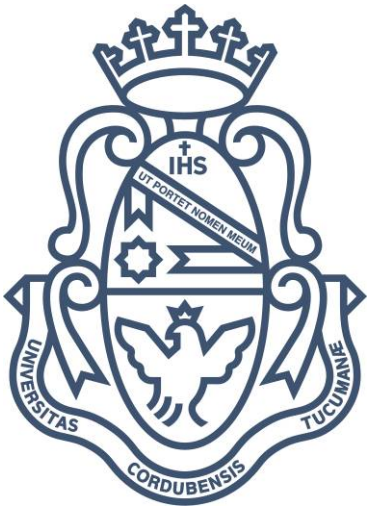
Facultad de Matemática, Astronomía, Física y Computación

Universidad Nacional de Córdoba



Juegos estocásticos con objetivo compuesto: recompensas totales sujetas a alcanzabilidad prioritaria

Tesis para obtener el título de

Licenciado en Ciencias de la Computación

Autor: Joaquín I. Feltes

Director: Pedro R. D'Argenio

Córdoba, Argentina 2024

Agradecimientos

A Abril, por el amor y el apoyo incondicional a lo largo de todos estos años.

A mi familia, por el constante acompañamiento, que me permitió concentrarme en hacer una carrera universitaria.

A mis compañeros de la facultad, en especial a Guido y César, por las tardes de estudio y su amistad durante toda la carrera.

A mis compañeros del ML lab, por motivarme para que me concentre en la tesis.

A todos los docentes que me crucé en la facultad, por su dedicación y enseñanza de calidad.

A Pedro por darme la oportunidad de trabajar con él, por todas las reuniones y por ayudarme con las inquietudes que me surgieron a lo largo de este año.

Resumen

El propósito de este trabajo es estudiar juegos estocásticos de dos jugadores con multiobjetivo. Uno de los objetivos es de alcanzabilidad de un conjunto de estados considerados exitosos y el otro es de recompensa total esperada, con la condición de que las recompensas serán 0 si no se llega a un estado exitoso.

Un juego estocástico es un grafo dirigido con transiciones probabilísticas y con dos jugadores, donde cada jugador tiene el control de varios estados en los que, por medio de acciones, se elige el siguiente estado. Estos jugadores son adversarios, es decir que uno quiere cumplir una serie de objetivos y el otro quiere impedirlo.

En los juegos estocásticos con recompensas totales cada estado tiene una recompensa y la meta de uno de los jugadores es maximizar su recolección. El objetivo de alcanzabilidad se manifiesta como un grupo de estados exitosos a los que el jugador maximizador quiere llegar.

En el pasado se ha trabajado con juegos estocásticos con recompensas totales bajo la condición de que el objetivo de alcanzabilidad se cumple con probabilidad 1. En nuestro caso, se agrega una nueva complejidad al problema ya que hay que tratar múltiples objetivos al mismo tiempo, teniendo que determinar algún orden de prioridad para trabajar sobre los mismos.

En este trabajo se introduce un algoritmo para la obtención del valor del juego, calculando la esperanza de llegar a los estados exitosos y la recompensa total esperada condicionada al objetivo de alcanzabilidad. Este algoritmo se divide en dos partes: la primera busca que el primer jugador maximice la probabilidad de llegar a los estados exitosos, la segunda busca que el primer jugador maximice la esperanza de la recompensa total acumulada condicionado a la maximización del éxito. En cualquiera de los dos casos, el segundo jugador se considera totalmente adversarial y, por consiguiente, busca minimizar ambos objetivos.

El algoritmo se implementó en una herramienta prototípica. Además se realizaron múltiples experimentos y se analizaron los resultados, evaluando los tiempos de ejecución y comparando los valores obtenidos si el minimizador se enfoca en la reducción de alcanzabilidad o de recompensas totales.

Abstract

The purpose of this paper is to study stochastic two-player games with multiple objectives. One objective is reachability of a set of states considered successful and the other is total rewards, with the condition that rewards will be 0 if a successful state is not reached.

A stochastic game is a directed graph with probabilistic transitions and two players, where each player has control of several states in which, by means of actions, the next state is chosen. These players are adversaries, i.e. one wants to accomplish a set of objectives and the other wants to prevent it.

In stochastic games with total rewards each state has a payoff and the goal of one of the players is to maximize its collection. The reachability goal manifests itself as a set of successful states that the maximizing player wants to reach.

In the past we have worked with stochastic games with total rewards under the condition that the reachability objective is met with probability 1. In our case, a new complexity is added to the problem since multiple objectives have to be dealt with at the same time, having to determine some order of priority to work on them.

In this work we introduce an algorithm for obtaining the value of the game, calculating the expectation of reaching the successful states and the total expected reward conditional on the reachability objective. This algorithm is divided into two parts: the first one seeks for the first player to maximize the probability of reaching the successful states, the second one seeks for the first player to maximize the expected total cumulative reward conditional on the maximization of success. In either case, the second player is considered fully adversarial and therefore seeks to minimize both objectives.

The algorithm was implemented in a prototypical tool. In addition, multiple experiments were performed and the results were analyzed, evaluating the execution times and comparing the values obtained whether the minimizer focuses on minimizing reachability or total rewards.

Índice general

Capítulo 1

Introducción

1.1. Introducción, motivación y objetivo

La teoría de juegos presenta una teoría matemática elegante y profunda. En las últimas décadas, ha recibido una gran atención desde Ciencias de la Computación porque tiene importantes aplicaciones para la síntesis y verificación de software. La analogía es atractiva, el funcionamiento de un sistema en un entorno no cooperativo (hardware defectuoso, agentes maliciosos, canales de comunicación poco fiables, etc.) puede modelarse como un juego entre dos jugadores (el sistema y el entorno), en el que el sistema intenta cumplir ciertos objetivos, mientras que el entorno trata de evitar que esto suceda. Este punto de vista es particularmente útil para la síntesis de controladores, es decir, para generar automáticamente políticas de toma de decisiones a partir de especificaciones de alto nivel. Por lo tanto, sintetizar un controlador consiste en calcular estrategias óptimas para un juego determinado.

En este proyecto nos enfocamos en juegos estocásticos de suma cero con información perfecta y para dos jugadores que alternan turnos y tratan de optimizar recompensas no negativas [10]. Intuitivamente, los dos jugadores juegan sobre un grafo moviendo una ficha por turnos. Algunos vértices son probabilistas en el sentido de que, si una ficha está en un vértice probabilista, el siguiente vértice se selecciona aleatoriamente. Además, los jugadores seleccionan sus movimientos utilizando estrategias. Asociado con cada vértice hay una recompensa que, en el contexto de este trabajo, asumimos no negativa. El objetivo del primer jugador es maximizar la cantidad esperada de recompensas recolectadas durante el juego, mientras que el segundo jugador busca minimizar este valor. Esto es lo que en [12] denominan *objetivo de recompensa total*. Este tipo de juegos ha resultado útil para razonar sobre varias clases de sistemas tales como vehículos autónomos, sistemas tolerantes a fallas, protocolos de comunicación, plantas de producción de energía, etc.

Para garantizar que el valor esperado de las recompensas acumuladas esté bien definido en cada jugada (quizás infinita), se necesita algún tipo de criterio de parada. Una forma común de hacer esto es obligar a las estrategias a decidir detenerse con alguna probabilidad positiva en cada decisión. Esto corresponde a los llamados juegos estocásticos con descuento [10, 11], y tiene las implicaciones de que las recompensas recolectadas pierden importancia a medida que avanza el juego (la "reducción de importancia" viene dada por el factor de descuento). Alternativamente, uno puede estar interesado en conocer la recompensa total esperada, es decir, la recompensa acumulada esperada sin que esta pierda

su valor a medida que pasa el tiempo. Para que este valor esté bien definido, el juego en sí debe detenerse. Es decir, independientemente de las estrategias que utilicen los jugadores, la probabilidad de llegar a un estado terminal debe ser 1 [8, 10].

Estudiaremos juegos con probabilidad 1 de llegada a un estado terminal, haciendo la distinción entre estados terminales exitosos y estados terminales no exitosos

El concepto de "detención exitosa" está asociado al logro de un objetivo exitoso. Así, por ejemplo, se puede pretender maximizar la superficie del terreno fotografiado por un vehículo aéreo no tripulado antes de que éste retorne a la base. Aquí consideramos que el problema se detiene exitosamente cuando el vehículo retorna a la base. Sin embargo podría ocurrir que por eventos circunstanciales (ej. entorno climático desfavorable o errores de navegación), el vehículo colapse en un accidente. Está claro que no tiene sentido tratar de maximizar la superficie fotografiada en una ruta que lo lleva inevitablemente al colapso, es decir, que llega a un estado terminal no exitoso. En este caso, nos encontramos en una situación donde la probabilidad de detención exitosa –es decir, la probabilidad de retornar a la base– puede ser menor a 1. Bajo este contexto, sería deseable poder obtener la estrategia que optimice la recompensa total esperada condicionado a que el juego alcanza la máxima probabilidad de detenerse exitosamente.

1.2. Ejemplo motivacional

Introduciremos un ejemplo en el que se presentarán una serie de reglas para motivar el trabajo. Estas reglas tendrán en cuenta la base teórica que se explicará a lo largo del escrito. El ejemplo es una modificación del juego "Roborta vs. the Fair Light" [3].

Roborta es un robot que se mueve dentro de una grilla de 4×4 celdas en base a las señales de un semáforo. Si este está en amarillo, Roborta se debe mover para alguno de los costados. Si está en verde, se debe mover para abajo, por último, si el semáforo está apagado, Roborta puede moverse como desee, ya sea hacia abajo o hacia alguno de los costados. Roborta y el semáforo cambian de estado en turnos, es decir, que en un determinado turno Roborta se mueve de casillero y luego en el turno siguiente el semáforo cambia de color, posteriormente Roborta debe moverse de casillero en base al nuevo color al que el semáforo cambió y continúa así hasta terminar el juego.

Además de estas reglas básicas de comportamiento, agregamos una serie de características al juego que hacen al problema más interesante. Primero, cada casillero tendrá una recompensa (no negativa). Segundo, algunas de las celdas restringen el movimiento a solo uno de sus lados. Tercero, consideramos que Roborta y el semáforo pueden fallar. Si Roborta falla, pierde la oportunidad de moverse y el semáforo puede volver a cambiar su color. Si el semáforo falla, se apaga, permitiendo que Roborta haga cualquier movimiento que la casilla le habilite. Los fallos se rigen cada uno por una probabilidad y duran solo un turno. Por último, agregaremos la posibilidad de que algunos casilleros estén 'flojos'. Esto hace que si Roborta se mueve hacia un casillero con esta característica, tendrá una posibilidad de romperse, haciendo que el juego termine y por lo tanto Roborta perderá, sin obtener recompensa.

Con esto en mente, Roborta tiene dos objetivos, primero completar el juego, esto se logra llegando a la parte inferior del tablero. Segundo, quiere conseguir la mayor cantidad de puntos como le sea posible. Por otro lado, el semáforo juega como su adversario o minimizador, es decir, tratando de impedir que Roborta cumpla sus objetivos.

Veremos que una versión de este tipo de juegos con dos objetivos son los juegos estocásticos con orden lexicográfico [5], donde el maximizador (Roborta) y el minimizador (semáforo) tienen un orden de prioridad para los objetivos. Es decir que priorizan maximizar (respectivamente minimizar) el primer objetivo y dadas las elecciones disponibles luego de hacer esto, intentan maximizar (minimizar) el siguiente objetivo.

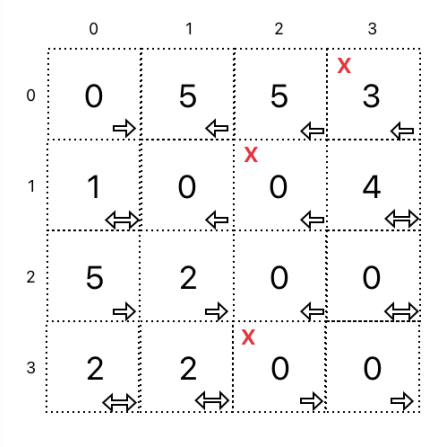


Figura 1.1: Ejemplo de un tablero 4×4

En nuestro caso Roborta primero quiere llegar al final del tablero y luego conseguir la mayor cantidad de puntos. El semáforo en cambio no tiene un orden de prioridad para minimizar los objetivos. Es decir que indiferentemente puede optar por minimizar la posibilidad de Roborta de llegar al final del juego o puede optar por minimizar la cantidad de puntos que se recolectarán.

Vemos en la figura 1.1 un ejemplo de tablero con 4×4 casillas. Todo tablero tiene como posición inicial la casilla $(0, 0)$ y termina abajo de la última fila, en este caso la fila 3. Cada casilla tiene en el centro el valor de la recompensa; en la esquina inferior derecha los movimientos disponibles para Roborta, representados con flechas con contorno negro y con una cruz roja en la esquina superior izquierda tenemos marcadas las casillas que tienen probabilidad de romperse. Por simplicidad en este ejemplo el semáforo y Roborta tienen probabilidad 0 de fallar.

En este juego el semáforo elige como primer movimiento prender la luz amarilla, para minimizar la probabilidad de Roborta de llegar al final del tablero. Esto la obliga a ir hacia la derecha. Luego de que se mueva, la luz vuelve a ponerse en amarillo, obligando a que Roborta se mueva hacia la izquierda y logrando que vuelva al casillero inicial. Vemos que si el semáforo actúa siempre de este modo, Roborta no llegará al objetivo final y el juego nunca va a terminar. Este comportamiento no es el que queremos simular, se quiere observar como el ambiente hace el intento de que Roborta se mueva entre las casillas flojas, intentando que pierda. También queremos ver si Roborta intenta esquivar estas casillas, recolectando la mayor cantidad de puntos y a su vez priorizando llegar al final del juego.

En [3] se soluciona este problema haciendo que el minimizador juegue de forma equitativa. Esto hace que si una elección es dada infinitas veces, se deben elegir todas las opciones posibles. Lo que provoca que el juego termine, ya que eventualmente va a tener que elegir prender la luz verde, y Roborta va a poder llegar al final del juego.

En nuestro caso, para no complejizar el juego con dos objetivos, no vamos a agregar un comportamiento equitativo al minimizador, si no que veremos dos casos en los que el juego puede progresar sin agregar esta condición. El primer caso es utilizando una de las reglas que ya presentamos, donde el semáforo se rompe con una probabilidad p. El segundo es haciendo una modificación al tablero. Para este caso agregaremos casillas con flechas hacia abajo, que obligan a Roborta a bajar, sin importar la elección del semáforo.

En el caso en que el semáforo puede romperse con una probabilidad p, Roborta elige avanzar hacia el fin del tablero casi cada vez que tiene la oportunidad. No solo eso, si no que también el semáforo, sabiendo que tiene probabilidad de romperse, cambia su estrategia. Esto es porque ya es imposible que Roborta se mantenga en un loop sin progresar en

el tablero. Lo cual nos da una forma de que el juego no se encuentre bloqueado por la estrategia del minimizador, dando una idea de fairness (equidad).

Podemos ver en la figura 1.2 un ejemplo de corrida con esta modalidad. En este ejemplo tenemos marcado con amarillo el recorrido que hace Roborta (dado que el semáforo no se haya roto en ningún momento). Y tenemos además señalado con una flecha negra en la esquina superior derecha de cada casilla en las que Roborta tiene una elección, la mejor estrategia para esa casilla. Vemos para este ejemplo que intentando minimizar las recompensas obtenidas por Roborta, el semáforo la guía por el camino con la menor cantidad de puntos posible, sin preocuparse por minimizar la probabilidad de que llegue al objetivo.

