Práctica 3: Planificación de equipos de robots móviles
60821 Evaluación y control de sistemas de producción
Máster Universitario en Ingeniería Informática

Cristian Mahulea
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Alberto Mur

21 de abril de 2024

Índice

1. Introducción

En esta práctica nos enfrentaremos a un problema de optimización que consiste en obtener la ruta que deben seguir un equipo de robots para conseguir la misión definida por el usuario. Para su resolución se utilizará un modelo de redes de Petri que se obtendrá a partir de la definición geométrica de un entorno.

2. Modelo de red de Petri del equipo de robots

Siguiendo las instrucciones del guión y del programa definimos un entorno sencillo con un robot y dos regiones de interés como la mostrada en la figura 1.

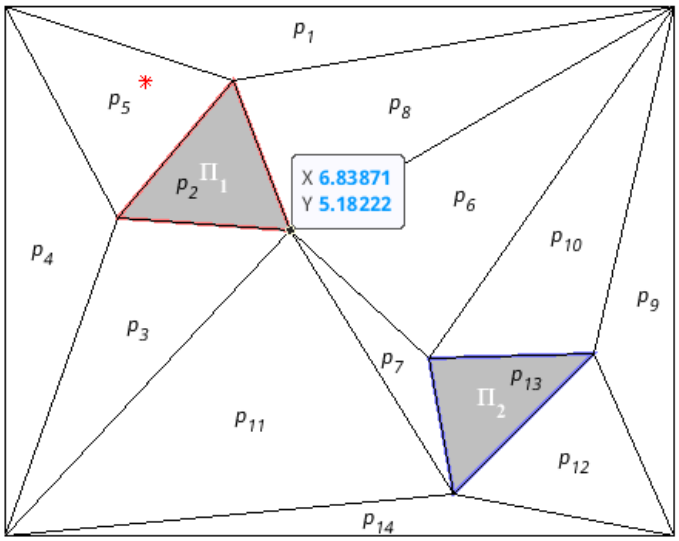

Figura 1: Entorno simple

Como se puede observar, se han definido las regiones de interés Π_1 y Π_2 las cuales se han asignados a los lugares $p2$ y $p3$ respectivamente. Y se ha posicionado un robot en el lugar $p5$.

Se ha creado una función en Matlab (Listado 1) que toma como entrada la estructura del entorno y devuelve las matrices PRE y $POST$, la matriz de incidencia C, el marcado inicial $m0$, la matriz de con los vectores característicos de cada observación, V y la matriz de adyacencia adj. El código esta comentado explicando cada uno de los procedimientos.

```
function [ PRE, POST, C, m0, V, adj] = petriNetBuilder( T )
%PETRINETBUILDER From an environment definition build the petri net model

% From adjacency matrix, remove diag.
adj = T.adj - sparse(T.Q,T.Q,ones(1,numel(T.Q)));

% We thus obtain the total number of transitions from one place to another,
% in both directions.
[ii,jj,~] = find(adj);

% We construct the PRE and POST matrices and use the non-zero indices from
% the previous matrix. The transitions are numbered sequentially in the
% columns, and the rows correspond to the places.
PRE =  sparse(ii,1:length(ii),ones(1,length(ii)));
POST =  sparse(jj,1:length(jj),ones(1,length(jj)));

% We create the incidence matrix.
C = POST - PRE;

% We construct the initial marking through the vector T.R0 which contains
% the position of each robot. We create a sparse matrix, taking advantage
% of the fact that the sparse function accumulates the value if the indices
% are repeated.
m0 = sparse(1,T.R0(:), 1, 1,numel(T.Q));

% We create the matrix with the characteristic vectors of each observation.
% This means that for each region of interest (row), the places
% (columns) that contain them are marked.
V = spalloc(numel(T.props), numel(T.obs), numel(cell2mat(T.props)));
for i=1:length(T.props)
    V(i,T.props{i}) = 1;
end

end
```

Listado 1: Función petriNetBuilder

3. Fórmulas booleanas y restricciones lineales

Para construir el sistema de restricciones lineales partimos de una condición *if then else*. Para un $N \gg m$ siendo m el número de robots se define una condición $N \cdot x \geq V \cdot m$ que indica que si un hay un robot en esta región se cumple y si no se cumplirá la condición $x \leq V \cdot m$.

