



Grado en Matemáticas en Informática

**SIMULACIÓN DE SUCESOS
DISCRETOS**

Curso 2017-18

ÍNDICE

1. **Conceptos básicos de la SSD**
2. **SSD de sistemas de espera complejos**
 - **Modelo G/G/1**
 - **Red de colas**
3. **SSD de un modelo de inventario probabilístico**
4. **SSD para conflictos aéreos**
5. **Software de SSD**
6. **Referencias**

2. Conceptos básicos de SSD

Existen distintos tipos de modelos de simulación:

- Los **modelos continuos** tratan con sistemas cuyo comportamiento cambia continuamente de forma con el tiempo (**por ejemplo**, el estudio de la dinámica de la población mundial). Normalmente se representan en la forma de ecuaciones diferenciales o en diferencias que describen las interacciones entre los diferentes elementos del sistema.
- Los **modelos discretos** tratan con sistemas cuyo comportamiento sólo cambia en instantes considerados. Un **ejemplo típico** ocurre en las líneas de espera, donde estamos interesados en la estimación de medidas como el tiempo de espera promedio o la longitud de la cola, las cuales sólo cambian cuando un cliente entra o sale del sistema. En todos los demás momentos no ocurre nada en el sistema desde el punto de vista de la inferencia estadística.

Los momentos en los que ocurren los cambios en el sistema identifican los **eventos** o **sucesos** del modelo. El hecho de que estos sucesos ocurren en instantes discretos da lugar al nombre de *simulación de sucesos discretos*.

Por lo tanto, estudiaremos sistemas que evolucionan en el tiempo en forma discreta → el *estado del sistema* cambia sólo en ciertos instantes del tiempo, no de forma continua. Para su estudio construimos un *modelo* o representación simplificada del sistema real.

En SSD podemos diferenciar tres **tipos de variables**:

- La *variable tiempo* t , que indica el tiempo transcurrido de la simulación.
- Las *variables contadores*, que llevan la cuenta del número de veces que se ha producido un evento.
- Las *variables de estado*, subconjunto mínimo de variables que describe el sistema en un instante particular del tiempo, y que permite predecir su comportamiento futuro. Los valores de las variables de estado en un instante t proporcionan el *estado del sistema* en ese instante.

En general, el estado del sistema podría cambiar como resultado de *actividades internas* o *endógenas*, o bien *actividades externas* o *exógenas*.

En SSD el estado del sistema cambiará cuando se produce un evento o suceso. Los sucesos que definen la evolución se generan en distintos instantes de tiempo y el paso del tiempo se controla mediante un mecanismo de reloj o reloj de la simulación.

Básicamente, hay dos **tipos de mecanismo de reloj**:

- En la *simulación sincrona u orientada a intervalos* el incremento del reloj se fija en una cantidad t , al término de la cual se actualizan las estadísticas y se cambia el estado del sistema. El proceso se repite hasta que se cumpla una condición de finalización, momento en el cual se imprimen los resultados y se concluye la simulación.

Este tipo de control del mecanismo del reloj de simulación tiene como *inconveniente* la selección del incremento de tiempo t :

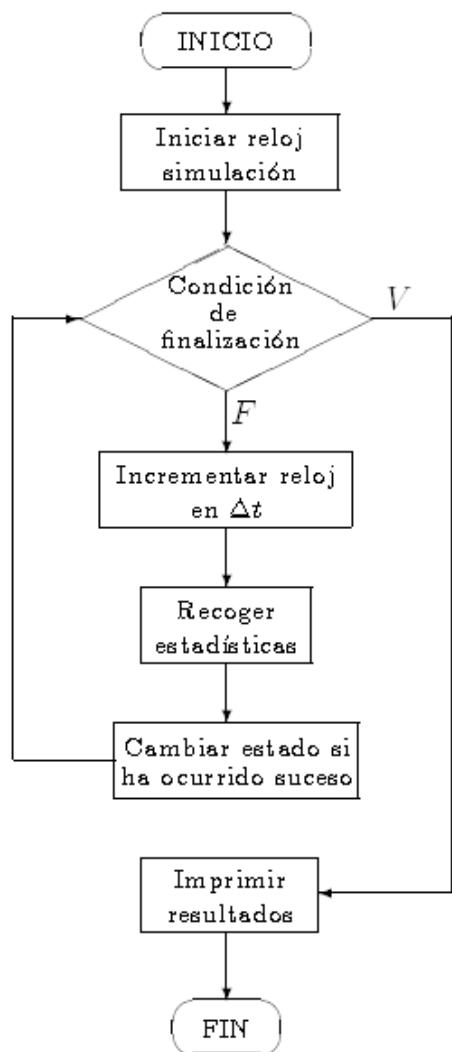
Demasiado grande → podría producirse más de un suceso durante ese intervalo y el orden en que se ha producido no se habría registrado (se asume que han tenido lugar al final del mismo).

Demasiado pequeño → puede que en la mayoría de intervalos de tiempo no ocurra ningún suceso, lo que llevará asociado una inefficiencia computacional.

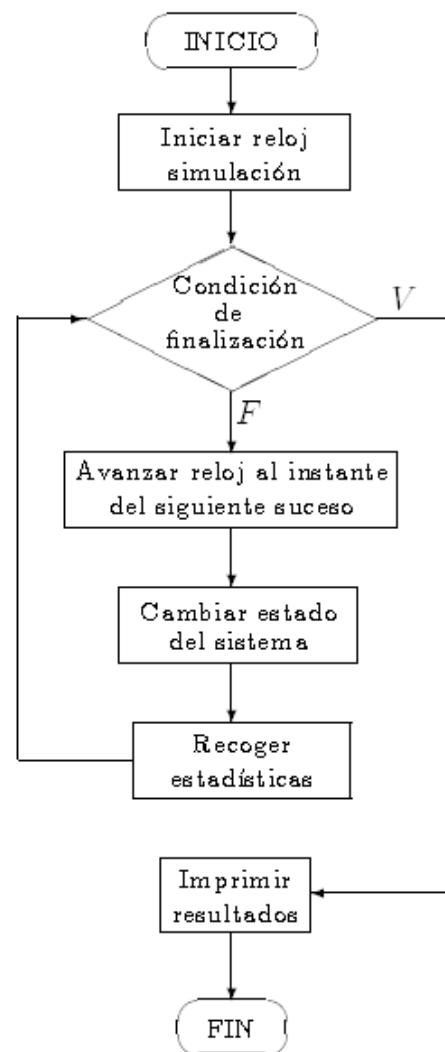
- En la *simulación asíncrona u orientada a sucesos* el reloj de simulación avanza hasta el instante en el que ocurre un nuevo suceso, se actualiza el estado del sistema y se recogen estadísticas. El proceso se repite hasta que su cumpla una condición de finalización, momento en el que se imprimen los resultados.

Debido a los inconvenientes de la simulación síncrona y a pesar de su sencilla implementación, en la SSD utilizaremos la segunda alternativa para el control del mecanismo del reloj de simulación, la simulación asíncrona.

Nota: la simulación síncrona, sin embargo, es una estrategia valiosa para sistemas en tiempo continuo que se estudian, por ejemplo, en Neelamkavil (1987).



Simulación síncrona



Simulación asíncrona

Al depender la evolución de la simulación del sistema de los instantes en los que suceden los eventos, debemos mantener una *lista de éstos y de los instantes en que tendrán lugar*, de forma que el mecanismo de reloj avanzará al instante en el que se produzca el primero de los eventos de la lista.

Dado un tipo de evento determinado, *el tiempo que transcurre entre ocurrencias consecutivas* de dicho evento puede ser *determinístico* o *aleatorio*, en cuyo caso se deberán generar valores a partir de la distribución de probabilidad que describe dicho comportamiento aleatorio.

Banks *et al.* (2000) especialmente orientado a lectores con poca base en cálculo, probabilidad y estadística, cubriendo de forma básica la SSD.

Banks (1998) orientado a lectores interesados en los aspectos más prácticos y no-vedosos de la SSD. Referencias a los investigadores más destacados y su aplicación en campos tan diversos como son el transporte, la sanidad o las aplicaciones militares.

ÍNDICE

1. Conceptos básicos de la SSD
2. **SSD de sistemas de espera complejos**
 - [Modelo G/G/1](#)
 - [Red de colas](#)
3. SSD de un modelo de inventario probabilístico
4. SSD para conflictos aéreos
5. Software de SSD
6. Referencias

3. SSD de sistemas de espera complejos

Una de las posibles aplicaciones de la SSD son los sistemas de espera (colas) complejos, en los cuales:

- es difícil encontrar procedimientos analíticos para modelizar su comportamiento porque se violen las hipótesis de estacionariedad, independencia o ergodicidad.
- se trata de una red de colas muy grande con disciplinas caprichosas, tiempos de llegada o de servicio arbitrarios, capacidad limitada de ciertos nodos y otros con comportamiento no estable...

Son muchos los libros sobre simulación en los que se incluyen ejemplos de simulación de sistemas de espera complejos, como:

Ross (1997), que además del modelo G/G/1 incluye sistemas con dos servidores en serie y en paralelo.

Karian y Dudewicz (1999), en el que se analizan los sistemas con disciplinas de colas con prioridades.

2.1. MODELO G/G/1

Sea un sistema de espera formado por un *único servidor*. Supongamos que los *tiempos entre llegadas* de clientes al sistema se modelizan mediante la variable aleatoria F . Si al llegar los clientes el sistema está vacío entonces pasan directamente a ser servidos, mientras que si ya hay algún cliente recibiendo servicio pasan a formar parte de la *cola*, en la que esperarán a que les llegue su turno.

Los *tiempos de servicio* de los clientes siguen una distribución G , siendo independientes entre sí e independientes de las llegadas de clientes. La cola se supone de *capacidad infinita* (no se rechazan clientes) y *disciplina FIFO* (First In First Out), es decir, el cliente que llegó primero al sistema será servido en primer lugar.

El sistema permite la entrada de clientes en el mismo hasta el instante T , momento en el cual no se permiten más entradas pero sí se sigue dando servicio a los clientes que están en el sistema hasta que todos ellos lo abandonan.

