# TRABAJO TEMA 6

LUIS ALBERTO ÁLVAREZ ZAVALETA

## Modelado 1: Flight Crew Assignment

Para poder modelar el problema de la asignación de viajes hemos representado cada uno de las tripulaciones y los viajes que reciben en una matriz de unos y ceros según los datos descritos en el problema, además del precio que tiene cada esta tripulación

```
int[][] viajesCrew = {
        {1,0,0,1,0,0,1,0,0,1,0,0},
        {0,1,0,0,1,0,0,1,0,0,1,0},
        {0,0,1,0,0,1,0,0,1,0,0,1},
        {0,0,0,1,0,0,1,0,1,1,0,1},
        {1,0,0,0,0,1,0,0,0,1,1,0},
        {0,0,0,1,1,0,0,0,1,0,0,0},
        {0,0,0,0,0,0,1,1,0,1,1,1},
        {0,1,0,1,1,0,0,0,1,0,0,0},
        {0,0,0,0,1,0,0,1,0,0,1,0},
        {0,0,1,0,0,0,1,1,0,0,0,1},
        {0,0,1,0,0,0,1,1,0,0,0,1},
        {0,0,0,0,0,1,0,0,1,1,1,1}};

int [] precioCrew= {2,3,4,6,7,5,7,8,9,9,8,9};

int numCrew = viajesCrew.length;
int numVuelos = viajesCrew[0].length;
```

### Variables

Para resolver el problema usaremos 2 variables

```
IntVar[] crewPlan = new IntVar[numCrew];

IntVar price;

for(int q = 0; q < numCrew; q++)
{
    int crew=q+1;
    crewPlan[q] = model.intVar("Crew_"+crew,0,1);
}

price = model.intVar("totalprice",0, Arrays.stream(precioCrew).sum());
```

cewPlan representa un array de 0 y 1 donde un 1 representa que a la tripulación participará en el plan y 0 que no serán incluidos en el plan final.

Se usa la variable precio para calcular el precio total de la tripulación y evaluar su calidad, en el peor de los casos se necesitará todas las tripulaciones disponibles por eso la variable tendrá de rango desde 0 hasta la suma de todos los elementos de precioCrew, lo que representaría el precio de contratar a todas las tripulaciones.

## Restricciones

Hemos utilizado 3 restricciones para modelar el problema, la primera de ellas es la que obligará que para el plan seleccionado cada vuelo tenga al menos 1 pasajero. Esto se consigue haciendo el producto escalar entre el array que contiene el plan seleccionado y la fila que representa el vuelo, esto debe ser siempre mayor o igual a 1 dado que se desea que al menos haya una tripulación asignada a cada vuelo que se tiene

```
for(int i  = 0; i < numVuelos-1; i++)
{
    model.scalar(crewPlan, viajesCrew[i], ">=", 1).post();

}
model.sum(crewPlan, ">=",3).post();
model.scalar(crewPlan, precioCrew, "=",price ).post();
```

La segunda restricción representa la condición de que es necesario contratar siempre a 3 tripulaciones como mínimo, aunque se tengan cubierto el plan con un menor número de tripulación.

La última restricción servirá para calcular el precio total que ha costado el plan que se ha generado y se usará para minimizar este resultado.

Para las extensiones del problema solo habrá que modificar la restricción que obliga a cada vuelo a tener un número mínimo de tripulación, a la cantidad indicada, para la extensión 1 es 2 tripulaciones mínimas por vuelo y para la extensión 2 son 3 tripulaciones por vuelo

```
for(int i  = 0; i < numVuelos-1; i++)
{
    model.scalar(crewPlan, viajesCrew[i], ">=", 2).post();

}

for(int i  = 0; i < numVuelos-1; i++)
{
    model.scalar(crewPlan, viajesCrew[i], ">=", 3).post();

}
```

## Evaluación de resultados

Podemos apreciar que se cumplen todas las restricciones, cubriéndose todos los vuelos y teniendo un mínimo de 3 tripulaciones para cubrir todos los vuelos. Con un coste total de 18 unidades de dinero.

```
<terminated> ViajesCrew [Java Application]
Crew_1: 0
Crew_2: 0
Crew_3: 1
Crew_4: 1
Crew_5: 0
Crew_6: 0
Crew_7: 0
Crew_8: 0
Crew_9: 0
Crew_10: 0
Crew_11: 1
Crew_12: 0
Total_Price: 18
```

