Universidad Nacional de Córdoba

FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y NATURALES

CARRERA INGENIERÍA EN COMPUTACIÓN

ESTUDIO EXPLORATORIO PARA LA DETERMINACIÓN DE POLÍTICA EN SISTEMAS CONTROLADOS POR REDES DE PETRI

Práctica Profesional Supervisada

Alumnos:
Casas, Nicolás
40574806, nicolas.casas@mi.unc.edu.ar, 3584305868
Ciarrapico, Nicolás Valentin
39880567, nicolas.ciarrapico@mi.unc.edu.ar, 2994840005

Supervisor: Ing. Luis Ventre
Tutor: Dr. Ing. Orlando Micolini

17 de noviembre de 2022

Índice

1. Introducción

1.1. Motivación

Los sistemas ciberfísicos (CPS) se refieren a sistemas que consisten en componentes cibernéticos (como implementaciones computarizadas) y físicos. La idea general es que los componentes cibernéticos y físicos se influyen mutuamente de tal manera que el componente cibernético puede hacer que el componente físico cambie de estado y que el cambio, a su vez, se retroalimente, lo que resultará en un cambio de estado en el componente cibernético[4]. Los mismos pueden entenderse como redes de procesos multi-físicos (mecánicos, eléctricos, bioquímicos, etc.) y computacionales (control, procesamiento de señales, inferencia lógica, planificación, etc.), que a menudo interactúan con un entorno altamente incierto y adverso, incluidos actores humanos y otros CPS. Los avances en la teoría y la práctica de estos sistemas aportan los medios para obtener un desempeño óptimo de los sistemas dinámicos, mejorar la productividad, aligerar la carga de muchas operaciones manuales, repetitivas y rutinarias, así como de otras actividades.

Los CPS entonces, tienen una interacción constante con el ambiente donde operan elpi. La manera en que toman decisiones para interactuar con el mismo es a través de lo que llamamos política. El problema, es que las diversas formas de implementar esta política en la actualidad (estáticas, aleatorias o round robin) resultan ser insatisfactorias cuando se aplican en sistemas reactivos, como los descriptos anteriormente. Es por esto que estudiaremos otras alternativas de realimentación que permitan obtener una política del sistema de manera dinámica, capaz de cumplir con requerimientos de usuario preestablecidos, y de esta forma optimizar los recursos empleados y el tiempo de respuesta del sistema.

1.2. Objetivos

El objetivo del presente trabajo es encontrar el compensador correcto para construir un sistema realimentado capaz de generar de manera dinámica una política que nos permita ordenar los disparos de una red de Petri de forma tal que se cumpla con requerimientos específicos definidos por el usuario. Además es importante poder realizar esto sin modificar el grafo de la red de Petri, teniendo en cuenta que la misma debe ser de tipo *S3PR* y no debe poseer deadlock. Para llevar a cabo esto estudiaremos las redes Bayesianas que usan la probabilidad para tratar la incertidumbre dentro de la inteligencia artificial y compararemos su rendimiento contra los compensadores más utilizados.

1.3. Requerimientos

Las definiciones de requerimientos del sistema especifican qué es lo que el sistema debe hacer (sus funciones) y sus propiedades esenciales y deseables[9]. Para crear definiciones de requerimientos del sistema requiere consultar con los clientes del sistema y con los usuarios finales, por lo que para obtener la información necesaria en esta etapa se llevaron adelante las siguientes tareas:

1. *Entrevistas*: Se realizaron una serie de entrevistas con el director de tesis Luis Ventre en conjunto con el co-director Orlando Micolini, en las cuales se definió de forma general los objetivos del proyecto. Además nos brindaron bibliografía de guía para estudiar posibles soluciones.

2. *Búsqueda y estudio de documentos*: Se analizó la bibliografía provista por los docentes, se buscó profundizar en material relacionado y se prepararon posibles soluciones que se discutieron en nuevas entrevistas.

A partir de esto se generó una lista de requerimientos para establecer las funcionalidades que tiene que cumplir el sistema.

1.3.1. Lista de requerimientos

1. El sistema debe ser capaz de controlar los disparos de una Red de Petri de forma tal que cumpla con requerimientos de invariantes de transición.

2. El sistema debe funcionar sin alterar o modificar el grafo de la red de Petri.
3. El sistema debe ser capaz de funcionar con cualquier Red de Petri S3PR.
4. Se debe implementar un modulo capaz de leer los archivos .html generados por Petrinator y generar las matrices de la red de petri en texto plano.
5. El sistema debe poder ejecutarse en Windows y Linux.
6. El usuario debe poder determinar los requerimientos por invariante de transición que debe seguir el sistema para una red en particular con el fin de generar las políticas de la misma.
7. El sistema debe ser capaz de ilustrar si el mismo convergió de forma correcta.

1.4. Modelo de Desarrollo

El modelo adoptado para el desarrollo del presente trabajo es el modelo iterativo. En este se realizan múltiples iteraciones en las que se desarrolla una versión del proyecto, se la expone a revisión por parte del cliente y se recolectan las correcciones del mismo. Estas se utilizan como entrada de la nueva iteración donde se genera una nueva versión repitiendo el proceso hasta obtener un sistema adecuado según los requerimientos planteados.

Las actividades de especificación, desarrollo y validación están entrelazadas en vez de separadas, con rápida retroalimentación a través de las actividades. Ian Sommervile[9].

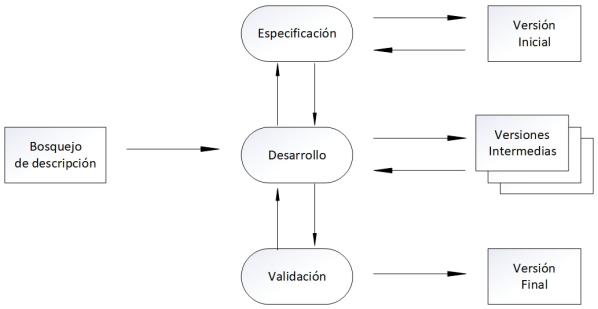

Figura 1: Desarrollo incremental.

En cada nueva versión se busca incorporar algunas de las funciones requeridas por el usuario. Es habitual comenzar por aquellas funcionalidades más importantes o urgentes. Esta interacción constante con el usuario permite controlar, desde una etapa temprana del desarrollo, que el producto es lo que se requiere. Si esto no se cumple, solo la ultima iteración se debe modificar y, probablemente, definir una nueva funcionalidad para próximas iteraciones.

1.5. Gestión de riesgos

De forma simple, se puede concebir un riesgo como una probabilidad de que una circunstancia adversa ocurra. Los riesgos son una amenaza para el proyecto, para el software que se está desarrollando y para la organización[9]. Se los puede agrupar en tres categorías:

- *Riesgos del proyecto*: Repercuten sobre el cronograma o los recursos del proyecto.
- *Riesgos del producto*: Afectan la calidad o el rendimiento del software que se quiere desarrollar.
- *Riesgos del negocio*: Influyen a la organización que desarrolla o comercializa el software.

Dado que las consecuencias de estos riegos pueden provocar que el proyecto fracase, es importante generar un *plan de gestión de riesgos* una vez definidos los requerimientos y antes de iniciar el desarrollo del primero. Esto cobra mayor importancia en los proyectos de software debido a las incertidumbres

inherentes con las que se enfrentan muchos proyectos. Es preciso anticiparse a los riesgos: comprender el impacto de éstos en el proyecto y en el producto final, y considerar los pasos para evitarlos. En el caso de que ocurran, se deben crear planes de contingencia para que sea posible aplicar acciones de recuperación. Las etapas que comprenden el proceso de gestión de riesgos son:

1. ***Identificación de riesgos.*** Identificar los posibles riesgos para el proyecto o el producto.
2. ***Análisis de riesgos.*** Valorar las probabilidades y consecuencias de estos riesgos.
3. ***Planificación de riesgos.*** Crear planes para abordar los riesgos, ya sea para evitarlos o minimizar su impacto en el proyecto.
4. ***Supervisión riesgos.*** Valorar los riesgos de forma constante y revisar los planes para la mitigación de riesgos tan pronto como se tenga nueva información sobre los mismos.

1.5.1. Identificación de riesgos

Para la realización de esta etapa se aplicaron dos técnicas:

- *Brainstorming*: Es una técnica creativa grupal cuyo objetivo es la generación de nuevas ideas sobre un tema o problema determinado. Permite obtener una lista de riesgos que luego serán analizados con mayor nivel de detalle.
- *Checklist*: consiste en un análisis detallado de los riesgos listados, descartando aquellos irrelevantes.

A continuación se presentan los riegos identificados, sobre que impactan y una breve descripción de los mismos.