Para realizar la SSD en el sistema propuesto definimos las siguientes variables:

- t \equiv tiempo transcurrido de simulación
- n \equiv nº de clientes en el sistema en el instante t (var. de estado).
- N_{LL} , $N_S \equiv$ nº de llegadas y de salidas hasta el instante t (variables contador).
- $LISTA\{t_{LL}, t_S\} \equiv$ registro con dos elementos en los que guardamos los tiempos en que sucederán los dos tipos de eventos.
- $LL(i) \equiv$ instante en el que llega el cliente i -ésimo al sistema.
- $S(i) \equiv$ instante en el que sale del sistema el cliente i -ésimo.
- $Serv(i) \equiv$ tiempo de servicio recibido por el cliente i -ésimo.

Estas tres últimas variables las utilizaremos para contabilizar los tiempos medios solicitados (en el sistema y en la cola).

- $T \equiv$ horizonte de la simulación.
- $T_p \equiv$ tiempo transcurrido desde T hasta que el último cliente abandona el sistema.

```
int simul_main (float T, float *T_p, float *t_med_sistema, float *t_med_cola) {
    float t = t_suc = 0.0;
    float LISTA.t_LL = LISTA.t_S = 1000;
    int N_LL = N_S = n = 0;
    float LL(i) = S(i) = Serv(i) = 0.0 para todo i
    float X = GenerarDistribucionF();

    if (X > T) {
        T_p = t_med_sistema = t_med_cola = 0.0;
        return -1;
    }
    else {
        rutina_llegada(X);
        while ((LISTA.t_LL != 1000) || (LISTA.t_S != 1000)) {
            if (LISTA.t_LL < LISTA.t_S) {
                t_suc = LISTA.t_LL;
                LISTA.t_LL = 1000;
                rutina_llegada(t_suc);
            }
        }
    }
}
```

```
if (LISTA.tS < LISTA.tLL) {  
    tsuc = LISTA.tS;  
    LISTA.tS = 1000;  
    rutina_servidor(tsuc);  
}  
}  
//tiempo desde T hasta que el último cliente abandona el sistema  
Tp = max (0.0, t-T);  
  
//tiempos medios en la cola y en el sistema  
float acumulo1 = acumulo2 = 0.0;  
for (int ind = 0; ind < NLL; ind++){  
    acumulo1 += S(ind) - LL(ind);  
    acumulo2 += S(ind) - LL(ind) - Serv(ind);  
}  
tmed_sistema = acumulo1 / NLL;  
tmed_colas = acumulo2 / NLL;  
}  
return 0;  
}
```

```
void rutina_llegada (int tsuc) {  
  
    t = tsuc;  
    n++;  
    NLL++;  
    LL(NLL) = t;  
    float X = GenerarDistribucionF();  
  
    if (t + X < T) LISTA.tLL = t + X;  
  
    if (n == 1) {  
        float Y = GenerarDistribucionG();  
        LISTA.tS = t + Y;  
        Serv(NS + 1) = Y;  
    }  
}
```

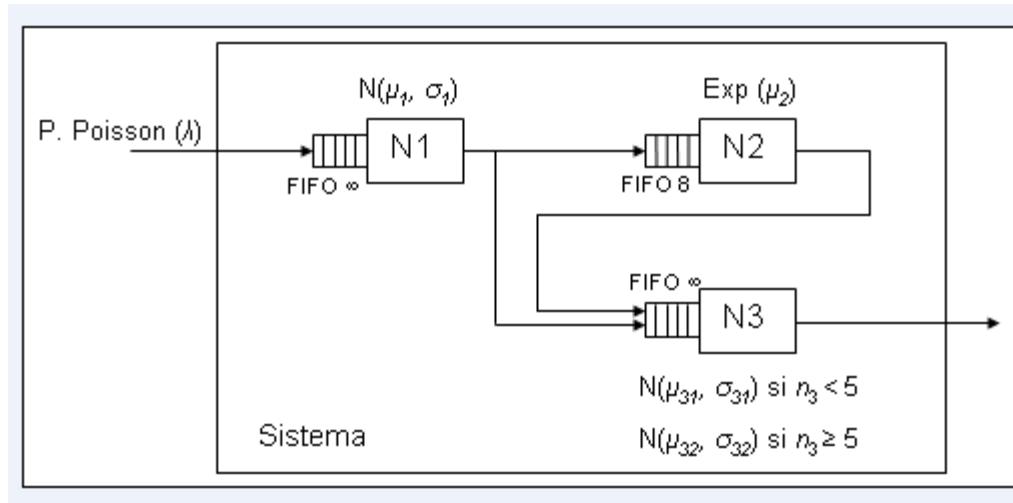
```
void rutina_servidor (int tsuc) {  
  
    t = tsuc;  
    n--;  
    NS++;  
    S(NS) = t;  
  
    if (n > 0) {  
        float Y = GenerarDistribucionG();  
        LISTA.tS = t + Y;  
        Serv(NS) = Y;  
    }  
}
```

ÍNDICE

1. Conceptos básicos de la SSD
2. SSD de sistemas de espera complejos
 - [Modelo G/G/1](#)
 - [Red de colas](#) ←
3. SSD de un modelo de inventario probabilístico
4. SSD para conflictos aéreos
5. Software de SSD
6. Referencias

2.2. RED DE COLAS

Consideremos el siguiente sistema formado por tres nodos:



Se permite la entrada de clientes hasta el instante T , a partir del cual únicamente se sigue dando servicio a los clientes que están en el sistema hasta que todos ellos lo abandonan. Determinar el *tiempo medio que pasan los clientes en el sistema*, el *número medio de clientes en cada nodo* y el *porcentaje de clientes que pasan por el nodo 2*. Obtendremos también el *tiempo transcurrido desde el instante T hasta que el último cliente abandona el sistema*.

Para realizar la SSD en el sistema propuesto definimos las siguientes variables:

- t \equiv tiempo transcurrido de simulación
- $n_1, n_2, n_3 \equiv$ nº de clientes en el nodo 1, 2 y 3 (var. de estado).
- $N_{LL1}, N_{LL2}, N_{LL3} \equiv$ nº de llegadas al primer, segundo y tercer nodo hasta el instante t (variables contador).
- $N_{SI}, N_{S2}, N_{S3} \equiv$ nº de salidas del primer, segundo y tercer nodo hasta el instante t (variables contador).
- $LISTA\{t_{LL1}, t_{S1}, t_{S2}, t_{S3}\} \equiv$ registro con cuatro elementos en los que guardamos los tiempos en que sucederán los cuatro tipos de eventos.
- $LL_j(i), S_j(i) \equiv$ instante en el que llega/sale el cliente i -ésimo al/del servidor j -ésimo, $j=1,2,3$, respectivamente.

Estas seis últimas variables las utilizaremos para contabilizar el tiempo medio que pasan los clientes en el sistema o en los nodos.

- T \equiv horizonte de la simulación.
- $n_med_nj \equiv$ número medio de clientes en el servidor j -ésimo, $j=1,2,3$.
- $T_p \equiv$ tiempo transcurrido desde T hasta que el último cliente abandona el sistema.

```
int simul_main (float T, float λ, float μ1, float σ1, float μ2, float μ31, float σ31, float μ32, float σ32, float *t_med_sistema, float %_clientes_n2, float *Tp, float *n_med_n1, float *n_med_n2, float *n_med_n3){\n\n    float t = tsuc = Tp = 0.0;\n    float LISTA.tLL1 = LISTA.tSI = LISTA.tS2 = LISTA.tS3 = 1000;\n    int NLL1 = NLL2 = NLL3 = NSI = NS2 = NS3 = 0;\n    int n1 = n2 = n3 = 0;\n    float LL1(i) = LL2(i) = LL3(i) = 0.0 para todo i\n    float S1(i) = S2(i) = S3(i) = 0.0 para todo i\n    float n_med_n1 = n_med_n2 = n_med_n3 = 0.0;\n\n    float X = GenerarExponencial(λ);\n\n    if (X > T) {\n        Tp = t_medio_sistema = %_clientes_n2 = 0.0;\n        n_med_n1 = n_med_n2 = n_med_n3 = 0;\n        return -1;\n    }\n}
```

```
else {  
    rutina_llegada_cliente(X);  
    while ((LISTA.tLLI≠1000) || (LISTA.tSI≠1000) || (LISTA.tS2≠1000 )  
        if (min(LISTA) == LISTA.tLLI) {  
            ||(LISTA.tS3≠1000)) {  
                tsuc = LISTA.tLLI; LISTA.tLLI = 1000;  
                rutina_llegada_cliente(tsuc);  
            }  
            if (min(LISTA) == LISTA.tSI) {  
                tsuc = LISTA.tSI; LISTA.tSI = 1000;  
                rutina_servicio_nodo1(tsuc);  
            }  
            if (min(LISTA) == LISTA.tS2) {  
                tsuc = LISTA.tS2; LISTA.tS2 = 1000;  
                rutina_servicio_nodo2(tsuc);  
            }  
            if (min(LISTA) == LISTA.tS3) {  
                tsuc = LISTA.tS3; LISTA.tS3 = 1000;  
                rutina_servicio_nodo3(tsuc);  
            }  
        } // fin del while  
    } //fin del else
```

```
//tiempo desde T hasta que el último cliente abandona el sistema
 $T_p = \max(0.0, t - T);$ 

//porcentaje de clientes que pasa por el segundo nodo
%_clientes_n2 = N_{LL2} / N_{LL1};

//tiempo medio que pasan los clientes en el sistema
float acumulo1, acumulo2, acumulo3 = 0.0;
for (int ind = 0; ind < N_{LL1}; ind++) acumulo1 += S_1(ind) - LL_1(ind);
for (ind = 0; ind < N_{LL2}; ind++) acumulo2 += S_2(ind) - LL_2(ind);
for (ind = 0; ind < N_{LL3}; ind++) acumulo3 += S_3(ind) - LL_3(ind);

t_med_sistema = (acumulo1 / N_{LL1}) + (%_clientes_nodo2 × acumulo2 / N_{LL2})
                + (acumulo3 / N_{LL3});

//número medio de clientes en cada nodo
n_med_n1 = n_med_n1 / t; n_med_n2 = n_med_n2 / t; n_med_n3 = n_med_n3 / t;

return 0;
}
```

```
void rutina_llegada_cliente (int tsuc) {  
  
    nmed_n1 += n1 × (tsuc - t);  
    nmed_n2 += n2 × (tsuc - t);  
    nmed_n3 += n3 × (tsuc - t);  
  
    n1++;  
    NLLI++;  
    LLI(NLLI) = tsuc;  
    t = tsuc;  
    float Y = GenerarExponencial(λ);  
  
    if (t + Y < T) LISTA.tLLI = t + Y;  
    if (n1 == 1) {  
        Y = GenerarNormal(μI, σI);  
        LISTA.tSI = t + Y;  
    }  
}
```

```
void rutina_servicio_nodo1 (int tsuc) {  
  
    nmed_n1 += n1 × (tsuc - t);  
    nmed_n2 += n2 × (tsuc - t);  
    nmed_n3 += n3 × (tsuc - t);  
    t = tsuc;  
    nI--; NSI++;  
    SI(NSI) = tsuc;  
  
    if (n2 < 9) { //el cliente va al segundo nodo  
        n2++; NLL2++;  
        LL2(NLL2) = t;  
        if (n2 == 1) {  
            float Z = GenerarExponencial(μ2);  
            LISTA.tS2 = t + Z;  
        }  
    }  
}  
  
else { //el cliente va al tercer nodo  
    n3++; NLL3++;  
    LL3(NLL3) = t;  
    if (n3 == 1) {  
        float W = GenerarNormal(μ3I, σ3I);  
        LISTA.tS3 = t + W;  
    }  
}  
  
if (nI > 0) {  
    float S = GenerarNormal(μI, σI);  
    LISTA.tSI = t + S;  
}
```

```

void rutina_servicio_nodo2 (int tsuc) {
    nmed_n1 += n1 × (tsuc - t);
    nmed_n2 += n2 × (tsuc - t);
    nmed_n3 += n3 × (tsuc - t);
    t = tsuc;
    n2--; NS22(NS2) = t;
    if (n2 > 0) {
        float Y = GenerarExponencial(μ2);
        LISTA.tS2 = t + Y;
    }
    //trato la entrada del cliente en el tercer nodo
    n3++; NLL33(NLL3) = t;
    if (n3 == 1) {
        float W = GenerarNormal(μ31, σ31);
        LISTA.tS3 = t + W;
    }
}

