
**UNIVERSIDAD DE GRANADA**

**E.T.S. DE INGENIERÍAS INFORMÁTICA y DE TELECOMUNICACIÓN**

**Departamento de Ciencias de la Computación e Inteligencia Artificial**

# SIMULACIÓN DE SISTEMAS

## Guión de Prácticas

### Práctica 3: Modelos de Simulación Dinámicos y Discretos

Curso 2022-2023

Grado en Informática

# Capítulo 1

# Mi Segundo Modelo de Simulación Discreto

## 1.1. Planteamiento del sistema a estudiar

Consideremos un sistema consistente en una entidad (servidor) que presta determinado servicio a una serie de entidades (clientes) que periódicamente llegan a solicitar dicho servicio. Por ejemplo, los clientes pueden ser máquinas o piezas que necesitan ser reparadas y el servidor un técnico reparador, o personas que van a ser atendidas por un cajero de un banco o de un supermercado, etcétera. Disponemos de información sobre los tiempos entre llegadas sucesivas de clientes, $A_1$, $A_2$, $A_3$,... (que son variables aleatorias independientes e idénticamente distribuidas; un generador de datos se encargará de proporcionar esos valores en tiempo de simulación). Un cliente que llega y encuentra al servidor libre pasa a ser atendido inmediatamente. También disponemos de información sobre los tiempos de servicio, $S_1$, $S_2$, $S_3$,... de los clientes sucesivos (que son variables aleatorias independientes e idénticamente distribuidas, e independientes de los tiempos entre llegadas; otro generador de datos proporcionará esos valores). Un cliente que llega y encuentra ocupado al servidor debe esperar turno. Cuando se completa un servicio, el servidor elige a un cliente de los que esperan, si lo hay, en una forma FIFO (primero en llegar, primero en ser servido, o sea, los clientes esperan formando una cola). Queremos simular este sistema hasta que $n$ clientes hayan sido atendidos, y estimar el número medio de clientes en cola, $Q(n)$, y el porcentaje de tiempo de ocio del servidor, $PTO(n)$.

Asumiremos que al empezar la simulación no hay clientes esperando y el servidor está libre. Utilizaremos un generador de datos de tiempos entre llegadas y uno de tiempos de servicio ambos exponenciales de medias `tlleg` y `tserv`, respectivamente, expresadas, por ejemplo, en horas. Así, `tlleg`=0.15 (9 minutos) y `tserv`=0.1 (6 minutos).

## 1.2. Gestión del tiempo. Métodos de incremento fijo e incremento variable de tiempo

Vamos a construir primero un sencillo modelo de simulación basado en lo que se llama método de *incremento fijo de tiempo*, y sin la estructuración que conviene utilizar (que ya se verá más adelante). El tiempo simulado se va a mover de unidad en unidad (como un reloj

convencional), aunque esa unidad de tiempo puede ser la que mejor nos parezca: minutos, medias horas, horas, días,... Posteriormente construiremos otro modelo con el método de *incremento variable de tiempo*.