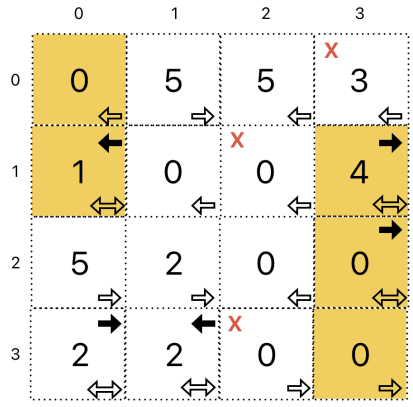


Figura 1.2: Tablero con el camino marcado siguiendo las estrategias de los jugadores

En el caso de modificar el tablero, como ya dijimos, agregaremos casillas que solo permiten a Roborta ir hacia abajo, sin importar la elección del semáforo. Esto hace que el ambiente no sea tan hostil con Roborta, y que eventualmente pueda llegar al final del juego. La existencia de estas casillas también hace que el minimizador cambie la estrategia, sabiendo que no puede evitar el progreso de Roborta. Cabe aclarar que esto es para un tablero que tenga una casilla de este tipo en cada fila, la cual sea alcanzable por Roborta, sin esta condición el semáforo puede igualmente hacer que el juego no termine.

En la sección 5 vamos a probar ambos tipos de ejemplos y comparar los resultados obtenidos.

1.3. Esquema de la tesis

La tesis esta estructurada de la siguiente manera: En el capítulo 2 vamos a mostrar las bases para poder trabajar con juegos estocásticos con múltiples objetivos y con las modificaciones específicas de este trabajo en concreto.

En 3 vamos a explicar primero qué y cómo son los juegos multiobjetivo y luego veremos diferentes estrategias que se realizaron en el pasado para resolver este problema.

Luego de la explicación de los juegos multiobjetivo y de los distintos algoritmos ya existentes tenemos en el capítulo 4 la definición del problema concreto que queremos resolver. Se demostrará por qué no cumple los requisitos o no puede ser abordado con los resultados de trabajos previos. También tendremos la resolución del problema, tanto el planteo teórico como el desarrollo del algoritmo.

Continuamos con el capítulo 5, en el cual se hablará de la implementación de una herramienta que incorpora los algoritmos mostrados en el capítulo anterior. También de un generador de tableros de Roborta, el cual se utilizará para crear múltiples ejemplos para correrlos con el algoritmo y poder compararlos en una tabla.

Por último en el capítulo 6 se resumirán los resultados del trabajo y se presentarán distintas posibilidades para continuar avanzando en la problemática.

Capítulo 2

Juegos estocásticos

En este capítulo daremos múltiples definiciones y resultados que serán útiles a lo largo del trabajo. Entre ellas vamos a definir los juegos estocásticos y cómo estos se relacionan con las cadenas de Markov (MC por sus siglas en inglés) y los Procesos de decisión de Markov (MDP). Así como los objetivos de alcanzabilidad y de recompensas totales, los cuales vamos a utilizar en nuestro caso de estudio. También aquí se va a definir la terminología que usaremos a lo largo del trabajo. La mayor parte de estas definiciones fueron tomadas de [2] y [3].

2.1. Definiciones básicas

Para empezar, diremos que una distribución de probabilidad (discreta) μ sobre un conjunto numerable S es una función:

$$\mu : S \to [0, 1] \tag{2.1}$$

tal que :

$$\mu(S) = \sum_{s \in S} \mu(s) \; = \; 1 \tag{2.2}$$

El conjunto de todas las distribuciones de probabilidad μ en S se denota con $\mathcal{D}(S)$.

Definición 1 (Distribución de Dirac). *Una distribución de Dirac es una distribución $\Delta_s \in \mathcal{D}(S)$ para un elemento $s \in S$. tal que:*

$$\Delta_s(s) \; = 1 \tag{2.3}$$
$$\Delta_s(s') \; = 0 \; si \; s \neq s' \tag{2.4}$$

Usaremos el concepto de distribución de Dirac cuando definamos las estrategias de un jugador.

Dado un conjunto V, denotamos con V^* (respectivamente V^∞) el conjunto de todas las secuencias finitas (resp. secuencias infinitas) de elementos de V. La concatenación se representa mediante yuxtaposición. Usamos las variables ω, ω', ... $\in V^\infty$ como elementos de las secuencias infinitas y las variables $\hat{\omega}$, $\hat{\omega}'$, ... $\in V^*$ como elementos de las secuencias finitas. El i-ésimo elemento de una secuencia finita $\hat{\omega}$ (resp. infinita ω) se denota con

$\hat{\omega}_i$ (resp. ω_i). Además, para cada secuencia finita $\hat{\omega}$, denotamos su largo con $\mid \hat{\omega} \mid$. Para $\omega \in V^{\infty}$, $inf(\omega)$ denota el conjunto de elementos que ocurre infinitamente en ω. Dado $S \subseteq V^*$, S^k es el conjunto obtenido por concatenar k veces las secuencias de S.

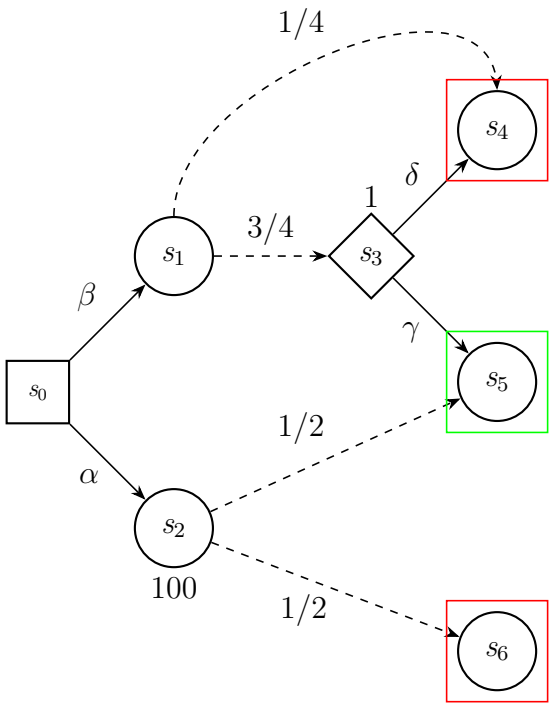


Figura 2.1: Juego estocástico, donde tenemos los cuadrados como estados de V_1, rombos como estados de V_2 y círculos como estados de V_P.

Definición 2 (Juego Estocástico). *Denotamos con $\mathcal{G}$ un juego estocástico, tal que:*

$$\mathcal{G} = (V, (V_1, V_2, V_P), \delta) \tag{2.5}$$

donde V es un conjunto finito de vértices (estados) y donde V_1, V_2, $V_P \subseteq V$ forman una partición de V. $\delta : V \times V \to [0, 1]$ es una función de transición probabilística, tal que para todo v que pertenece a $V_1 \cup V_2$, $\delta(v, v') \in \{0, 1\}$, para cualquier $v' \in V$; y $\delta(v, \cdot) \in \mathcal{D}(V)$ para todo $v \in V_P$. Si $V_P = \emptyset$, entonces $\mathcal{G}$ es llamado grafo de dos jugadores. Además, si $V_1 = \emptyset$ o $V_2 = \emptyset$, entonces $\mathcal{G}$ es un Proceso de decisión de Markov (MDP). Por último si $V_1 = \emptyset$ y $V_2 = \emptyset$, entonces $\mathcal{G}$ es una Cadena de Markov (MC).

Podemos ver en la Figura 2.1 un ejemplo de un juego estocástico, donde $s_0 \in V_1$, $s_3 \in V_2$ y $s_1, s_2, s_4, s_5, s_6 \in V_P$, y con la función de transición δ definida para cada estado con las flechas que salen de cada uno en la figura. Por ejemplo para s_0 y s_1 tenemos:

$\delta(s_0, s_1) = 1$, $\delta(s_0, s_2) = 1$ y $\delta(s_0, s_i) = 0, i \notin \{1, 2\}$.
$\delta(s_1, s_4) = 1/4$, $\delta(s_1, s_3) = 3/4$ y $\delta(s_1, s_i) = 0, i \notin \{3, 4\}$.

En general vamos a omitir para las figuras algunas transiciones (y otros detalles) que consideramos obvias por contexto. Por ejemplo en esta figura cada uno de los estados s_4, s_5

y s_6 tienen una única transición que genera un bucle hacia si mismos. Además marcamos las transiciones probabilísticas con flechas punteadas, para facilitar la visualización. También agregamos a las transiciones de los estados de los jugadores una etiqueta, para poder señalizarlas en el futuro. Por último marcamos con cuadrados de distintos colores algunos estados para hacer referencia en el futuro y decir que pertenecen a cierto conjunto.

Vemos que si nos quedamos con los estados s_1, s_3, s_4, s_5 y sus transiciones tenemos un Proceso de Decisión de Markov, ya que $V_1 = \emptyset$. También podemos ver que si solo nos quedamos con los estados s_2, s_5, s_6 y sus transiciones tenemos una Cadena de Markov ya que $V_1 = V_2 = \emptyset$.

Por lo general diremos que los estados pertenecientes a V_1 pertenecen al jugador 1, los de V_2 al jugador 2 y que los de V_P son estados probabilistas.

Definición 3 (Conjunto de sucesores y predecesores). *Para todos los estados $v \in V$, definimos*

$$post_\delta(v) = \{v' \in V \mid \delta(v,\ v') > 0\} \tag{2.6}$$

como el conjunto de sucesores de v. De manera similar,

$$pre_\delta(v) = \{v' \in V \mid \delta(v',\ v) > 0\} \tag{2.7}$$

como el conjunto de predecesores de v.

Por lo general omitiremos el superíndice δ ya que queda claro en el contexto. Además, vamos a asumir que $post(v) \neq \emptyset$ para todo $v \in V$. Tenemos como ejemplo, tomando la figura 2.1 $post(s_0) = \{s_1, s_2\}$ y $pre(s_5) = \{s_5, s_3, s_2\}$

También definimos las clausulas finitas de $post(v)$ y $pre(v)$, así también como la definición para un subconjunto B de V:

- $post_\delta(B) = \bigcup_{v \in B} post_\delta(v)$
- $post^*(B) = \bigcup_{k \geq 0} post^k_\delta(B)$, donde $post^0_\delta(B) = B$ y $post^{k+1}_\delta(B) = post_\delta(post^k_\delta(B))$
- $pre_\delta(B) = \bigcup_{v \in B} pre_\delta(v)$
- $pre^*(B) = \bigcup_{k \geq 0} pre^k_\delta(B)$, donde $pre^0_\delta(B) = B$ y $pre^{k+1}_\delta(B) = pre_\delta(pre^k_\delta(B))$
- $post^*(v) = post^*(\{v\})$
- $pre^*(v) = pre^*(\{v\})$

De nuevo ejemplificaremos con el juego $\mathcal{G}$ de la figura 2.1, para los estados s_1 y s_2 tenemos: $post^*(s_1) = \{s_3, s_4, s_5\}$, $pre^*(s_3) = \{s_0, s_1\}$. Y dados $S = \{s_1, s_2\}$ y $T = \{s_4, s_6\}$, tenemos: $post^*(S) = \{s_3, s_4, s_5, s_6\}$ y $pre^*(T) = \{s_0, s_1, s_2, s_3, s_4, s_6\}$

También vamos a fijar un estado inicial para un juego. Usaremos la notación $\mathcal{G}_v$ para indicar que un juego empieza por v. Cuando el estado inicial sea deducible por contexto omitiremos esta notación y usaremos simplemente $\mathcal{G}$. Por ejemplo en 2.1 es evidente que el estado inicial es s_0.

Por último decimos que un estado $v \in V$ es *terminal* o absorbente si $\delta(v,\ v) = 1$ y $\delta(v,\ v') = 0$ para toda $v \neq v'$. En 2.1 tenemos como estados terminales a s_4, s_5 y s_6.

Definición 4 (Componente Final). *Un componente final de $\mathcal{G}$ es un par (V', δ') tal que:*

a. $V' \subseteq V$.

b. $\delta'(v, \cdot) = \delta(v, \cdot)$ *para* $v \in V_P$.

c. $\emptyset \neq post^{\delta'}(v) \subseteq post_\delta(v)$ *para* $v \in V_1 \cup V_2$.

d. $post_\delta(v) \subseteq V'$ *para todo* $v \in V'$.

e. El grafo subyacente de (V', δ') es fuertemente conectado.

Nótese que un componente final también puede considerarse como un juego. El conjunto de componentes finales de $\mathcal{G}$ se denota como $EC(\mathcal{G})$.

Definición 5 (Camino). *Un camino en un juego $\mathcal{G}$ es una secuencia infinita de vértices $v_0 v_1 ...$ tal que*
$\delta(v_k, v_{k+1}) > 0$ para todo $k \in \mathbb{N}$. Para un Juego Estocástico $\mathcal{G}$ tenemos $Paths_\mathcal{G}$, el conjunto de todos los caminos de $\mathcal{G}$ y $FPaths_\mathcal{G}$ el conjunto de todos los prefijos finitos de caminos de $\mathcal{G}$. También tenemos $Paths_{\mathcal{G}_v}$, los caminos que empiezan en v, y el equivalente con $FPaths_{\mathcal{G}_v}$.

Un ejemplo de camino de 2.1 es $s_0 s_1 s_3 s_5 s_5 s_5 ...$. Este se da siguiendo las aristas β y γ, pasando por el medio por la elección probabilista con valor 3/4.

Definición 6 (Estrategia). *Una estrategia para el Jugador i (para $i \in 1,\ 2$) en un juego $\mathcal{G}$ es una función:*

$$\pi_i : V^* V_i \to \mathcal{D}(V) \tag{2.8}$$

que asigna una distribución de probabilidad para cada secuencia finita de estados tal que $\pi_i(\hat{\omega} v)(v') > 0$ solo si $v' \in post(v)$.

El conjunto de todas las estrategias para el Jugador i se denota con Π_i. Una estrategia π_i se le llama *pura* o *determinista* si por cada $(\hat{\omega} v) \in V^* Vi$, $\pi_i(\hat{\omega} v)$ es una distribución de Dirac y es llamado *sin memoria* (o *memoryless* en inglés) si $\pi_i(\hat{\omega} v) = \pi_i(v)$ para todo $\hat{\omega} \in V^*$. Diremos que los conjuntos Π_i^M y Π_i^D son los conjuntos de todas las estrategias memoryless y de todas las estrategias deterministas, respectivamente, para el jugador i. Definimos el conjunto de todas las estrategias memoryless y deterministas como $\Pi_i^{MD} = \Pi_i^M \cap \Pi_i^D$. Por lo general usaremos estrategias deterministas y se ignorará el superíndice D cuando esto sea evidente por contexto.

Un ejemplo de un par de estrategias memoryless deterministas π_1 y π_2 para los jugadores 1 y 2 en el grafo de la figura 2.1 puede ser:

$$\pi_1(s_0, s_1) = 1, \pi_1(s_0, s_i) = 0, i \neq 1$$

$$\pi_2(s_3, s_4) = 1, \pi_2(s_3, s_i) = 0, i \neq 4$$

Para describir más simplemente una estrategia memoryless determinista en lenguaje natural vamos a decir que el jugador i tomó el camino μ para el estado s_j. En el ejemplo dado, podemos decir que el jugador 1 eligió el camino β para el estado s_0 y que el jugador 2 eligió el camino δ para el estado s_3.

Dadas dos estrategias $\pi_1 \in \Pi_1$, $\pi_2 \in \pi_2$ y un estado inicial v, el *resultado* de un juego es una Cadena de Markov, que la denotaremos como $\mathcal{G}_v^{\pi_1,\pi_2}$.