Riesgo	Repercute en	Descripción
Planificaciones demasiado optimistas	Proyecto	La planificación efectuada se basa en situaciones ideales las cuales no se reproducen en la realidad.
Problema de coordinación entre los miembros del grupo	Proyecto	Descoordinación entre los miembros del equipo pueden causar retrasos en el desarrollo del proyecto.
Duplicación de código	Proyecto	Que los integrantes inviertan tiempo desarrollando la misma funcionalidad debido a una mala comunicación.
Subestimación de tamaño o complejidad	Proyecto	Que el proyecto a realizar sea más complejo que lo esperado.
Subestimación de los tiempos	Proyecto	Los tiempos para la finalización del proyecto pueden ser mayores a los estimados.
Aumento de la complejidad del proyecto	Proyecto	El proyecto se va haciendo cada vez más complejo, generando un mayor esfuerzo y tiempo en su realización.
Metodologías inadecuadas en el desarrollo del proyecto	Proyecto	La metodología empleada puede no adecuarse al proyecto, o hasta ser contraproducente.
Abandono de integrante del equipo	Proyecto	Un integrante abandona el proyecto.
Priorizar programación	Proyecto y producto	Dejar de lado otras actividades fundamentales en el desarrollo del proyecto por concentrar los esfuerzos en la programación del mismo.
Cambio de requerimientos	Proyecto y producto	Mayor cantidad de cambios a los requerimientos que los anticipados.
Pérdida del trabajo realizado	Proyecto y producto	Pérdida del proyecto por falla en la nube o dispositivo que lo contenga.
Enfoque incorrecto de las necesidades cubiertas	Producto	El proyecto realiza funcionalidades que no cumplen con las necesidades esperadas.

Cuadro 1: Riesgos identificados

1.5.2. Análisis de riesgos

Durante este proceso, se considera por separado cada riesgo identificado y se decide acerca de la probabilidad y la seriedad del mismo. No existe una forma fácil de hacer esto (recae en la opinión y experiencia del gestor del proyecto). No se hace una valoración con números precisos sino en intervalos[9]:

- La probabilidad del riesgo se puede valorar como muy bajo (< 10 %), bajo (10-25 %), moderado (25-50 %), alto (50-75 %) o muy alto (>75 %).
- Los efectos del riesgo pueden ser valorados como catastrófico, serio, tolerable o insignificante.

Este análisis se puede observar en el cuadro 2.

Riesgo	**Probabilidad**	**Efectos**
Planificaciones demasiado optimistas	Alta	Tolerable
Problema de coordinación entre los miembros del grupo	Baja	Tolerable
Duplicación de código	Muy baja	Tolerable
Subestimación de tamaño o complejidad	Alta	Serio
Subestimación de los tiempos	Moderada	Serio
Aumento de la complejidad del proyecto	Alta	Tolerable
Metodologías inadecuadas en el desarrollo del proyecto	Baja	Serio
Abandono de integrante del equipo	Muy baja	Catastrófico
Priorizar programación	Alta	Tolerable
Cambio de requerimientos	Moderado	Tolerable
Pérdida del trabajo realizado	Muy baja	Catastrófico
Enfoque incorrecto de las necesidades cubiertas	Baja	Tolerable

Cuadro 2: Análisis de riesgos

1.5.3. Planeación de riesgos

El proceso de planificación de riesgos considera cada uno de los riesgos clave que han sido identificados, así como las estrategias para gestionarlos. Otra vez, no existe un proceso sencillo que nos permita establecer los planes de gestión de riesgos. Depende del juicio y de la experiencia del gestor del proyecto[9]. En esta etapa se seleccionaron los riesgos con mayor gravedad o mayor probabilidad de ocurrencia y se generaron los siguientes planes:

- *Plan de mitigación*: Tiene por objetivo reducir la probabilidad de ocurrencia del riesgo o el impacto que el mismo puede provocar. El mismo puede observarse en la tabla 3.
- *Plan de contingencia*: Esta comprendido por las acciones que se deben realizar en caso de que el riesgo se presente. Este puede apreciarse en la tabla 4

Riesgo	**Estrategia**
Subestimación de tamaño o complejidad	Se utilizarán estrategias con diferentes enfoques para lograr una buena estimación
Subestimación de los tiempos	Se utilizarán estrategias con diferentes enfoques para lograr una buena estimación
Metodologías inadecuadas en el desarrollo del proyecto	Se profundizará en la bibliografía y se analizará en profundidad con el fin de escoger la metodología más adecuada a cada etapa del proyecto
Pérdida del trabajo realizado	Se generaran semanalmente copias de seguridad en múltiples dispositivos
Abandono de integrante del equipo	Continuación el trabajo de forma individual

Cuadro 3: Plan de mitigación

Riesgo	**Estrategia**
Subestimación de tamaño o complejidad	Se buscará dividir el problema en partes de menor complejidad y resolver aquellas de mayor importancia
Subestimación de los tiempos	Se buscará dividir el proyecto en partes de menor complejidad y resolver aquellas de mayor importancia
Metodologías inadecuadas en el desarrollo del proyecto	Se profundizará en la bibliografía y se aplicará una nueva metodología
Pérdida del trabajo realizado	Restauración de la copia de seguridad más reciente
Abandono de integrante del equipo	Se continuara el trabajo de forma individual

Cuadro 4: Plan de contingencia

2. Marco Teórico

2.1. Redes de Petri

Las Redes de Petri son modelos matemáticos utilizados para la representación de sistemas con paralelismo, concurrencia, sincronización e intercambio de recursos[7]. La red de Petri esencial fue definida por Carl Adam Petri. Son una generalización de la teoría de autómatas que permite expresar un sistema como eventos concurrentes.

Las redes de Petri están fuertemente asociadas a la teoría de grafos, ya que las mismas pueden representarse como un grafo dirigido bipartito compuesto por cuatro elementos:

- *Plazas*: Representan estados del sistema. El estado de una plaza está dado por la cantidad de marcas o tokens que esta contiene.
- *Token*: Figuran como puntos negros dentro de las plazas. Estos representan el valor específico de una condición o estado y generalmente se traducen en la presencia o ausencia de algún recurso del sistema.
- *Transiciones*: Representan el conjunto de sucesos cuya ocurrencia produce la modificación de los estados de las plazas, y en consecuencia del estado global del sistema.
- *Arcos*: Indican las interconexiones entre las plazas y las transiciones, estableciendo el flujo de tokens que sigue el sentido de la flecha.

2.1.1. Red de Petri Generalizada

Una Red de Petri generalizada no marcada es una cuádrupla $< P, T, Pre, Post >$ donde:

- $P = P_1, P_2, \ldots, P_n$ es un conjunto finito, no vacío, de plazas.
- $T = T_1, T_2, \ldots, T_m$ es un conjunto finito, no vacío, de transiciones, donde $P \cap T = \emptyset$, i.e. los conjuntos P y T son inconexos.
- $Pre : PxT \to \mathbb{N}^P$ es la función de incidencia de entrada y
- $Post : PxT \to \mathbb{N}^P$ es la función de incidencia de salida.

$Pre(p_i, t_j)$ contiene el peso del arco que va de P_i a T_j.
$Post(p_i, t_j)$ contiene el peso del arco que va de T_j a P_i.

2.1.2. Red de Petri Marcada

Una Red de Petri Marcada está definida por el par (N, M), donde N es una Red de Petri y $M : P \to \mathbb{N}^P$ (donde $|P| = p$ es una aplicación llamada **marcado**. $m(N)$ define el marcado de la RdP y m_{pi} indica el marcado de la plaza p_i, es decir, el número de tokens contenido en la plaza i. La marca inicial se denota m_0 y da la cantidad inicial de tokens en todas las plazas de la red, por lo que especifica el estado inicial del sistema.

2.1.3. Tipos de Arcos

- *Común*: Consume o produce una cierta cantidad de tokens de una plaza, de acuerdo al peso del mismo.
- *Inhibidor*: Siempre en la dirección plaza a transición. En el caso que el marcado de la plaza sea mayor o igual al peso del arco, la transición no está sensibilizada. No consume tokens al producirse el disparo de la transición asociada.

- *Lector*: Siempre en la dirección plaza a transición. La transición está sensibilizada si la marca en la plaza es mayor al peso del arco. No consume tokens al producirse el disparo de la transición asociada.
- *Reset*: Siempre en la dirección plaza a transición. Consume todos los tokens de la plaza al dispararse la transición.

2.1.4. Matrices

Para una red con n plazas y m transiciones, mas matrices tienen un tamaño nxm. Cada fila representa una plaza, mientras que cada columna representa una transición. Se conforman de la siguiente manera:

- *Matriz de Incidencia*: Está compuesta por las matrices I^+ e I^-, las cuales son función de los arcos comunes.