```

```

void rutina_servicio_nodo3 (int tsuc) {
    float R;
    nmed_n1 += n1 × (tsuc - t);
    nmed_n2 += n2 × (tsuc - t);
    nmed_n3 += n3 × (tsuc - t);
    t = tsuc;
    n3--; NS33(NS3) = t;
    if (n3 > 0) {
        if (n3 < 5) R = GenerarNormal(μ31, σ31);
        else R = GenerarNormal(μ32, σ32);
        LISTA.tS3 = t + R;
    }
}

```

ÍNDICE

1. Conceptos básicos de la SSD
2. SSD de sistemas de espera complejos
 - [Modelo G/G/1](#)
 - [Red de colas](#)
3. **SSD de un modelo de inventario probabilístico** 
4. SSD para conflictos aéreos
5. Software de SSD
6. Referencias

3. SSD de un modelo de inventario probabilístico

Consideremos un almacén de un determinado producto cuyo precio de venta al público es de 2 euros la unidad (r). La **llegada de clientes** al almacén se distribuye según un proceso de Poisson de parámetro $\lambda = 0.5$ clientes por hora y la cantidad de **productos demandados** por cada uno de ellos tiene la siguiente distribución:

Demanda	1 unidad	2 unidades	3 unidades	4 unidades
Probabilidad	0.3	0.4	0.2	0.1

Para satisfacer la demanda de sus clientes el dueño del almacén mantiene un stock de productos, de forma que cuando el nivel del inventario es bajo solicita al distribuidor más unidades.

La **política de pedidos** al distribuidor es por lo tanto aperiódica, es decir, cuando el nivel del inventario es menor que una determinada cantidad ($p=30$) y no hay actualmente ningún pedido pendiente (esperando a que nos llegue), se solicitan unidades del producto para que el nivel del inventario llegue a $P=100$ unidades.

Asociado a cada **pedido** que realizamos al proveedor existe un coste fijo K (coste de preparación) de 10 euros, independientemente de las unidades demandadas. Adicionalmente, el coste por unidad incluida en el pedido depende de la cantidad solicitada, habiendo **descuentos por cantidad**, es decir, si el número de unidades demandadas es menor que 50 ($n_descuent$) el precio es de 1 euro la unidad (p_1), mientras que si se piden más de 50 el precio desciende a 75 céntimos (p_2).

El tiempo que tarda en ser servido el pedido por los proveedores (**tiempo líder**), sigue una distribución normal de media $\mu=48$ horas y desviación típica $\sigma=0.8$, pagándose en ese momento.

Se ha llegado a un acuerdo con el proveedor de forma que, tomando como referencia las 48 horas que tarda en media un pedido en ser servido, por cada 3 horas (lim_penal) de retraso en la entrega del mismo se realizará un descuento del 0.01% del valor del pedido ($%_penal$), encareciéndose en la misma cantidad en el caso de que el pedido llegue con adelanto.

El dueño del almacén debe pagar $h=0.0001$ euros por unidad del producto y unidad de tiempo, asociado al **almacenamiento físico** de los productos (alquiler del local, refrigeración,...).

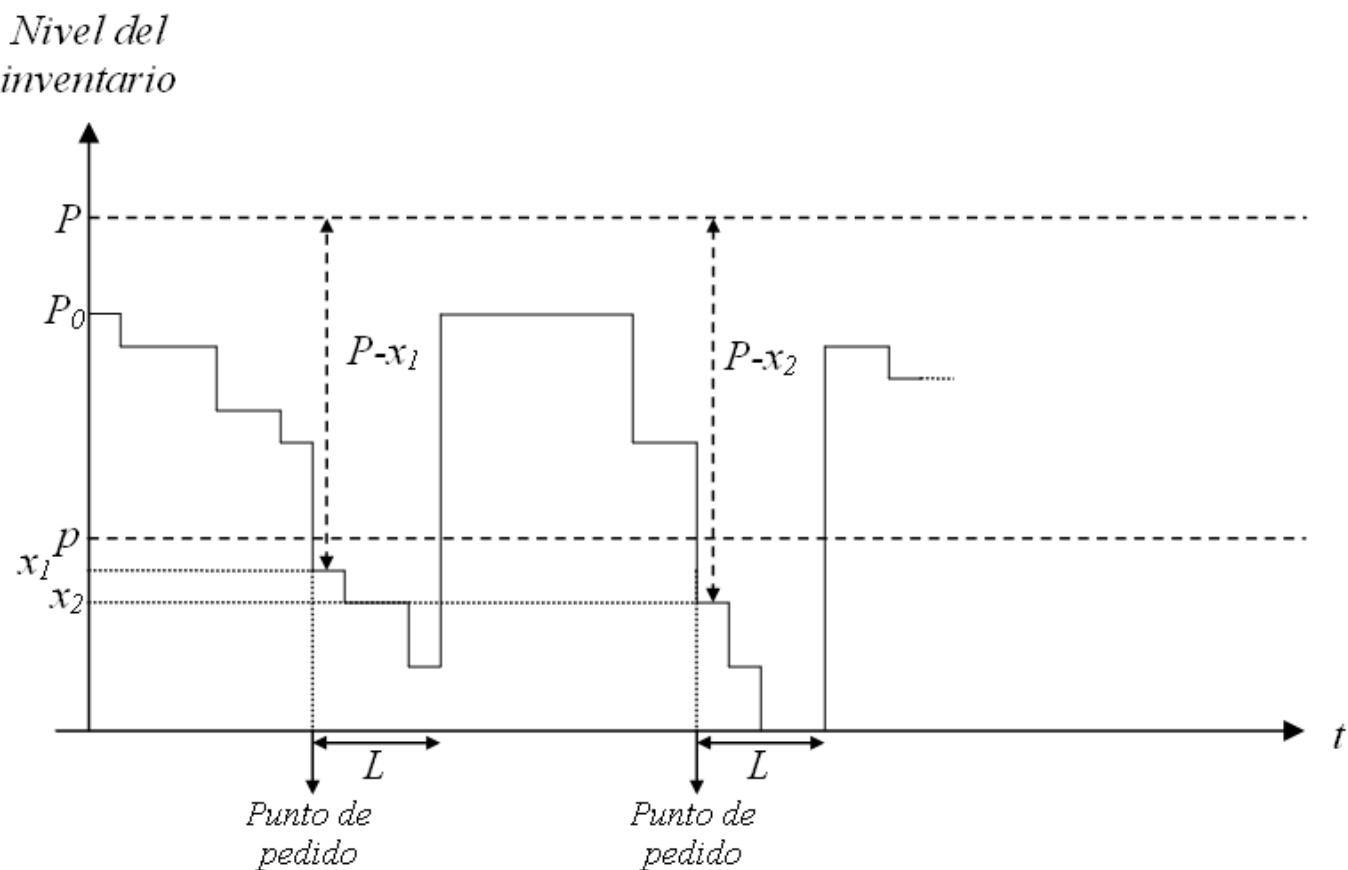
En el caso de que al llegar un cliente éste solicite una cantidad mayor que la que hay en inventario, se le sirve lo que queda, perdiendo la venta restante.

A.Simular el comportamiento de almacén durante un periodo de tiempo de $T=5$ meses para estimar el beneficio esperado, la proporción de clientes cuya demanda se satisface completamente y el porcentaje de tiempo que el nivel del inventario permanece a cero. Para ello, supondremos que el nivel del inventario inicial es de $P_0=70$ unidades del producto.

A.Representar gráficamente la evolución del nivel del inventario durante los 5 meses.