### 1.2.1. Pseudocódigo del programa de simulación con incremento fijo

A continuación aparece el pseudocódigo para la programación del modelo de simulación.

```
inicializar generadores de datos //imprescindible (por ejemplo, srand(time(NULL)))
infinito = 10e30; //tiempo en que ocurren cosas que sabemos no pueden ocurrir
atendidos = 0; //al principio no hay nadie ya atendido
inicio_ocio = 0.0; //marcará el momento en que el servidor empieza a estar ocioso
acum_cola = 0.0; //acumulador de número de clientes en cola por el tiempo en que
       //están en dicha cola. El cliente que está siendo atendido NO está en cola
reloj = 0; //marca el valor del tiempo simulado, inicialmente cero
servidor_libre = true; //inicialmente el servidor no está atendiendo a nadie
encola = 0; //no hay nadie en cola todavía
tiempo_llegada = reloj + generallegada(tlleg); //llegará el primer cliente
tiempo_salida = infinito; //nadie puede salir si nadie ha entrado aun
while (atendidos < total_a_atender) //simularemos hasta que hayamos atendido a
                                    //cierto número de clientes, total_a_atender
     {
      if (reloj == tiempo_llegada) //si estamos en el instante en que llega alguien
       {
        tiempo_llegada = reloj + generallegada(tlleg); //determinamos cuando llegará
                                                       //el siguiente cliente
        if (servidor_libre) //si el servidor está ocioso
             {
              servidor_libre = false; //deja de estarlo
              tiempo_salida = reloj + generaservicio(tserv); //determinamos cuando
                                          //saldrá ese cliente que acaba de llegar
              ocio += reloj - inicio_ocio; //acumulamos el ocio hasta este momento
             }
        else { //el servidor está ocupado
              acum_cola += (reloj - tultsuc) * encola; //acumulamos el número de
                      //clientes en cola por el tiempo en que han estado en cola
              tultsuc = reloj; //para saber en qué momento cambió el tamaño de la
                               //cola, en este caso aumentó en uno más
              encola ++;  //hay un cliente más en cola
             }
       }
      if (reloj == tiempo_salida) //si estamos en el instante en que se va alguien
       {
        atendidos ++; //se ha atendido a un cliente más
        if (encola > 0) //si quedan clientes en cola
```

```
            {
             acum_cola += (reloj - tultsuc) * encola; // acumulamos el número de
                       //clientes en cola por el tiempo en que han estado en cola
             tultsuc = reloj; //para saber en qué momento cambió el tamaño de la
                              //cola, en este caso disminuyó en uno
             encola  --; //hay un cliente menos en cola
             tiempo_salida = reloj + generaservicio(tserv); //determinamos cuando
                                        //saldrá ese cliente que acaba de entrar
            }
        else { //no quedan clientes en cola
              servidor_libre = true; //el servidor se queda ocioso por falta de clientes
              inicio_ocio = reloj; //marcamos cuando empieza a estar ocioso
              tiempo_salida = infinito; //nadie puede salir puesto que nadie hay
             }
       }
       reloj ++;   //el tiempo se incrementa en una unidad
      }
porcent_ocio = ocio*100/reloj; //calculamos el % de tiempo de ocio del servidor
printf(porcent_ocio);
media_encola = acum_cola/reloj; //calculamos el número medio de clientes en cola
printf(media_encola);
```

Los generadores de datos mencionados, `generallegada` y `generaservicio`, se pueden implementar del siguiente modo (método de inversión):

float generallegada(`tlleg`)

```
u = (float) random(); // o también rand() en lugar de random()
u = (float) (u/(RAND_MAX+1.0); //RAND_MAX es una constante del sistema
return (-tlleg*log(1-u));
```

float generaservicio: es igual, cambiando `tlleg` por `tserv`.

En nuestro caso, y dado que vamos a mover el tiempo en unidades enteras, esos generadores hay que modificarlos ligeramente. En primer lugar, hay que redondear la salida (que es un float) al entero más próximo; también, para evitar problemas con el manejo del tiempo (y no dejarnos cosas perdidas en el pasado), si el valor devuelto por el generador es 0, debe cambiarse por un 1 (pensad por qué); por último, hay que decidir la unidad de tiempo que vamos a emplear: si se trata de horas, entonces `tlleg`=0.15 y `tserv`=0.1; si se trata de minutos, entonces `tlleg`=9 (0.15*60) y `tserv`=6 (0.1*60), y así sucesivamente.

### 1.2.2. Tareas a realizar

Construid un programa que implemente este modelo. Ejecutadlo, para un número de clientes a atender elevado (p.e. 10000), y empleando diferentes unidades de medida de tiempo (p.e. horas, medias horas, cuartos de horas, minutos, segundos, décimas de segundo, décimas de hora,

centésimas de hora, décimas de minuto,...). Repetid varias veces ¿Qué observais en relación a los valores devueltos por las diferentes simulaciones? ¿Son los resultados obtenidos empleando diferentes unidades de tiempo coherentes entre sí?

### 1.2.3. Pseudocódigo del programa de simulación con incremento variable de tiempo

Ahora vamos a construir otra versión del modelo, que emplea otra técnica de control del tiempo, denominada *incremento variable de tiempo*, más eficiente y precisa (pero todavía seguimos sin emplear la estructuración apropiada).