  En la matriz I^+, denominada matriz de incidencia de entrada o *post*, cada elemento $post(P_i, T_j)$ contiene el peso del arco que va desde T_j a P_i. Indica la cantidad de tokens generados al disparar la transición.

  En la matriz I^-, denominada matriz de incidencia de salida o *pre*, cada elemento $pre(P_i, T_j)$ contiene el peso del arco que va desde P_i a T_j. Indica la cantidad de tokens consumidos por la transición al realizar el disparo.

  Finalmente, la matriz de incidencia I se forma de la siguiente forma:

$$I = I^+ - I^- \tag{1}$$

- *Matriz de inhibición*: Contiene en cada uno de sus elementos $inh(P_i, T_j)$, el peso del arco de inhibición que va desde $P_i a T_j$
- *Matriz de Reset*: Contiene en cada uno de sus elementos $res(P_i, T_j)$ un 1 si existe un arco de reset que va desde P_i a T_j
- *Matriz de lectura*: Contiene en cada elemento $lec(P_i, T_j)$ el peso del arco lector que va desde P_i a T_j.

2.1.5. Sensibilizado de una transición

Una transición está sensibilizada si todas las plazas de entrada a la transición tienen una marca igual o mayor al peso del arco que une cada plaza con la transición.

Previo a expresar la condición de sensibilizado de manera general, son necesarias las siguientes definiciones:

- $\bullet T_j$ es el conjunto compuesto por las plazas entrante a T_j
- $T_j \bullet$ es el conjunto compuesto por las plazas salientes a T_j.
- $M_k(P_i)$ es el marcado de la plaza P_i antes de disparar la transición T_j.
- $M_{k+1}(P_i)$ es el marcado de la plaza P_i después de dispara la transición T_j.
- w_{ij} es el peso del arco $P_i \rightarrow T_j$
- w_{ji} es el peso del arco $T_j \rightarrow P_i$

De esta forma, el sensibilizado de una transición T_j se expresa como:

$$T_j \text{ está sensibilizada } sii \ \forall P_i \in \ \bullet T_j \ \rightarrow M_k(P_i) \geq w_{ij} \tag{2}$$

Lo definido anteriormente se cumple para redes cuyos arcos son todos comunes. Para redes con arcos inhibidores y/o de lectura cambia la condición de sensibilizado. De esta forma, dependiendo del tipo de arco deben cumplirse ciertas condiciones:

- *Arco de inhibición*: El marcado de la plaza de la cual parte debe es menor que el peso del arco.
- *Arco de lectura*: El marcado de la plaza de la cual parte debe ser mayor o igual al peso del arco.

Por su parte los arcos de reset no alteran la condición de sensibilizado de una transición.

2.1.6. Disparo de una transición

Dada una marca $M_k(P)$, cualquier transición que se encuentre sensibilizada puede ser disparada, este disparo nos llevará a una nueva marca $M_{k+1}(P)$ dada por:

$$M_{k+1}(P) = M_k(P) + I^+(P_i, T_j) - I^-(P_i, T_j) \forall P_i \in P \tag{3}$$

Al disparar la transición T_j se extraen tantos tokens de $\bullet T_j$ como indiquen los arcos que unen estas plazas con T_j. Se añaden a $T_j\bullet$ la cantidad de tokens que indiquen los arcos que unen a T_j con estas plazas. El disparo de una transición T_j se denota $M_k \rightarrow T_j \rightarrow M_{k+1}$.

2.1.7. Ecuación de estado

La ecuación de estado representa matemáticamente el comportamiento dinámico del sistema [6]. Esta permite obtener el estado del sistema luego del disparo de una transición. Se utiliza el marcado en un instante k para calcular el marcado de la red en un instante de tiempo $k + 1$. Por lo tanto la ecuación de estado para una Red de Petri con n plazas y m transiciones se define de la siguiente forma:

$$M_{k+1} = M_k + I * \sigma \tag{4}$$

Siendo:

1. I la matriz de incidencia.
2. σ vector de disparo. Tiene dimensión m x 1 y contiene 1 en la posición de la transición que se quiere disparar.
3. M_k: vector de marcado actual.
4. M_{k+1}: vector de marcado del estado siguiente.

2.1.8. Extensión de la ecuación de estado

La ecuación de estado descrita previamente es útil únicamente en presencia de arcos comunes. Para representar matemáticamente la existencia de los demás arcos nombrados anteriormente se requiere de una matriz para indicar la conexión plaza transición para cada uno de estos.

- *Vector de transiciones des-sensibilizadas por arco inhibidor*: Es un vector de valores binarios de dimensión $m \times 1$, que indica con un cero cuáles transiciones están inhibidas por el arco y con un uno cuales no.
- *Vector de transiciones des-sensibilizadas por arco lector*: Es un vector de valores binarios de dimensión m x 1, que indica con un cero cuáles transiciones están inhibidas por el arco con un uno las que no.
- *Vector transiciones des-sensibilizadas por tiempo*: Es un vector de valores binarios de dimensión m x 1, que indica con un cero cuáles transiciones están inhibidas porque no se ha alcanzado o se ha superado el intervalo de tiempo transcurrido desde que la transición fue sensibilizada.
- *Vector de transiciones reset*: Es un vector de valores enteros de dimensión m x 1, que tiene el valor de la marca de la plaza que se quiere poner a cero, mientras que las otras componentes son uno.

Finalmente utilizando los vectores descriptos anteriormente se puede obtener el vector de sensibilizado extendido E_x:

$$E_x = E \; and \; B \; and \; L \; and \; Z \tag{5}$$

Para poder introducir el brazo reset hay que multiplicar elemento a elemento (#) al vector que resulte de la conjunción por A. Obteniendo así la ecuación de estado extendida:

$$M_{k+1} = M_k + I * ((\sigma \; and \; E_x)\#A) \tag{6}$$

2.1.9. Propiedades de las Redes de Petri

2.1.9.1 Propiedad de limitación

Dada una Red de Petri PN, se dice que una plaza P_i está **k-limitada** por una marcado inicial M_0 si hay un entero natural k tal que, para todas las marcas alcanzables desde M_0, el número de tokens en P_i no es mayor que k. Es decir:

$$\exists \; k \in \mathbb{N} \; / \; \forall M \in marcados(PN) \rightarrow M(P) \leq k \tag{7}$$

Una Red de Petri PN está limitada para un marcado inicial M_0 si todos los lugares están limitados para M_0. Es decir, PN está limitada por k si todas sus plazas están limitadas por k.

A partir de esta definición surgen varios conceptos, entre los cuales se encuentran los siguientes:

- Una Red de Petri es **segura** si todas sus plazas son **1-limitadas**.
- Una Red de Petri es **cíclica** si siempre existe la posibilidad de alcanzar el marcado inicial desde cualquier otro marcado alcanzable.
- Una Red de Petri es **repetitiva** si existe una secuencia de disparos que contiene a todas las transiciones y que lleva a la red desde el marcado actual al mismo marcado.
- Una Red de Petri es **conservativa** si se cumple que el número de tokens en el marcado es siempre el mismo.

2.1.9.2 Propiedad de vivacidad

La **vivacidad** de una transición indica que, en todo instante de la evolución de la red, su disparo es posible. Este concepto es particularmente relevante ya que determina si la ejecución de la red puede o no detenerse en un estado determinado. A partir de esto se puede definir la vivacidad de una red de Petri. Esta propiedad indica que una red es viva para un marcado si todas sus transiciones lo son.

Por otro lado, la **cuasi-vivacidad** de una transición expresa la posibilidad de dispararla al menos una vez a partir de un marcado inicial M_0. De la misma manera que para el caso de la vivacidad, una red de Petri es cuasi-viva si todas sus transiciones lo son.

Gracias a esta última definición, se puede definir la vivacidad en función de la cuasivivacidad de la siguiente manera: una transición es viva si la misma es cuasi-viva en la red para todo marcado alcanzable desde M_0.

La vivacidad está directamente asociada con la ausencia de **deadlock** o **interbloqueo**. En términos generales, el deadlock es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos. En el caso de una red de Petri, esto suele ocurrir cuando dos o más transiciones esperan mutuamente por el disparo de la otra, produciendo el bloqueo permanente de esa porción de la red. Una red de Petri viva garantiza la ausencia de interbloqueo sin importar la secuencia de disparos.

2.1.9.3 Propiedad de alcanzabilidad

La **alcanzabilidad** de una Red de Petri es fundamental para el análisis de las propiedades dinámicas de un sistema. A grandes rasgos, permite determinar si el sistema modelado puede alcanzar un determinado estado.

Un marcado M_i es alcanzable desde el marcado inicial M_0 si existe una secuencia finita de disparos que me haga llegar a esa marca. Un marcado M_i es alcanzable desde M_0 si existe una secuencia finita de disparos σ tal que $M_0 \xrightarrow{\sigma} M_i$.