B.Mediante replicaciones de simulación identificar qué cambios se podrían realizar para mejorar el comportamiento del modelo de inventario (mayor beneficio y mayor porcentaje de clientes servidos satisfactoriamente).

Nota. El almacén permanece abierto 8 horas al día de lunes a sábado, descansando el domingo. El tiempo líder se aplica sobre el horario en que está abierto el almacén.



Consideramos las siguientes variables:

- x ≡ número de unidades del producto en el inventario (variable de estado).
- y ≡ n° de unidades incluidas en un pedido que está pendiente de llegar. Si es cero, entonces no hay ningún pedido pendiente (variable de estado).
- t ≡ tiempo transcurrido de simulación.
- $\text{LISTA}\{t_C, t_P\}$ ≡ registro con dos elementos en los que guardamos los tiempos en que sucederán los dos tipos de eventos.

Las **variables contador**, que usaremos para tomar estadísticas y responder a la preguntas sobre el comportamiento del sistema, son:

- C ≡ coste total por pedidos hasta el instante t .
- H ≡ coste total por almacenamiento hasta el instante t .
- R ≡ beneficios por ventas a clientes hasta el instante t .
- N_{C+} ≡ número de clientes servidos satisfactoriamente.
- N_{C-} ≡ número de clientes a los que no hemos podido satisfacer completamente su demanda.
- t_0 ≡ tiempo en el que el nivel del inventario está a cero.

```
simul_main (70, 30, 100,  $5 \times 30 \times 8$  horas, 0.5, (1,2,3,4), (0.3,0.4,0.2,0.1), 2 euros, 0.0001, 48, 0.8, 10 euros, 30, 1 euro, 0.75 euros, 0.001%, 48 horas, 3 horas, *benef, *%_cl_satisf, *%_x_0)
```

```
int simul_main (int  $P_0$ , int  $p$ , int  $P$ , float  $T$ , float  $\lambda$ , int *dem, float *prob, float  $r$ , float  $h$ , float  $\mu$ , float  $\sigma$ , float  $K$ , int  $n\_descuent$ , float  $p_1$ , float  $p_2$ , float %_penal, float  $L_{ref}$  float  $lim\_penal$ , float *benef, float *%_cl_satisfchos, float *%_x_0) {
```

```
    int  $x = P_0$ ;           int  $y = N_{C+} = N_{C-} = 0$ ;  
    float  $C = R = H = 0.0$ ;   float  $LISTA.t_C = LISTA.t_P = 1000$ ;  
    float  $t = t_{suc} = t_0 = 0.0$ ;  benef = %_cl_satisf = %_x_0 = 0.0;  
    float  $var\_auxiliar = 0.0$ ; (instante en que  $x$  se pone a 0)  
    float  $L_{aux} = 0.0$ ; (último  $L$  generado)
```

```
    float  $Z = GenerarExponencial(\lambda)$ ;
```

```
    if ( $Z > T$ ) return -1;
```

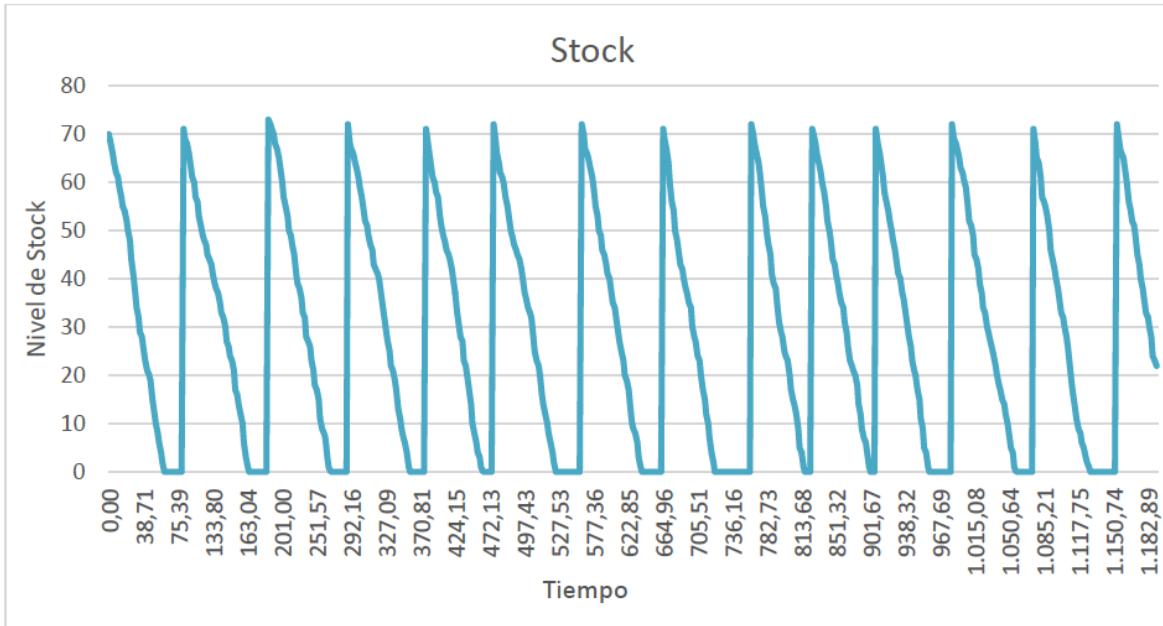
```
else {
    rutina_llegada_cliente(Z);
    while ((LISTA.tP ≠ 1000) || (LISTA.tC ≠ 1000)) {
        if (LISTA.tC < LISTA.tP) {
            tsuc = LISTA.tC; LISTA.tC = 1000;
            rutina_llegada_cliente(tsuc);
        }
        if (LISTA.tP < LISTA.tC) {
            tsuc = LISTA.tP; LISTA.tP = 1000;
            rutina_llegada_pedido(tsuc);
        }
    } //fin del else
} //fin del while

benef = R - C - H ;
%_cl_satisf = NC+ / (NC+ + NC-);
%_x_0 = t0 / t;
return 0;
}
```

```
void rutina_llegada_cliente (int tsuc) {  
  
    H += (tsuc - t) × h × x;    t = tsuc;  
    float demanda = GenerarDiscreta (dem, prob);  
  
    if (demanda ≤ x) {  
        R += demanda × r;    x -= demanda;    NC+++;  
        if (x==0) var_auxiliar = t;  
    }  
    else {  
        R += x × r;    x = 0;    NC-++;  
        if (var_auxiliar == 0) var_auxiliar = t;  
    }  
    if ((x ≤ p) && (y == 0)) {  
        y = P - x;  
        Laux=L = GenerarNormal(μ,σ);    LISTA.tP = t + L;  
    }  
    float Y=GenerarExponencial(λ);  
    if (t + Y < T) LISTA.tC = t + Y;  
}
```

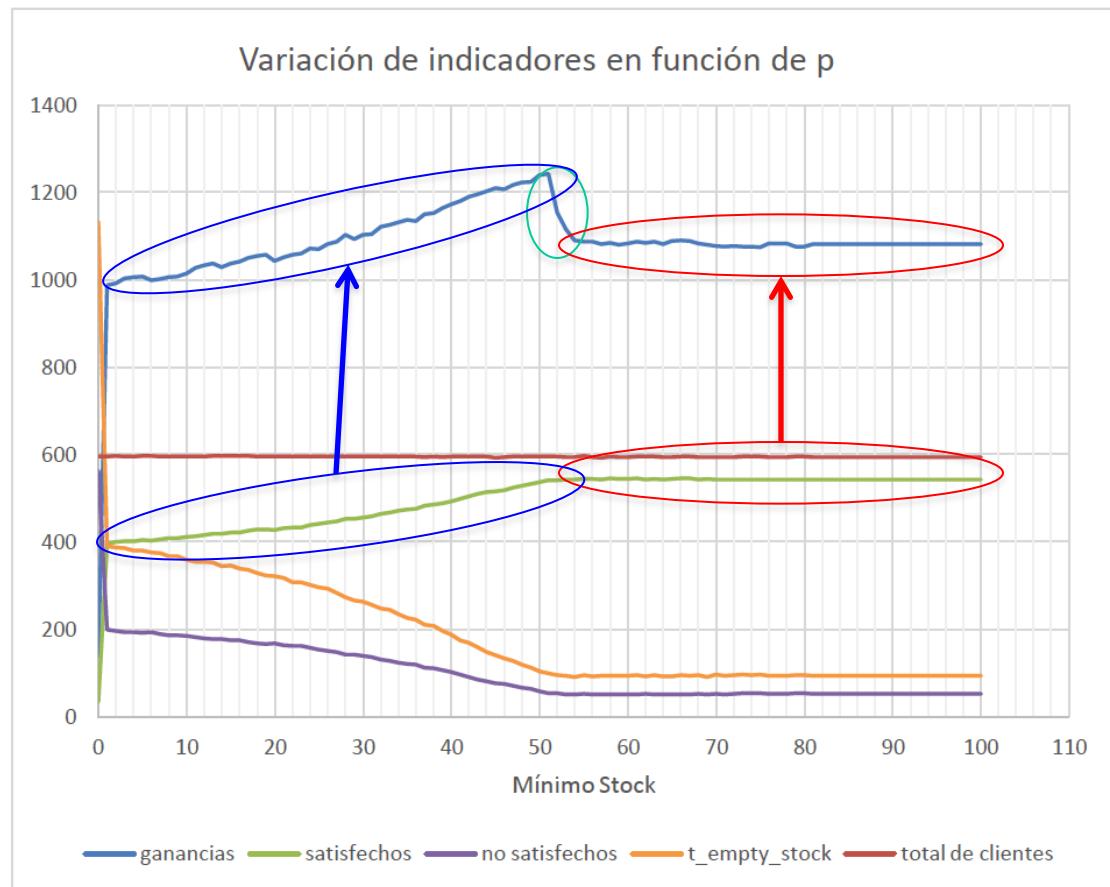
```
void rutina_llegada_pedido (int tsuc) {  
  
    H += (tsuc - t) × h × x;  
    t = tsuc;  
  
    float Cinit=0.0;  
    if (y < num_descuent) Cinit=K + y × p1;  
    else Cinit=K + y × p2;  
  
    if (Laux-Lref ≤ lim_penal) C +=Cinit × (1+%_penal)  
    else C +=Cinit × (1-%_penal)  
  
    x +=y;      y = 0;  
  
    if (var_auxiliar > 0) {  
        t0 +=(t - var_auxiliar);  
        var_auxiliar = 0;  
    }  
}
```