En este caso la variable tiempo no tiene que ser entera, puede ser float, y los generadores de datos pueden emplearse sin ninguna modificación. Unicamente hay que tener en cuenta que si modificamos la unidad de medida del tiempo (p.e. minutos en vez de horas), los parámetros `tlleg` y `tserv` deben modificarse en consecuencia.

```
inicializar generadores de datos
infinito = 10e30;
atendidos = 0;
inicio_ocio = 0.0;
acum_cola = 0.0;
reloj = 0.0;
servidor_libre = true;
encola = 0;
tiempo_llegada = reloj + generallegada(tlleg);
tiempo_salida = infinito;
while (atendidos < total_a_atender) do
     {
      reloj = min(tiempo_llegada,tiempo_salida); //una función que calcula el mínimo
      if (reloj == tiempo_llegada)
       {
        tiempo_llegada = reloj + generallegada(tlleg);
        if (servidor_libre)
             {
              servidor_libre = false;
              tiempo_salida = reloj + generaservicio(tserv);
              ocio += reloj - inicio_ocio;
             }
        else {
              acum_cola += (reloj - tultsuc) * encola;
              tultsuc = reloj;
              encola ++;
             }
       }
      if (reloj == tiempo_salida)
       {
```

```
        atendidos ++;
        if (encola > 0)
             {
              acum_cola += (reloj - tultsuc) * encola;
              tultsuc = reloj;
              encola  --;
              tiempo_salida = reloj + generaservicio(tserv);
             }
        else {
              servidor_libre = true;
              inicio_ocio = reloj;
              tiempo_salida = infinito;
             }
       }
      }
porcent_ocio = ocio*100/reloj;
printf(porcent_ocio);
media_encola = acum_cola / reloj;
printf(media_encola);
```

Se puede observar que el cambio con respecto al modelo desarrollado anteriormente es muy pequeño, en este caso afecta únicamente a dos líneas de código (pero esto representa una diferencia importantísima).

### 1.2.4. Tareas a realizar

Construid un programa que implemente este modelo. Comparad la eficiencia de los métodos de incremento fijo y variable midiendo los tiempos de ejecución de ambos modelos de simulación, para un número de clientes atendidos elevado (el mismo en todos los casos, por ejemplo 10000), usando diferentes unidades de tiempo (horas, minutos, segundos, décimas de segundo). Esto implica modificar los parámetros `tlleg` y `tserv`. ¿Qué conclusiones se pueden extraer?

Comparad también los resultados obtenidos mediante incremento fijo y variable en cuanto a precisión, para las diferentes elecciones de la unidad de tiempo del reloj. Para ello, realizar simulaciones de cada sistema, con un número elevado de clientes atendidos. Se pueden comparar los resultados obtenidos con valores teóricos (siempre que `tserv`<`tlleg`). Concretamente, si $\rho = \frac{\texttt{tserv}}{\texttt{tlleg}}$, entonces $Q(n) \to \frac{\rho^2}{1-\rho}$, y $PTO(n) \to 100 * (1-\rho)$, cuando $n \to +\infty$. ¿Se obtiene alguna conclusión sobre la precisión de los dos métodos?

# Capítulo 2

# Mi Tercer Modelo de Simulación Discreto

Ahora vamos a modelizar un sistema más complejo, utilizando el método de incremento variable de tiempo y también la estructuración típica de un modelo de simulación. Esto implica manejar una *lista de sucesos* para el control del tiempo, y disponer de procedimientos de *inicialización*, *temporización*, generación de *informes* y de gestión de cada tipo de *suceso*. La forma de detener la simulación también será diferente: en vez de parar cuando se haya atendido cierto número de clientes (o en general cuando haya ocurrido algún suceso específico), pararemos cuando haya transcurrido cierta cantidad de tiempo.

## 2.1. Planteamiento del sistema a estudiar

Un centro de trabajo dispone de $k$ máquinas idénticas (p.e. fotocopiadoras) y un único operario encargado de manejarlas. Los trabajos que van llegando deben cargarse en una máquina (si hay alguna disponible) por parte del operario. Si todas las máquinas están ocupadas, los trabajos deben esperar turno (en una única cola). Incluso si hay máquinas disponibles, los trabajos deben esperar hasta que el operario esté libre para cargarlos en una máquina. Una vez cargado un trabajo, la máquina lo procesa automáticamente sin requerir la intervención del operario hasta que termina. No obstante, cuando una máquina termina un trabajo, este debe de ser descargado de la misma por el operario. Cuando se de la circunstancia de que haya trabajos que deben ser cargados y otros descargados, la prioridad es que el operario se encarga primero de descargar trabajos. Disponemos de información sobre los tiempos entre llegadas de trabajos, los tiempos de carga y descarga de trabajos por parte del operario y los tiempos de procesamiento de los trabajos en las máquinas.