En caso de la extensión con un mínimo de 2 tripulaciones por vuelo podemos apreciar que se utilizan 6 tripulaciones para cubrir los 12 vuelos.

```
Crew_1: 1
Crew_2: 0
Crew_3: 1
Crew_4: 1
Crew_5: 1
Crew_6: 0
Crew_7: 0
Crew_8: 0
Crew_9: 0
Crew_10: 0
Crew_11: 1
Crew_12: 1
Total_Price: 36
```

En caso de 3 tripulaciones por vuelo podemos observar que se necesitan a todas las tripulaciones menos a la 10.

```
<terminated> ViajesCrewExt2 [Java Application]
Crew_1: 1
Crew_2: 0
Crew_3: 1
Crew_4: 1
Crew_5: 1
Crew_6: 1
Crew_7: 1
Crew_8: 1
Crew_9: 1
Crew_10: 0
Crew_11: 1
Crew_12: 0
Total_Price: 56
```

# Modelado 2: Planificación temporal para reparto de pedidos

Para modelar el reparto de los pedidos, tenemos como datos del problema los siguientes valores, la duración de cada pedido y las franjas temporales donde los clientes pueden atender a ese pedido. Además, el desplazamiento entre localizaciones tiene un coste, en este problema son de 30 minutos, este coste se deberá añadir después de cada entrega.

Por otro lado, los tiempos se han representado en forma matricial, en este caso cada matriz representa el horario en el que el cliente esta disponible y se le puede entregar el paquete.

Para el caso de P1, en cliente está disponible de 9 a 13:30 y de 17:00 a 20:30.

```
int[] durations = {30, 60, 30, 120, 60};

int deplazamiento = 30;

int[][] parcelEarlyTimes = {
        {9*60,13*60+30 }, //P1
        {12*60,14*60+30}, //P2
        {14*60,17*60 },   //P3
        {9*60,17*60 },    //P4
        {12*60,14*60+30 },//P5


};
int[][] parcelLateTimes = {
        {17*60, 20*60+30},  //P1
        {16*60+30, 18*60+30},  //P2
        {18*60,20*60+30 },  //P3
        {18*60+30,20*60+30  },  //P4
        {18*60+30,19*60+30  }, //P5
};
```

## Variables

las variables que utilizaremos serán 3, dos arrays que nos permitirán establecer el tiempo de inicio y final de reparto para cada paquete y una variable extra que nos permitirá obtener el tiempo del pedido final calculando en máximo tiempo de entrega máxima.

La variable endTimes se puede calcular a partir de startTimes y la duración de entra del paquete, pero dado que se debe obtener el valor máximo ha sido de utilidad la creación de esta nueva variable.

```
// Time variables
IntVar[] startTimes = model.intVarArray("startTimes", 5, 12 * 60+desplazamiento, (20 * 60) - desplazamiento)
IntVar[] endTimes = model.intVarArray("endTimes", 5, 12 * 60+desplazamiento, (20 * 60) - desplazamiento);
IntVar maxEndtime=model.max("max", endTimes);
```

Dado que el camión sale del depósito a las 12 y debe estar antes de las 20:00 en el depósito el rango de las variables endTimes y startTimes están limitadas a estos valores ya que el primer paquete podrá sen entregado a partir de las 12:30 (30 minutos de viaje hasta la primera localización del paquete) y el último deberá finalizar antes de las 19:30 para poder volver al depósito.

## Restricciones

En cuanto a restricciones, la primera permite definir la duración temporal de cada tarea y la segunda restringe el inicio y fin de la entrega a los horarios de los clientes, esto permite asignar las entregas de cada paquete a un horario donde el cliente pueda recibir el paquete.

```
for (int i = 0; i < 5; i++) {

    // Task [Start+Duration = end]
     model.arithm(endTimes[i],"=",startTimes[i],"+",durations[i]).post();

     // La tarea se encuentra entre los horarios que el cliente esta disponible
     model.or(
             model.and(model.arithm(startTimes[i],">=",parcelEarlyTimes[i][0]),
                     model.arithm(endTimes[i],"<=",parcelEarlyTimes[i][1])),
             model.and(model.arithm(startTimes[i],">=",parcelLateTimes[i][0]),
                     model.arithm(endTimes[i],"<=",parcelLateTimes[i][1]))
            ).post();

    }
```

Además, es necesario evitar el solapamiento entre las tareas, dado que solo se tiene un camión habrá que asegurarse que la entrega de dos paquetes no se pueda solapar.