Resultados de la simulación



- Beneficio esperado: 1102.78 €
- Número total de clientes: 595 clientes
- Número de clientes satisfechos: 456 clientes
- Número de clientes insatisfechos: 139 clientes
- Tiempo que el almacén permanece vacío: 263.28 horas
- Ventas perdidas: 289 productos

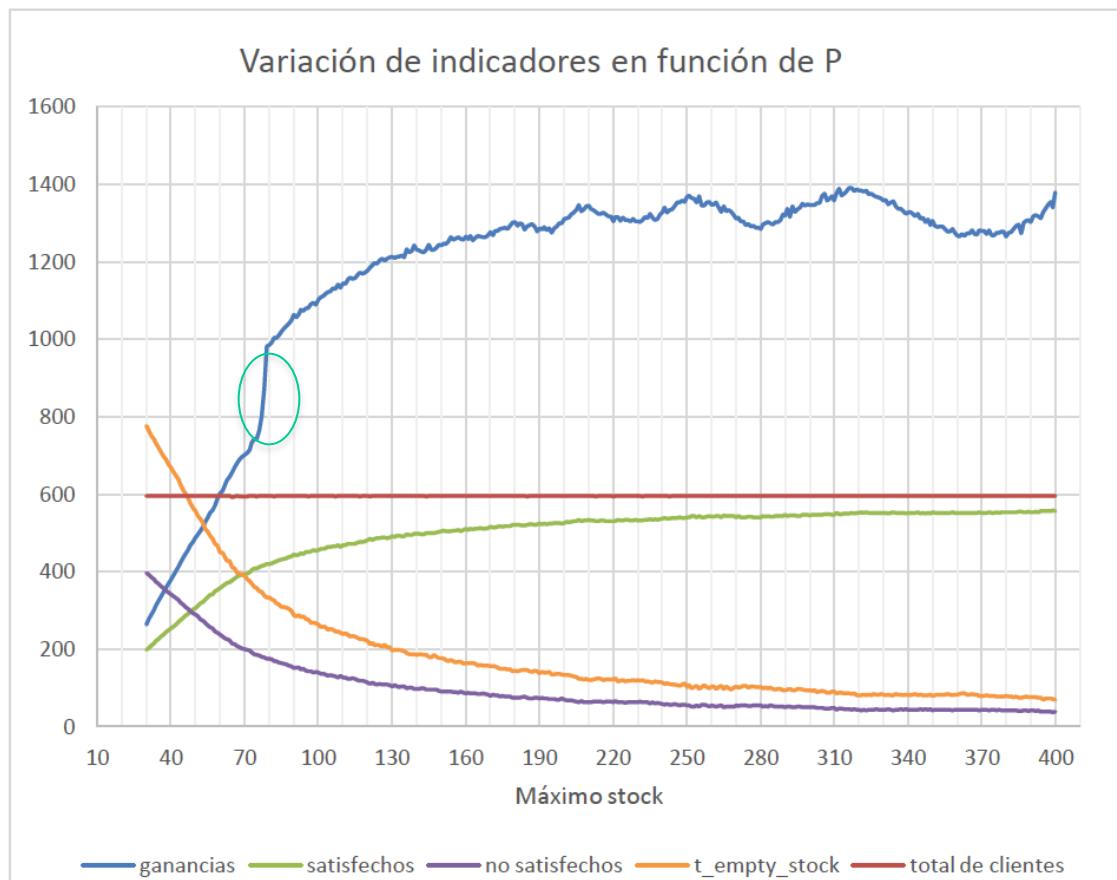
Variación de los indicadores en función de p manteniendo $P=100$



Resultado de la simulación para $p=50$ y $P=100$

- Beneficio esperado: 1239.18 €
- Número total de clientes: 595 clientes
- Número de clientes satisfechos: 537 clientes
- Número de clientes insatisfechos: 58 clientes
- Tiempo que el almacén permanece vacío: 104.04 horas
- Ventas perdidas: 116 productos

Variación de los indicadores en función de P manteniendo $p=30$



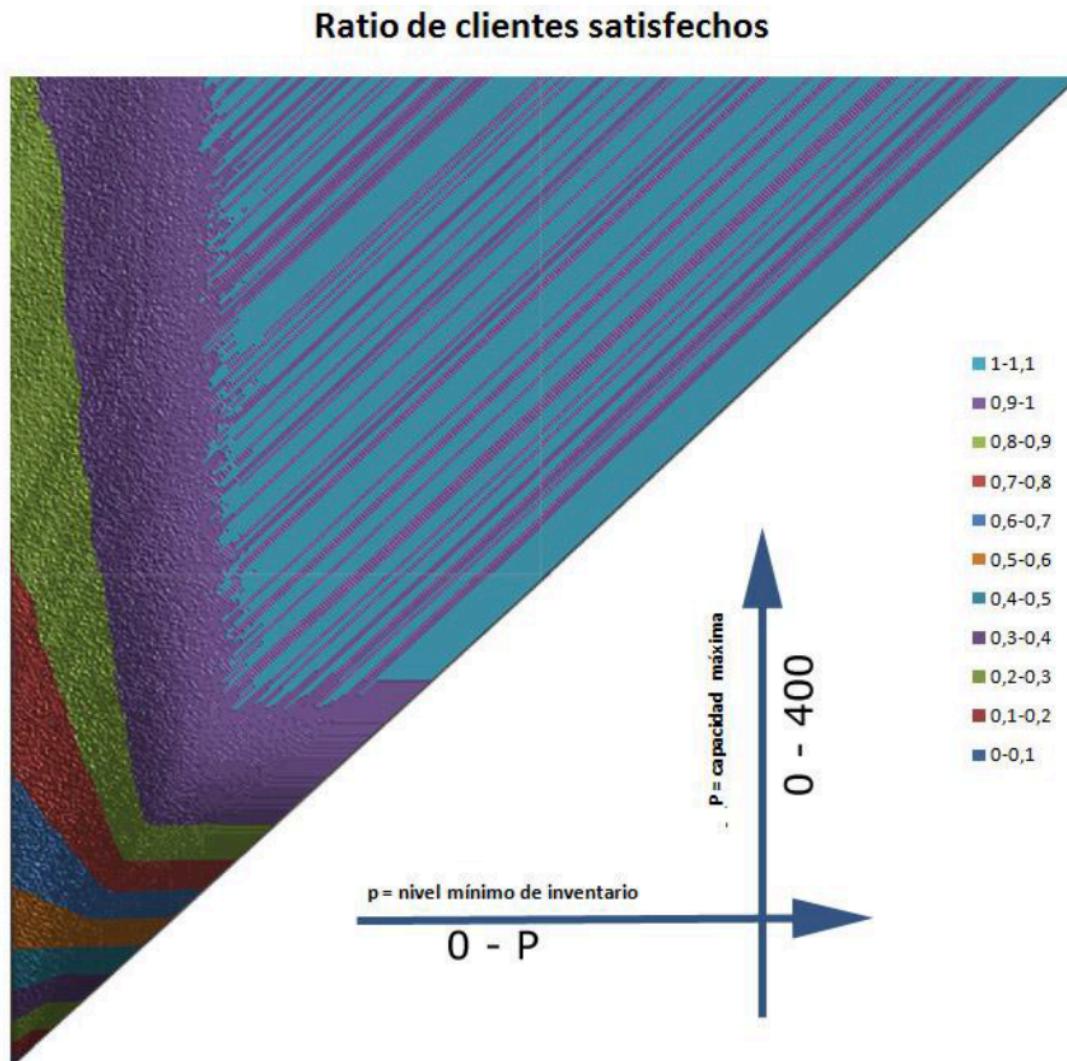
Resultado de la simulación para $p=30$ y $P=317$

- Beneficio esperado: 1390.53 €
- Número total de clientes: 595 clientes
- Número de clientes satisfechos: 551 clientes
- Número de clientes insatisfechos: 44 clientes
- Tiempo que el almacén permanece vacío: 83.46 horas
- Ventas perdidas: 92 productos