La cadena subyacente del ejemplo anterior es la que podemos ver en la figura 2.2.

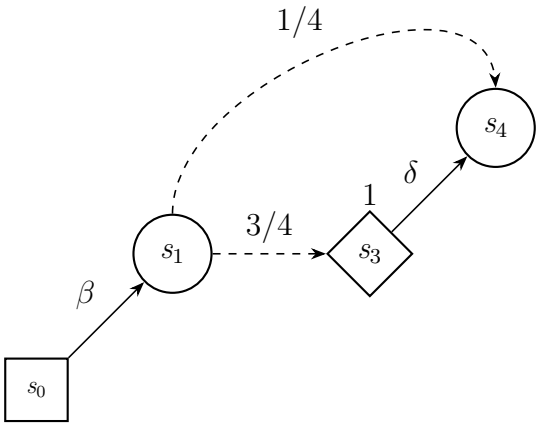


Figura 2.2: Cadena de Markov subyacente de fijar dos estrategias en 2.1

A continuación definiremos un conjunto llamado *evento*, el cual usaremos para definir la probabilidad de las propiedades que nos interesa demostrar.

Un evento $\mathcal{A}$ es un conjunto medible en la σ-álgebra de Borel generada por los conos de $Paths_{\mathcal{G}}$. El *cono* (también llamado *cilindro*) generado por el camino finito $\hat{\omega}$ en $FPaths_{\mathcal{G}}$ es el conjunto $cyl(\hat{\omega}) = \{\omega \in Paths_{\mathcal{G}} \mid \forall 0 \leq i < |\hat{\omega}| : \omega_i = \hat{\omega}_i\}$. Esto quiere decir que el conjunto derivado del camino finito $\hat{\omega}$ consiste en todos los caminos infinitos que empiezan con $\hat{\omega}$.

De nuevo, fijando dos estrategias π_1, π_2 y un nodo inicial v llamamos $Prob_v^{\pi_1,\pi_2}$ a la medida de probabilidad asociada al juego $\mathcal{G}$ [4]. Entonces intuitivamente $Prob_v^{\pi_1,\pi_2}(\mathcal{A})$ es la probabilidad de que las estrategias π_1 y π_2 generen un camino que pertenezca al conjunto $\mathcal{A}$ cuando el juego $\mathcal{G}$ empieza en v.

Se dice que un juego estocástico $\mathcal{G}$ tiene parada (*to be stopping* en inglés), si para todas las estrategias π_1, π_2 la probabilidad de llegar a un estado terminal es 1 [8]. Este concepto es de gran importancia para decidir si un juego está determinado, y lo veremos como condición necesaria en algunas de las estrategias que se han estudiado.

Decimos intuitivamente que un juego estocástico está determinado cuando sin importar el orden de elección de las acciones de los jugadores, se llega a un mismo valor para el

juego. El valor del juego depende del tipo de objetivo que se selecciona. Por ejemplo para objetivos de alcanzabilidad, el valor del juego es la esperanza de llegar al conjunto de estados finales. Para un objetivo de recompensas totales, puede ser el valor esperado de recompensa acumulada que se obtiene al final del juego. Veremos luego que hay diferentes formas de definir el valor de un juego dependiendo de las reglas que se aplican. Vamos a mostrar un ejemplo de esto en el siguiente capítulo para juegos multiobjetivo con orden lexicográfico. Allí se define el *lex-value*, un vector donde cada componente es la esperanza de alcanzar cada objetivo [5].

Definición 7 (Determinación). *Decimos formalmente que un juego $\mathcal{G}$ está determinado para un vértice v y para un objetivo ϕ si y solo si:*

$$\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \mathbb{E}^{\pi_1,\pi_2}_{\mathcal{G},v}(\phi) = \inf_{\pi_2 \in \Pi_2} \sup_{\pi_1 \in \Pi_1} \mathbb{E}^{\pi_1,\pi_2}_{\mathcal{G},v}(\phi) \tag{2.9}$$

Es decir, que tenemos el mismo valor esperado fijando primero la estrategia óptima del minimizador y luego dejando que el maximizador elija su mejor estrategia contraria, a que si el maximizador elige primero su mejor estrategia y luego el minimizador elige luego su mejor contra estrategia.

Una estrategia óptima para un jugador i es una que para cualquier contra estrategia del otro jugador, se tiene el mejor valor esperado (ya sea el máximo para el maximizador o el mínimo para el minimizador). Formalemente:

Definición 8 (Estrategia óptima. Jugador 1). *Dado un juego estocástico $\mathcal{G}$ y $\pi_2 \in \Pi_2$, y un objetivo ϕ a maximizar, decimos que una estrategia óptima para el jugador 1 (maximizador) es cuando existe una estrategia $\pi_1^* \in \Pi_1$ tal que*

$$\mathbb{E}^{\pi_1^*,\pi_2}_{\mathcal{G},v}(\phi) = \sup_{\pi_1 \in \Pi_1} \mathbb{E}^{\pi_1,\pi_2}_{\mathcal{G},v}(\phi) \tag{2.10}$$

Similarmente definimos una estrategia óptima para el jugador 2 como:

Definición 9 (Estrategia óptima. Jugador 2). *Dado un juego estocástico $\mathcal{G}$ y $\pi_1 \in \Pi_1$, y un objetivo ϕ a minimizar, decimos que una estrategia óptima para el jugador 2 (minimizador) es cuando existe una estrategia $\pi_2^* \in \Pi_2$ tal que*

$$\mathbb{E}^{\pi_1\pi_2^*}_{\mathcal{G},v}(\phi) = \inf_{\pi_2 \in \Pi_2} \mathbb{E}^{\pi_1,\pi_2}_{\mathcal{G},v}(\phi) \tag{2.11}$$

2.2. Notación LTL

Para representar conjuntos específicos de caminos usamos notación LTL (por sus siglas en inglés *Linear Temporal Logic*). La notación LTL nos proporciona una forma precisa de expresar propiedades sobre la relación de estados en el tiempo, dada una ejecución. En la notación LTL se agregan a los operadores de lógica proposicional dos importantes operadores: $\Diamond$ "eventually"(eventualmente en el futuro) y $\Box$ "always"(ahora y siempre en el futuro). La siguiente es la definición intuitiva de $\Diamond$ y $\Box$: $\Diamond\varphi$ nos asegura que φ va a ser verdadero eventualmente en el futuro. $\Box\varphi$ se satisface si y solo si no se da $\neg\varphi$ eventualmente en el futuro. Esto es equivalente a decir que φ es verdadero desde ahora y para siempre. Para mayor detalle en la notación LTL, ver [2].

2.3. Objetivos

Como se ha mencionado, se mostraran juegos estocásticos con diferentes tipos de objetivos. Un objetivo es una propiedad que uno de los jugadores quiere cumplir, en nuestro caso el jugador uno o maximizador.

La primera propiedad que nos interesa analizar en el trabajo es la de alcanzabilidad, es decir de llegar a un conjunto de estados. La formalizaremos de la siguiente manera:

Definición 10 (Alcanzabilidad). *Dado $\mathcal{G} = (V, (V_1, V_2, V_P), \delta)$ un juego estocástico y $T \subseteq V$ un conjunto de estados, la propiedad de alcanzabilidad se denota simplemente con $\Diamond T$. Donde $\Diamond T = \{s_0 s_1 \ldots \in Paths_{\mathcal{G}} \mid \exists i \geq 0 \colon s_i \in T\}$*

El valor $\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} Prob_v^{\pi_1,\pi_2}(\Diamond T)$ de los objetivos de alcanzabilidad pueden calcularse usando la función $f^{\Diamond}$, que introduce las ecuaciones de Bellman, definidas para cada $s \in V$ por:

$$f^{\Diamond}(x_s) = \begin{cases} \max\{x_{s'} \mid s \to s'\}, & \text{si s es maximizador y } s \notin T \cup N \\ \min\{x_{s'} \mid s \to s'\}, & \text{si s es minimizador y } s \notin T \cup N \\ \sum_{s' \in V_p} \delta(s, s') \cdot x_{s'}, & \text{si s es probabilista y } s \notin T \cup N \\ 1, & \text{si } s \in T \\ 0, & \text{si } s \in N \end{cases} \tag{2.12}$$

donde T es el conjunto objetivo y N el conjunto de vértices desde donde T no puede ser alcanzado, formalmente $N = \{s \in S \mid s \notin pre^*(T)\}$.

En [8] se demuestra que existe una estrategia determinista sin memoria óptima para alcanzabilidad, lo cual nos será de utilidad para nuestro caso concreto. Además se prueba en este trabajo que los juegos de un solo objetivo están determinados para los juegos estocásticos simples.

Luego de definir alcanzabilidad, nos interesa presentar la propiedad de recompensas totales. Esta nos permite contar cuantos puntos se recolectan en un juego.

Definición 11 (Recompensas totales). *Una función de recompensa es una función medible $f^{rew} : V^{\infty} \to \mathbb{R}$. Definamos $\mathbb{E}_{\mathcal{G},v}^{\pi_1,\pi_2}[f^{rew}]$ como el valor esperado de la función*

medible f^{rew} *bajo la probabilidad* $Prob_{\mathcal{G},v}^{\pi_1,\pi_2}$. *La ecuación que usaremos para definir esta recompensa es la siguiente:*

$$f^{rew}(\sigma) = \begin{cases} \lim_{n\to\infty} \sum_{i=0}^{n} rew(\sigma_i), & si\ \sigma \vDash \Diamond T \\ 0, & si\ \sigma \nvDash \Diamond T \end{cases} \tag{2.13}$$

dados $\sigma \in V^\infty$ *y* σ_i *la i-ésima posición de* σ.

Para poder definir el objetivo de recompensas totales consideramos la función dada $rew : V \to \mathbb{R}^+$. Esta le da una recompensa positiva a cada nodo, que es la que el maximizador quiere recolectar. Vamos a requerir, además, que $rew(v) = 0$ para todo nodo terminal.

Para el ejemplo de 2.1 tenemos la función *rew* definida por las etiquetas de los nodos. Aquellos nodos que no están etiquetados tienen recompensa 0.

Entonces tenemos $rew(s_2) = 100$, $rew(s_3) = 1$ y $rew(s_i) = 0$ si $i \notin \{2, 3\}$. Por lo tanto, para $T = \{s_4, s_6\}$ y $\sigma = s_0 s_1 s_3 s_4 s_4 ...$ tenemos $f^{rew}(\sigma) = 1$. En cambio si $T = \{s_5\}$ tenemos $f^{rew}(\sigma) = 0$ ya que $s_0 s_1 s_3 s_4 s_4 ... \nvDash \Diamond\{s_5\}$.

Ahora definiremos una estrategia equitativa o justa (fair strategy), lo cual nos será útil para entender una de las tácticas que son utilizadas para trabajar con juegos estocásticos con recompensas totales.

Definición 12 (Jugadas justas). *Dado un juego estocástico* $\mathcal{G} = (V, (V_1, V_2, V_P), \delta)$ *el conjunto de jugadas justas para el jugador 2, denotado* (FP^2) *se define de la siguiente manera:*

$$FP^2 = \{\omega \in Paths_{\mathcal{G}} \mid \forall v' \in V_2 : v' \in \text{ínf}(\omega) \Rightarrow \text{post}\,(v') \subseteq \text{ínf}(\omega)\} \tag{2.14}$$

Entonces para un juego estocástico $\mathcal{G}$ diremos que una estrategia $\pi_2 \in \Pi_2$ es casi seguro justa (o simplemente justa) si y solo si se mantiene que para todo $\pi_1 \in \Pi_1$ y $v \in V$: $Prob_{\mathcal{G},v}^{\pi_1,\pi_2}(FP^2) = 1$.

En [3] se define cuando un juego tiene parada bajo jugadas justas y se prueba que para juegos estocásticos con parada garantizada dado un minimizador justo, la condición de recompensas totales está determinada.

Capítulo 3

Juegos multiobjetivo

En esta sección vamos a hablar de distintos tipos de juegos estocásticos. Se definirán y se mostrarán estrategias para abordar juegos estocásticos con múltiples objetivos, con objetivos probabilísticos, booleanos y juegos estocásticos con objetivos con orden lexicográfico.

3.1. ¿Qué es un juego multiobjetivo?

Un juego multiobjetivo es un tipo de juego estocástico donde el maximizador debe cumplir múltiples objetivos, que pueden ser de diferentes tipos y pueden potencialmente ser conflictivos [5].

3.2. Objetivos probabilísticos

En [6] se presentan los juegos estocásticos con objetivos de precisión, es decir, donde uno de los jugadores quiere alcanzar exactamente un valor esperado de la función objetivo. Por otro lado el objetivo del segundo jugador es que no se consiga este valor esperado.

Para el jugador que quiere conseguir el valor esperado (lo llamaremos **Preciser**) se construye el objetivo utilizando dos objetivos probabilísticos. Esto se logra haciendo que para un objetivo de, por ejemplo, alcanzabilidad, se tomen las estrategias maximizadoras y minimizadoras. Se puede ver esto simplemente dado un juego estocástico donde el jugador minimizador (**Spoiler**) no tenga ningún estado ($V_2 = \emptyset$), lo que nos deja con un MDP. Dadas estas estrategias, se consigue un valor esperado de alcanzabilidad de l (maximizando) y h (minimizando). Entonces cualquier valor $x \in [l, h]$ se puede conseguir *precisamente* eligiendo las estrategias minimizadoras y maximizadoras con cierta probabilidad Θ y $(1 - \Theta)$. Cabe remarcar que las estrategias de los jugadores son no deterministas en esta solución.

Se demuestra luego a lo largo del trabajo que por más que sea intuitivo pensar que este resultado se puede llevar a cabo con los juegos estocásticos, esto no es posible, ya que cuando el **Preciser** fija una estrategia, el **Spoiler** puede elegir cualquier estrategia subóptima para asegurarse que la función objetivo tenga diferente valor al esperado.

Dado esto, queda demostrado que los juegos estocásticos multiobjetivos *en general* no están determinados (veremos algunos casos concretos en los que sí lo están).

Para la demostración de la no determinación en los juegos estocásticos con objetivos precisos, se definen dos problemas: el problema de síntesis del controlador y el problema de la estrategia de contraataque (en inglés *Controller synthesis problem* y *Counter-strategy problem*).

Intuitivamente el problema de contraataque es decidir si **Spoiler** no tiene estrategia ganadora, es decir que para cada estrategia de este, existe una estrategia de **Preciser** que logra el objetivo. Por el otro lado, el problema de síntesis es decidir si **Preciser** tiene estrategia ganadora. Es decir que se pueda fijar una estrategia de este y para cualquier estrategia de **Spoiler**, siempre se logra el objetivo.

Para alguna función objetivo y un valor objetivo de esa función, se dice que el juego $\mathcal{G}$ está determinado si una respuesta positiva del problema de contraataque implica una respuesta positiva a la del problema de síntesis. Es decir, que si **Spoiler** no tiene estrategia ganadora, entonces **Preciser** tiene estrategia ganadora.

3.3. Objetivos booleanos

En [7] se presentan los juegos estocásticos donde el objetivo de un jugador es satisfacer una fórmula dada como una combinación booleana positiva de objetivos de recompensa total esperada, mientras que el objetivo del otro jugador es impedir esto. La idea es que las fórmulas booleanas expresan una combinación de funciones de recompensa total con limites superior o inferior para la recompensa esperada.

Se prueba a lo largo del trabajo que el problema de encontrar una estrategia determinista y ganadora es indecidible. Para algunos casos en concreto, como el de disyunción de objetivos, se muestra que estrategias MD para el jugador 1 son suficientes para que gane y que la existencia de tales estrategias es un problema NP-completo.