El conjunto de marcados alcanzables puede ser representado mediante un grafo, en el cual cada nodo representa un marcado. El arco entre estos nodos representa la transición que se necesita disparar para llegar de un marcado a otro.

Ejemplo:

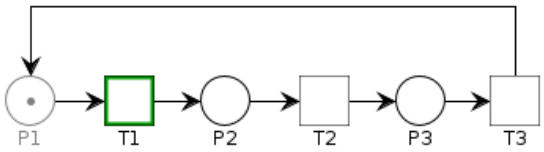

Figura 2: Red de Petri de tres estados.

Esta red cuenta con tres estados:

1. Marcado inicial o al disparar T3 desde m_2: $m_0 = [1, 0, 0]$
2. Marcado al disparar T1 desde m_0: $m_1 = [0, 1, 0]$
3. Marcado al disparar T2 desde m_1: $m_2 = [0, 0, 1]$

Con estos estados es posible generar el siguiente grafo:

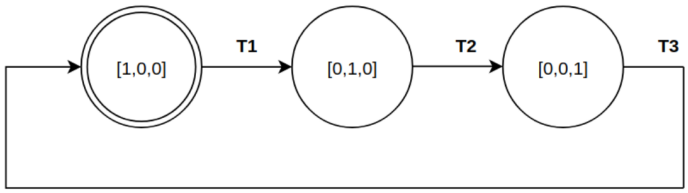

Figura 3: Grafo de alcanzabilidad.

2.1.9.4 Cobertura de una Red de Petri

Si la red de Petri no es acotada habrá plazas cuya cantidad de tokens aumentará indefinidamente, lo que también se repetirá en los nodos del grafo de alcanzabilidad, por lo que el algoritmo para obtener el árbol de alcanzabilidad no convergerá. Para estos casos existe otro tipo de análisis denominado **grafo de cobertura**.

El grafo de cobertura consiste en graficar todas las marcas posibles de la red, pero colapsando en un único marcado genérico, representado con el sımbolo ω, aquellas plazas cuya cantidad de tokens crecerá infinitamente.

2.1.10. Invariantes

Las invariantes de una red son propiedades independientes tanto del marcado inicial como de la secuencia de disparos, y pueden asociarse a ciertos subconjuntos de plazas o de transiciones. De esta forma surgen dos conceptos:

- **Invariante de plaza** o **p-invariante**: Se llama así a un conjunto de plazas si dado un determinado vector de ponderación con enteros positivos o cero se cumple que:

$$q_1 \text{ x } M(P_1) + q_2 \text{ x } M(P_2) + \cdots + q_n \text{ x } M(P_n) = K$$

  Siendo K constante para todo marcado. Es un componente conservativo independiente del marcado inicial. Sin embargo, el valor de la constante sí depende del marcado inicial.

- **Invariante de transición** o **t-invariante**: conjunto de transiciones que una vez disparadas secuencialmente generan el mismo marcado del cual habían partido, pudiendo disparar la secuencia indefinidamente.

2.2. Reinforcement Learning

Reinforcement Learning (RL) [11] es una rama del *machine learning* donde el aprendizaje se da interactuando con el ambiente. Es un aprendizaje orientado a objetivos, donde al algoritmo no se le enseña qué acciones debe tomar; sino que aprende de las consecuencias de sus acciones.

Este tipo de aprendizaje se complementa con los ya estudiados anteriormente como se ve en la siguiente Fig.4:

Figura 4: Tipos de aprendizajes.

El Reinforcement Learning entonces, intentará hacer aprender a la máquina basándose en un esquema de "premios y castigos" en un entorno en donde hay que tomar acciones y que está afectado por múltiples variables que cambian con el tiempo [3].

2.2.1. Componentes del RL

- *El agente:* Es el modelo que queremos entrenar para que aprenda a tomar decisiones
- *Ambiente:* Es el entorno en donde interactúa y "se mueve" el agente. El ambiente contiene las limitaciones y reglas posibles en cada momento.
- *Acción:* Son las posibles acciones que puede tomar en un momento determinado el Agente (cambiar de estado).

- *Estado:* Es el indicador de cómo se encuentran los diversos elementos que componen el ambiente en un momento dado.
- *Recompensas:* Cada acción tomada por el Agente tendrá como consecuencia un refuerzo positivo o negativo que orientará al Agente hacia la forma correcta de comportarse.
- *Política:* Define el comportamiento del agente en el ambiente.

Figura 5: Interacción entre el agente y el entorno.

El agente se encuentra en un primer momento en un "estado inicial" y luego realiza una acción, lo cual genera que esto influya en el ambiente, luego de esto el agente obtiene dos cosas: Un nuevo estado y la recompensa. Si esta ultima es negativa el agente actuara de forma distinta frente a dicha situación y si es positiva reforzara este comportamiento. Esta interacción se ve mas claramente en la Fig.5.

Al finalizar la instancia de aprendizaje todo el conocimiento aprendido es almacenado en la polıtica.

Figura 6: Interacción entre el agente y el entorno.

Se debe lograr un equilibrio entre la recompensa y el agente, dado que si el agente recibe una recompensa lo suficientemente alta el mismo siempre realizará la misma acción no consiguiendo generalizar el sistema. Fig.6.

2.2.2. Tipos de algoritmos de RL

Existen dentro del aprendizaje por refuerzo ciertos algoritmos a emplear, algunos de ellos son:

2.2.2.1 Q-Learning(Value Learning):

El objetivo principal al entrenar el modelo a través de las simulaciones es ir completando una matriz de Políticas de manera que las decisiones que tome nuestro agente obtengan "la mayor recompensa" evitando el sobreajuste.

- A la política se la denomina Q.
- Q(estado, acción) nos indica el valor de la política para un estado y una acción determinados.

Para completar la matriz de políticas se utiliza la ecuación de Bellman Fig.7.

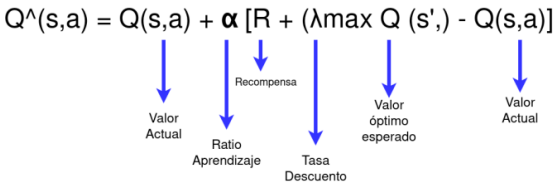

Figura 7: Ecuación de Bellman.

Esta ecuación determina cómo se irán actualizando las políticas Qˆ(s,a), en base a su valor actual más una recompensa recibida como consecuencia de dicha acción. Hay dos ratios que afectan a la manera en que influye esa recompensa: el ratio de aprendizaje, que regula "la velocidad" en la que se aprende, y la "tasa de descuento" que tendrá en cuenta la recompensa a corto o largo plazo.

2.2.2.2 Policy learning

Este método de aprendizaje por refuerzo tiene como principal diferencia a Value Learning (Q-Learning) que en esta última se busca tener una Red Neuronal/Política que aprenda a aproximar la función Q [1], para obtener un valor $Q_(s, a)$ de un estado dada una acción, y luego usamos este valor para inferir cuál es la mejor acción a tomar, esta es nuestra política.

Por otra parte **Policy Learning** busca directamente aprender la política pudiendo usar una NN o, en casos más simples, una matriz para luego poder alimentar la misma con una entrada y como salida se tendrá qué acción debemos tomar. Esto simplifica la situación, ya que para obtener la acción a tomar, es decir la acción que maximizar´a la recompensa dado un estado, simplemente se debe muestrear desde la función de política sin necesidad de realizar numerosas valuaciones.

En resumen en **Value Learning** se aproxima una función Q y se la usa para inferir la política óptima, mientras que en **Policy Learning** se optimiza la política de forma directa.

La política en esta metodología será alimentada con el estado actual, y como salida tendrá las probabilidades correspondientes a cada acción posible.

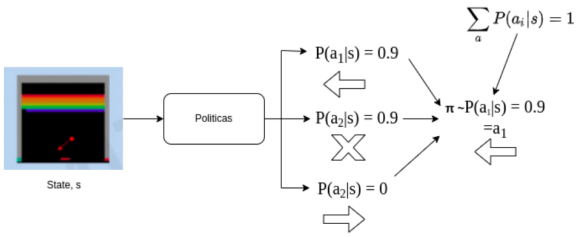

Figura 8: Policy gradient.

En la figura 8 se tiene como ejemplo las probabilidades de los distintos tipos de acciones que puede tomar el agente (la barra) con el fin de pegarle a la bola roja. Aquí la opción óptima es la de moverse a la izquierda, pero dado que esta es una distribución de probabilidad, la vez que se muestree podría

llegar a elegirse quedarse en el lugar (acción 2) dado que también cuenta con un valor no nulo de probabilidad.