Además podemos definir la misión mediante una fórmula booleana. Por ejemplo, para este entorno, $!A\&B$, es decir, la región A correspondiente a Π_1 es un obstáculo y/o no queremos robots en esa región. Y queremos que haya al menos un robot en la región B correspondiente a Π_2. Es decir $A \leq 0$ y $B \geq 1$.

Mediante la función `formula2constraints` obtenemos la ecuación en forma matricial $\Delta \cdot Var \leq b$.

4. Cálculo del marcado final y trayectorias

Con toda la información generada en apartados anteriores podemos iniciar la resolución de un problema de optimización de la forma:

$$\begin{aligned} &minf \cdot x \\ &A_{ineq} \cdot x \leq B_{ineq} \\ &A_{eq} \cdot x = B_{eq} \end{aligned}$$

Queremos calcular un estado final (m), una trayectoria que corresponde con los disparos de las transiciones (σ) y el cumplimiento de la misión (Var).

Por tanto:

$$x = \begin{bmatrix} m \\ \sigma \\ Var \end{bmatrix}$$

Como lo que queremos es minimizar la trayectoria

$$1^T\sigma = f \cdot x$$

$$1^T\sigma = f \cdot \begin{bmatrix} m \\ \sigma \\ x \end{bmatrix}$$

$$f = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

Ahora obtendremos los valores de A_{eq} y b_{eq} partiendo de la ecuación de la red de Petri:

$$m = m_0 + C \cdot \sigma$$

$$m - C\sigma = m_0$$

$$A_{eq} \cdot \begin{bmatrix} m \\ \sigma \\ x \end{bmatrix} = b_{eq}$$

$$A_{eq} = \begin{bmatrix} I & -C & 0 \end{bmatrix}$$

$$b_{eq} = m_0$$

A continuación obtendremos los valores de A_{ineq} y b_{ineq} para lo cuál haremos uso de las desigualdades obtenidas anteriormente:

$$\Delta \cdot Var \leq b$$

$$N \cdot Var \geq V \cdot m \rightarrow V \cdot m - N \cdot Var \leq 0$$

$$Var \leq V \cdot m \rightarrow Var - V \cdot m \leq 0$$

$$A_{ineq} \cdot x \leq b_{ineq}$$

$$\begin{bmatrix} 0 & 0 & A \\ V & 0 & -N \\ -V & 0 & I \end{bmatrix} \cdot \begin{bmatrix} m \\ \sigma \\ x \end{bmatrix} \leq \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}$$

Por último utilizaremos la función `cplexmilp` que resuelve problemas de programación lineal entera mixta. Solo queda definir una seria de parámetros adicionales como es el límite inferior, en nuestro caso 0. Esto es porque el limite inferior en todos los lugares es 0. Además debemos indicar el tipo de variables del sistema, en nuestro caso m y σ son enteros(`I`) y Var es booleano(`B`). Se ha programado una función en Matlab `findPaths` que toma como parámetros la matriz de incidencia C, la matriz con las observaciones V, el marcado inicial m_0, la fórmula booleana y el número de variables que la componen. Esta función devuelve el marcado final (m_f), las transiciones disparadas (σ) y las variables de la misión Var, así como información devuelta por el solucionador.

```
function [mf, sigma, compliance, fval,exitflag,output] = findPaths(C, V, m0, formula,
    nr_props)
%FINDPATHS From petriNetBuilder return values and a boolean formula returns
% minimal step to accomplish the mission

% We obtain the constraints in matrix form.
[A,b] = formula2constraints(formula, nr_props);

% We calculate sizes for later use.
[nr_restrictions, ~] = size(A);
[m_size,sigma_size] = size(C);
[nr_obs, ~ ] = size(V);

% We calculate the size of the target variable vector.
x_m = m_size + sigma_size + nr_props;

fprintf("Number of variables (%d): m(%d) + \\sigma(%d) + props(%d)\n", x_m , m_size,
    sigma_size, nr_props);

% Vector f
% x = [    m                  \sigma              Var         ]
f = [ zeros(1,m_size) ones(1,sigma_size) zeros(1, nr_props) ];

% Calculation of N strictly greater than the number of robots.
N = sum(m0) + 1;

% Aineq = [ 0   0   A
%           V   0   -N
%          -V   0   I  ]
Aineq = [
    zeros(nr_restrictions, m_size), zeros(nr_restrictions,sigma_size),  A;
    V,                              zeros(nr_obs, sigma_size),          -N*eye(nr_obs,
    nr_props);
    -V,                             zeros(nr_obs, sigma_size),          eye(nr_obs,
    nr_props);
];

% bineq = [ b 0 0 ]
bineq = [b; zeros(nr_obs,1); zeros(nr_obs,1)];

% Aeq = [ I  -C  0 ]
Aeq = [eye(m_size,m_size), -C, zeros(m_size, nr_props)];
beq = m0';

% Cplex Solver
sostype=[];
sosind=[] ;
soswt=[];
ub=[];
% Set the lower limit.
lb = zeros(size(f));
% Create type vector
ctype = char([ones(1,m_size)*'I' ones(1,sigma_size)*'I' ones(1, nr_props)*'B']);

[x,fval,exitflag,output] = cplexmilp(f,Aineq,bineq,Aeq,beq,sostype,sosind,soswt,lb,ub,
    ctype);

% Final state
mf = x(1:m_size);
```

```
offset = m_size;
% Triggers
sigma = x(offset+1:offset+sigma_size);
offset = offset + sigma_size;

% Compliance
compliance = x(offset+1:offset+nr_props);

end
```