Se quiere calcular el número medio de trabajos en espera de procesamiento (en espera de ser cargados), el número medio de trabajos en espera de ser descargados, el tiempo medio de estancia de los trabajos en el centro, el tiempo de ocio del operario y el total de trabajos procesados.

## 2.2. Sucesos y grafos de sucesos. Estructura de un programa de simulación dinámico y discreto

### 2.2.1. Sucesos y Grafo de Sucesos

Inicialmente podemos considerar los siguientes sucesos:

- Llegada de trabajo
- Comienzo de carga
- Fin de carga
- Comienzo de procesamiento
- Fin de procesamiento
- Comienzo de descarga
- Fin de descarga

El grafo de sucesos podría ser el representado en la figura 2.1.

Si reducimos el grafo al mínimo, tenemos los siguientes sucesos, y el grafo correspondiente se representa en la figura 2.2.

- Llegada de trabajo
- Fin de carga
- Fin de procesamiento
- Fin de descarga

### 2.2.2. Variables de estado

Las variables de estado esenciales en este modelo son: el `reloj`, la situación del operario, `operario_libre`, el número de máquinas desocupadas, `maquinas_libres`, el número de trabajos esperando ser cargados (y procesados), `encola_lleg` y el número de trabajos esperando ser descargados `encola_sal`.

También se necesitan otras variables, `tdus_lleg`, `tdus_sal`, para almacenar los tiempos del último suceso que modifica `encola_lleg` y `encola_sal` (necesarios para calcular el número medio de trabajos en cada cola), y `comienzo_ocio` (necesaria para calcular los tiempos de ocio del operario). También se necesitan dos estructuras cola, `cola_llegadas` y `cola_salidas`, para almacenar el tiempo en que llegó al sistema cada trabajo que se encuentre en cola para cargarse o descargarse (necesarias para poder calcular los tiempos de estancia).