Este método de RL puede trabajar con valores continuos. Siguiendo con el ejemplo previo, podría no solo obtenerse la dirección a la cual moverse, sino también la velocidad con la cual hacerlo. Esto puede realizarse visualizando la salida como una distribución probabilística dependiendo de cual se adapte más a la situación. En el ejemplo siguiente se toma una distribución gaussiana.

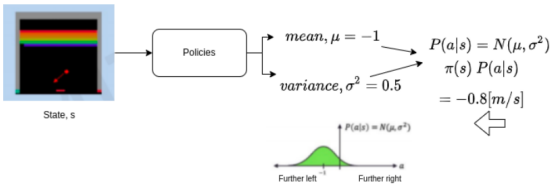

Figura 9: Policy gradient.

En la Fig.9 puede observarse que el agente debe moverse hacia la izquierda con una velocidad cercana a -1 m/s. Al muestrear en esta distribución, se puede notar que la velocidad en concreto que tomará el agente es de 0.8 m/s hacia la izquierda. De esta forma también se puede ver que a pesar de que la media de esta distribución es -1, no estamos limitados a ese número exacto.

Todo esto abre la posibilidad para escenarios donde puede llegar a tenerse un sin fin de acciones posibles a tomar.

2.3. Redes de Bayes

2.3.1. La estadística de Bayes

Es necesario definir ciertos términos que se utilizan con frecuencia en el manejo de la probabilidad Bayesiana. Estos son:

- **Probabilidad previa o a priori:** Es la probabilidad que se tiene de que cierto modelo sea cierto, antes de obtener datos a través de observaciones. Es necesario advertir que las observaciones cambian lo que sabemos del modelo.
- **Probabilidad posterior o a posteriori:** Es la probabilidad de que el modelo sea cierto después de las observaciones.
- **Variables aleatorias:** Representa una "parte"del mundo cuyo estado se desconoce. Se representan en mayúscula y sus posibles valores en minúscula. Las v.a pueden ser de distintos tipos:
  - Booleanas: El conjunto de valores posible esta formado solo por dos valores: $true$ (verdadero) y $false$ (falso).
    - Notación: a y $\neg a$ son equivalentes a $A = true$ y $A = false$ respectivamente.
  - Discretas: Sus posibles valores componen un conjunto discreto. Incluyen a las booleanas.
  - Continuas: El conjunto de valores posibles es un conjunto no numerable.
- **Proposiciones:** Usando las conectivas proposicionales y las variables, podemos expresar proposiciones.
  - Conectivas: $\vee(or)$, $\wedge(and)$, $\sim(nop)(\neg)$
  - Se asignan probabilidades a las proposiciones para expresar el grado de creencia que se tiene en las mismas.

2.3.2. Probabilidad: definición axiomática

Una función de probabilidad es una función definida en el conjunto de proposiciones (respecto de un conjunto dado de variables aleatorias), verificando las siguientes propiedades:

- $0 \leq P(a) \leq 1$ para toda proposición a.
- $P(true) = 1$ y $P(false) = 0$
  - donde true y false representan a cualquier proposición redundante o insatisfacible, respectivamente.
- $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$, para cualquier par de proposiciones a y b.

El calculo de probabilidades se construye sobre estos tres axiomas. Por ejemplo:

- $P(\neg a) = 1 - P(a)$
- $P(a \vee b) = P(a) + P(b)$, si a y b son independientes.
- $\sum_{i=0}^{n} P(D = d_i) = 1$, siendo D una variable aleatoria y $di, i = 1, 2, \ldots, n$ sus posibles valores.

2.3.3. Teorema de Bayes

Sea $\{A_1, A_2, \ldots, A_i, \ldots, A_n\}$ un conjunto de sucesos mutuamente excluyentes y exhaustivos tales que la probabilidad de cada uno de ellos es distinta de cero ($P[A_i] \neq 0$ para $i = 1, 2, \ldots, n$). Si B es n suceso cualquiera del que se conocen las probabilidades condicionales $P(B|A_i)$ entonces la probabilidad $P(A_i|B)$ viene dada por la expresión:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)}$$

donde:

- $P(A_i)$ son las probabilidades a priori.
- $P(B|A_i)$ son las probabilidades de B en la hipótesis A_i.
- $P(A_i|B)$ son las probabilidades a posteriori.

2.3.4. Fórmula de Bayes

Con base en la definición de probabilidad condicionada se obtiene la Fórmula de Bayes, también conocida como Regla de Bayes:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{k=1}^{n} P(B|A_k)P(A_k)} \cdots$$

Esta fórmula nos permite calcular la probabilidad condicional $P(A_i|B)$ de cualquiera de los eventos A_i dado B.

2.3.5. Redes Bayesianas

Es un grafo que representa la relación entre las variables de un problema, permitiendo una representación compacta y son de ayuda en la toma de decisiones [5]. Están conformadas por:

- *Nodos:* círculos que representan una variable aleatoria.
- *Flechas:* relaciones causales entre las variables aleatorias (conexiones entre nodos).

Las redes Bayesianas nos permiten realizar inferencias[2], existen tres tipos:

- Razonamiento de predicción.
- Razonamiento de diagnostico.
- Razonamiento de justificación.

2.3.5.1 Razonamiento de predicción

Es explicado por la cadena de causalidad.

Se usa la causa (información dada) para inferir algo

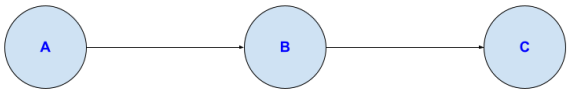

Figura 10: Cadena de Causalidad.
Red de Bayes con tres nodos.
Nodo A causa Nodo B y Nodo B causa Nodo C.

Aquí se utiliza la información dada en la causa para poder inferir algo. Si se tiene conocimiento sobre A, entonces se puede inferir le probabilidad de que ocurra B, y a su vez, la probabilidad de que ocurra algo sobre C.

El evento A causa el evento B y el evento B causa el evento C

Por lo que si se conoce el se puede inferir C y no es necesario conocer A. Esto se llama **bloqueo**, el evento B *bloquea* el evento A.

2.3.5.2 Razonamiento de diagnostico

Es explicado por una red de causa común.

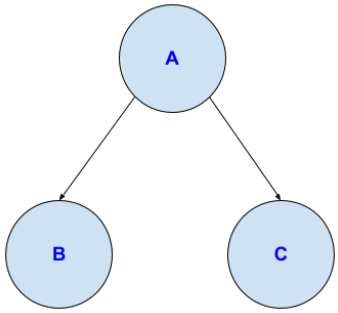

Figura 11: Red de causa común.
Nodo A causa Nodo B y también causa Nodo C.

2.3.5.3 Razonamiento de justificación

Dos o más eventos son causales de un tercero. Estas redes presentas características como:

- Independencia condicional
- Dependencia condicional

Entonces:

- Si A es verdadero es posible inferir la probabilidad de que sus padres A y B también sean verdaderos.
- Se puede explicar que ha causado C.

Cuando esto ocurre, si se conoce que uno de los eventos es verdadero, se puede inferir que la probabilidad que otro evento causante sea verdadero es menor. Esto es interesante dado que A y B son independientes.

En este tipo de redes también puede ocurrir un bloqueo. Existen casos donde un nodo puede bloquear varios nodos, esto es conocido con D-separación.

La figura 12, aclara estas ideas con un ejemplo, muestra si un vuelo puede estar retrasado; el retraso puede ser debido al mal tiempo o al exceso de trafico aéreo. La red esta compuesta por un nodo para cada variable y las relaciones entre estos (nodo Vr vuelo, nodo Mt mal clima, nodo Ta trafico aéreo). Las relaciones representada por flechas son causales donde los padres causan a los hijos.

En el ejemplo el retraso puede deberse al mal tiempo o al exceso de trafico aéreo y también por ambos.

Estas relaciones son probabilísticas y la fuerza de la causa depende del valor de la probabilidad que las relaciona. Por lo que en las aplicaciones, cuando sea posible conviene, que los eventos resultantes discretos sean mutuamente excluyentes, es decir tengamos solo un valor del grupo posible. Esto facilita el cálculo.

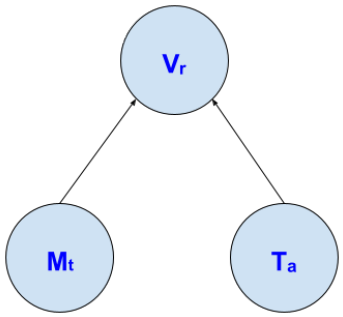

Figura 12: Red de efecto común.
Nodo A y Nodo B son causa de Nodo C.

Para el desarrollo de este ejemplo se supone a todas las variables aleatorias como binarias (V,F). Ahora, con esta información, es posible saber si el vuelo esta retrasado o no.

