Práctica 3: Planificación de equipos de robots móviles

En ésta práctica se desarrollará un método automatizado para la planificación de un equipo de robots móviles que tienen que cumplir una misión definida como una formula lógica sobre algunas regiones de interés. El entorno donde se mueven los robots se descompone en regiones utilizando algoritmos "*cell decomposition*" y la misión se define como una función lógica sobre los estados finales de los robots. Para cumplir con el objetivo, se obtendrá un modelo de tipo red de Petri para modelar las capacidades de movimiento del equipo. La misión se traduce a un conjunto de restricciones lineales y utilizando la ecuación de estado de la red de Petri, se resolverá un problema de programación lineal entera mixta cuya solución proporcionará las trayectorias de los robots.

Asumimos un entorno rectangular en el que se mueve un equipo de robots móviles idénticos. En el entorno se encuentran algunas regiones de interés que se pueden definir con el ratón en MATLAB ejecutando el fichero `entorno.m` (lo encontráis en Moodle). Denominamos las regiones de interés *observaciones*. El programa `entorno.m` hace una división del espacio robot utilizando un algoritmo de descomposición triangular teniendo en cuenta las observaciones definidas por el utilizador. Después de su ejecución, en la estructura T, se encuentra la siguiente información:

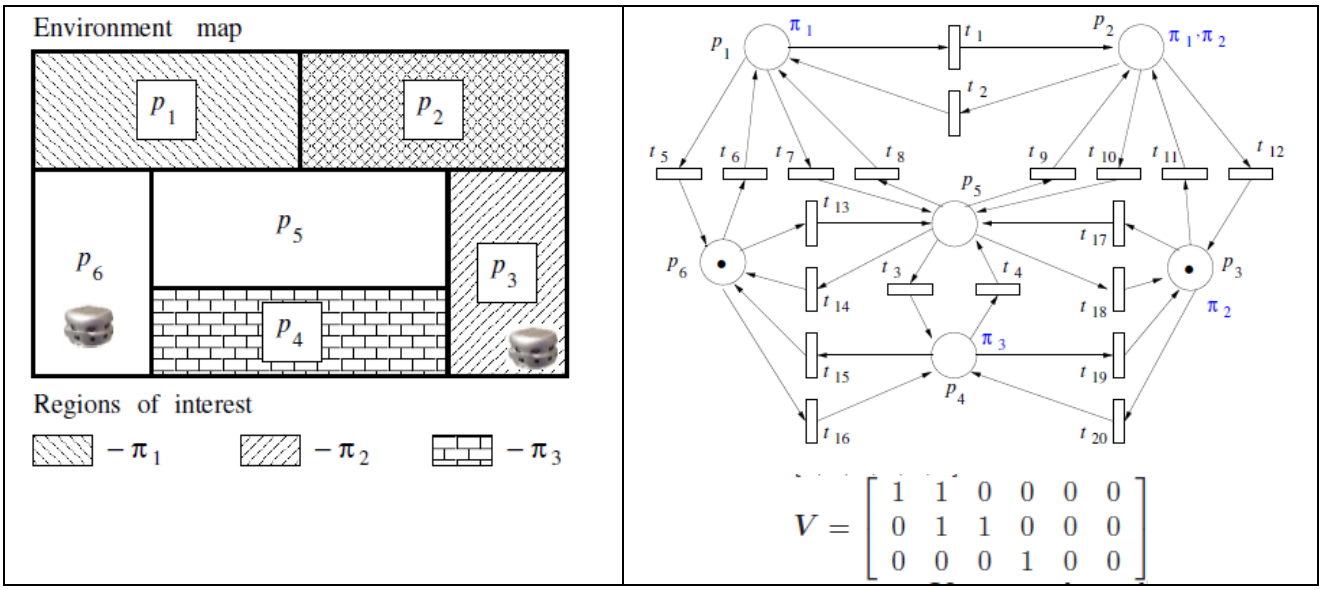
- **`T.Q`**: es el conjunto de estados discretos, cada elemento corresponde a una región del entorno particionado;
- **`T.Vert`**: son los vértices de las regiones del entorno particionado;
- **`T.adj`**: es una matriz binaria (elementos de `0` y `1`) tal que `T.adj(i,j)=1` si las regiones i y j son adyacentes. Nótese que hay un *selfloop* (auto-bucle) en cada nodo, i.e., `T.adj(k,k)=1` para todo k;
- **`T.OBS_set`**: contiene todas las *observaciones* posibles. A una región se le asigna un conjunto de observaciones y si en la región se encuentra por lo menos un robot, todas las observaciones asignadas están activas. Si se han definido regiones de interés independientes (no existe intersección entre las regiones definidas con el ratón) entonces cada región tiene como máximo una observación asignada. Por el contrario, si la intersección de dos regiones de interés no es nula, en algunas regiones de **`T.Q`** se puede cumplir más de una observación. El último elemento de **`T.OBS_set`** corresponde a la observación nula (regiones del entorno que tiene intersección nula con toda región de interés);
- **`T.obs`**: vector de dimensión igual al número de regiones del entorno correspondientes a las observaciones en cada región. El valor de cada elemento es el índice correspondiente en el vector **`T.OBS_set`**.
- **`T.mid_X`** y **`T.mid_Y`**: las coordinadas de los puntos centrales de las regiones que se pueden utilizar para dibujar las trayectorias;
- **`T.props`**: vector de dimensión igual al número de observaciones, y para cada observación se especifican las regiones del modelo discreto a los que se les ha asignado la observación;
- **`T.RO`**: vector conteniendo las regiones iniciales de cada robot.