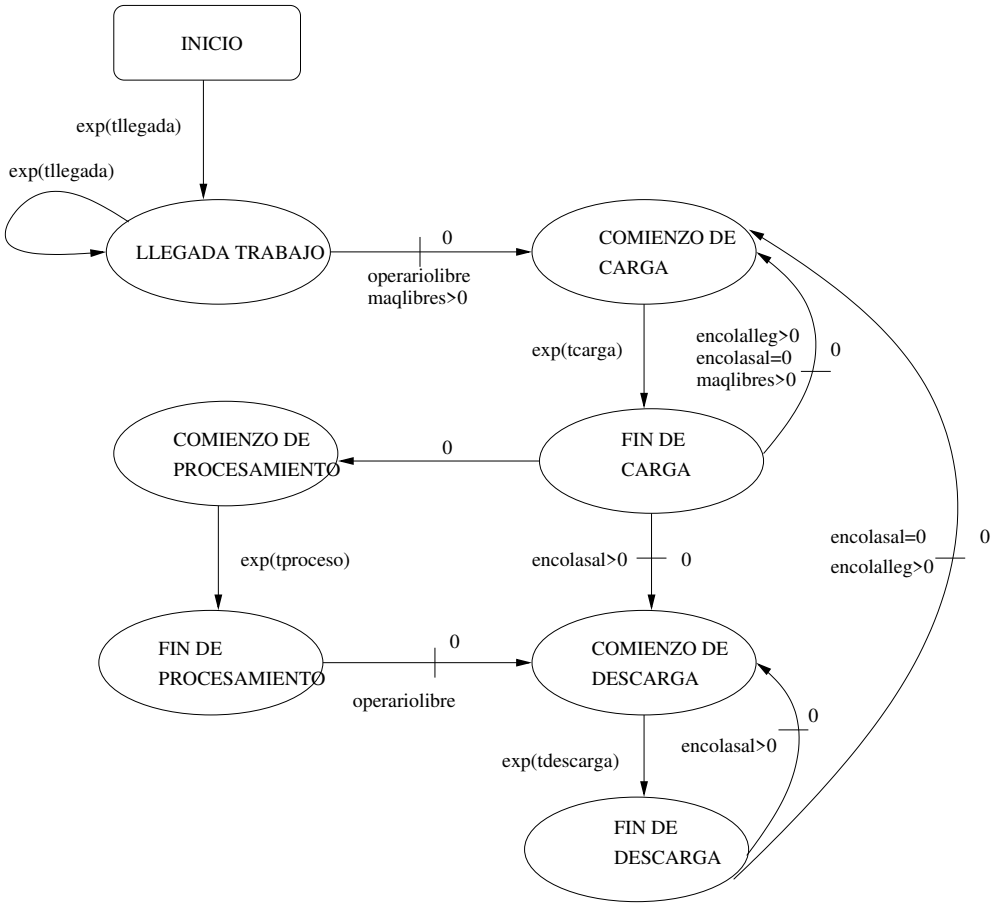


Figura 2.1: Grafo de sucesos original.

### 2.2.3. Contadores estadísticos

Se necesitan acumuladores para calcular las medidas de rendimiento deseadas, `acum_lleg` y `acum_sal` para el número medio de trabajos en las colas, `acum_ocio` para el tiempo de ocio del operario, `total_procesados` para el número de trabajos procesador y `acum_estancia` para el tiempo medio de estancia de los trabajos.

### 2.2.4. Parametros de entrada

`num_maquinas`, `tllegada`, `tcarga`, `tdescarga`, `tproceso` (tiempos medios entre llegadas, de carga, descarga y proceso) y `tparada` (tiempo que durará la simulación).

### 2.2.5. Rutinas del Modelo de Simulación

Programa principal

```
inicializacion();
while (!parar)
```

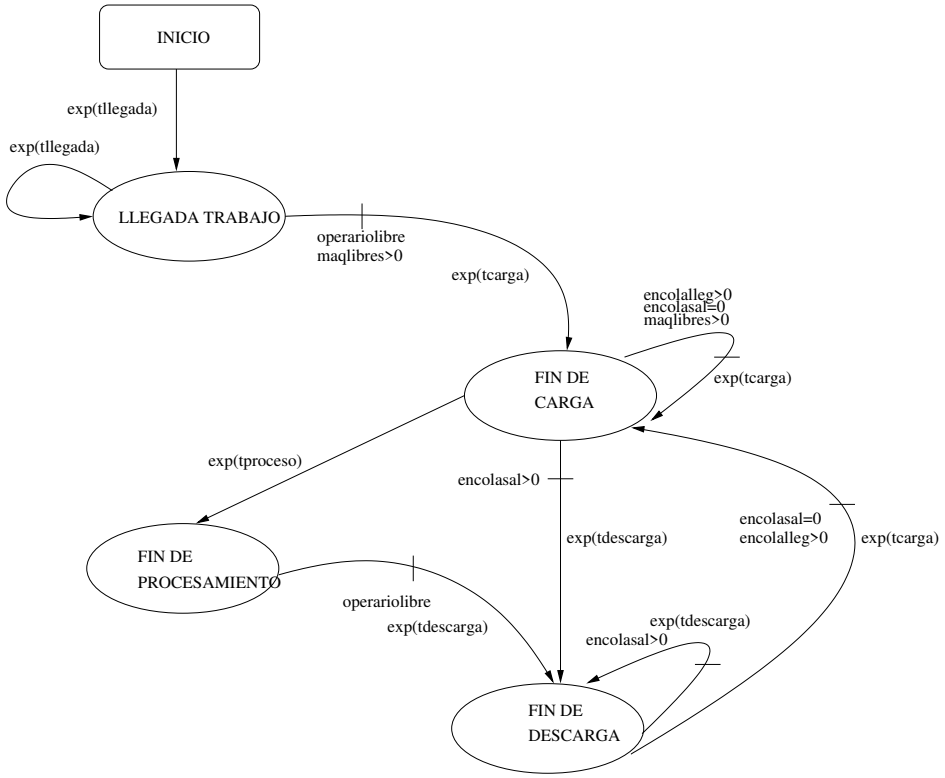


Figura 2.2: Grafo de sucesos reducido.

```
    {
     temporizacion();
     suceso();
    }
//generarInforme(); el generador de informes es la rutina del suceso fin de simulacion
```