Para esto es necesario construir una tabla de probabilidades condicionales para saber que valores son necesarios utilizar en el calculo de la probabilidad final, dependiendo del mal tiempo y del exceso de trafico.

Valores de los padre		**Probabilidad para vuelos retrasados**	
Mal tiempo	Trafico aereo	Retrasado=Verdadero	Retrasado=Falso
v	v	$p(Vr \cap Mt \cap Ta)$	$p(\sim Vr \cap Mt \cap Ta)$
v	f	$p(Vr \cap Mt \cap \sim Ta)$	$p(\sim Vr \cap Mt \cap \sim Ta)$
f	v	$p(Vr \cap \sim Mt \cap Ta)$	$p(\sim Vr \cap \sim Mt \cap Ta)$
f	f	$p(Vr \cap \sim Mt \cap \sim Ta)$	$p(\sim Vr \cap \sim Mt \cap \sim Ta)$

Cuadro 5: Tabla de probabilidades condicionales para las variables Vr, Mt y Ta

La tabla de probabilidades condicionales debe construirse siempre para todos los nodos hijos de la red. Los nodos padres tendrán una tabla de probabilidad a priori.

3. Desarrollo

3.1. Curso sklearn master

En esta primera instancia, se llevó a cabo el tutorial sklearn-master, a modo de tener una primera aproximación al machine learning o aprendizaje automático.

El machine learning cubre principalmente tres áreas:

- *Aprendizaje supervisado*: A grandes rasgos, a partir de ciertos datos categorizables (de los cuales se conoce su categoría), se obtienen modelos matemáticos que permiten estimar, dada una entrada de categoría desconocida, a cuál pertenece.
- *Aprendizaje no supervisado*: En este caso, no se busca inferir a qué categoría pertenece algo, si no más bien encontrar características comunes entre los datos de entrada.
- *Aprendizaje por refuerzo*: Esta última área, tiene un enfoque distinto, ya que lo que busca es tomar decisiones de acciones a realizar a futuro, en base a una recompensa obtenida al haber realizado dicha acción en el pasado.

El curso, hace énfasis en las dos primeras áreas, con teoría, ejemplos y ejercicios. Los notebooks resueltos se encuentran en `https://github.com/los2nicos/PPS_y_PI/tree/main/tutorial-sklearn`.

3.2. Curso MIT: Introduccion to deep learning

De este curso[1] se prestó atención a las partes referidas a Reinforcement Learning o Aprendizaje por refuerzo, Deep Learning o aprendizaje profundo, para finalmente confluir en Deep Reinforcement Learning.

A partir de los ejercicios de este curso, se desarrolló un primer acercamiento al trabajo utilizando la librería de Tensorflow y se crearon algunas redes neuronales simples.

3.3. Componentes de software

A continuación se detallan los paquetes de software mas importantes que serán utilizados en el desarrollo del sistema del presente trabajo.

3.3.1. TensorFlow

TensorFlow es una es una plataforma de código abierto que ofrece herramientas para aprendizaje automático. Permite trabajar con tensores (matrices multidimensionales con un tipo uniforme), variables, desarrollar redes neuronales y bucles de aprendizaje para las mismas, etc.

Principalmente el manejo de tensores resultará de utilidad para el desarrollo del presente trabajo.

3.3.2. Matplotlib

Matplotlib es una biblioteca para la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o arrays en el lenguaje de programación Python.

Estos gráficos permitirán analizar el comportamiento del sistema.

3.3.3. NumPy

NumPy es el paquete más utilizado para la computación científica en Python[10]. Esta librería provee objetos de arreglos multidimensionales, de los que derivan las matrices, y una amplia variedad de rutinas para operar rápidamente con arreglos que incluyen operaciones matemáticas, lógicas, manipulación de forma y tamaño, ordenamiento, I/O, álgebra lineal básica, y más.

Como se detalló en el marco teórico, las Redes de Petri se pueden representar como un conjunto de matrices, por lo que es fundamental contar con una herramienta que permita manipularlas eficientemente.

3.3.4. re — Operaciones con Expresiones Regulares

Este módulo proporciona operaciones de coincidencia de expresiones regulares [8].

El manejo de expresiones regulares permitirá identificar cuando se complete un invariante de transición, al buscar en un archivo de log de actividades de la Red de Petri, la secuencia de disparo de transiciones que conforman el invariante.

El poder llevar un control de los invariantes completados permitirá comparar este numero con los requerimientos del usuario.

3.3.5. pyAgrum

PyAgrum es una librería desarrollada en C++ y Python dedicada al manejo de Redes Bayesianas y otros modelos gráficos de probabilidad. Permite crear estos modelos y realizar las operaciones necesarias para el cálculo de probabilidades de manera rápida y eficiente.

Esta librería nos permite realizar las gráficas de la Red de Bayes y de las tablas de probabilidad de cada nodo.

3.4. Condiciones

En esta sección detallaremos las condiciones que van a formar parte de la recompensa retornada por el ambiente al disparar una transición. Las mismas, son las siguientes:

- *Recompensa por poder dispararse*: Es la más simple de todas. Si dado un estado, puedo disparar una transición, va a tener esta componente. Sirve como caso base de prueba y equiponderar las transiciones disparables al deshabilitar el resto de las recompensas.
- *Recompensa por uso de recursos*: Si la transición disparada representa el uso de recurso, la recompensa otorgada por esta componente es directamente proporcional a la cantidad de recursos utilizados.
- *Recompensa por liberación de recursos*: Si la transición disparada representa la liberación de un recurso, la recompensa de esta componente es directamente proporcional a la cantidad de recursos liberados.
- *Recompensa por la liberación de recursos requeridos en la inmediatez*: Es un plus que se agrega a la recompensa anterior, en el caso en que el recurso liberado, sea necesario. i.e. permita que una nueva transición que tenga como plaza de entrada la que contenga el recurso liberado, se sensibilice por el disparo de la primer transición.
- *Recompensa por transiciones sensibilizadas*: Otorga recompensas si el disparo de la transición produce la sensibilización de otras que no lo estaban antes del disparo.
- *Recompensa por invariante completado*: La más importante de todas porque simboliza la finalización de un proceso o circuito. Otorga recompensa si la transición disparada, es la última necesaria para terminar un invariante. Notar el caso particular de que no sólo depende del disparo de una transición, si no de que se hayan disparado previamente el resto de transiciones pertenecientes a dicho invariante.

La recompensa obtenida será una suma ponderada de la siguiente forma:

$$R_t = \sum_{i=1}^{n} \alpha_i * R_i \tag{8}$$

donde:

- R_i es la recompensa iésima, y su valor representa la cantidad de veces que se cumplió esa condición,
- α_i es el peso o valor numérico de cumplir la recompensa iésima una vez y
- n es el número de condiciones que otorgan recompensas

3.5. Iteraciones

Para el trabajo integrador, se propone, en primera instancia, desarrollar las siguientes iteraciones:

- *Iteración 1*: Implementación de la lógica de ejecución de Redes de Petri y comunicación con Petrinator.
  1. Implementar la lógica para soportar Redes de Petri a partir de sus matrices.
  2. Implementar una herramienta que facilite la obtención de las matrices desde Petrinator.
- *Iteración 2*: Implementación del environ.
  1. Adaptación de la red, para poder ser implementada como un 'environ' propio del modelo de Reinforcement Learning.
  2. Implementar lógica de los métodos step y get_reward. Para este último considerar:
     - Puntos por poder disparar la transición
     - Puntos por utilizar un recurso
     - Puntos por liberar un recurso
     - Puntos por sensibilizar transiciones
     - Puntos por liberar un recurso que podría ser utilizado en la inmediatez
     - Puntos por completar invariantes
- *Iteración 3*: Implementar el agente y el sesgo inicial.
  1. Implementar la lógica de un sesgo inicial: En una red de Petri es muy probable que la cantidad de transiciones que no estén sensibilizadas i.e. no se puedan disparar, sea un número proporcionalmente grande, llevando a que el sistema tarde en converger o bien, diverja.
  2. Implementar la lógica de distintas políticas de decisión del autómata. Una determinística (la que genere mayor recompensa) y una probabilística (más probabilidad si mayor recompensa).
- *Iteración 4*: Implementar redes de bayes como sistema de control sobre conflictos.
  1. Implementar lógica para determinar situaciones de conflicto en Redes de Petri.
  2. Implementar lógica para obtener información para inicializar las tablas a priori de las redes Bayesianas.
  3. Implementar lógica para soportar redes de Bayes.
  4. Implementar lógica para la generación de políticas de la Red de Petri utilizando redes de Bayes para la resolución de conflictos.
- *Iteración 5*: Especificación de requerimientos por invariante de transición.
  1. Implementar una lógica para la especificación de requerimientos por invariante de transición.
  2. Implementar una lógica para contabilizar la cantidad de invariantes realizados.
  3. Implementar lógica de control para que la cantidad invariantes completados se ajuste a los requerimientos.
  4. Implementar una lógica para determinar si el sistema converge o no a los requerimientos especificados.
- *Iteración 6*: Extensión del control sobre todas los estados, no sólo casos de conflicto.
  1. Implementar la lógica que permita extender el control sobre todos los estados, no solo el conflicto.
  2. Realizar análisis de los resultados obtenidos respecto de la iteración anterior.