Una fórmula multiobjetivo (MQ por las siglas de multi-objective query) ϕ es una combinación booleana positiva, es decir conjunciones y disyunciones de predicados (objetivos) de la forma $r \bowtie v$ donde r es una función de recompensa, $v \in \mathbb{Q}$ es un limite y $\bowtie \in \{\geq, \leq\}$ es un operador de comparación. La validez de una MQ se define inductivamente en la estructura de la fórmula, un objetivo $r \bowtie v$ es verdadero en un estado s de $\mathcal{G}$ bajo un par de estrategias (π_1, π_2) si y solo si $\mathbb{E}_{\mathcal{G},s}^{\pi_1,\pi_2}[rew(r)] \bowtie v$ y el valor de las disyunciones y conjunciones se definen de manera directa. Se dice que el jugador 1 gana el juego en un estado s si tiene una estrategia π_1 tal que para toda estrategia π_2 del jugador 2 la fórmula ϕ evalúa a verdadero bajo (π_1, π_2).

Una MQ ϕ es una fórmula de conjunciones (CQ) si es una conjunción de objetivos. Es una fórmula de disyunciones (DQ) si es una disyunción de objetivos. Nos importan estas definiciones porque veremos que para fórmulas tan simples como CQ de dos objetivos, ya es difícil encontrar una estrategia óptima.

Se pueden enriquecer las fórmulas MQ con objetivos de *alcanzabilidad*. Se nombra en [7] que es posible reducir fórmulas MQ con alcanzabilidad a fórmulas con recompensas totales. Esta noción de reducción de objetivos de alcanzabilidad a objetivos de recompensa total es una idea conocida que utilizaremos a lo largo del trabajo.

El primer teorema de [7] es sobre la no determinación en MQ. Este dice que los juegos estocásticos con múltiples objetivos no están determinados en general, y que las estrategias óptimas pueden no existir, incluso para las CQ de dos objetivos. Este resultado se deduce directamente del trabajo [6], ya que el caso de estudio de este (alcanzar un conjunto

de estados terminales T con probabilidad p) se puede representar como una CQ de dos objetivos: $\phi = f^{\Diamond}(x_s) \geq p \wedge f^{\Diamond}(x_s) \leq p$, donde x_s es el estado inicial.

3.4. Juegos multiobjetivo con orden lexicográfico

El orden lexicográfico es un orden de prioridad que se da sobre los objetivos. Esto ayuda a la hora de resolver los posibles objetivos conflictivos que puede tener un juego. El orden lexicográfico de los objetivos es útil en muchos escenarios. Por ejemplo, podemos tener el caso de un vehículo autónomo que puede tener como principal objetivo evitar choques y como objetivo secundario optimizar el rendimiento de la energía utilizada para viajar. La idea principal es que dado un orden lexicográfico sobres los objetivos, se los puede considerar secuencialmente. Dado esto, después de considerar cada objetivo, se remueven las acciones que no son óptimas para este objetivo, para cada jugador. Haciendo que en las próximas iteraciones solo se consideren la acciones óptimas localmente. En [5] se introduce un algoritmo para la solución de este tipo de problemas para objetivos de alcanzabilidad y *safety*. Vamos a dar algunas de las definiciones básicas de este tipo de juegos, para tener una base y luego poder mostrar qué ideas son útiles y cuáles no en el caso de juegos multiobjetivo con nuestras características.

Dados dos vectores x e y de largo n, definimos el orden lexicográfico $\leq_{\mathbf{lex}}$ en un espacio de n objetivos $\mathbb{R}^n$ como $x \leq_{\mathbf{lex}} y$ si y solo si $x_i \leq y_i$ donde $i \leq n$ es la mayor posición tal que para todo $j < i$ se mantiene que $x_j{=}y_j$. Es decir que la i-ésima posición hace el desempate para determinar que vector es mayor al otro. Notar que existen el supremo y el ínfimo en el orden lexicográfico para conjuntos arbitrarios $S \subseteq [0,\ 1]^n$ [5] .

También daremos la definición de un objetivo lexicográfico y del valor lexicográfico de un objetivo (lexicográfico) en un estado. Un objetivo lexicográfico de alcanzabilidad-safety es un vector $\boldsymbol{\Omega} = (\Omega_1, \ldots, \Omega_n)$ tal que $\Omega_i \in \{\, \mathbf{Reach}\,(V_i)\,, \mathbf{Safe}\,(V_i)\}$ con $V_i \subseteq V$ para todo $1 \leq i \leq n$. El valor lexicográfico de $\boldsymbol{\Omega}$ en el estado $s \in V$ se define como:

$$^{\boldsymbol{\Omega}}\mathbf{v}^{\mathrm{lex}}(s) = \sup_{\sigma \in \Pi_1} \inf_{\tau \in \Pi_2} \mathbb{P}_s^{\sigma,\tau}(\boldsymbol{\Omega})$$

donde $\mathbb{P}_s^{\sigma,\tau}(\boldsymbol{\Omega})$ denota al vector $(\mathbb{P}_s^{\sigma,\tau}\,(\Omega_1)\,, \ldots, \mathbb{P}_s^{\sigma,\tau}\,(\Omega_n))$ y el supremo y el ínfimo se toman con respecto al orden $\leq_{\mathrm{lex}}$ en $[0,1]^n$. Entonces el valor lexicográfico en el estado s es el vector de probabilidades supremo lexicográfico que el maximizador puede asegurar contra todas las posibles estrategias del minimizador. A lo largo del trabajo de orden lexicográfico [5] se prueba que estos pueden ser intercambiados, es decir que el juego está determinado para un juego multiobjetivo de alcanzabilidad-safety con orden lexicográfico y algunas condiciones dadas.

Capítulo 4

Objetivos prioritarios en entornos adversariales

En este capítulo hablaremos de juegos estocásticos con dos objetivos. El primer objetivo es de alcanzabilidad de un conjunto de estados exitosos y el segundo es de recompensas totales. Con un detalle, diremos que si no se cumple el objetivo de alcanzabilidad, la recompensa total obtenida es 0. De este modo, para el jugador 1 o maximizador, que quiere cumplir los objetivos, habrá un orden de prioridades, como se mostró en 3.4. Donde lógicamente el objetivo de alcanzabilidad es el principal y el de maximización de recompensas es secundario. En cambio, para el jugador 2 o minimizador, no hay un orden específico para estos objetivos, puede deliberadamente elegir entre minimizar alcanzabilidad, recompensas totales o una combinación de estos. Este tiene una conducta adversarial, como hemos visto en los ejemplos del capítulo 3. Además cabe recordar que trabajaremos con un caso en el que el juego estocástico tiene recompensas no negativas y que los estados que pertenecen a los conjuntos de terminación exitosa y no exitosa son estados finales con recompensa 0. Esto quiere decir que al llegar a un estado terminal no aumenta la recompensa.

Veremos en este capítulo las diferencias con las distintas estrategias que fuimos mostrando en el capítulo anterior, así como unos ejemplos que se fueron construyendo para presentar mejor la problemática. También daremos nuestra estrategia para abordarla, con su correspondiente algoritmo. Luego en el capítulo 5 se mostrarán los distintos experimentos efectuados utilizando el algoritmo.

4.1. Problemática

Como dijimos, el objetivo principal cuando se trabaja con juegos estocásticos es poder afirmar o negar la hipótesis de que un tipo de juego en concreto sea determinado (2.9) o no. Para eso, tenemos que dar las estrategias óptimas para cada uno de los jugadores. Para nuestro caso definimos como el valor a optimizar a la esperanza de la recompensa total, condicionada a alcanzar los estados de terminación exitosa, dado que se eligieron las estrategias óptimas para alcanzabilidad para el maximizador. Es decir, que dadas las estrategias maximizadoras de alcanzabilidad, se eligen las estrategias que maximizan la esperanza de la recompensa condicionada al objetivo de alcanzabilidad. Elegimos restringir

primero las estrategias de alcanzabilidad ya que queremos que la probabilidad de alcanzar el objetivo sea la mayor posible, antes de maximizar la recompensa condicionada. La esperanza de la recompensa *condicionada* es necesaria para el algoritmo que consigue las estrategias maximizadoras de recompensa, esta decisión será explicada en profundidad en 4.5.

Para explicar mejor el porqué recortamos las estrategias del jugador 1 que maximizan alcanzabilidad antes de calcular las recompensas totales podemos ver en la figura 4.1. Aquí la esperanza de la recompensa condicionada de tomar el camino α es mucho mayor que la de tomar el camino β. Ya que la primera tiene una recompensa de 10^{25} y la segunda de 1. Pero vemos que la probabilidad de llegar al estado objetivo s_6 es tan baja por el camino α, que realmente no tiene sentido tomar esa elección. En el caso en que decidimos utilizar las estrategias maximizadoras de alcanzabilidad antes de maximizar la esperanza condicionada, se elige el camino con β, ya que el otro camino es descartado. En cambio si se elige maximizar la esperanza condicionada sin restringir previamente las estrategias que maximizan alcanzabilidad, se elegiría el camino con α. Nosotros deseamos el primer caso, lo que implica que tenemos un orden lexicográfico de los objetivos para el jugador maximizador. Si pensamos en un ejemplo del mundo real, en el que el objetivo principal es llegar sin accidentes a un destino y el objetivo secundario es minimizar el tiempo de llegada, podemos ver intuitivamente que vamos a querer elegir un camino que nos tome más tiempo, si es 99 % seguro que llegaremos a salvo, en comparación con un camino más rápido que tiene una probabilidad de 1 % de llevarnos a salvo a destino.

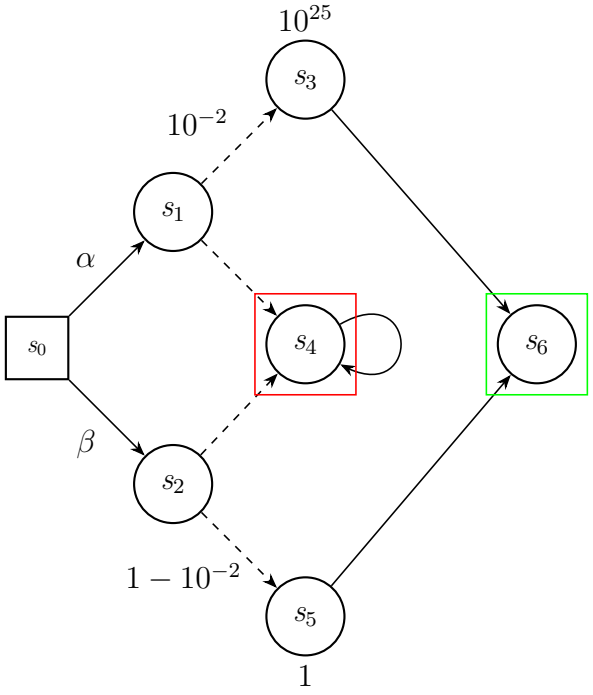


Figura 4.1: Juego estocástico con una elección.

A continuación veremos como calcular la esperanza de la recompensa y la esperanza de la recompensa condicionada al objetivo de alcanzabilidad.

4.1.1. Esperanza de la función de recompensa

Teniendo en cuenta la función f^{rew} definida en 2.13, y dado T el conjunto de estados que representan una terminación exitosa, $\pi_1 \in \Pi_1$ y $\pi_2 \in \Pi_2$, diremos que $\mathbb{E}^{\pi_1,\pi_2}(f^{rew} \mid \Diamond T)$ representa la esperanza de la función de recompensa condicionada a la probabilidad de alcanzar un estado exitoso, para un par de estrategias π_1 y π_2.

Para el cálculo de la Esperanza condicionada, primero tenemos que conocer el valor de le Esperanza de la función de recompensa $\mathbb{E}^{\pi_1,\pi_2}(f^{rew})$. Vamos a demostrar que para un par de estrategias π_1 y π_2 esta es igual a la sumatoria de las probabilidades de los caminos $\hat{\rho}$ que llegan al conjunto objetivo, multiplicado por la recompensa total recolectada a lo

largo de dicho camino. Es decir:

$$\mathbb{E}^{\pi_1,\pi_2}(f^{rew}) = \sum_{\hat{\rho}\in(B\setminus T)\mathbf{U}T} Prob^{\pi_1,\pi_2}(\hat{\rho}) \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i) \tag{4.1}$$

Para demostrar esto definiremos algunos conjuntos, funciones y lemas que nos serán útiles a lo largo del proceso. Primero definiremos usando notación LTL que $B = \Diamond T$, es decir B será el conjunto de trazas que alcanzan al conjunto T. También definimos $\mu^{\pi_1,\pi_2}(cyl(\hat{\rho})) = Prob^{\pi_1,\pi_2}(\hat{\rho})$ Como la probabilidad de un conjunto de trazas infinitas.

Con esto en mente, usaremos un lema con dos partes, que nos serán útiles para la definición de la esperanza de la función de recompensa.

Lema 4.1.1. *Sea* $\hat{\rho} \in (B \setminus T)\mathbf{U}T$

(a) $\mu^{\pi_1,\pi_2}(cyl(\hat{\rho}) \cap (V^* T^\omega)^c) = 0$

(b) si $\rho \in T^\omega$, $f^{rew}(\hat{\rho}\rho) = \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i)$

Demostración. Primero vemos para la parte (a) del lema que por definición, si $\hat{\rho} \in (B \setminus T)\mathbf{U}T$, los elementos de $cyl(\hat{\rho})$ son de la forma $s_0 s_1 ...\ t\ s_n s_{n+1} ...$ con $t \in T$. Vemos que la segunda parte de la intersección de (a) son trazas que o no llegan al objetivo, o que luego de llegar a este, no se quedan infinitamente en él. Por eso, la intersección tiene solo a elementos de $cyl(\hat{\rho})$, pero que no se quedan infinitamente en T.

Como dijimos al principio del capítulo los estados exitosos son estados terminales, por lo que nuestras trazas que tienen un estado exitoso, solo tienen el mismo estado infinitamente luego de este. Esto hace que en la intersección solo haya trazas que no son posibles en nuestro caso de estudio, por lo que tienen probabilidad 0.