12 horas aprox.
Pentium Core 5
4 GB RAM

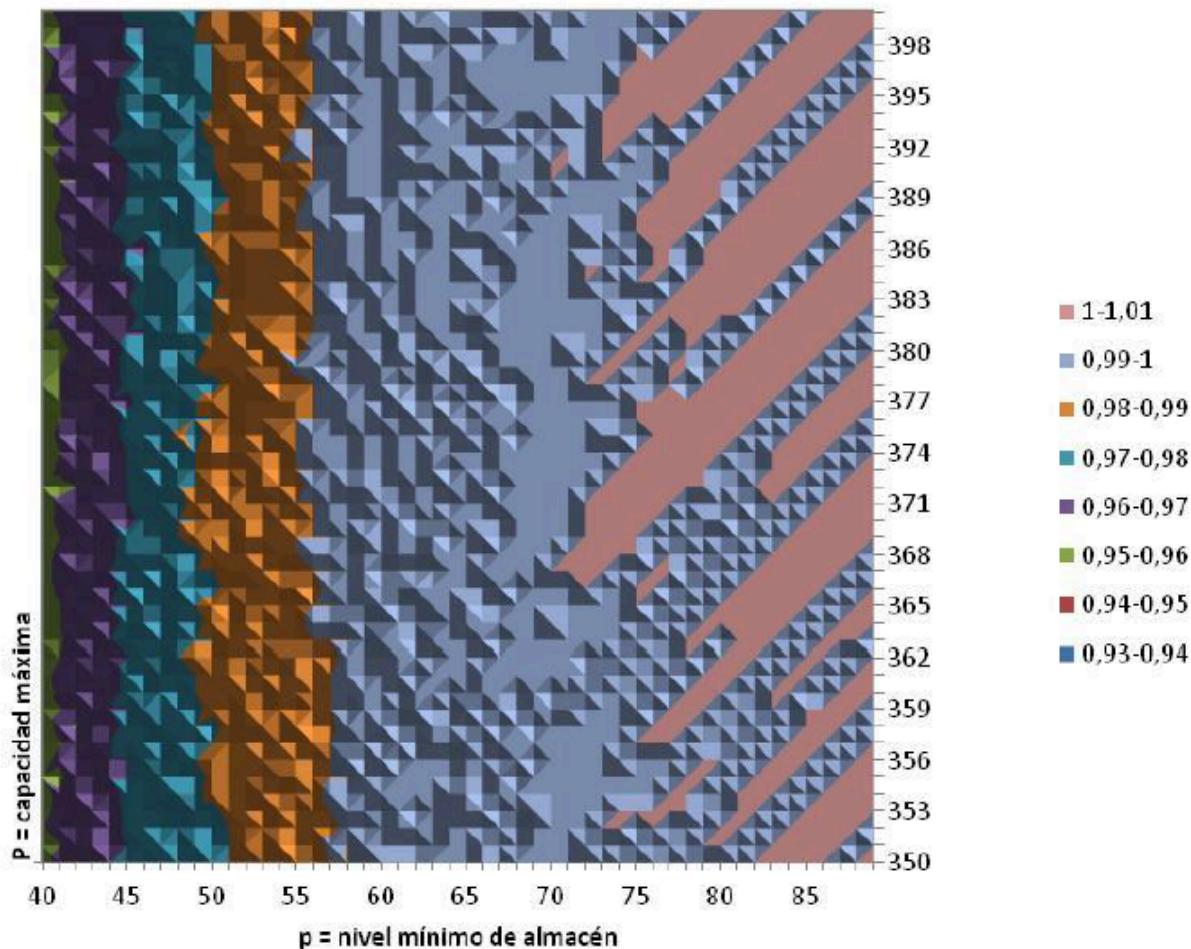


Metaheurísticas?



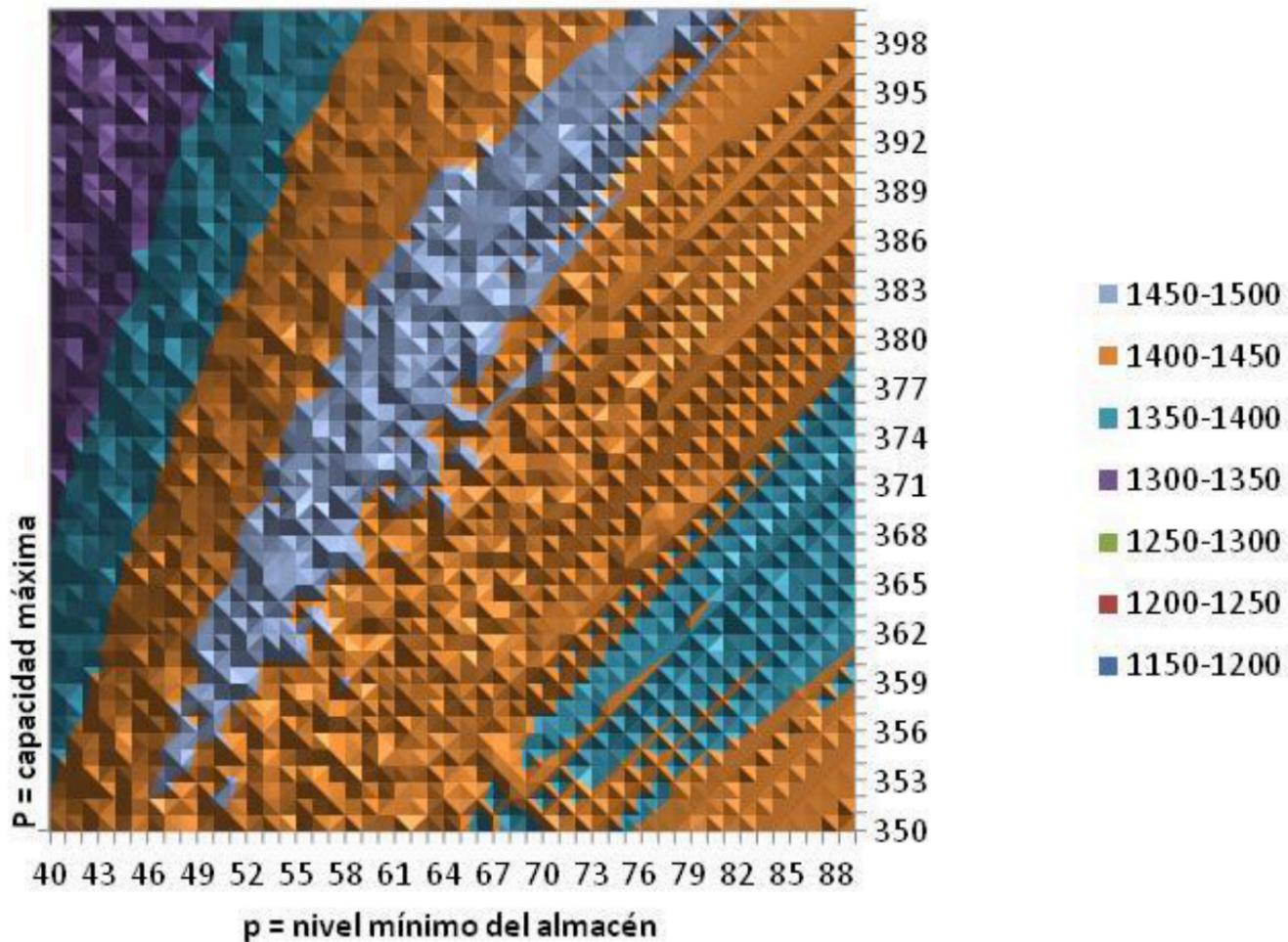
Ratio de clientes satisfechos en función de P (eje vertical) y de p (eje horizontal)

Ratio de clientes satisfechos



Ratio de clientes satisfechos en el entorno del óptimo alcanzado

Beneficio esperado (€)



Beneficio esperado en el entorno del óptimo alcanzado

- Beneficio esperado: 1468.43 €
- Número total de clientes: 594 clientes
- Número de clientes satisfechos: 594 clientes
- Número de clientes insatisfechos: 0 clientes
- Tiempo que el almacén permanece vacío: 0.55 horas
- Ventas perdidas: 0 productos

Resultado de la simulación para $p = 73$ y $P = 390$.

- Beneficio esperado: 1476.07
- Número total de clientes: 595 clientes
- Número de clientes satisfechos: 590 clientes
- Número de clientes insatisfechos: 5 clientes
- Tiempo que el almacén permanece vacío: 7.60 horas
- Ventas perdidas: 9 productos

Resultado de la simulación para $p = 58$ y $P = 370$.

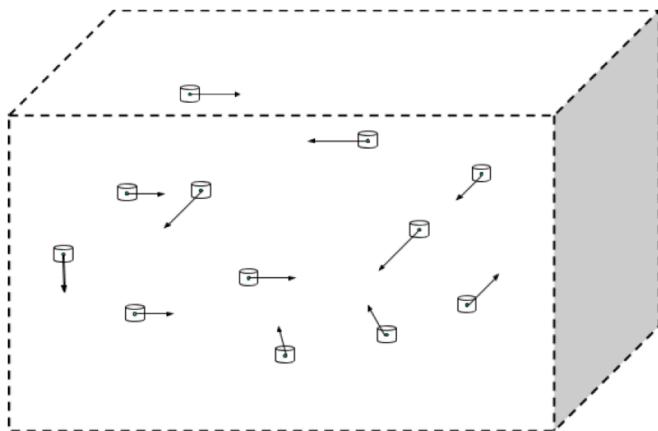
ÍNDICE

1. Conceptos básicos de la SSD
2. SSD de sistemas de espera complejos
 - Modelo G/G/1
 - Red de colas
3. SSD de un modelo de inventario probabilístico
4. **SSD para conflictos aéreos**
5. Software de SSD
6. Referencias



Descripción del problema

Sector aéreo



Controladores aéreos:

- Planificador
- Ejecutivo

Volumen de seguridad

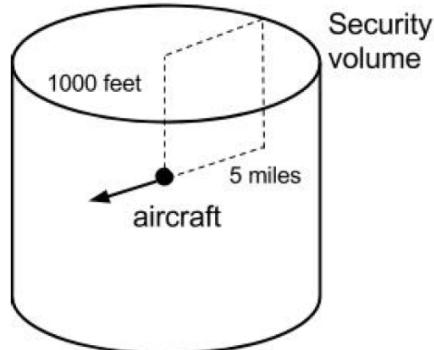


Fig. 2. Security volume of an aircraft

Parámetros

Características del avión

$v_{min,i}$: minimum velocity for the aircraft i ,

$v_{max,i}$: maximum velocity for the aircraft i ,

$z_{min,i}$: minimum altitude for the aircraft i ,

$z_{max,i}$: maximum altitude for the aircraft i ,

$\beta_{max,i}$: maximum variation of the angle for the aircraft i .