- *Iteración 7*: Análisis del comportamiento del transitorio.

    1. Implementar la lógica que permita obtener los parámetros de las gráficas de convergencia vistos como lazos de sistemas de control.
    2. Análisis del comportamiento de las gráficas al modificar parámetros del sistema o incorporar nuevos recursos.

3.6. Iteración 1

3.6.1. Introducción

En esta iteración se busca implementar un ambiente capaz de representar los elementos de una red de Petri (plazas, arcos y transiciones) representados como matrices, y su lógica de disparo así como también una herramienta que nos permita obtener la información necesaria para dicha representación desde los archivos .html generados por Petrinator.

3.6.2. Objetivos

- Implementar la lógica para soportar Redes de Petri a partir de sus matrices.
- Implementar una herramienta que facilite la obtención de las matrices desde Petrinator.

3.6.3. Desarrollo

3.6.3.1 Lógica de redes de petri

En esta etapa se creó una clase que realice las operaciones necesarias para manipular redes de petri. Indica el marcado de la red en un momento dado, qué transiciones están sensibilizadas en dicho momento, permite disparar transiciones y actualiza el estado de la red.

El constructor de la clase toma como parámetros los elementos que caracterizan a una red de petri, tales como el marcado inicial y las matrices de incidencia.

3.6.3.2 HTML_Parser

Para la representación de las redes de Petri en el sistema se necesita obtener desde Petrinator los siguientes elementos de cada red:

- La matriz de incidencia de entrada W^-.
- La matriz de incidencia de salida W^+.
- El marcado inicial M_0.
- Los invariantes de transición.
- Entero natural k al que esta limitada la red.

Petrinator permite obtener estos datos y guardarlos en archivos con formato .html. Se decidió implementar la clase *HTML_Parser* capaz de interpretar el contenido de estos archivos y recuperar e imprimir los elementos listados en formato requerido por Python.

3.6.4. Conclusiones

En esta iteración se logro con éxito incorporar la lógica de las redes de Petri al sistema y generando un base que permitirá implementar sobre esta un sistema que genere dinámicamente la política de disparo de la red.

3.7. Iteración 2

3.7.1. Introducción

En esta iteración se busca adaptar el código de la red de petri para que pueda ser implementada como un 'environ' propio del modelo de Reinforcement Learning. Se implementa la lógica de recompensas y del método *step*.

3.7.2. Objetivos

1. Adaptación de la red, para poder ser implementada como un 'environ' propio del modelo de Reinforcement Learning.
2. Implementar lógica de los métodos step y get_reward. Para este último considerar:
   - Puntos por poder disparar la transición
   - Puntos por utilizar un recurso
   - Puntos por liberar un recurso
   - Puntos por sensibilizar transiciones
   - Puntos por liberar un recurso que podría ser utilizado en la inmediatez
   - Puntos por completar invariantes

3.7.3. Desarrollo

En el modelo de Reinforcement Learning, el environ es algo a partir del cual se ejecuta una acción y se observa qué sucede. En la práctica, esto se hace mediante el método *step*, cuyo prototipo es el siguiente:

$$state, reward, done, info\ step(action) \tag{9}$$

donde:

- *action*: Valor que recibe la función por parámetro. Es la acción a realizar sobre el ambiente. Va de 0 a $n\_actions - 1$. En este caso, es la transición a disparar.
- *state*: Valor retornado por la función. Representa la información que se puede obtener del ambiente luego de realizar la acción. En este caso, es el nuevo estado de la red.
- *reward*: Valor retornado por la función. Es un indicador de qué tan bueno (o malo) fue el resultado de ejecutar la acción. Mientras mayor sea numéricamente hablando, mejor habrá sido la decisión tomada. En este caso, será una combinación lineal de una serie de factores a considerar, que se definirán en la próxima sección.
- *done*: Valor booleano que retorna la función. Indica que la ejecución terminó, sea porque se cumplió un objetivo o porque hay un fallo del que no se puede retornar. En el caso de la red de Petri, solo será True en caso de intentar disparar una transición no sensibilizada.
- *info*: Valor de retorno de la función. Información extra, en nuestro caso se omite este campo. Siempre será null, pero se incluye para mantener compatibilidad con el standard de environ de reinforcement learning.

3.7.3.1 Recompensa obtenida

Para el cálculo de la recompensa se tuvieron en cuenta distintos factores
Luego, el cálculo, es el siguiente:

3.8. Iteración 3

3.8.1. Introducción

En esta iteración se lleva a cabo la implementación del agente y del sesgo inicial.

3.8.2. Objetivos

1. Implementar la lógica de un sesgo inicial: En una red de Petri es muy probable que la cantidad de transiciones que no estén sensibilizadas i.e. no se puedan disparar, sea un número proporcionalmente grande, llevando a que el sistema tarde en converger o bien, diverja.

2. Implementar la lógica de distintas políticas de decisión del autómata. Una determinística (la que genere mayor recompensa) y una probabilística (más probabilidad si mayor recompensa).

3.8.3. Desarrollo

En esta iteración, se implementa el agente. Como ya se mencionó, el agente es quien toma acción sobre el ambiente. La decisión de qué acción realizar se lleva a cabo a través partir de:

- *choose_action()*: Este método recibe por parámetro el estado actual y la política a utilizar.
- *Políticas*:
  - *eg_policy*: Política épsilon greedy. En base a parámetros de exploración, determina si se inclina por la transición que mayor recompensa otorge o por una al azar. Utiliza tabla Q.
  - *prob_eg_policy*: Política épsilon greedy probabilísitca. En base a parámetros de exploración, determina si para seleccionar la acción a realizar se realiza mediante una distribuición de probabilidad que priorice a aquellas que más recompensa otorguen o si elige una al azar i.e. distribución equiprobable. En ambos casos, nunca seleccionará una transición que no se pueda disparar en el estado pasado por parámetro. Utiliza una tabla Q probabilística, que se explica más adelante en esta iteración.
  - *Otras políticas*: Que se explicarán en otras iteraciones.

Durante esta etapa del trabajo, el objetivo es maximizar el número de condiciones cumplidas, i.e. maximizar la recompensa obtenida por el agente.

A diferencia del trabajo final en el que se basa este trabajo, donde las tablas de Q-learning se utilizaban en caso de conflictos, la idea es extender el algoritmo para controlar todas las transiciones, independientemente del estado de la red.

Tras encarar este problema como un problema de Q learning tradicional, surgen un par de problemas:

- *Tamaño de la tabla Q*
- *Tiempo en converger*

3.8.3.1 Tamaño de la tabla Q

Una tabla Q tiene dimensiones *n_states* x *n_actions*, donde *n_actions* es el número de transiciones a disparar, y *n_states* es la cardinalidad del conjunto de marcados posibles de la red de petri, dado un estado inicial.

Para determinar los estados posibles, se desarrolló una búsqueda por amplitud (cuyo nodo raíz, es el estado inicial) disparando todas las transiciones y generando un nodo a explorar en caso de poder disparar la transición i.e. environ.can_shoot(T_i) == *true* y que el nodo no haya sido explorado previamente.

3.8.3.2 Sesgo inicial

Al haber redes grandes en las cuales, dado un estado, solo un subconjunto pequeño del conjunto de transiciones es factible de disparar, inicializar la tabla Q en ceros lleva a que el sistema demore un tiempo considerable en poder aprender a disparar unas pocas transiciones. Surge entonces la idea de crear una tabla sesgo. Esto es, una tabla no inicializada en ceros.

Como el problema principal, es que, dado un estado, la mayoría de las transiciones no están sensibilizadas, se optó por aprovechar la búsqueda por amplitud del punto anterior, para completar una estructura que vinculara, $estado_{actual} \rightarrow estados_{alcanzables}$ (en dicho estado).

Para esto, cada nodo de la búsqueda por amplitud (recordemos que hay un nodo por estado posible) contenía en su estructura un vector de dimensión M i.e. número de transiciones posibles. El vector tiene en su posición iésima, el identificador del estado al que se llega desde el estado actual tras disparar la transición i. En caso de que la transición i no se pueda disparar, la posición contiene -1. Cabe destacar que el identificador de un estado alcanzable es un número entero no negativo.