Procedimiento inicializacion()

```
// inicializar generador de numeros pseudoaleatorios;
// si se van a hacer varias simulaciones, hacerlo en el main
  reloj = 0.0;
  operario_libre = true;
  maquinas_libres = num_maquinas;
  encola_lleg = 0;
  encola_sal = 0;
  tdus_lleg = 0.0;
  tdus_sal = 0.0;
  comienzo_ocio = reloj;
  acum_lleg = 0.0;
  acum_sal = 0.0;
  acum_estancia = 0.0;
```

```
acum_ocio = 0.0;
total_procesados = 0;
nodo.tllego = 0; //almacena el tiempo en que llega un trabajo
nodo.suceso = SUCESO_FIN_SIMULACION;
nodo.tiempo = reloj+tparada;
insertar_lsuc(nodo);
nodo.suceso = SUCESO_LLEGADA_TRABAJO;
nodo.tiempo = reloj+genera_trabajo(tllegada);
insertar_lsuc(nodo);
parar=false;
```

Procedimiento temporizacion()

Supuesto que `lsuc` es una estructura de datos con los registros ordenados en orden creciente del campo tiempo, simplemente extrae el registro con menor valor del tiempo, actualiza el valor de `reloj` a ese valor y determina el tipo de suceso de que se trata, para llamar después a la rutina de suceso correspondiente.

```
nodo = lsuc.front();
lsuc.pop_front();
reloj = nodo.tiempo;
```

Procedimiento suceso()

```
switch(nodo.suceso) {
  case SUCESO_LLEGADA_TRABAJO: llegada_trabajo(); break;
  case SUCESO_FIN_CARGA: fin_carga(); break;
  case SUCESO_FIN_DESCARGA: fin_descarga(); break;
  case SUCESO_FIN_PROCESAMIENTO: fin_procesamiento(); break;
  case SUCESO_FIN_SIMULACION: fin_simulacion(); break;
}
```

Procedimiento llegada_trabajo()

```
nodo.suceso = SUCESO_LLEGADA_TRABAJO;
nodo.tiempo = reloj+genera_trabajo(tllegada);
nodo.tllego = 0;
insertar_lsuc(nodo); //programa la proxima llegada
if (operario_libre && maquinas_libres>0) {
  nodo.suceso = SUCESO_FIN_CARGA;
  nodo.tiempo = reloj+genera_carga(tcarga);
  nodo.tllego = reloj;
  insertar_lsuc(nodo);
  operario_libre = false;
  maquinas_libres --;
  acum_ocio += reloj-comienzo_ocio;
```

```
}
else {
      acum_lleg += (reloj-tdus_lleg)*encola_lleg;
      tdus_lleg = reloj;
      encola_lleg ++;
      cola_llegadas.push_back(reloj);
}
```

Procedimiento fin_carga()

```
nodo.suceso = SUCESO_FIN_PROCESAMIENTO;
nodo.tiempo = reloj+genera_procesamiento(tproceso);
//nodo.tllego se deja como esta, se hereda del suceso actual
insertar_lsuc(nodo);
if (encola_sal > 0)
      {
      acum_sal += (reloj-tdus_sal)*encola_sal;
      tdus_sal = reloj;
      encola_sal --;
      cuandollegue = cola_salidas.front();
      cola_salidas.pop_front();
      nodo.suceso = SUCESO_FIN_DESCARGA;
      nodo.tiempo = reloj+genera_descarga(tdescarga);
      nodo.tllego = cuandollegue;
      insertar_lsuc(nodo);
      }
   else if (encola_lleg > 0 && maquinas_libres > 0)
          {
           acum_lleg += (reloj-tdus_lleg)*encola_lleg;
           tdus_lleg = reloj;
           encola_lleg --;
           maquinas_libres --;
           cuandollegue = cola_llegadas.front();
           cola_llegadas.pop_front();
           nodo.suceso = SUCESO_FIN_CARGA;
           nodo.tiempo = reloj+genera_carga(tcarga);
           nodo.tllego = cuandollegue;
           insertar_lsuc(nodo);
          }
        else {
              operario_libre = true;
              comienzo_ocio = reloj;
        }
```