Entrada al sector

$v_{ini,i}$: velocity of the aircraft i upon entry,

$z_{ini,i}$: altitude of the aircraft i upon entry,

$x_{ini,i}$: abscise of the aircraft i upon entry,

$y_{ini,i}$: ordinate of the aircraft i upon entry, and

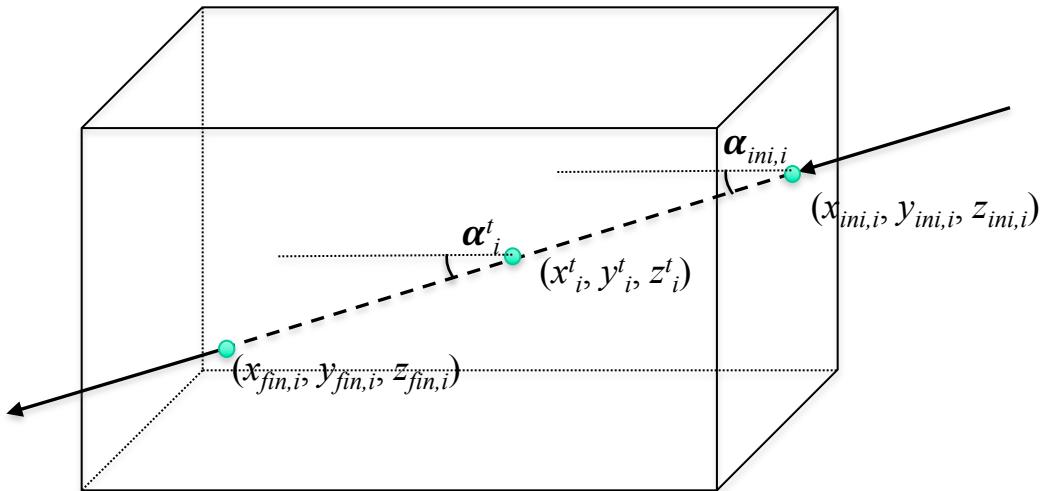
$\alpha_{ini,i}$: angle of the aircraft i with respect to the horizontal upon entry.

Salida óptima del sector

$x_{fin,i}$: estimated abscise of the aircraft i when leaving the aerial sector,

$y_{fin,i}$: estimated ordinate of the aircraft i when leaving the aerial sector.

$z_{fin,i}$: estimated z-axis of the aircraft i when leaving the aerial sector.



Estado de las aeronaves en el instante t

v_i^t : velocity of the aircraft i at the time t ,

z_i^t : aircraft i altitude at the time t ,

x_i^t : aircraft i abscise at the time t ,

y_i^t : aircraft i ordinate at the time t ,

t_i : time necessary to arrive to the bound of the aerial sector with a constant velocity v_i^t ,

α_i^t : angle with respect to the horizontal of the aircraft i at the instant t , and

man_i^t : number of maneuvers performed by the aircraft i at the time t since it entered in the aerial sector.

Riesgo de colisión

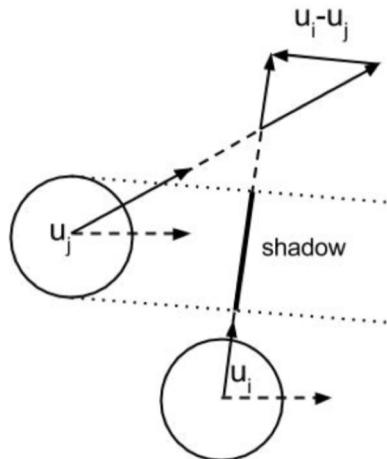


Fig. 3. Detecting conflict situations

$$\vec{u}_i = (v_{new,i} \times \cos(\alpha_{new,i}), v_{new,i} \times \sin(\alpha_{new,i}))$$

$$\vec{u}_j = (v_{new,j} \times \cos(\alpha_{new,j}), v_{new,j} \times \sin(\alpha_{new,j}))$$

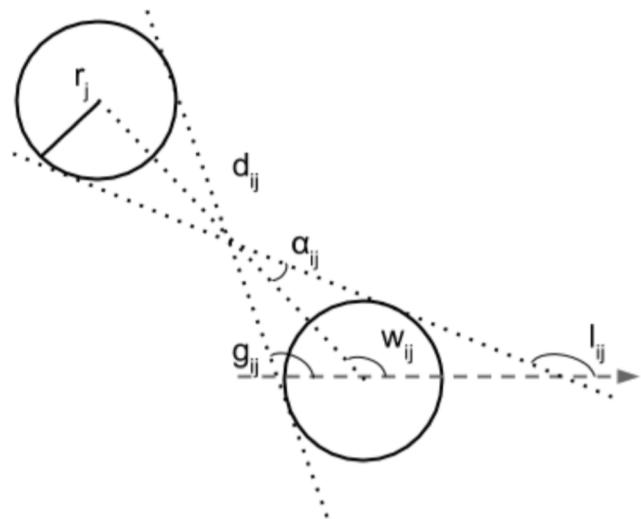


Fig. 4. Main angles and distances

No hay conflicto si

$$\frac{v_{new,i} \times \sin(\alpha_{new,i}) - v_{new,j} \times \sin(\alpha_{new,j})}{v_{new,i} \times \cos(\alpha_{new,i}) - v_{new,j} \times \cos(\alpha_{new,j})} \geq \tan(l_{ij}),$$

$$\frac{v_{new,i} \times \sin(\alpha_{new,i}) - v_{new,j} \times \sin(\alpha_{new,j})}{v_{new,i} \times \cos(\alpha_{new,i}) - v_{new,j} \times \cos(\alpha_{new,j})} \leq \tan(g_{ij}).$$

Funciones objetivo

Minimizar la magnitud de las maniobras

$$\min f_1 = a_1 + \frac{w_1}{n} \sum_{i=1}^n |a_1 - |q_i||,$$

$$\text{with } a_1 = \frac{1}{n} \sum_{i=1}^n |q_i|$$

Minimizar los riesgos de colisiones

horizontal collision risk ($r_{hor,ij}$)

$$r_{hor,ij} = \min\{\gamma_{ij}, \delta_{ij}\}$$

$$\gamma_{ij} = \tan(l_{ij}) - \frac{v_{new,i} \times \sin(\alpha_{new,i}) - v_{new,j} \times \sin(\alpha_{new,j})}{v_{new,i} \times \cos(\alpha_{new,i}) - v_{new,j} \times \cos(\alpha_{new,j})},$$

$$\delta_{ij} = \frac{v_{new,i} \times \sin(\alpha_{new,i}) - v_{new,j} \times \sin(\alpha_{new,j})}{v_{new,i} \times \cos(\alpha_{new,i}) - v_{new,j} \times \cos(\alpha_{new,j})} - \tan(g_{ij})$$

vertical collision risk ($r_{ver,ij}$)

$$r_{ver,ij} = \begin{cases} \max\left\{z_{new,j} - z_{new,i} + s_{ver,j}, z_{new,j} - z_{new,i} + s_{ver,i}\right\}, & \text{if } z_{new,i} > z_{new,j} \\ \max\left\{z_{new,i} - z_{new,j} + s_{ver,i}, z_{new,i} - z_{new,j} + s_{ver,j}\right\}, & \text{if } z_{new,i} \leq z_{new,j} \end{cases}$$

$$z_{new,i} = \begin{cases} 0, & \text{if } ac_i = 0 \\ z_i + q_i(z_i - z_{min,i}), & \text{if } ac_i = 1 \wedge q_i \leq 0 \\ z_i + q_i(z_{max,i} - z_i), & \text{if } ac_i = 1 \wedge q_i > 0 \end{cases}$$

Funciones objetivo

Minimizar el número de maniobras

$$\min f_3 = \sum_{i=1}^n (vc_i + ac_i + tc_i),$$

with $vc_i + ac_i + tc_i \leq 1 \forall i$

Minimizar distancias puntos óptimos de salida

$$\min f_5 = a_5 + \frac{w_5}{n} \sum_{i=1}^n |a_5 - d_i|$$

with $a_5 = \frac{1}{n} \sum_{i=1}^n d_i$,

$$d_i = \sqrt{(x_{fin,i} - x_{fin,new,i})^2 + (y_{fin,i} - y_{fin,new,i})^2 + (z_{fin,i} - z_{fin,new,i})^2}$$

$$x_{fin,new,i} = \text{intersection}(x_i, \alpha_{new,i})$$

$$y_{fin,new,i} = \text{intersection}(y_i, \alpha_{new,i})$$

$$z_{fin,new,i} = \text{intersection}(z_i, \alpha_{new,i})$$

Minimizar retrasos

$$\min f_4 = a_4 + \frac{w_4}{n} \sum_{i=1}^n |a_4 - |t_{new,i} - t_i||,$$

with $a_4 = \frac{1}{n} \sum_{i=1}^n |t_{new,i} - t_i|$ and

$$t_{new,i} = \frac{\sqrt{(x_{fin,i} - x_i)^2 + (y_{fin,i} - y_i)^2}}{v_{new,i}}$$

Minimizar la dispersión de las maniobras

$$\min f_6 = \frac{1}{n} \sum_{i=1}^n |a_6 - man_{new,i}|,$$

with $a_6 = (\sum_{i=1}^n man_{new,i})/n$

$$man_{new,i} = \begin{cases} man_i, & \text{if } vc_i = 0 \wedge ac_i = 0 \wedge tc_i = 0 \\ man_i + 1, & \text{if } vc_i = 1 \vee ac_i = 1 \vee tc_i = 1 \end{cases}$$

Técnicas de solución: Recocido simulado

Emula la forma en que se enfrian los materiales para alcanzar un estado de máxima entropía.



```
solución = crearSoluciónInicial();
eval = evaluar(solución);
t = establecerTemperatura();
while convergencia() == False and its < maxIts do
    soluciónGenerada = generarSolución(solución);
    evalGenerada = evaluar(soluciónGenerada);
    if evalGenerada < eval then
        | solución = soluciónGenerada;
        | eval = evalGenerada;
    else
        | [solución,eval] =
        | elegirProbabilísticamente(solucion,solucionGenerada,eval,evalGenerada,t);
    end
    actualizarTemperatura(t);
end
```

Técnicas de solución: Nubes de partículas

Se inspira en el comportamiento social de las bandadas de pájaros o bancos de peces que evolucionan teniendo en cuenta la mejor solución encontrada en su recorrido y al líder.



El algoritmo emplea una población de soluciones (partículas) que incluyen unas a otras para recorrer el espacio de búsqueda. En concreto, se basan en la velocidad actual de la partícula, su mejor solución encontrada hasta el momento y la mejor solución encontrada por el enjambre en su totalidad.