Utilizando los valores devueltos podemos obtener la información de la trayectoria y el marcado final. En la imagen 2, observamos la trayectoria del robot mediante flechas grises y un multiplicador que indica el número de veces que se se realica ese desplazamiento. Los asteríscos rojos representan el estado incial y los circulos azules el estado final.

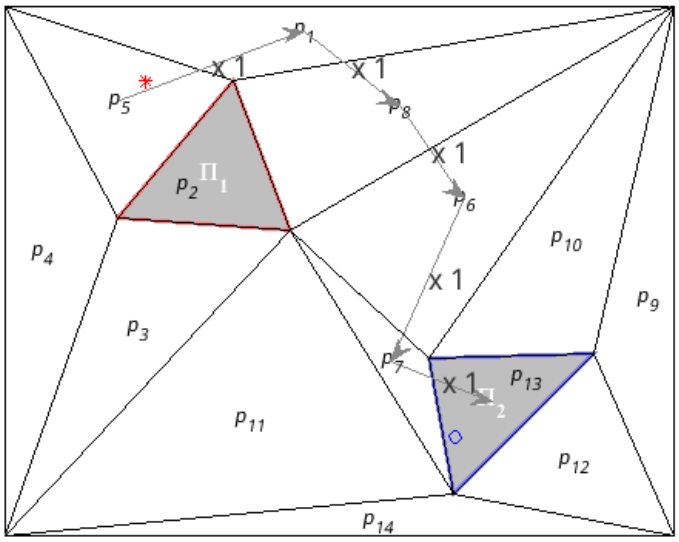

Figura 2: Solución de entorno de ejemplo 1

5. Otros escenarios

En el escenario de la figura 3 se definen cuatro robots, tres regiones de interés y la fórmula boolean $A\&B\&!C$. Observamos que de la región de interés $\Pi 3_3$ la flecha lleva asociado un multiplicador $x4$ que indica que todos los robots salen de esa región. El marcado final de la imagen sitúa: un robot en Π_1, un robot en Π_2, dos robots en el lugar p_9 y ninguno en la región Π_3. Por lo tanto vemos que se cumple la misión. Es importante destacar, que la misión se cumple a nivel de equipo.