Esto se consigue asegurando una de dos opciones, que la entrega de cada paquete se produzca antes o que se produzca después de la entrega del paquete actual.

Tambien es necesario tener en cuenta que después de entregar el paquete, es necesario desplazarse hasta la nueva localización por lo que el tiempo de finalización de un paquete deberá tener una diferencia de 30 minutos con el tiempo del siguiente paquete

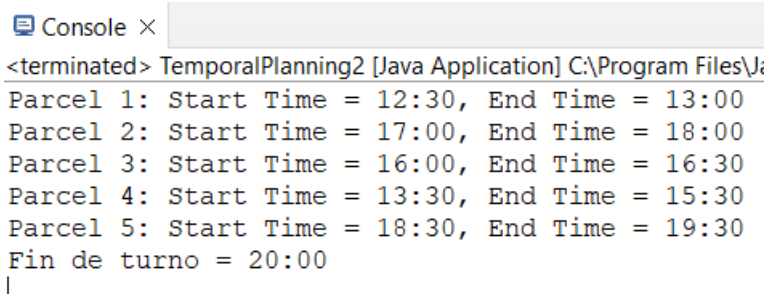
```
// Evitar solapamiento entre tareas
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if(i!=j) {
             model.or(
                     model.and(model.arithm(endTimes[i],"<=",startTimes[j],"-",desplazamiento),
                             model.arithm(endTimes[i],"<",endTimes[j])
                             ),

                     model.and(model.arithm(startTimes[i],">=",endTimes[j],"+",desplazamiento),
                             model.arithm(startTimes[i],">",startTimes[j])
                             )

                    ).post();
        }
    }
}
```

## Evaluación de resultados

Para poder compara los resultados de este problema se comparará un plan temporal y la solución óptima. Para obtener una solución simplificaremos el problema para que no se optimice la solución

```
Solution solution = model.getSolver().findSolution();
```

Podemos apreciar que el orden de entrega de los paquetes que se genera es 1-4-3-2-5, esto permite cumplir todas las restricciones del problema, permitiendo atender a todos clientes y volviendo a las 20:00 al depósito.

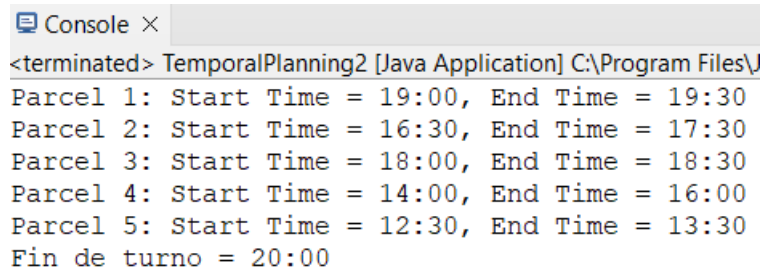



Dado que se quiere minimizar el tiempo de entrega se utilizará el último paquete para ver el tiempo final del camión y se minimizará este tiempo.

```
Solution solution = model.getSolver().findOptimalSolution(maxEndtime,false);
```

Podemos comprobar que en la solución óptima. El orden de entrega será el siguiente 5-2-4-1-3.

Dado que el tiempo que existe es justo para realizar todo el recorrido no hay una diferencia significativa temporal entre el problema modelado sin la minimización del tiempo. Esto podría cambiar si varían los datos de entrada , con clientes con diferentes horarios o diferentes tiempos de entrega.



# Conclusiones

La utilización de Choco CSP, como herramienta para abordar problemas de asignación y planificación temporal, en Eclipse nos ha permitido la modelización de estos problemas, permitiendo una representación clara y sencilla utilizando Java como lenguaje de programación.

El modelado del reparto se vuelos ha sido fácil de modelar gracias a la utilización de variables binarias numéricas que ha permitido representar la asignación tanto de los datos de entrada, como del plan optimo a implementar. Algunas ideas que se podrían usar para ampliar el problema podrían ser restricciones relacionadas con la tripulación como el que si la tripulación 1 participa la 4 no puede incluirse en el plan dado que 1 persona de la tripulación 1 se encuentra tambien en el grupo 4.

El modelo de planificación temporal para el reparto de pedidos ha sido más difícil de implementar debido a la mayor cantidad de restricciones que planteaba el problema, como el tiempo limitado de entrega de cada cliente, la planificación temporal de todos los pedidos y la restricción horaria. Como futuro proyecto se podría incluir la modelización de problemas con más de un camión o alguna otra restricción de optimalidad como la minimización del camino total recorrido por el camión.