Luego para (b), usando la definición de f^{rew} y separando la sumatoria en dos, la primera hasta $\hat{\rho}$, y la segunda con ρ, obtenemos:

$$\begin{aligned} f^{rew}(\hat{\rho}\rho) &= \lim_{n\to\infty} \sum_{i=0}^{n} rew((\hat{\rho}\rho)_i) \\ &= \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i) + \lim_{n\to\infty} \sum_{i=0}^{n} rew(\rho_i) \\ &= \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i) \end{aligned}$$

En el último paso borramos el segundo sumando ya que al ser $\rho \in T^\omega$, tenemos que para todo $v \in T$, $rew(v) = 0$, entonces $rew(\rho_i) = 0$ para todo i. □

Demostrado el lema, podemos continuar con la demostración de 4.1. Comenzando por la definición de la esperanza para un valor continuo 4.2.

$$\mathbb{E}^{\pi_1,\pi_2}(f^{rew}) = \int_{\rho \in V^\omega} f^{rew}(\rho)\, d\mu^{\pi_1,\pi_2}(\rho) \tag{4.2}$$

$$= \left[\sum_{\hat{\rho} \in (B \setminus T)^* T} \int_{\rho \in cyl(\hat{\rho})} f^{rew}(\rho)\, d\mu^{\pi_1,\pi_2}(\rho) \right]$$

$$+ \int_{\rho \notin \Diamond T} f^{rew}(\rho)\, d\mu^{\pi_1,\pi_2}(\rho) \tag{4.3}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \int_{\rho \in cyl(\hat{\rho})} f^{rew}(\rho)\, d\mu^{\pi_1,\pi_2}(\rho) \tag{4.4}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \int_{\rho \in cyl(\hat{\rho}) \cap (V^* T^\omega)} f^{rew}(\rho)\, d\mu^{\pi_1,\pi_2}(\rho)$$

$$+ \int_{\rho \in cyl(\hat{\rho}) \cap (V^* T^\omega)^c} f^{rew}(\rho)\, d\mu^{\pi_1,\pi_2}(\rho) \tag{4.5}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \int_{\rho \in cyl(\hat{\rho}) \cap (V^* T^\omega)} f^{rew}(\rho)\, d\mu^{\pi_1,\pi_2}(\rho) \tag{4.6}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \int_{\rho \in cyl(\hat{\rho}) \cap (V^* T^\omega)} \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i)\, d\mu^{\pi_1,\pi_2}(\rho) \tag{4.7}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i) \int_{\rho \in cyl(\hat{\rho}) \cap (V^* T^\omega)} d\mu^{\pi_1,\pi_2}(\rho) \tag{4.8}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i)\ \mu^{\pi_1,\pi_2}(cyl(\hat{\rho}) \cap (V^* T^\omega)) \tag{4.9}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i)\ \mu^{\pi_1,\pi_2}(cyl(\hat{\rho})) \tag{4.10}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i)\ Prob^{\pi_1,\pi_2}(\hat{\rho}) \tag{4.11}$$

$$= \sum_{\hat{\rho} \in (B \setminus T)^* T} Prob^{\pi_1,\pi_2}(\hat{\rho}) \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i) \tag{4.12}$$

Seguimos en 4.3 dividiendo la integral entre los caminos que pertenecen a $\Diamond T$ y los que no. Dividimos los primeros utilizando los cilindros de $\hat{\rho}$. En 4.4 se elimina lo que previamente era la segunda integral ya que tiene valor 0. Esto es porque $f^{rew}(\rho) = 0$ para $\rho \notin \Diamond T$. En 4.5 se divide el conjunto de $cyl(\hat{\rho})$ entre los que intersecan con $(V^* T^\omega)$ y los que intersecan con su complemento, para poder utilizar el lema que definimos previamente. En 4.6 eliminamos la segunda parte del sumando, ya que por lema 4.1.1(a) tenemos $\mu^{\pi_1,\pi_2}(\rho) = 0$ para $\rho \in cyl(\hat{\rho}) \cap (V^* T^\omega)^c$.

A continuación usamos el lema 4.1.1(b) para el paso 4.7. Ya que $\rho \in cyl(\hat{\rho}) \cap (V^*T^\omega)$ implica que ρ tiene la forma $\hat{\rho}\rho'$ con $\rho' \in T^\omega$. Puede darse el caso en que $\rho' \in (V^*T^\omega)$, con alguna cantidad finita de elementos de $(V \setminus T)$ antes de los elementos de T. Pero como dijimos previamente en la demostración del lema 4.1.1(a), la probabilidad de que una traza tenga algún elemento de T y luego continúe con elementos de $(V \setminus T)$ es 0 ya que los estados exitosos son terminales. Como este caso entonces suma 0 a la esperanza, lo eliminamos de la ecuación. En 4.8 sacamos la sumatoria porque es una constante en la integral y en 4.9 hacemos la integración. En 4.10, cambiamos $\mu^{\pi_1,\pi_2}(cyl(\hat{\rho}) \cap (V^*T^\omega))$ por $\mu^{\pi_1,\pi_2}(cyl(\hat{\rho}))$. Ya que por lema (a) $\mu^{\pi_1,\pi_2}(cyl(\hat{\rho}) \cap (V^*T^\omega)^c = 0$, entonces podemos hacer la unión de este con el primero y quedarnos con $\mu^{\pi_1,\pi_2}(cyl(\hat{\rho}))$. Luego en 4.11 usamos la definición de μ^{π_1,π_2} para reemplazarla por $Prob^{\pi_1,\pi_2}$. Por último en 4.12 como $Prob^{\pi_1,\pi_2}(\hat{\rho})$ es una constante de la segunda sumatoria, la dejamos afuera de esta por distributividad.

Así llegamos a la igualdad de 4.1, ahora continuaremos con la esperanza de la recompensa condicionada al objetivo de alcanzabilidad.

4.1.2. Esperanza condicionada

Definida $\mathbb{E}^{\pi_1,\pi_2}(f^{rew})$ correctamente, daremos una simplificación de $\mathbb{E}^{\pi_1,\pi_2}(f^{rew} \mid \Diamond T)$ para poder calcularlo con facilidad. Primero, usando la definición de probabilidad condicionada de [1, p.212 (3)] Tenemos que:

$$\mathbb{E}^{\pi_1,\pi_2}(f^{rew} \mid \Diamond T) = \frac{\mathbb{E}^{\pi_1,\pi_2}(f^{rew} I_{\Diamond T})}{Prob^{\pi_1,\pi_2}(\Diamond T)}$$

Luego, por definición de la función característica $I_{\Diamond T}$ y la definición de f^{rew} 2.13, tenemos:

$$f^{rew} I_{\Diamond T}(\rho) = f^{rew}(\rho) \cdot I_{\Diamond T}(\rho) = \begin{cases} f^{rew}(\rho) & si\ \rho \in \Diamond T \\ 0 & si\ \rho \notin \Diamond T \end{cases} = f^{rew}(\rho)$$

Gracias a estas dos ecuaciones llegamos a que la esperanza condicionada es igual a la esperanza de la función de recompensa dividida entre la probabilidad de llegar a los estados finales, para un par de estrategias π_1 y π_2.

$$\mathbb{E}^{\pi_1,\pi_2}(f^{rew} \mid \Diamond T) = \frac{\mathbb{E}^{\pi_1,\pi_2}(f^{rew})}{Prob^{\pi_1,\pi_2}(\Diamond T)} \tag{4.13}$$

Utilizando 4.1 y 4.13, tenemos finalmente que la fórmula que utilizaremos para el calculo de la esperanza de la recompensa condicionada a llegar al objetivo de alcanzabilidad es la siguiente:

$$\mathbb{E}^{\pi_1,\pi_2}(f^{rew} \mid \Diamond T) = \frac{\sum_{\hat{\rho} \in (B \setminus T)\mathbf{U}\, T} Prob^{\pi_1,\pi_2}(\hat{\rho}) \sum_{i=0}^{|\hat{\rho}|-1} rew(\hat{\rho}_i)}{Prob^{\pi_1,\pi_2}(\Diamond T)} \tag{4.14}$$

4.2. Diferencias metodológicas con investigaciones previas

Hemos mencionado en el capítulo 3 varias formas de abarcar los problemas que queremos estudiar en esta investigación. Ahora mostraremos por qué no utilizamos estas estrategias. En particular, veremos por qué no obligamos al jugador 2 a ser justo (agregar la condición de *fairness*) y por qué tampoco usamos el algoritmo de orden lexicográfico para resolver el problema de multiobjetivo.

4.2.1. Adversario justo

Como introdujimos en el ejemplo motivacional 1.2, el jugador 2 puede actuar de forma tal que provoque al juego a quedarse infinitamente en un grupo de casillas, sin poder progresar. Por lo que es imposible cumplir el objetivo de alcanzabilidad, pero a la vez no se detiene el cálculo de recompensas. Como ya dijimos, el paper [3] soluciona este problema haciendo que el minimizador juegue de forma equitativa (*fairness*). Esto implica que para cada elección que se dé infinitas veces, se tomarán todas las decisiones infinitas veces también.

En nuestro caso tenemos dos formas de solucionar el problema de estos bucles infinitos creados por el minimizador. Estas nos dan una noción de equitatividad y evita la necesidad de agregar *fairness* al jugador 2, que complejiza los cálculos para múltiples objetivos.

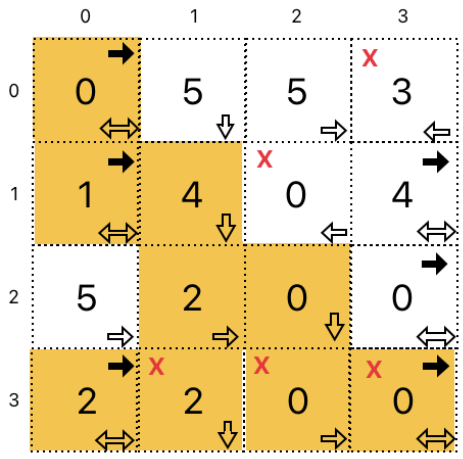


Figura 4.2: Ejemplo de un tablero 4x4 con flechas hacia abajo. Jugador 2 minimiza alcanzabilidad.

Primero, la introducción de la regla en que el minimizador puede fallar. Esto hace que por más que tome la elección de mantenernos en el mismo grupo de casilleros sin permitirnos avanzar, existe una probabilidad p de que su elección no importe, porque se averió. Cómo explicamos anteriormente, en este caso el jugador 1 puede tomar cualquier decisión, ya sea avanzar o recolectar más puntos moviéndose a los lados. Sabiendo que tiene una probabilidad de fallar, el semáforo toma decisiones distintas a las que tomaría en otro caso, esta vez intentando guiar a Roborta hacia los casilleros que están flojos para impedir que llegue al final del tablero.

La otra forma en que se soluciona el problema es agregando nuevos tipos de casillas al ejemplo de Roborta. Estas casillas permiten el progreso en el juego, ya que obligan al semáforo a darnos solo la elección de bajar. Estos tipos de casilleros son agregados en los grupos de estados en los que el semáforo puede mantener infinitamente a Roborta. Un ejemplo de esto es el que dimos en la introducción (1.1), en donde se hace un bucle entre las casillas (0,0) y (0,1). En 4.2 agregamos los casilleros que obligan a bajar al tablero. Logrando que al empezar desde el casillero inicial arriba a la izquierda, Roborta siempre pueda tomar una elección que la lleve al final del tablero. En este tablero tenemos al igual que en la introducción 1 casillas con las elecciones de Roborta con una flecha de color negro en la esquina superior izquierda

y casillas amarillas que marcan el camino que dió como resultado correr el algoritmo. Como explicaremos en la siguiente sección, el jugador 2 puede elegir minimizar el objetivo de alcanzabilidad o las recompensas totales, en este caso se concentra en lo primero, pero sin ignorar las recompensas, si tiene la posibilidad de minimizarlas sin aumentar la esperanza de llegar a un estado exitoso.

Recordemos que cada vez que Roborta pasa por una casilla marcada con una cruz roja hay una probabilidad p de que esta *se rompa*, haciendo que el juego termine de manera no exitosa.

Entonces en el juego, tenemos que en el primer casillero el semáforo elige ponerse en verde, haciendo que Roborta baje a la siguiente fila. Esto es porque si se pone en amarillo, ella se moverá hacia la derecha e inevitablemente bajará pero habiendo recolectado más puntos. En el casillero (0,1) se pone en amarillo, evitando los 5 puntos de debajo y esto lo lleva en un par de turnos a la posición (2,1). Aquí el semáforo decide que la obligará a ir a la derecha, para que no haya otra opción que pasar por toda la fila 3 antes de llegar al final del tablero. Esta fila tiene múltiples oportunidades de terminar el juego de forma no exitosa antes de llegar al final. Como es lógico, Roborta elige en ambas oportunidades de esta fila moverse hacia la derecha, para llegar a la casilla que le permite bajar hasta el final del tablero, pero debe pasar por todas las casillas marcadas antes, pudiendo perder el juego.

En definitiva, la solución que agregamos para no utilizar *fairness* en nuestro caso de estudio implica la modificación de tableros para evitar casos en los que no haya progreso en el juego. Con esto logramos teóricamente que todos los juegos tengan parada garantizada, ya sea exitosa o no exitosa.

4.2.2. Orden lexicográfico

Como dijimos anteriormente, nuestros juegos estocásticos tienen dos objetivos, el primero de alcanzabilidad, es decir que se debe llegar a un conjunto de estados considerados exitosos, y el segundo de acumulación de recompensas. También, se agregó la condición de que la recompensa será cero si no se llega a un estado exitoso. Teniendo esto en cuenta, el jugador 1 tiene una noción de orden lexicográfico para trabajar con los objetivos. En cambio para el jugador 2 no queremos que exista esa condición, ya que tiene un comportamiento adversarial, en el que puede elegir minimizar con mayor o menor prioridad cualquiera de estos objetivos.

Teniendo esto en cuenta, no se utilizará el algoritmo de recorte presentado en [5]. Ya que este implica que para cada objetivo se recorten las estrategias que no son óptimas para ambos jugadores. Lo que haremos en su lugar es recortar las estrategias del jugador 1 que no maximicen el objetivo de alcanzabilidad y luego se maximizará el otro objetivo con las estrategias restantes.

Para mostrar las diferencias en ambos enfoques, podemos ver la figura 4.3. En este ejemplo tenemos al jugador 1 en el estado inicial s_0, con dos acciones disponibles α y β. Luego tenemos el estado s_1 para el jugador 2, el cual también tiene dos acciones γ y δ y el estado s_2 para el jugador 1 con solo una acción, por lo que no lo analizaremos. Luego tenemos los estados probabilistas s_3 y s_4, con probabilidades de 0,35 y 0,3 de llegar al

estado exitoso s_5, respectivamente. El resto de las probabilidades llevan al estado final s_6 que imposibilita que cumplamos el objetivo de alcanzabilidad.

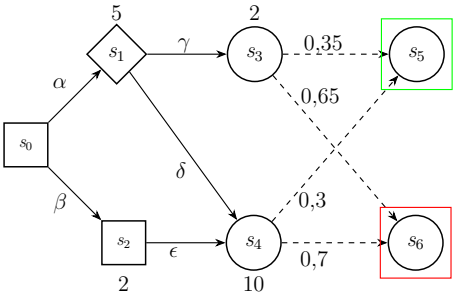


Figura 4.3: Juego donde usar el algoritmo de orden lexicográfico da resultados diferentes a los deseados.

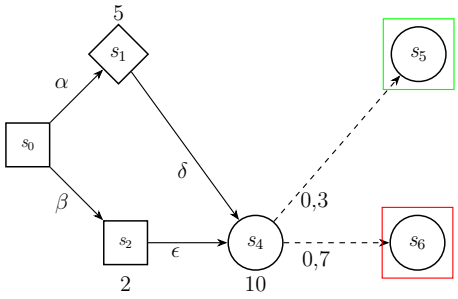


Figura 4.4: juego 4.3 sin el camino γ

En este ejemplo vemos como si el jugador 2 recorta las estrategias que no considera óptimas para alcanzabilidad, tendremos un resultado distinto a si elige una acción en la siguiente etapa.

Para empezar, cuando el jugador 1 maximiza alcanzabilidad, no descarta ninguna acción en s_0, ya que interpretando que el jugador 2 va a minimizarla, tendrá en ambos caminos probabilidad de llegar a s_5 de 0,3 como mínimo. Ahora el jugador 2 cuando se sigue el algoritmo de [5] descarta γ, en pos de minimizar la probabilidad de llegar al estado exitoso. Cuando llega el momento de hacer el cálculo de la esperanza condicionada, tenemos que para el algoritmo lexicográfico ahora trabajamos con el grafo de 4.4. En este escenario el jugador 1 toma la acción α, ya que le garantiza 15 puntos de recompensa, sobre 12 que tendría al elegir β.