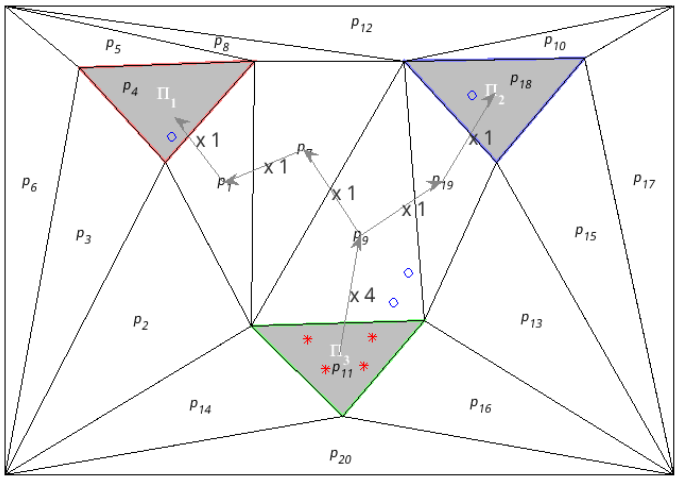

Figura 3: Solución de entorno de ejemplo 2

En la imagen de la figura 4 se ha definido un entorno más complejo con un robot y una especie de laberinto con tres regiones verticales (Π_1,Π_2 y Π_3) y la región Π_4. Se ha definido la fórmula booleana $!A\&!B\&!C\&D$, esperando conseguir una ruta serpenteante. Como se observa en la imagen, el algoritmo necesitaria mayor información para tratar las regiones Π_1,Π_2 y Π_3 como obstáculos a evitar, ya que en su resolución atraviesa estas regiones.

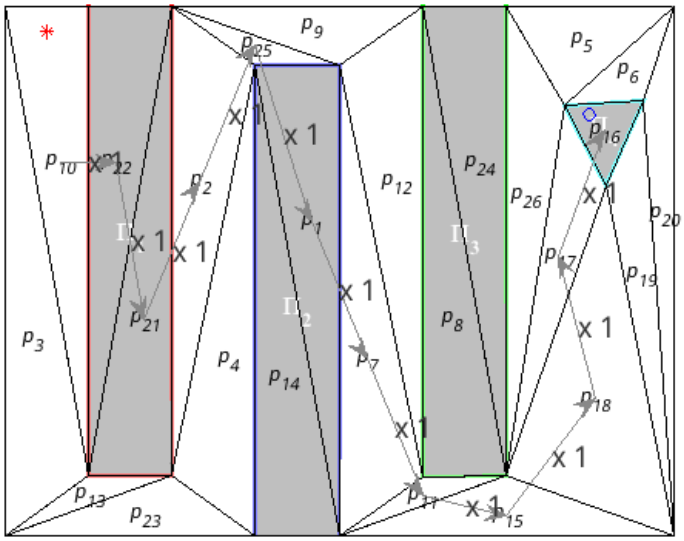

Figura 4: Solución de entorno de ejemplo 3

Y por último el entorno de la figura 5, donde se han posicionado: cuatro robots y cinco regiones de interés; y se ha planteado la misión $A\&B\&C\&!D\&!E$. La idea inicial era plantear un cuello botella con las regiones Π_4 y Π_5. Debido a la limitación anterior se observa que los robots atraviesan dichas regiones.

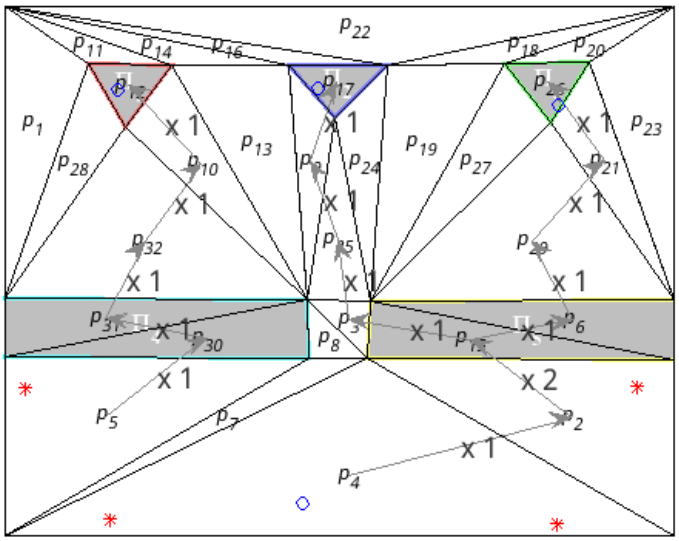

Figura 5: Solución de entorno de ejemplo 4

6. Conclusión

Con esta práctica hemos aprendido como podemos utilizar las redes de Petri para modelar diferentes sistemas y aplicar algoritmos de optimización para encontrar soluciones, en este caso, al problema de encontrar rutas óptimas para robots. También hemos aprendido a solucionar dichos problemas con programación y lineal y explorado sus limitaciones.