Procedimiento fin_procesamiento()

```
if (operario_libre) {
   nodo.suceso = SUCESO_FIN_DESCARGA;
   nodo.tiempo = reloj+genera_descarga(tdescarga);
   //nodo.tllego se deja como esta, se hereda del suceso actual
   insertar_lsuc(nodo);
   operario_libre = false;
   acum_ocio += reloj-comienzo_ocio;
   }
else
   {
   acum_sal += (reloj-tdus_sal)*encola_sal;
   tdus_sal = reloj;
   encola_sal ++;
   cola_salidas.push_back(nodo.tllego);
   }
```

Procedimiento fin_descarga()

```
maquinas_libres ++;
total_procesados ++;
acum_estancia += reloj-nodo.tllego;
if (encola_sal > 0)
      {
      acum_sal += (reloj-tdus_sal)*encola_sal;
      tdus_sal = reloj;
      encola_sal --;
      cuandollegue = cola_salidas.front();
      cola_salidas.pop_front();
      nodo.suceso = SUCESO_FIN_DESCARGA;
      nodo.tiempo = reloj+genera_descarga(tdescarga);
      nodo.tllego = cuandollegue;
      insertar_lsuc(nodo);
      }
   else if (encola_lleg > 0)
           {
            acum_lleg += (reloj-tdus_lleg)*encola_lleg;
            tdus_lleg = reloj;
            encola_lleg --;
            maquinas_libres --;
            cuandollegue = cola_llegadas.front();
            cola_llegadas.pop_front();
            nodo.suceso = SUCESO_FIN_CARGA;
            nodo.tiempo = reloj+genera_carga(tcarga);
            nodo.tllego = cuandollegue;
            insertar_lsuc(nodo);
```

```
   }
else {
      operario_libre = true;
      comienzo_ocio = reloj;
}
```

Procedimiento fin_simulacion (equivale al generador de informes)

```
parar=true;
acum_lleg += (reloj-tdus_lleg)*encola_lleg;
acum_sal += (reloj-tdus_sal)*encola_sal;
if (operario_libre) acum_ocio += reloj-comienzo_ocio;
printf("\nNumero de trabajos esperando ser cargados = %f",acum_lleg/reloj);
printf("\nNumero de trabajos esperando ser descargados = %f",acum_sal/reloj);
printf("\nTotal de trabajos procesados= %f",total_procesados);
printf("\nTiempo de estancia de los trabajos = %f",acum_estancia/total_procesados);
printf("\nPorcentaje de tiempo de ocio del operario = %f",acum_ocio*100/reloj);
```

### 2.2.6. Tareas a realizar

El programa `multiplesmaquinas1trabajador.cpp` implementa el modelo descrito anteriormente.

Probad el programa (que permite especificar el número de simulaciones[1]). Utilizad el número de simulaciones necesario para que los resultados sean razonablemente estables.

Estudiad qué ocurre si reemplazáis todos los generadores de datos exponenciales por generadores determinísticos que siempre devuelven los correspondientes valores medios, o por generadores uniformes (con la misma media que los anteriores)[2].

Investigad qué ocurriría si en lugar de las $x$ máquinas de las que disponemos, con tiempo de procesamiento `tproceso` cada una, tuviésemos una sola pero $x$ veces más rápida, con tiempo de proceso `tproceso/x` (cantidad versus calidad).

## 2.3. Modificando el modelo original

Nos planteamos ahora la posibilidad de disponer de más de un operario para manejar las máquinas, de manera que mientras uno esté haciendo una tarea (cargar o descargar una máquina), otros puedan estar haciendo tareas con otras máquinas.

### 2.3.1. Tareas a realizar

Modificad el modelo y el programa de simulación anteriores para que pueda haber varios operarios. Describid en detalle los cambios que hay que realizar.

[1] Para cambiar los otros parámetros de entrada hay que modificar el correspondiente fichero .h

[2] El objetivo es comprobar que no basta con tener información sobre comportamiento medio, la existencia de aleatoriedad y el tipo de distribución también pueden ser muy importantes. Los resultados pueden ser diferentes manteniendo los valores medios pero cambiando las distribuciones.