En cambio, con nuestro algoritmo continuamos con el grafo de 4.3, ya que no hay recortes por parte del minimizador en la etapa anterior. Este ahora elige quedarse con la acción γ, para usar el camino con menor recompensa esperada (el camino de γ tiene 0.7 mientras que el de δ tiene 3). Con esto en mente, cuando el jugador 1 tiene que elegir una acción para maximizar recompensas, usando el calculo de recompensa condicionada, elige el camino de β ya que le da 12 por sobre 7 de esperanza de la recompensa condicionada (Recordemos 4.14).

Las estrategias tienen entonces distintos resultados. El ejemplo con el algoritmo de orden lexicográfico toma el camino $s_0 s_1 s_4$, recolectando 15 de recompensa y llegando al objetivo con probabilidad 0.3, mientras que en nuestro caso toma el camino $s_0 s_2 s_4$, recolectando 12 puntos y llegando con la misma probabilidad. Vemos que en el caso del orden lexicográfico estamos asumiendo el comportamiento del jugador 2, donde este toma una decisión que no es la mejor para la reducción de la recompensa. Esto es porque al actuar con el mismo orden de prioridad de objetivos, no tiene un comportamiento completamente adversarial, que es el que deseamos reproducir en esta investigación.

4.3. Algoritmo de maximización de alcanzabilidad y recompensas

Para obtener el valor óptimo para el juego, dadas las condiciones planteadas, proponemos un algoritmo con dos pasos principales. Al tener una noción de orden lexicográfico para los objetivos para el jugador 1, hacemos que en un primer paso nuestro algoritmo calcule las estrategias óptimas de alcanzabilidad, restringiendo así las acciones del maximizador que no tienen la mayor probabilidad de llegar a los estados exitosos. Este es un problema que ya ha sido previamente resuelto y se conoce que está determinado [8].

En el siguiente paso, se hace un calculo de maximización de recompensas totales, usando los resultados de [3]. Para este, se necesita como condición que sea segura la llegada a los estados finales. Teniendo eso en cuenta, es que necesitamos calcular la esperanza de la recompensa *condicionada* al cumplimiento del objetivo de alcanzabilidad. Con esta restricción, hacemos un paso extra antes de calcular las recompensas. Este paso consta de recortar y redistribuir las probabilidades de los caminos que no nos lleven a estados exitosos. Una vez hecho los recortes, se puede seguir con el algoritmo usual de recompensas totales.

En resumen, separamos las estrategias del jugador 1 que maximizan la probabilidad de alcanzabilidad y con ese subconjunto hacemos el calculo de maximización de recompensas condicionado a alcanzar el objetivo, haciendo el debido recorte.

4.4. Cálculo de alcanzabilidad

Para la primer parte del algoritmo vamos a seleccionar las estrategias de Π_1 tal que maximicen la alcanzabilidad del conjunto de llegada exitosa T.

En 2.12 mostramos la función para la probabilidad de llegar a un conjunto de estados. Usando esta fórmula, buscaremos las mejores estrategias del jugador 1 para alcanzar los estados exitosos T. Dados Π_1 y Π_2, los conjuntos de estrategias del jugador 1 y jugador 2 respectivamente, nos quedaremos con las estrategias π_1 que pertenezcan al siguiente conjunto:

$$\left\{\pi_1 \mid \forall v \in V_1 \colon \pi_1 \in \arg \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} Prob_v^{\pi_1,\pi_2}(\Diamond T)\right\} \tag{4.15}$$

Este está conformado de estrategias que para cada decisión minimizadora de alcanzabilidad de Π_2, dan la mejor opción posible para maximizarla.

Este conjunto tiene una sutil falla para nuestra configuración de juegos. Como hemos dicho, el jugador 2 no juega siempre minimizando solo la alcanzabilidad. Este puede, en pos de minimizar las recompensas, tomar una elección que no sea la de menor probabilidad de llegar al conjunto de estados exitosos. Con esto en mente, hay que calcular para el jugador 1 las mejores estrategias de alcanzabilidad en esos casos y no dejar elecciones sin maximizar.

Podemos ver en la figura 4.5 que el camino que da menor alcanzabilidad es el que se da si el minimizador elige la acción α, que nos lleva hacia s_2. Con esto en mente, un scheduler π_1 calculado desde el estado s_0 puede elegir para la decisión en el estado s_3 el camino por δ en vez del camino por γ (este último es el que nos lleva a maximizar el objetivo de

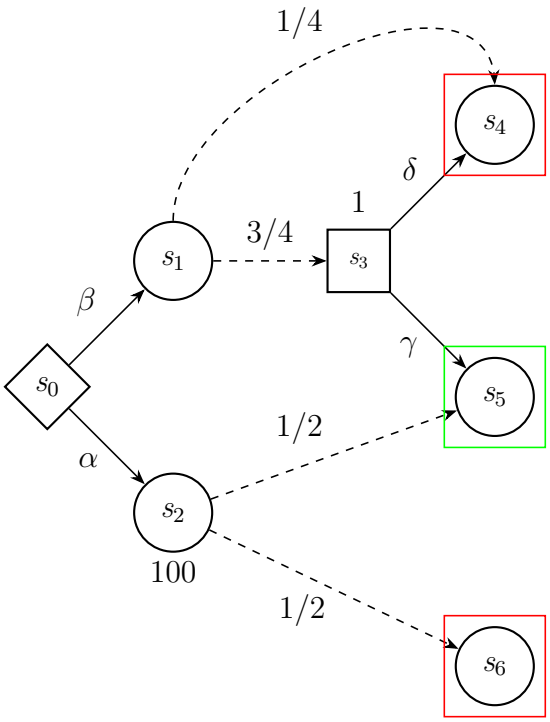


Figura 4.5: Juego estocástico con $s_0 \in V_2$ y $s_3 \in V_1$.

alcanzabilidad). Esto es porque en un principio no importa que decisión se tome en s_3, si este estado no será alcanzado. Lo que se daría si el jugador 2 elige siempre α, minimizando la probabilidad de alcanzar el conjunto T. Pero como hemos demostrado, puede haber ocasiones en las que priorice minimizar la recompensa esperada. En este caso, lo haría eligiendo la acción β, para evitar los 100 puntos de recompensa en s_2. Esto provocaría que el jugador 1, que había definido su estrategia sin tener en cuenta esta posibilidad, elija la acción δ y nos lleve al estado no exitoso s_4.

Como vemos, necesitamos que las estrategias del jugador 1 tengan en cuenta todas las posibles decisiones del jugador 2, y que no asuma que algún camino no va a ser tomado. Para eso, vamos a introducir el conjunto $\rho \Diamond T$, que lo definiremos de la siguiente forma:

$$\rho \Diamond T = \left\{ \sigma \in V^\omega \mid \rho \text{ es prefijo de } \sigma \wedge \exists i \geq |\rho| \colon \sigma_i \in T \right\} \tag{4.16}$$

Es decir, el conjunto de trazas tal que hay un prefijo ρ de estados y para un índice i mayor al largo de ρ, tenemos un estado que pertenece al conjunto T.

Ahora introducimos el conjunto de estrategias Π_1^*, de las estrategias maximizadoras de alcanzabilidad para el jugador 1, teniendo en cuenta todas las posibles estrategias del jugador 2:

$$\Pi_1^* = \left\{ \pi_1^* \mid \forall \rho \in V^* \colon \forall v \in V \colon \pi_1^* \in \arg \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} Prob_v^{\pi_1,\pi_2}(\rho \Diamond T) \right\} \tag{4.17}$$

La intuición detrás de este conjunto es que para todas las trazas ρ que empiecen desde un estado v, se seleccionan solo los π_1^* que maximicen $Prob_v^{\pi_1,\pi_2}(\rho \Diamond T)$, donde $\rho \Diamond T$ nos permite abarcar todos los posibles caminos elegidos por el jugador 2.

Así, si volvemos al ejemplo 4.5, el maximizador elegirá la acción γ, ya que tuvo en cuenta para el calculo de su estrategia la posibilidad de que se de la traza $s_0 s_1 s_3$.

4.4.1. Algoritmo de maximización de alcanzabilidad

Para el algoritmo de maximización de alcanzabilidad utilizamos el algoritmo de *value iteration*, buscando resolver las ecuaciones de Bellman de la función 2.12. Para eso, inicializamos con ceros un vector de probabilidad de alcanzar el estado objetivo. Luego, al tener valor 1 para los estados finales, la probabilidad se irá propagando por el vector en cada iteración.

Algorithm 1 reverseDFS

Require: δ las transiciones del juego.
Require: T los estados finales exitosos.
  $\delta' \leftarrow reverseTransitions(\delta)$
  $PT \leftarrow [\,]$ ▷ Conjunto de estados que llegan a T
  $i \leftarrow 0$
  while $i < |\,T\,|$ **do**
    $PT \leftarrow recursiveReverseDFS(T[i], \delta', PT)$
    $i \leftarrow i + 1$
  end while
  $PT \leftarrow PT \setminus T$
  return PT

En nuestra versión del algoritmo, no actualizamos los estados que pertenecen a N, es decir los que no tienen probabilidad de llegar al objetivo, ya que estos siempre tendrán valor 0. Para esto, primero determinamos que estados pueden alcanzar al conjunto T mediante el algoritmo de *reverse DFS*.

En 1 tenemos el algoritmo de *reverse DFS*, que nos devuelve el conjunto de estados que alcanzan al conjunto T. Es decir que al terminar de correr el algoritmo, tenemos como resultado que PT es igual a $pre^*(T) \setminus T$. Esto se logra corriendo recursivamente el algoritmo 2 con los estados finales como parámetro principal.

El algoritmo recursivo recibe un estado v, las transiciones invertidas y los estados que ya fueron agregados a PT. El propósito de este conjunto es no repetir la recursión para estados ya agregados. Primero se agrega al conjunto PT el estado actual y luego se itera por los estados previos a v, llamando recursivamente a la función *recursiveReverseDFS*. Finalmente el resultado de este algoritmo es un conjunto con todos los estados que tienen un camino hacia v sumados a los estados que ya se encontraban previamente en PT.

Algorithm 2 recursiveReverseDFS

Require: v el estado para comenzar el DFS.
Require: δ' las transiciones del juego invertidas.
Require: PT los estados que alcanzan a v o a un estado de T.
  $RecursivePT \leftarrow PT \cup \{v\}$
  $i \leftarrow 0$
  while $i < |\delta'(v)|$ **do**
    if $\delta'(v)[i] \notin RecursivePT$ **then**
      $RecursivePT \leftarrow recursiveReverseDFS(\delta'(v)[i], \delta', RecursivePT)$
    end if
    $i \leftarrow i + 1$
  end while
  return $RecursivePT$

Una vez que tenemos el conjunto de estados que alcanzan a T, podemos continuar con el algoritmo 3.

Para este se requiere un valor ϵ además de los estados que pertenecen a PT. Este valor se usará como criterio de parada, deteniendo el algoritmo cuando sea menor que este la diferencia entre los vectores de probabilidad de la iteración actual y la anterior. Se inicializa el vector de probabilidad x' con ceros y la diferencia con 1. En cada iteración se guarda el valor de aplicar la función $f^{\Diamond}$ con los parámetros x' y PT, el primero tiene los valores de probabilidad de la iteración anterior, y el segundo

es el conjunto que tiene la información de cada estado, como a que jugador pertenece y si es o no estado final. $f^{\Diamond}(x, PT)$ se corrseponde con $f^{\Diamond}(x)$ en la ecuación 2.12 y PT se corresponde con el conjunto $S \setminus T \cup N$ en esa misma ecuación.

Algorithm 3 valueIterationReachability

Require: PT los estados que alcanzan a un estado de T.
Require: ϵ un decimal.
  $x' \leftarrow [0]* \mid PT \mid$ ▷ Vector de largo $\mid PT \mid$
  $diff \leftarrow 1$
  while $diff > \epsilon$ **do**
    $x \leftarrow f^{\Diamond}(x', PT)$
    $diff \leftarrow \mid\mid x - x' \mid\mid$
    $x' \leftarrow x$
  end while
  return x'

Al terminar la ejecución del algoritmo obtenemos el vector x', que tendrá para cada estado la probabilidad de llegar a los estados finales exitosos, teniendo en cuenta el tipo de jugador. Cabe recalcar que si el estado inicial tiene probabilidad 0 de llegar a los estados exitosos, significa que no tiene sentido continuar con el cálculo de recompensas, ya que estas serán 0 si no se llega al final.

Posteriormente, para cada estado de los jugadores 1 y 2, guardaremos las estrategias que obtienen las probabilidades de x'.

4.5. Cálculo de recompensa total condicionada

Una vez que tenemos las estrategias Π_1^* del jugador 1 podemos calcular las estrategias que maximicen las recompensas. Buscamos entonces las estrategias que cumplan la siguiente ecuación.

$$\sup_{\pi_1^* \in \Pi_1^*} \inf_{\pi_2 \in \Pi_2} \mathbb{E}^{\pi_1^*, \pi_2}(f^{rew} \mid \Diamond T) \tag{4.18}$$

Esta nos va a proporcionar el valor esperado del juego. Utilizamos la esperanza de la recompensa *condicionada* a que se cumpla el objetivo de alcanzabilidad ya que para correr el próximo paso del algoritmo, debe ser seguro que se va a alcanzar el conjunto de estados finales.

En general, tendremos un valor distinto al calcular la esperanza de la recompensa que al calcular la esperanza condicionada. Esto no significa que un valor represente mejor lo que queremos calcular que el otro, solo es necesario recalcarlo para mostrar las posibles restricciones que implica trabajar con este método.

4.5.1. Diferencias entre esperanza de recompensa y esperanza de recompensa condicionada

Usaremos la figura 4.6 como ejemplo para distinguir el uso de una esperanza y la otra. Vemos primero que tenemos cuatro estados finales, de los cuales s_9 y s_{11} son exitosos (marcados con verde), y los demás son estados no exitosos (marcados con rojo). También vemos que solo tenemos un estado controlado por el jugador 1, s_0, mientras que los demás son estados probabilistas o controlados por el jugador 2 (círculos y rombos, en ese orden), donde solo nos interesa s_1, ya que es el único que tiene más una elección para el jugador 2.

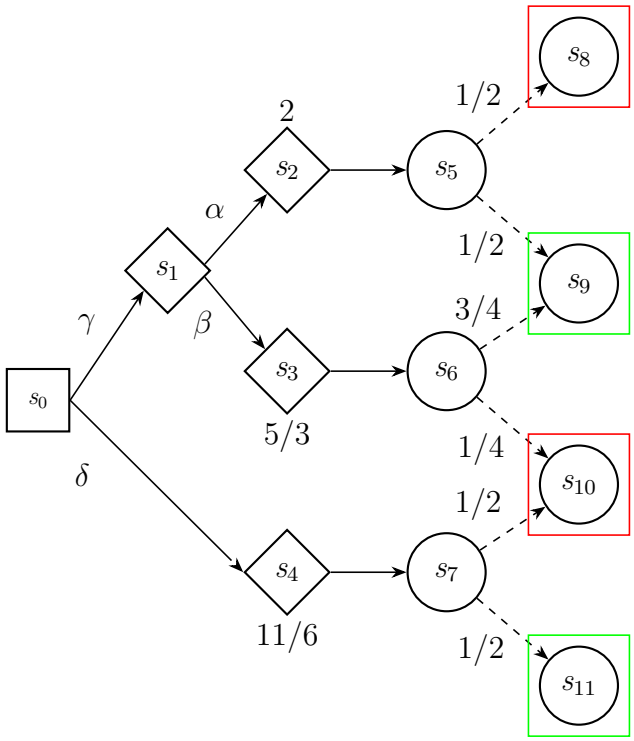


Figura 4.6: Juego estocástico donde se ven las diferencias en calcular la esperanza condicionada.

