       err3 = err3 + 1;
       if (err3 > 100)
         goto Salto;
      }
    while (tt[n_per_p1b, n_per_m1] == 0 || n_per_p1a == n_per_p1b);
   do
     {
      n_per_p2a = Convert.ToInt32(rnd.Next(0, n));
     }
   while (tt[n_per_p2a, n_per_m2] == 0);

   do
     {
      n_per_p2b = Convert.ToInt32(rnd.Next(0, n));
      err4 = err4 + 1;
      if (err4 > 100)
        goto Salto;
     }
  while (tt[n_per_p2b, n_per_m2] == 0 || n_per_p2a == n_per_p2b);
}

int[,] tt_per1 = new int[n, mps[0]];
int[,] tt_per2 = new int[n, mps[0]];
int[,] tt_per3 = new int[n, mps[0]];

for (int i = 0; i < n; i++)
    {
     for (int r = 0; r < mps[0]; r++)
         {
          tt_per1[i, r] = tt[i, r];
          tt_per2[i, r] = tt[i, r];
          tt_per3[i, r] = tt[i, r];
         }
    }

// se generan las tres perturbaciones

tp1 = tt_per1[n_per_p1a, n_per_m1];
tp2 = tt_per1[n_per_p2a, n_per_m2];

tt_per1[n_per_p1a, n_per_m1] = tt_per1[n_per_p1b, n_per_m1];
tt_per1[n_per_p2a, n_per_m2] = tt_per1[n_per_p2b, n_per_m2];

tt_per1[n_per_p1b, n_per_m1] = tp1;
tt_per1[n_per_p2b, n_per_m2] = tp2;


tp3 = tt_per2[n_per_p1a, n_per_m1];
tp4 = tt_per2[n_per_p1b, n_per_m1];

tt_per2[n_per_p1a, n_per_m1] = tt_per2[n_per_p2a, n_per_m2];
tt_per2[n_per_p1b, n_per_m1] = tt_per2[n_per_p2b, n_per_m2];

tt_per2[n_per_p2a, n_per_m2] = tp3;
tt_per2[n_per_p2b, n_per_m2] = tp4;
```

```
tp5 = tt_per3[n_per_p1a, n_per_m1];
tp6 = tt_per3[n_per_p1b, n_per_m1];

tt_per3[n_per_p1a, n_per_m1] = tt_per3[n_per_p2b, n_per_m2];
tt_per3[n_per_p1b, n_per_m1] = tt_per3[n_per_p2a, n_per_m2];

tt_per3[n_per_p2b, n_per_m2] = tp5;
tt_per3[n_per_p2a, n_per_m2] = tp6;

//Se generan los ficheros necesarios para evaluar las tres programaciones

int[, ,] tab_per1 = new int[n, 4, m];
int[, ,] tab_per2 = new int[n, 4, m];
int[, ,] tab_per3 = new int[n, 4, m];

for (int i = 0; i < n; i++)
    {
     for (int s = 0; s < 4; s++)
         {
         for (int j = 0; j < m; j++)
             {
              tab_per1[i, s, j] = 0;
              tab_per2[i, s, j] = 0;
              tab_per3[i, s, j] = 0;
             }
         }

    }

for (int j = 0; j < mps[0]; j++)
    {
     prec1 = 0;
     time1 = 0;

     prec2 = 0;
     time2 = 0;

     prec3 = 0;
     time3 = 0;

for (int i = 0; i < n; i++)
    {
     if (tt_per1[i, j] != 0)
        {
         tab_per1[tt_per1[i, j] - 1, 0, 0] = tt_per1[i, j] - 1;
         tab_per1[tt_per1[i, j] - 1, 1, 0] = j;
         tab_per1[tt_per1[i, j] - 1, 2, 0] = (int)time1;
         time1 = time1 + p_ij[tt_per1[i, j] - 1, 0] + S_ijk[tt_per1[i, j]
                 - 1, 0, prec1];
         tab_per1[tt_per1[i, j] - 1, 3, 0] = (int)time1;
         prec1 = tt_per1[i, j];
        }
     if (tt_per2[i, j] != 0)
        {
         tab_per2[tt_per2[i, j] - 1, 0, 0] = tt_per2[i, j] - 1;
         tab_per2[tt_per2[i, j] - 1, 1, 0] = j;
         tab_per2[tt_per2[i, j] - 1, 2, 0] = (int)time2;
         time2 = time2 + p_ij[tt_per2[i, j] - 1, 0] + S_ijk[tt_per2[i, j]
                  - 1, 0, prec2];
         tab_per2[tt_per2[i, j] - 1, 3, 0] = (int)time2;
         prec2 = tt_per2[i, j];
        }
```

```
     if (tt_per3[i, j] != 0)
        {
         tab_per3[tt_per3[i, j] - 1, 0, 0] = tt_per3[i, j] - 1;
         tab_per3[tt_per3[i, j] - 1, 1, 0] = j;
         tab_per3[tt_per3[i, j] - 1, 2, 0] = (int)time3;
         time3 = time3 + p_ij[tt_per3[i, j] - 1, 0] + S_ijk[tt_per3[i, j]
                  - 1, 0, prec3];
         tab_per3[tt_per3[i, j] - 1, 3, 0] = (int)time3;
         prec3 = tt_per3[i, j];
        }
   }
}

float[] part_ready_per1 = new float[n];
float[] part_ready_per2 = new float[n];
float[] part_ready_per3 = new float[n];

int coste_per1 = 0;
int coste_per2 = 0;
int coste_per3 = 0;

int[] Ctime_per1 = new int[n];                // complection time!
int[] Ctime_per2 = new int[n];
int[] Ctime_per3 = new int[n];

for (int i = 0; i < n; i++)
    {
      Ctime_per1[i] = 0;
      Ctime_per2[i] = 0;
      Ctime_per3[i] = 0;

       part_ready_per1[i] = tab_per1[i, 3, 0];
       part_ready_per2[i] = tab_per2[i, 3, 0];
       part_ready_per3[i] = tab_per3[i, 3, 0];
    }

type = 1;
coste = 0;  // se inicializa a cero de forma que el programa entre
               siempre en el primero de los if que hay a continuación

// se envian las nuevas secuencias a Cuing para que se completen
Cuing(ref tab_per1, ref tt_per1, ref coste_per1, ref Ctime_per1, ref
part_ready_per1);

if (coste == 0 || coste > coste_per1)
    {
     coste = coste_per1;
     for (int i = 0; i < n; i++)
          {
            for (int r = 0; r < mps[0]; r++)
                 {
                  tt[i, r] = tt_per1[i, r];
                 }
          }
    }

Cuing(ref tab_per2, ref tt_per2, ref coste_per2, ref Ctime_per2, ref
part_ready_per2);

if (coste > coste_per2)
    {
     coste = coste_per2;
```

```
     for (int i = 0; i < n; i++)
         {
          for (int r = 0; r < mps[0]; r++)
              {
               tt[i, r] = tt_per2[i, r];
              }
          }
    }

 Cuing(ref tab_per3, ref tt_per3, ref coste_per3, ref Ctime_per3, ref
part_ready_per3);

  if (coste > coste_per3)
     {
      coste = coste_per3;
      for (int i = 0; i < n; i++)
          {
           for (int r = 0; r < mps[0]; r++)
              {
               tt[i, r] = tt_per3[i, r];
              }
          }
      }
  Salto: ;
    }
  }

// se guarda finalmente la mejor sulución
SR_file(tab_best, Ctime_best, coste_best);

StreamWriter final = new StreamWriter(dirección save, true);

final.Write("\t");
final.Write(coste_best);
final.Close();
```