Nota: Después de crear un entorno se recomienda guardar la estructura `T` en un fichero .mat (utilizando la instrucción 'save') para poder repetir los algoritmos para el mismo sistema.

Modelo de red de Petri del equipo de robots

L1. Escribir un programa Matlab teniendo como parámetro de entrada la estructura **`T`** para obtener a la salida un modelo de tipo red de Petri para el equipo de robots (las matrices **`Pre`**, **`Post`** y el vector de marcado inicial $\mathbf{m}_0$). La red de Petri se construye asignando un lugar a cada región obtenida después de la partición (o estado discreto de **`T`**), existiendo una transición entre dos lugares si corresponden a regiones adyacentes. El marcado inicial corresponde al número de robots presentes en cada región.

L2. Escribir un programa Matlab para obtener una matriz **`V`**, de dimensión: (#observaciones X #regiones) de forma que cada fila de la matriz sea el vector característico de una observación, i.e., `V(i,j)=1` si la región *`i`* definida por el utilizador tiene intersección no vida con la región *`j`* obtenida después de la partición y **`V(i,j)=0`** en caso contrario. En las figuras de abajo tenéis un ejemplo de un entorno, su modelo de red de Petri y la matriz `V` correspondiente.



Para cada observación, se define una variable binaria que será evaluada a ***1*** si la observación es activa en el marcado (estado) actual. Por ejemplo, en la figura de arriba, se definen tres variables binarias, `A` correspondiente a π_1, `B` para π_2 y `C` para π_3 y como en el estado actual ($\mathbf{m}_0$) hay un robot en p_3 (lugar a que hemos asignado la observación π_2), las variables binarias para $\mathbf{m}_0$ tienen los siguientes valores: `A=C=0` y `B=1`. Como se quiere utilizar el modelo de red de Petri para la planificación, los valores de las variables binarias dependen del marcado de la red de Petri. Las siguientes restricciones asignan a la variable `A`, el valor `0` ó `1` dependiendo de si la observación es activa o no en el marcado **`m`**:

$$\begin{cases} N \cdot A \geq V(A,:) \cdot m \\ A \leq V(A,:) \cdot m \end{cases}$$

Donde `N` es un número grande (estrictamente mayor que el número de robots) y **`V(A,:)`** es la fila de la matriz `V` correspondiente a la observación `A`. Para un marcado alcanzable **`m`**, si π_1 es activa (hay por lo menos un robot en una región con la observación `A`) entonces **`V(A,:)*m ≥ 1`**. Por el contrario, si no hay robots en las regiones donde se puede observar `A`, entonces **`V(A,:)*m=0`**. Sabiendo eso, es fácil ver que si **`V(A,:)*m ≥ 1`**, la solución del sistema anterior es `A=1` de acuerdo con la primera restricción. Por otro lado, si **`V(A,:)*m = 0`**, la segunda restricción implica `A=0`.

Fórmulas booleanas y restricciones lineales

Es conocido que cualquier fórmula booleana en forma normal conjuntiva (*conjunctive normal form* - CNF en inglés) se puede convertir en un conjunto de restricciones lineales. Por ejemplo la fórmula: `'(A or B) and not C and not D'` se convierte el siguiente sistema de ecuaciones:

$$\begin{cases} A + B \geq 1 \\ C \leq 0 \\ D \leq 0 \end{cases}$$

Definiendo el vector `Var=[A B C D]`^{T}, el sistema de restricciones anterior se puede poner en una forma matriceal:

$$\Delta \cdot Var \leq b \text{, donde } \Delta = \begin{bmatrix} -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ y } b = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

Las matrices `Δ` y `b` para una formula Booleana se obtienen ejecutando la función `formula2constraints` que la podéis bajar de moodle. Como ejemplo, para nuestra formula:

```
formula='(A|B)&!C&!D';
[A,b] = formula2constraints(formula, 4)
```

L3. Realiza un programa Matlab para calcular un marcado final m_f en una red de Petri que modela un equipo de robots utilizando la ecuación de estado de la red de Petri y programación matemática. En el marcado final m_f, las observaciones activas tienen que cumplir una fórmula booleana dada como parámetro de entrada. Para el problema de optimización se debe utilizar el software CPLEX. En nuestro caso, siendo la solución entera se debe utilizar la función `cplexmilp` desde Matlab.

L4. Utilizando la solución obtenida en **L3**, obtén las trayectorias como secuencias de regiones para cada robot. En este caso, utilizando la estructura de la red de Petri y sabiendo que los robots son idénticos, se puede demonstrar que las trayectorias se pueden obtener disparando las transiciones de una en una hasta disparar todo el vector de disparo.