En este caso el paso de conseguir las mejores estrategias de alcanzabilidad no elimina caminos para el jugador 1 dejándonos con la misma figura. Para el cálculo de recompensa total, tomando la esperanza (sin estar condicionada), el camino $\gamma\alpha$ da 1 de recompensa esperada, $\gamma\beta$ da 5/4 y δ da 11/12. Por lo que elegir la opción δ es la que proporciona menor recompensa en todos los casos para el jugador 1, entonces elige γ. En cambio si tomamos la esperanza condicionada, el camino $\gamma\alpha$ nos da valor 2, $\gamma\beta$ da 5/3 y δ da 11/6. Aquí elegir γ para el jugador 1 hace que el minimizador elija β, que es el camino con menor recompensa condicionada, esto nos da un valor de 5/3. Por lo que en este caso es mejor para el maximizador elegir δ, que nos deja una esperanza de recompensa condicionada de 11/6.

4.5.2. Algoritmo de maximización de recompensa condicionada

Ahora calcularemos para las estrategias maximizadoras del jugador 1, la recompensa condicionada a alcanzar un estado exitoso. Para eso primero eliminamos las acciones del jugador 1 que no son seleccionadas por la estrategia óptima de alcanzabilidad. Luego para calcular la esperanza condicionada, debemos efectuar otro paso de recorte. En este paso se recortan las ramas probabilistas que tienen probabilidad 0 de llegar al objetivo y se re-balancean los valores de probabilidad de manera que las distribuciones de probabilidad de cada transicion quedan condicionadas a la llegada del objetivo. Por último, aplicamos el algoritmo de *value iteration* para recompensas totales.

El algoritmo 4 efectúa el primer paso de recorte, dados los parámetros *bestStrategies*,

el vector obtenido luego de maximizar la probabilidad de alcanzar el objetivo, y *states*, una lista de todos los estados con su información (*bestStrategies* contiene las mismas estrategias que Π_1^*). Para efectuar el recorte iteramos entre los estados y para cada uno que sea del jugador 1, solo le dejamos las transiciones que forman parte de una estrategia maximizadora ($states[i].t$ son las transiciones para el estado s_i).

Luego continuamos con el recorte de caminos que tienen probabilidad 0 de alcanzar el objetivo, lo que es realizado por el algoritmo 5. Tenemos como valores de entrada las probabilidades de alcanzar un estado exitoso *statesReachability* y nuevamente la lista de estados *states*. Otra vez iteramos entre los estados, y para cada uno de los que pertenezcan al jugador 1 o sean probabilistas, iteramos entre sus estados vecinos. Cada estado vecino que tenga probabilidad 0 de cumplir el objetivo de alcanzabilidad será eliminado y se redistribuirá la probabilidad ($states[i].p$) de tomar ese camino entre los demás vecinos. La redistribución de probabilidades se hace para cada estado que no ha sido eliminado, dividiendo su probabilidad actual sobre el total de probabilidades restantes luego de eliminar el camino que no tiene posibilidades de llegar al objetivo.

Algorithm 4 prunePathsMAXReachability

Require: *bestStrategies* Estrategias que maximizan alcanzabilidad.
Require: *states* estados del juego con sus transiciones.
  $i \leftarrow 0$
  while $i < |\,states\,|$ **do**
    if $states[i] \in V_1$ **then**
      $states[i].t \leftarrow bestStrategies[i]$
    end if
    $i \leftarrow i + 1$
  end while
  return *states*

Algorithm 5 prunePathsNOReachability

Require: *states* estados del juego con sus transiciones.
Require: *statesReachability* Probabilidades de alcanzar un estado exitoso.
  $i \leftarrow 0$
  while $i < |\,states\,|$ **do**
    if $states[i] \in V_P$ **then**
      $j \leftarrow 0$
      while $j < |\,states[i].t\,|$ **do**
        $nextStateIdx \leftarrow states[i].t[j].idx$
        if $statesReachability[nextStateIdx] == 0$ **then**
          $states[i].t \leftarrow states[i].t \setminus states[i].t[j]$
          $redistributeProbabilities(states.p)$
        end if
        $j \leftarrow j + 1$
      end while
    end if
    $i \leftarrow i + 1$
  end while
  return *states*

Por ejemplo si de un nodo s_i salen caminos x, y, z; con probabilidades p_x, p_y, p_z. Y el camino que pasa por z tiene probabilidad 0 de llegar a un estado exitoso, se lo elimina y se debe redistribuir su probabilidad p_z. Las nuevas probabilidades de los caminos x e y son $\frac{p_x}{(p_x+p_y)}$ y $\frac{p_y}{(p_x+p_y)}$, respectivamente.

Por último, habiendo condicionado el juego a tener parada garantizada, podemos usar las siguientes ecuaciones de Bellman en 4.19 para encontrar su mínimo punto fijo [3], utilizando nuevamente *value iteration*.

$$\Gamma(v) = \begin{cases} rew(v) + \sum_{v' \in post(v)} \delta(v, v')\ rew(v'), & \text{si } v \in V_P \setminus T \\ rew(v) + max\{rew(v') \mid v' \in post(v)\}, & \text{si } v \in V_1 \setminus T \\ rew(v) + min\{rew(v') \mid v' \in post(v)\}, & \text{si } v \in V_2 \setminus T \\ 0, & \text{si } v \in T \end{cases} \tag{4.19}$$

En el algoritmo 6 inicializamos el vector con las recompensas iniciales de cada estado

($states.r$). Esto es para ahorrar algunas iteraciones y condiciones que serían necesarias si inicializamos el vector con ceros.

Luego se aplica iterativamente la función Γ hasta que la diferencia con la iteración anterior sea menor o igual a ϵ. Este algoritmo nos devuelve la esperanza de la recompensa condicionada a llegar a un estado exitoso, habiendo restringido las estrategias del jugador 1 para tener la mayor probabilidad de llegar a este tipo de estados.

Algorithm 6 valueIterationRewards

Require: *states* estados del juego con sus transiciones.
Require: ϵ un decimal.
  $x' \leftarrow states.r$
  $diff \leftarrow 1$
  while $diff > \epsilon$ **do**
    $x \leftarrow \Gamma(x')$
    $diff \leftarrow || x - x' ||$
    $x' \leftarrow x$
  end while
  return x'

Luego separaremos las estrategias del jugador 1 que consigan este valor. Para el jugador 2, separaremos dos estrategias distintas, para ver los resultados de que este busque minimizar recompensas o probabilidad de llegar a un estado exitoso. Hay que tener en cuenta que el minimizador podría no elegir ninguna de estas estrategias, buscando un punto medio entre los dos objetivos.

Entonces luego de correr el algoritmo de dos pasos tendremos como resultado las estrategias maximizadoras de esperanza de recompensa condicional para el jugador 1 y estrategias minimizadoras de alcanzabilidad y de recompensa condicionada para el jugador 2. Además de las probabilidades esperadas de cumplir el objetivo de alcanzabilidad y las esperanzas de recompensas condicionadas, para cada estado. Obviamente de estas últimas nos importan los valores para el estado inicial, para determinar cual es el valor del juego.

Cabe destacar que no se llegó a calcular si un juego estocástico con las características y objetivos dados está determinado. Pero notamos que tenemos dos estrategias del minimizador, con un valor diferente para cada una. Por esto es interesante analizar en el futuro que pasa si el jugador 2 fija una estrategia que trate de abarcar ambos objetivos al mismo tiempo, por ejemplo buscando el valor mínimo de una curva de Pareto [7].

Capítulo 5

Implementación y experimentación

En este capítulo hablaremos primero de la implementación de la herramienta para computar los algoritmos mostrados en la sección anterior, seguido de un programa para la creación de tableros de Roborta y por último mostraremos los resultados de corridas del algoritmo para diferentes ejemplos.

5.1. Implementación

Se implementaron los algoritmos presentados en el capítulo 4 usando el lenguaje de programación Python en la versión 3.11. Para esto se creo un tipo abstracto de datos que representa un juego estocástico. Este tiene los diferentes estados con su tipo, recompensas, transiciones y una lista con los estados finales exitosos. Luego se creó una clase para cada tipo de estados y una clase *Solver* que corre los algoritmos mostrados en el capítulo anterior. Dentro de esta se llaman a métodos de cada estado para ejecutar los algoritmos, para simplificar la comprensión y notar las diferencias en cada tipo de estado.

El código fuente de la herramienta se encuentra en github [9]. El input es un diccionario de Python el cual contiene cuatro listas. Las primeras tres de largo n, siendo n la cantidad de estados. La primera enumera las recompensas de los estados, la segunda que jugador lo controla y la tercera representa las transiciones, para cada estado hay una lista de pares (acción, estado) para los estados de jugador 1 y 2 o (probabilidad, estado) para los estados probabilistas. La última lista contiene el conjunto de estados finales. El output que proporciona el algoritmo es un archivo de texto que proporciona para cada corrida, en caso de que se pueda resolver el juego (tiene alcanzabilidad de estados exitosos mayor a 0), la cantidad de estados, transiciones, iteraciones utilizadas para los cálculos de alcanzabilidad y recompensas totales, las estrategias para maximizar/minimizar cada objetivo y por último y más importante los resultados de las iteraciones para cada estado. De estas nos interesa principalmente el estado inicial, ya que es el que nos da el valor de todo el juego. Entonces de estos resultados tenemos la probabilidad de alcanzar un estado exitoso cuando el minimizador se enfoca en alcanzabilidad y cuando se enfoca en minimizar recompensas, así también como las recompensas esperadas condicionadas a que se cumpla el objetivo de alcanzabilidad. Por último tenemos el tiempo de ejecución en segundos.

Para la sección de experimentación se implementó un generador de tableros de Roborta, que dados múltiples parámetros (largo, ancho, recompensas, probabilidades) genera un

tablero aleatorio en el formato del tipo abstracto de datos nombrado previamente. Además, se agregó otro algoritmo que permite generar el juego estocástico a partir de un tablero, tomando como parámetros los movimientos, recompensas, casillas flojas y probabilidades de que se rompan robot, semáforo y casillero, con el fin de poder probar distintos tableros y modificaciones de estos.

5.2. Experimentación

Para esta parte vamos a correr el algoritmo con los distintos ejemplos mostrados en el trabajo, para tener una idea de como funciona el algoritmo con ejemplos simples. También podremos ver los resultados para las distintas estrategias del minimizador. Además lo correremos con otro grupo de ejemplos, que serán tableros de Roborta de distintos tamaños, para analizar como se comporta el algoritmo en casos más complejos. Todos los ejemplos fueron ejecutados en una Macbook Air con chip Apple M1 y 16 GB de RAM.

En el cuadro 5.1 se reportan los resultados de todos los ejemplos que se fueron mostrando a lo largo del trabajo. Las primeras nueve filas son los grafos que fueron usados para explicar distintas características del problema. Se puede ver como se agregaron tres modificaciones del ejemplo 4.6, para ver como cambian los comportamientos de los jugadores en un entorno que ya hemos analizado. Para el primero, se intercambiaron que estados pertenecen al jugador 1 y cuales al jugador 2, así vemos como al minimizar recompensas, la probabilidad de alcanzar un estado exitoso aumenta y las recompensas condicionadas disminuyen, con respecto al ejemplo anterior y con el mismo ejemplo cuando se minimiza la alcanzabilidad.

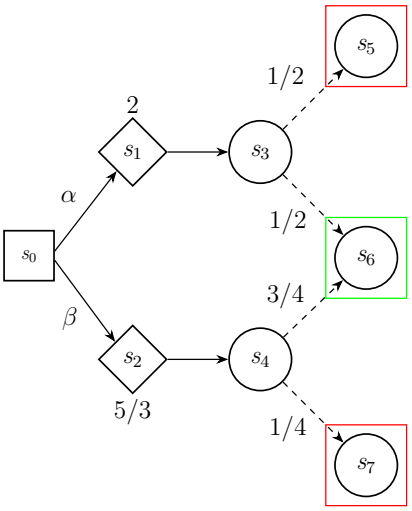


Figura 5.1: Juego estocástico simple.

Luego tenemos el mismo ejemplo con todos los estados no probabilistas perteneciendo al jugador 1, lo que hace que se maximice la alcanzabilidad, llegando a 0.75, ya que prioriza esto en la primer parte del algoritmo. Por último tenemos todos los estados del jugador 1 y todas las decisiones probabilistas con probabilidad de 0.5 . Esto hace que el maximizador busque el camino con mayor recompensa, ya que la probabilidad de cumplir el primer objetivo será la misma por cualquier camino.

Agregamos también los ejemplos de las figuras 5.1 y 5.3, que fueron de interés a lo largo de la investigación para resolver distintos problemas. Por ejemplo como actúa nuestro algoritmo en los casos en que tenemos loops. Estos simulan mejor las situaciones del mundo real, ya que las decisiones no solo dividen entre llegar o no al objetivo, también pueden hacernos volver hacia estados por los que ya pasamos en el pasado. En este escenario vemos que tenemos una mayor cantidad de iteraciones del algoritmo que en los ejemplos más simples. Para este último ejemplo tuvimos un tiempo de ejecución menor a 0.004 segundos. Para el resto de ejemplos más simples, tuvimos tiempos de ejecución menores a 0.0002 segundos.

Toda la información de las corridas se encuentra en el repositorio de Github [9]. Los datos de entrada se encuentran en el directorio *inputs/* y los resultados de las ejecuciones en el directorio *outputs/*. Todos estos grafos se encuentran en el archivo de *paper_games*, *paper_games.py* para los datos de entrada y *paper_games.txt* para los de salida.

Por último en el mismo cuadro 5.1 tenemos dos tableros de Roborta. El primero donde se corrió el algoritmo con una probabilidad de que Roborta y el semáforo se rompan y pierdan el turno de 0.1. Esto hace que la recompensa condicionada a la alcanzabilidad cuando el minimizador se enfoca en las recompensas, que es el caso mostrado en la figura 1.2 sea de 5.555. Es interesante notar que en este caso, al enfocarse en las recompensas, el minimizador no se preocupa en hacernos pasar por casillas flojas, por lo que la probabilidad de llegar al final es 1. En cambio, la ruta para minimizar alcanzabilidad nos hace recolectar mayor recompensas y la probabilidad de llegar al final disminuye a 0.738. La siguiente fila representa el mismo tablero que tiene flechas hacia abajo para ayudar al progreso, pero se agregó además del ejemplo 4.2 donde se ve la ruta cuando el minimizador se enfoca en la alcanzabilidad, la figura 5.2, donde se ve que pasa cuando se concentra en las recompensas. Así podemos comparar como a veces al concentrarse en un valor este disminuye y el otro aumenta. Vemos como el semáforo se pone en verde en el caso de las recompensas al llegar al casillero (3,3), permitiendo a Roborta a llegar al final del tablero y sin permitirle recoger más puntos. A diferencia con el caso de alcanzabilidad, donde el semáforo se pone en amarillo, obligando a Roborta a pasar por toda la última fila, lo que la hace pasar sobre múltiples casillas flojas teniendo una posibilidad de perder.

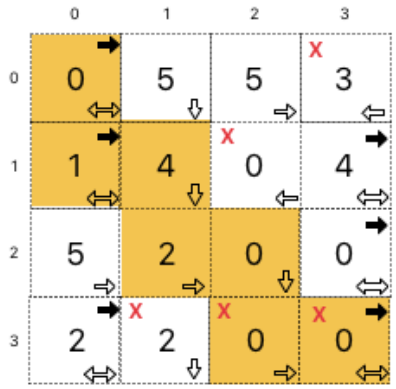


Figura 5.2: Tablero 4x4 con flechas hacia abajo. Jugador 2 minimiza recompensas.