Luego, a cada estado se le asigna un valor igual a la mayor recompensa que se puede obtener, considerando todas las posibles m^n combinaciones de disparos partiendo de dicho estado. Donde n es un valor parametrizable y representa la profundidad deseada.

En la práctica, para cada estado se crea un tensor de rango igual a la profundidad de análisis elegida, donde cada elemento del tensor representa la recompensa obtenida tras disparar en orden las coordenadas del tensor, i.e. $tensor[0][2][1][3] = 67$ (tensor de rango 4), indica que dado un estado (al que pertenece el tensor), la recompensa de disparar las transiciones $0-2-1-3$ (en ese orden) es de 67. Luego, el valor que retornará el método $recursive\_reward\_getter()$ es el máximo valor encontrado en el tensor.

Vinculando esta información se pueden obtener vectores de dimensión M en los que, dado un estado, se obtiene la máxima recompensa esperada por el estado al que se llega tras disparar la transición i. Solo resta sumarle a cada elemento la recompensa de llegar desde el estado actual al nuevo estado.

Finalmente, al superponer estos vectores, se obtiene una tabla $Q(estado, transicion)$ tal que:

$$Q(estado, t_i) = \begin{cases} r > 0 & si\ t_i\ es\ disparable \\ r = 0 & en\ caso\ contrario \end{cases}$$

Superponiendo estos vectores se obtiene una *Q_value_table*, que no permite disparar transiciones no sensibilizadas y en la que al elegir la transición a disparar, elige la que mayor recompensa otorgue en el corto plazo.

3.8.3.3 Haciendo la tabla probabilística

Si bien es cierto que eligiendo siempre la transición cuya recompensa es la mayor, la recompensa global debería tender a ser la mayor, esto, en el escenario de representar un proceso productivo, podría llevar a realizar siempre el mismo proceso, condenando a los demás a no completarse nunca.

Para ello, se generó una tabla similar a la anterior, pero en la que se dividió cada elemento de Q_{ij} por la suma de la fila perteneciente al estado, i.e. $sum(Q_i)$. De esta forma:

$$Qprob_{ij} = \frac{Q_{ij}}{\sum_k Q_{ik}} \tag{10}$$

Con esta nueva tabla, se puede generar un número aleatorio, utilizando la fila Q_i como distribución de probabilidad y así elegir una transición con un peso ponderado a la recompensa de dispararla respecto de disparar las demás.

3.9. Conclusiones

3.10. Iteración 4

3.10.1. Introducción

En esta iteración se busca implementar una red de Bayes que represente de manera adecuada la dependencia de las variables de la red de Petri así como también un algoritmo inteligente para la generación automática de políticas para redes de Petri utilizando redes de Bayesianas.

3.10.2. Objetivos

- Implementar lógica para determinar situaciones de conflicto en Redes de Petri.
- Implementar lógica para obtener información para inicializar las tablas a priori de las redes Bayesianas.
- Implementar lógica para soportar redes de Bayes.
- Implementar lógica para la generación de políticas de la Red de Petri utilizando redes de Bayes para la resolución de conflictos.

3.10.3. Desarrollo

3.10.3.1 Determinación de conflictos

3.10.3.2 Recolección de experiencias

Se utiliza el agente desarrollado en la iteración anterior para disparar la red de Petri y se generara un archivo de log donde se registrara en cada disparo la información necesaria para calcular las probabilidades a priori y condicionales de las tablas de cada red Bayesiana. Los datos registrados son:

- Estado de la red antes del disparo.
- La transición que se disparo.
- Si se resolvió un conflicto, el número del conflicto.
- Por cada condición que se haya cumplido, el número de condición.

3.10.3.3 Redes de Bayes

Con el objetivo de resolver los conflictos de la red de Petri con el uso de redes Bayesianas se decidió generar una de estas para cada conflicto. Se probaron dos diseños:

- Red de Bayes stateless: Representada en la figura 13, la misma está compuesta por un nodo que representa el conflicto pudiendo tomar valores entre 0 y el número de transiciones de la red de Petri que forman parte del conflicto, y un nodo por cada una de las condiciones listadas en la sección 3.4. Consideramos que el hecho de que las condiciones se cumplan o no (su valor sea 1) depende de la transición que se resuelva el conflicto es por eso que los arcos van desde el nodo del conflicto hacia los nodos de condiciones.

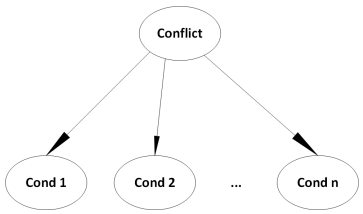

Figura 13: Red de Bayes sin estado.

- Red de Bayes stateful: Se puede observar en la figura 14. Su estructura es similar a la anterior pero se agrega un nodo que representa el estado de la red de Petri donde la tabla de este nodo tendrá tantas entradas como estados existan. El estado de la red de Petri es una variable observable por lo que su valor sera evidencia a la hora de calcular la probabilidad posterior conjunta del nodo C.

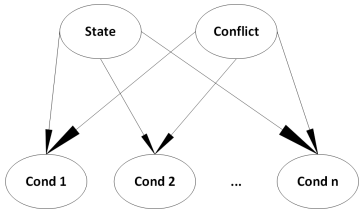

Figura 14: Red de Bayes con estado.

Para el soporte de la lógica bayesiana y su utilizacion para la resolución de conflictos fue necesario implementar 4 métodos encargados de generar la red de Bayes para cada conflicto, generar las tablas de cada nodo y cargarlas con los datos obtenidos de las experiencias, actualizar las tablas en tiempo de ejecución a medida que se tenga nueva información y seleccionar la transición con la que se debe resolver un conflicto en cada caso. Estos métodos son:

- *get_bns_list*: Método encargado de generar una red de Bayes para cada conflicto de la red de Petri. Aquí se generan los nodos los nodos de la red de Bayes, se generan las tablas de los mismos indicando cuantos valores posibles tiene cada uno, y se crean los arcos que representan la relación entre estos nodos. Como se requiere una red de Bayes para cada conflicto estas se almacenan en un arreglo llamado BN_list para facilitar su manejo donde se almacenan.
- *read_log*: Método dedicado a recolectar la información generada en la sección 3.10.3.2. Los datos de estas experiencias permitirán calcular las probabilidades a priori y condicionales para llenar las tablas de las redes de Bayes.
- *update_tables*: Método donde se calculan las probabilidades a priori y condicionales para rellenar las tablas de cada nodo de la red de Bayes.
- *politica_bayesiana*: Método encargado de obtener la probabilidades posterior del conflicto a resolver a partir de la red de Bayes del mismo y con estos valores asignar una probabilidad de ser disparada a cada transición del conflicto. Finalmente utilizando estos valores como distribución de probabilidad se escoge la acción a tomar.

3.10.4. Test realizados

3.10.5. Conclusiones

Se concluye que fue posible generar un sistema capaz de utilizar redes Bayesianas para la resolución de conflictos en una red de Petri y generar dinámicamente la política de disparo de la misma,

Referencias

[1] Alexander Amini. Deep reinforcement learning. mit, 6:s191, 2019.

[2] Nicolas Arrioja Landa Cosio. *Inteligencia Artificial, Sistemas Inteligentes con C#*. USERS, 2011.

[3] Juan Ignacio Bagnato. Aprendizaje por refuerzo. `http://aprendemachinelearning.com/aprendizaje-por-refuerzo/.`

[4] Paulo Carreira, Vasco Amaral, and Hans Vangheluwe, editors. *Foundations of multi-paradigm modelling for cyber-physical systems*. Springer Nature, Cham, Switzerland, 1 edition, September 2020.

[5] José Luis Iglesias Feria. Ia probabilidad - redes bayesianas. `https://www.youtube.com/playlist?list=PLYWD-VqrD5BCr-QeESSOvpEqkAcvQOrQ0`, 2018.

[6] Micolini Orlando, Cebollada Marcelo, Eschoyez Maximiliano, Ventre Luis O., and Schild Marcelo. Ecuación de estado generalizada para redes de petri no autónomas y con distintos tipos de arcos. *XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016)*, 2016.

[7] J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Independently Published, 2019.

[8] Python.org. 3.11.0 documentation.la biblioteca estándar de python.servicios de procesamiento de texto.re — operaciones con expresiones regulares. `https://docs.python.org/es/3/library/re.html#id1`, 2022.

[9] Ian Sommerville. *Ingenieria del software*. Pearson Educacion, 7 edition, 2005.

[10] the NumPy community. *NumPy User Guide. Release 1.23.0*. NumPy, 2022.

[11] Simonini Thomas. Deep reinforcement learning course. `https://simoninithomas.github.io/deep-rl-course/`.