Simulación de sucesos discretos

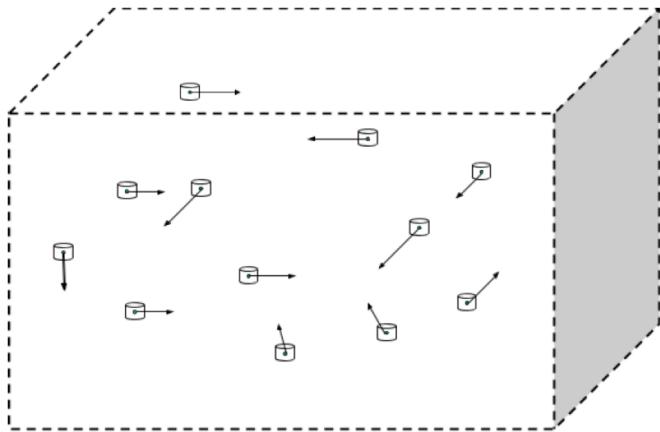


Fig. 1. *Aerial sector with several aircrafts*

Eventos:

- **Entrada de una avión en el sector aéreo →**
 - Guardar datos sobre el punto e instante de entrada y del óptimo de salida (si no realiza maniobras).
 - Lanzar nuestro algoritmo para identificar la/s maniobras a realizar por los aviones.
- **Salida de un avión del sector aéreo →**
 - Guardar datos sobre el punto e instante de salida y número y magnitud de maniobras realizadas desde que entró al sector.

ÍNDICE

1. Conceptos básicos de la SSD
2. SSD de sistemas de espera complejos
 - [Modelo G/G/1](#)
 - [Red de colas](#)
3. SSD de un modelo de inventario probabilístico
4. SSD para conflictos aéreos
5. [Software de SSD](#) 
6. Referencias

5. Software de SSD

La implementación de modelos complejos con varios tipos de sucesos y procesos y numerosas interacciones puede resultar complicada → se han desarrollado numerosos *productos de software específicos de la simulación*.

En cierto sentido son comparables a lenguajes generales como C, Pascal..., pero cuentan además con *capacidades o herramientas específicas* que facilitan el proceso de modelización. Esencialmente, tales capacidades evitan al analista crear herramientas y procedimientos que aparecen en casi todas las aplicaciones de simulación. Entre estas herramientas específicas suelen estar:

- Un marco general para la *creación y descripción de un modelo* en términos de procesos, sucesos, entidades, atributos, recursos e interacciones entre componentes del modelo.
- La inclusión de un *mecanismo de reloj de simulación*.
- La inclusión de *métodos para secuenciar la ocurrencia de sucesos*.

5. Software de SSD

La implementación de modelos complejos con varios tipos de sucesos y procesos y numerosas interacciones puede resultar complicada → se han desarrollado numerosos *productos de software específicos de la simulación*.

En cierto sentido son comparables a lenguajes generales como C, Pascal..., pero cuentan además con *capacidades o herramientas específicas* que facilitan el proceso de modelización. Esencialmente, tales capacidades evitan al analista crear herramientas y procedimientos que aparecen en casi todas las aplicaciones de simulación. Entre estas herramientas específicas suelen estar:

- Un marco general para la *creación y descripción de un modelo* en términos de procesos, sucesos, entidades, atributos, recursos e interacciones entre componentes del modelo.
- La inclusión de un *mecanismo de reloj de simulación*.
- La inclusión de *métodos para secuenciar la ocurrencia de sucesos*.

- *Generadores de números aleatorios* y de las principales variables aleatorias.
- *Herramientas para la recolección y el análisis de estadísticas* relativas al uso de recursos y otras entidades.
- Herramientas para la *descripción gráfica del modelo y de la simulación*.
- Herramientas para la *validación y la verificación del modelo*.

Los lenguajes de simulación presentan como **inconvenientes** que suelen ser caros, no están disponibles para todos los sistemas operativos y requieren mayor tiempo de ejecución que los lenguajes generales.

Los lenguajes de simulación discreta disponibles entran en dos amplias categorías:

- **Programación de eventos.** El usuario detalla las acciones asociadas con la ocurrencia de cada evento. La contribución principal del lenguaje es automatizar el proceso de muestreo a partir de las distribuciones, el almacenamiento cronológico y recuperación de eventos, así como la recopilación de estadísticas del modelo.

GASP (Pritsker, 1974; Rimvall y Cellier, 1982), **SLAM** (Pritsker y O'Reilly, 1999), **SIMAN** (Pegden *et al.* 1995) y **SIMSCRIPT** (Fayek, 1988).

- **Orientados a procesos.** Usan bloques o nodos que se pueden unir para formar una red que describa los movimientos, transacciones o entidades en el sistema. Se conducen internamente por las mismas acciones que se usan en los lenguajes de programación de eventos. La diferencia es que esas acciones están automatizadas para liberar al usuario de los tediosos detalles de cálculo.

Están basados en el concepto de entrada-salida del enfoque de "caja negra", proporcionando de esta forma una mayor simplicidad y facilidad de uso a costa de una menor flexibilidad.

GPSS (Solomon, 1983; Karian y Dudewicz; 1999) fue el primero que se creó (en 1960 por IBM), está específicamente diseñado para la *simulación de sistemas de espera*, y requiere que el usuario domine el "trabajo interno" de aproximadamente 80 bloques diferentes.

Otros lenguajes que implementan esta estrategia son:

SIMULA (Pooley, 1987; Holmevik, 1994)

(<http://www.engin.umd.umich.edu/CIS/course.des/cis400/simula/simula.html>)

SIMNET II (Taha, 1991)

(<http://web.ineg.uark.edu/simnet2/simnet.htm>).

Otros lenguajes, como **SIMSCRIPT II.5** (Rusell, 1999) (<http://www.simscrip.com>) o **MODSIM** (Belanger, 1993) (<http://www.caci.com>), permiten modelizar con ambas estrategias.

Los **simuladores** son paquetes de software de sencillo manejo que permiten desarrollar modelos para *situaciones específicas*. Obviamente su uso es más fácil y permiten un desarrollo rápido de modelos, pero su escasa generalidad limita su interés como herramientas generales. En la *revista OR/MS Today* se realiza bienalmente un estudio del software en el mercado para SSD (último, 2013).

También se han desarrollado simuladores de propósito general. Dos muy conocidos y utilizados, que permiten componentes discretos y continuos simultáneamente, son

- **Extend Suite** (Image, 2006) (http://www.imaginethatinc.com/prods_suite.html), que permite una descripción gráfica del modelo para facilitar la implementación mediante la importación de bloques de librerías, conectados para indicar el flujo de entidades a través del sistema y detallados mediante el uso cajas de diálogo; y
- **Arena** (Kelton et al., 2004) (<http://www.arenasimulation.com/>).

ÍNDICE

1. Conceptos básicos de la SSD
2. SSD de sistemas de espera complejos
 - [Modelo G/G/1](#)
 - [Red de colas](#)
3. SSD de un modelo de inventario probabilístico
4. SSD para conflictos aéreos
5. Software de SSD
6. [Referencias](#) ←

6. Referencias

- Banks, J. (1998). [Handbook of Simulation: Principles, Methodology, Application and Practice](#), Wiley.
- Banks, J., Nelson, B. y Carson, J. (2000). [Discrete-Event System Simulation](#), Prentice Hall.
- Belanger, R. (1993). [MODSIM II: The High-Level Object-Oriented Language](#), CACI Products Company.
- Bryant, R.E. (1977) [Simulation of Packet Communications Architecture Computer Systems](#), MIT-LCS-TR-188, Massachusetts Institute of Technology.
- Chandy, K.M. y Misra, J (1979) Distributed Simulation: a Case Study in Design and Verification of Distributed Programs, [IEEE Trans. Software Engineering](#) 5 (5), 440-452.
- Fayek, A.M. (1988). [Introduction to Combined Discrete-Continuous Simulation Using PC SIMSCRIPT II.5](#), CACI Products Company.
- Fujimoto, R.M. (1990). Parallel Discrete Event Simulation, [Communications of the ACM](#) 33 (10), 30-53.

Holmevik (1994). Compiling SIMULA: A Historical Study of Technological Genesis, [Annals of the History of Computing](#) **16** (4), 25-37.

Imagine That, Inc. (2006) [Extend User's Manual](#), versión 7.

Jefferson, D.R. (1985) Virtual Time, [ACM Trans. Programming Languages and Systems](#) **7** (3), 404-425.

Karian, Z.A. y Dudewicz, E.J. (1999). [Modern Statistical, Systems and GPSS Simulation](#), 2^a Ed., CRC Press.

Kelton, W.D., Sadowski, R.P. y Sturrock, D.T. (2004) [Simulation with Arena](#), McGraw-Hill.

Lin Y.-B. y Fishwick, A. (1995). Asynchronous Parallel Discrete Event Simulation, [IEEE Trans. Systems, Man and Cybernetics](#), vol. XX.

Misra, J. (1986). Distributed Discrete-Event Simulation, [Computing Surveys](#) **18** (1), 40-65.

Neelamkavil, F. (1987). [Computer Simulation and Modelling](#), Wiley.

Pegden, C.D., Shannon, R.E. y Sadowski, R.P. (1995). [Introduction to Simulation Using SIMAN](#), McGraw-Hill Inc.

- Pooley, R.J. (1987). [An Introduction to Programming in SIMULA](#), Blackwell Scientific Publications.
- Pritsker, A.A. (1974). [The GASP-V Simulation Language](#), Wiley.
- Pritsker, A.A. y O'Reilly, J.J. (1999). [Simulation with Visual SLAM and AweSim](#), Wiley.
- Rimvall, C.M. y Cellier, F.E. (1982). The GASP-VI Simulation Package for Process-Oriented Combined Continuous and Discrete System Simulation, [Proceedings of the 10th IMACS World Congress on Simulation and Scientific Computation](#), 413-416.
- Ross, S. (1997). [Simulation](#), Academic Press.
- Russell, E.C. (1999). [Building Simulation Models with SIMSCRIPT II.5](#), CACI Products Company.
- Solomon, S.L. (1983). [Simulation of Waiting-Line Systems](#), Prentice Hall.
- Taha, H.A. (1991). Simulation with SIMNET II, [Proceedings of the 23rd Conference on Winter Simulation](#), 132-135.