Podemos ver en estos dos ejemplos que la cantidad de estados y transiciones incrementó notablemente, aún más en el ejemplo en que hay probabilidad de que los jugadores pierdan un turno, ya que se agregan múltiples estados probabilistas para cada decisión. Esto también afecta lógicamente a la cantidad de iteraciones que debe hacer el algoritmo en cada paso. El último ejemplo tardó poco más de 0.1 segundos en ejecutarse y el de flechas hacia abajo tomó menos de 0.003 segundos. Los inputs y outputs se encuentran en los directorios correspondientes y tienen los siguientes nombres
"manual_robot_Roborta_1_w4_l4_r5_rb10_lb10_tb10_" y
"manual_robot_arrow_down_w4_l4_r5_rb10_lb10_tb10_force_down", para el de pro-

babilidad de romperse los jugadores y para el de flechas hacia abajo, respectivamente.

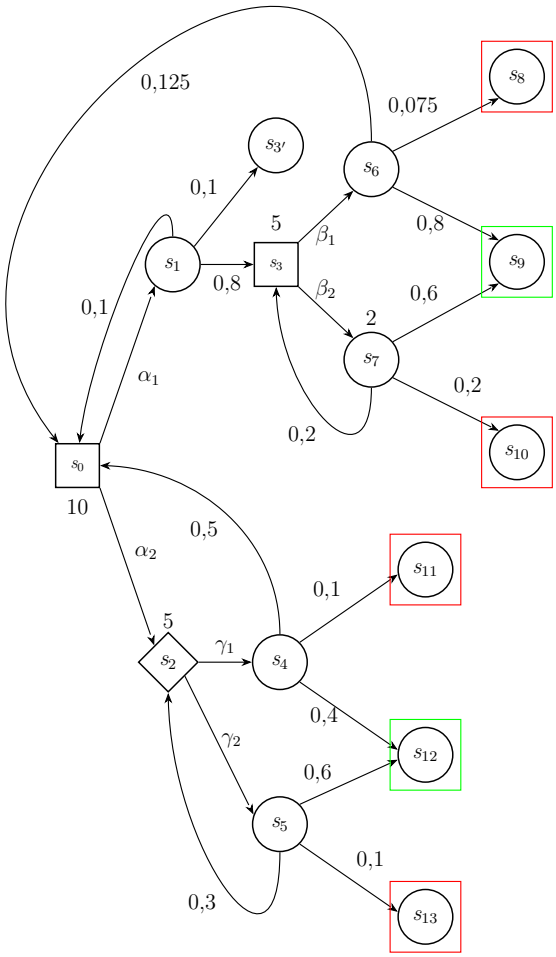


Figura 5.3: Juego estocástico con loops y múltiples elecciones.

En el cuadro 5.2 reportamos los resultados de distintos tableros de Roborta, con diferentes tamaños. Para cada distinta configuración de largo y ancho se corrió un tablero *Normal*, donde no hay casilleros con flechas hacia abajo para forzar progreso; y un tablero con flechas hacia abajo. Es importante notar que como dijimos en la introducción del trabajo, los tableros normales precisan que el semáforo tenga una probabilidad de romperse para que Roborta pueda progresar. Con esta característica todo tablero tiene solución y alguna probabilidad de completar el juego. Para el caso de tableros con flechas hacia abajo, como se puede ver en el código, tenemos que todas las filas tienen al menos una casilla que apunta hacia abajo. Pero incluso con esta restricción podemos tener tableros

para los que no se llega al final, ya que el minimizador puede mantenernos en un par de casillas infinitamente. Por esto se corrieron múltiples iteraciones para cada tamaño con el generador de tableros aleatorios. Así, los ejemplos que mostramos en la tabla fueron los de tableros con parada asegurada.

Vemos que los tableros con estados probabilistas para que fallen los jugadores tienen más estados y transiciones, y también que hay mayor cantidad de iteraciones para encontrar una solución. Por otro lado, vemos que la cantidad de iteraciones no depende estrictamente del tamaño del tablero, ya que puede ser que uno tenga caminos más simples por más que sea más grande. Lo que si podemos ver es que para cada tablero particular, tenemos menor alcanzabilidad cuando el minimizador se enfoca en reducir esta y lo mismo pasa con las recompensas. Cada tablero tiene sus datos de entrada y salida en los directorios correspondientes con el nombre
"robot_SEMILLA_wANCHO_lLARGO_r6_rb10_lb10_tb10_lt30", donde SEMILLA, LARGO y ANCHO son los respectivos enteros para cada dato y para los tableros con flechas hacia abajo un sufijo "_force_down" . Los ejemplos tienen semillas que varían entre los valores 40 y 51. Los tiempos de ejecución van de 0.01 segundos para los ejemplos más simples a 5 segundos para los más complejos.

Cuadro 5.1: Ejemplos mostrados a lo largo del trabajo. La primer columna tiene la referencia al ejemplo, las siguientes tres columnas tienen la cantidad de estados, transiciones e iteraciones que se usaron para correr el algoritmo, en ese orden. La quinta columna divide cada ejemplo en dos, para ver los resultados cuando el minimizador se enfoca en alcanzabilidad de estados exitosos o en recompensas totales. Las últimas dos columnas tienen el valor del juego para alcanzabilidad y recompensas totales condicionadas, para cada caso del minimizador.

Ejemplo	Cantidad de			Jugador 2 minimiza	$Prob^{\pi_1,\pi_2}(\Diamond T)$	$\mathbb{E}^{\pi_1,\pi_2}(f^{rew} \mid \Diamond T)$
	Estados	Trans.	Iter.			
Ejemplo 4.1	7	10	alc. 4	Alcanz.	0.99	1
			rec. 3	Recomp.	0.99	1
Ejemplo4.3	7	11	alc. 4	Alcanz.	0.3	12
			rec. 4	Recomp.	0.3	12
Ejemplo4.5	7	11	alc. 4	Alcanz.	0.5	100
			rec. 3	Recomp.	0.75	1
Ejemplo4.6	12	17	alc. 4	Alcanz.	0.5	1.833
			rec. 4	Recomp.	0.5	1.833
Ejemplo4.6 jugadores intercambiados	12	17	alc. 5	Alcanz.	0.5	1.833
			rec. 5	Recomp.	0.75	1.667
Ejemplo4.6 todas jugador 1	12	17	alc. 5	Alcanz.	0.75	1.667
			rec. 5	Recomp.	0.75	1.667
Ejemplo4.6 todo jugador 1, mismas probabilidades	12	17	alc. 4	Alcanz.	0.5	2
			rec. 5	Recomp.	0.5	2
Ejemplo5.1	8	11	alc. 4	Alcanz.	0.75	1.667
			rec. 4	Recomp.	0.75	1.667
Ejemplo5.3	10	23	alc. 22	Alcanz.	0.8	18.789
			rec. 27	Recomp.	0.8	18.789
Ejemplo 1.2	162	287	alc. 255	Alcanz.	0.738	29.308
			rec. 322	Recomp.	1	5.555
Ejemplo 4.2	66	89	alc. 20	Alcanz.	0.729	11
Ejemplo 5.2			rec. 19	Recomp.	0.81	7

Cuadro 5.2: Tableros de Roborta. En la primera columna se describe el tamaño del tablero (ancho x alto). La segunda columna describe si el tablero tiene o no casilleros con flechas hacia abajo. Las siguientes dos columnas tienen la probabilidad de que el robot y el semáforo se rompan y pierdan un turno. el resto de columnas sigue el formato de la tabla anterior.

Versión del tablero		Prob. de romper		Cantidad de			Jugador 2 minimiza	$Prob^{\pi_1,\pi_2}(\Diamond T)$	$\mathbb{E}^{\pi_1,\pi_2}(f^{rew} \mid \Diamond T)$
		Robot	Semáforo	Estados	Trans.	Iter.			
5 x 5	Normal	0.1	0.1	252	469	alc. 305	Alcanz.	0.55	9.135
						rec. 33	Recomp.	0.901	2.789
	Con ↓	-	-	102	130	alc. 15	Alcanz.	0.81	9
						rec. 13	Recomp.	0.9	7
10 x 5	Normal	0.1	0.1	502	918	alc. 272	Alcanz.	0.433	21.188
						rec. 495	Recomp.	0.801	2.633
	Con ↓	-	-	202	271	alc. 47	Alcanz.	0.729	9
						rec. 17	Recomp.	0.81	6
20 x 10	Normal	0.1	0.1	2002	3690	alc. 397	Alcanz.	0.194	34.032
						rec. 526	Recomp.	0.839	5.876
	Con ↓	-	-	802	1071	alc. 77	Alcanz.	0.430	22
						rec. 45	Recomp.	0.656	9
40 x 10	Normal	0.1	0.1	4002	7403	alc. 427	Alcanz.	0.289	33.052
						rec. 594	Recomp.	0.867	8.325
	Con ↓	-	-	1602	2211	alc. 52	Alcanz.	0.531	11
						rec. 27	Recomp.	0.590	9

Capítulo 6

Conclusión y trabajo futuro

6.1. Conclusión

En este trabajo nos enfocamos en juegos estocásticos multiobjetivos, donde un jugador se enfoca en maximizar y el otro en minimizar dichos objetivos. En particular, se trabajó con un objetivo de maximización de recompensas y un objetivo de alcanzabilidad, con la condición de que si no se cumple el objetivo de alcanzabilidad, la recompensa será 0. Esto llevó a investigar diversos trabajos del área, como el agregado de un orden lexicográfico en los objetivos, juegos estocásticos con objetivos probabilísticos, juegos estocásticos donde el minimizador se comporta de forma justa, entre otros.

Para esto se trabajó con múltiples ejemplos, que ayudaron a pensar en la problemática y en sus posibles soluciones, que incluyó un algoritmo para calcular las mejores estrategias dividido en dos partes, lo cual llevó también a trabajar la fórmula de la esperanza de la función de recompensa total condicionada a la llegada de un estado exitoso. Las particularidades del trabajo, como hacer que el minimizador se comporte de forma no colaborativa en el orden de los objetivos, hizo que no se pueda utilizar en su totalidad las técnicas aprendidas de la bibliografía, lo cual nos llevó a mostrar las diferencias particulares con estos casos.

Se programó un algoritmo para conseguir la estrategia óptima para el maximizador y dos estrategias diferentes para el minimizador, una que minimiza la alcanzabilidad y otra que minimiza las recompensas totales condicionadas. Luego se evaluó el algoritmo con los ejemplos mostrados en el trabajo y con diferentes tableros de Roborta, para el cual se creó un generador de tableros aleatoreos, con diferentes datos de entrada, como el largo y ancho de este, la probabilidad de que una casilla sea floja, entre otros. Para los tableros de Roborta se analizaron dos tipos de juegos, los que tienen la probabilidad que el Semáforo se rompa y los que tienen tableros con flechas hacia abajo. Esto fue para tener juegos con parada casi garantizada, impidiendo que los algoritmos se encuentren computando un problema sin solución. Para estos experimentos notamos que los tableros con flechas hacia abajo tienen menos estados que los tableros con probabilidades de que el semáforo falle, y con esto vemos también una diferencia en la cantidad de iteraciones que toma al algoritmo en resolver el problema. También vemos la diferencia entre la esperanza de cumplir el objetivo de alcanzabilidad o de las recompensas esperadas condicionales cuando cambiamos las estrategias del minimizador.

6.2. Trabajo futuro

Para el problema de juegos estocásticos con un objetivo de alcanzabilidad y un objetivo de recompensas totales hay todavía múltiples ramas en las que todavía se pueden explorar distintas posibilidades.

Por ejemplo para el caso en que se fuerza al minimizador en comportarse de manera justa. Como dijimos en el capítulo anterior, para el caso de Roborta, hay tableros en los que se puede dar que el minimizador mantenga a ella en un bucle entre dos casilleros, lo que hace que el juego no termine. Si se agrega *fairness* al minimizador, se podrá experimentar con otros juegos además de los dos tipos mostrados en la experimentación. Además de esto, el restringir al minimizador a ser justo nos permitirá calcular la esperanza de la recompensa, sin la parte de estar condicionada al cumplimiento del objetivo de alcanzabilidad. Esto es porque para el cálculo de recompensas utilizamos el algoritmo de recompensas totales de [3], pero para usarlo debemos asegurarnos que los juegos tengan parada garantizada.

También se introdujo en los preliminares la noción de estrategias sin memoria, puesto que varios artículos en la bibliografía trabajan con este concepto. Sin embargo no se calculó si las estrategias óptimas de los jugadores en nuestro problema serían iguales si no tuvieran esta condición. Para el algoritmo creado para la parte de experimentación no utilizamos memoria en los estados, por lo que la intuición dice que debería ser igual teóricamente, pero falta probarlo.

Por último, se puede continuar con la investigación explorando si este tipo de juegos está o no determinado. Es decir, definiendo un valor para el juego (por ejemplo la esperanza de la recompensa), se quiere ver si para todos los juegos hay un único valor cuando ambos jugadores utilizan sus estrategias óptimas. En nuestro caso en concreto no decidimos cual sería este valor, ya que dijimos que el minimizador puede tomar una estrategia inesperada, con el objetivo de minimizar uno de los objetivos. Pero se puede definir el valor a calcular usando por ejemplo una curva de Pareto, como se hace en el trabajo con objetivos booleanos [7]. En [5] se muestra que los juegos estocásticos multiobjetivos sin orden lexicográfico podrían no estar determinados, pero creemos que, dado nuestro método para calcular la estrategia óptima del maximizador, se puede llegar a ver que este tipo de juegos está determinado si se elige un valor de juego acorde.

Bibliografía

[1] R. B. Ash and C. A. Doléans-Dade. *Probability and Measure Theory*. Harcourt/Academic Press, second edition, 1999.

[2] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

[3] P. F. Castro, P. R. D'Argenio, R. Demasi, and L. Putruele. Playing against fair adversaries in stochastic games with total rewards. In S. Shoham and Y. Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 48–69. Springer, 2022. doi: 10.1007/978-3-031-13188-2_3.

[4] K. Chatterjee and T. A. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2011.05.002. URL `https://www.sciencedirect.com/science/article/pii/S0022000011000511`. Games in Verification.

[5] K. Chatterjee, J. Katoen, M. Weininger, and T. Winkler. Stochastic games with lexicographic reachability-safety objectives. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 398–420. Springer, 2020. doi: 10.1007/978-3-030-53291-8_21.

[6] T. Chen, V. Forejt, M. Z. Kwiatkowska, A. Simaitis, A. Trivedi, and M. Ummels. Playing stochastic games precisely. In M. Koutny and I. Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2012. doi: 10.1007/978-3-642-32940-1_25. URL `https://doi.org/10.1007/978-3-642-32940-1_25`.

[7] T. Chen, V. Forejt, M. Z. Kwiatkowska, A. Simaitis, and C. Wiltsche. On stochastic games with multiple objectives. In K. Chatterjee and J. Sgall, editors, *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2013. doi: 10.1007/978-3-642-40313-2_25. URL `https://doi.org/10.1007/978-3-642-40313-2_25`.

[8] A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. doi: 10.1016/0890-5401(92)90048-K. URL `https://doi.org/10.1016/0890-5401(92)90048-K`.

[9] J. Feltes. conditionalrewards, 2024. URL `https://github.com/joaquinfeltes/conditionalrewards`.

[10] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 978-0-387-94805-8. doi: 10.1007/978-1-4612-4054-9.

[11] L. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39 (10):1095–1100, 1953. ISSN 0027-8424. doi: 10.1073/pnas.39.10.1095.

[12] M. Svorenová and M. Kwiatkowska. Quantitative verification and strategy synthesis for stochastic games. *Eur. J. Control*, 30:15–30, 2016. doi: 10.1016/j.ejcon.2016.04.009.