Investigad, ya que existe la posibilidad de disponer de varios operarios, qué sería más beneficioso para el sistema: disponer de varios operarios o de un único operario mucho más eficiente (de nuevo cantidad frente a calidad). Concretamente, comparad un sistema con $m$ operarios, cada uno con tiempos medios de carga y de descarga `tcarga`, `tdescarga`, respectivamente, con un sistema con un único operario pero con tiempos de carga y descarga $m$ veces más rápidos, `tcarga/m`, `tdescarga/m` (también por ejemplo $m/2$ operarios con tiempos de carga y descarga el doble de rápidos, `tcarga/2`, `tdescarga/2`, o en general cuando el cociente entre los tiempos de carga o descarga y el número $m$ de operarios sea constante).

# Capítulo 3

# Análisis de Salidas y Experimentación

Para el modelo de simulación del sistema con $k$ máquinas y $m$ operarios, desarrollado en el capítulo anterior, vamos a realizar diversos tipos de análisis de salidas y experimentación.

## 3.1. ¿Cuánto hay que simular?

Se pretende analizar y comparar las siguientes dos configuraciones alternativas del sistema, desde el punto de vista del número medio de trabajos en espera de carga, $NTEC$:

(A) sistema con 10 máquinas y 2 operarios con tiempos de carga y descarga de 3 y 2 minutos.

(B) sistema con 10 máquinas y 1 operario con tiempos de carga y descarga de 1.5 y 1 minutos.

Utilizad el programa del capítulo anterior para hacer lo siguiente:

- Haced una simulación de cada sistema, obtened los valores estimados, $NTEC_A$ y $NTEC_B$ para cada sistema, y señalad como preferible el sistema que produzca un valor de $NTEC$ menor. Repetid 100 veces este proceso, obteniendo 100 valores de $NTEC_A$ y $NTEC_B$, y el porcentaje de veces en que el sistema (A) es preferible al (B) y viceversa. Esos porcentajes representan un estimador de la probabilidad de, en cada caso, tomar la decisión equivocada al decidir que un sistema es mejor que el otro, sobre la base de hacer *una única* simulación.

- Haced ahora cinco simulaciones de cada sistema, y obtened los valores estimados de $NTEC_A$ y $NTEC_B$ como la media de los resultados de las cinco simulaciones de cada sistema. Repetid este proceso 100 veces (o sea, haciendo un total de 500 simulaciones de cada sistema), obteniendo de nuevo 100 valores de $NTEC_A$ y $NTEC_B$, y el porcentaje de veces en que el sistema (A) es preferible al (B) y viceversa. Los porcentajes ahora representan un estimador de la probabilidad de, en cada caso, tomar la decisión equivocada al decidir que un sistema es mejor que el otro, sobre la base de hacer *cinco* simulaciones.

- Reiterad este proceso, con 10, 50, 100, 500, 1000, 10000 simulaciones. ¿Cómo varían las probabilidades de equivocarse en función del número de simulaciones?

Repetid todo el proceso anterior, pero ahora compararemos los mismos sistemas (A) y (B) pero desde el punto de vista del total de trabajos procesados, $TTP$ (en este caso resultará preferible el sistema que de mayor valor de la medida de rendimiento).

¿Qué conclusiones se pueden extraer?

## 3.2. Intervalos de confianza

- Utilizad alguna de las técnicas de comparación de dos sistemas alternativos mediante intervalos de confianza para decidir si resulta preferible un sistema con 10 máquinas y 2 operarios con tiempos de carga y descarga de 3 y 2 minutos cada uno respectivamente, frente a un sistema con 10 máquinas y 1 operario con tiempos de carga y descarga 1.5 y 1 minutos respectivamente. La medida de rendimiento a utilizar es el número medio de trabajos en espera de carga, y también el tiempo medio de estancia de los trabajos.

## 3.3. Comparación de más de dos sistemas

Utilizad una técnica de elección del mejor de entre un conjunto de $k = 4$ sistemas para decidir cuál de las cuatro configuraciones siguientes es preferible:

- Sistema con 10 máquinas y 1 operario con tiempos de carga y descarga 0.6 y 0.4 minutos.
- Sistema con 10 máquinas y 2 operarios con tiempos de carga y descarga de 1.2 y 0.8 minutos.
- Sistema con 10 máquinas y 5 operarios con tiempos de carga y descarga de 3 y 2 minutos.
- Sistema con 10 máquinas y 10 operarios con tiempos de carga y descarga de 6 y 4 minutos.

La medida de rendimiento para realizar esta comparación es el tiempo medio de estancia de los trabajos.