UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

APRENENTATGE PER REFORÇ APLICAT A UN CAS DE RECURSOS COMPARTITS

Autor: Arnau Finol Peñalver

Directora: Dra. Maite López Sánchez
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 24 de gener de 2022

Abstract

This thesis explores the theoretical concepts needed to generate an ethical embedding, as well as the development of prior theoretical knowledge for understanding. Ethical embedding involves generating a Markow decision process where optimal policies are ethical based on a multi-objective Markow decision process where at least one of them follows an ethical criterion.

Finally, it includes the implementation of the knowledge through the adaptation of the Common Game problem proposed by the company DeepMind and its subsequent resolution through the algorithms previously developed in a theoretical way.

Resum

En aquest treball s'exploren els conceptes teòrics necessaris per a generar una inserció ètica, així com el desenvolupament dels coneixements teòrics previs per a la seva comprensió. Una inserció ètica consisteix a generar un procés de decisió de Markow on les polítiques òptimes són ètiques partint d'un procés de decisió de Markow amb múltiples objectius on almenys un dells és segueix un criteri ètic.

Finalment, s'inclou la posada en pràctica dels coneixements a través de l'adaptació del problema Common Game proposat per l'empresa DeepMind i la seva posterior resolució a través dels algorismes desenvolupats prèviament de forma teórica.

Índex

1 Introducció

En l'àmbit de la intel·ligència artificial sovint es tracta el tema de com fer que els agents d'aprenentatge autònom ho facin d'una manera ètica. En aquest treball tractarem les claus per garantir que un agent que realitza un aprenentatge per reforç ho faci sempre d'una forma ètica.

Objectius

L'objectiu d'aquest treball és aplicar les innovacions proposades pel doctorant Manel Rodriguez Soto en els articles [9] i [8] sobre un cas pràctic més complex del que s'ha fet fins al moment. Comprovant així, que els resultats teòrics que s'ha obtingut es poden dur a la pràctica i se'n poden extreure resultats tangibles. Volem aplicar les seves aportacions sobre una adaptació d'un entorn desenvolupat per l'empresa DeepMind en l'article [5] conegut com a Commons Game. Aplicar aquests coneixements sobre un problema d'aquesta magnitud suposa un repte innovador.

Per fer-ho possible primer ens proposem conèixer la teoria necessària en la qual es fonamenten les innovacions proposades en els articles publicats per Manel Rodriguez Soto. Un altre objectiu és la implementació dels algorismes proposats en els articles esmenats, així com l'adaptació d'un problema proposat per tal de poder aplicar els algorismes.

Per últim ens proposem extreure resultats sobre el problema en el qual aplicarem els algorismes.

Estructura de la Memòria

En el capítol 2 s'expliquen els fonaments teòrics necessaris per poder treballar en el problema d'inserció ètica. En el capítol 3 s'expliquen els algorismes d'iteració de valors i Q-aprenentatge. En el capítol 4 s'introdueix a l'aprenentatge amb múltiples objectius i un es detalla un algorisme per aprendre totes les polítiques òptimes en aquests entorns. En el capítol 5 es formalitzen els problemes ètics en un entorn multiobjectiu, es detalla problema d'inserció ètica i es realitza un estudi teòric sobre l'algorisme ètic d'inserció. El resultat d'aquest treball es troba en el capítol 6 on apliquem el coneixement a un problema pràctic i per finalitzar trobem les conclusions.

2 Fonaments teòrics

En aquest capítol desenvoluparem els conceptes clau en els quals es basa aquest treball. Donarem un seguit de definicions, explicarem mètodes i desenvoluparem resultats que constitueixen els fonaments de la matèria.

2.1 Agents, accions i estats

Definició 2.1. *Un agent és un sistema capaç de percebre el seu entorn i d'actuar en aquest. Diem que un agent és racional si en cada possible seqüència de percepcions pren aquella acció que maximitza el valor de la seva mesura de rendiment o en el cas d'incertesa, en maximitza el resultat esperat.*

Per tal de dur a terme la tasca d'escollir quina és la millor acció l'agent es basa en les evidències que aporten les percepcions i la informació que aquest sigui capaç d'emmagatzemar. Més endavant comentarem com es mesura el rendiment d'un agent.
Per tal de modelitzar la situació de l'agent en el medi definirem l'espai d'estats:

Definició 2.2. *L'espai d'estats és el conjunt S de totes les configuracions possibles del medi on actua l'agent, incloent-hi la seva disposició dintre de l'entorn.*

En aquest treball tractarem amb problemes que podrem modelitzar de forma discreta. En cada estat $s \in S$ l'agent podrà triar una acció dins d'un conjunt d'accions possibles $A(s)$. Realitzar una d'aquestes accions el durà al següent estat, que anomenarem estat successor de l'estat que l'ha precedit. En entorns deterministes realitzar la mateixa acció en el mateix estat situarà l'agent sempre en el mateix estat successor. Per contra, en els entorns incerts, hi haurà unes certes probabilitats de transitar a diversos estats. És a dir, realitzar la mateixa acció no duu a l'agent sempre al mateix estat successor. Aquesta incertesa es podrà modelitzar en alguns casos com una variable aleatòria.

Definició 2.3. *Direm que un entorn és completament observable si l'agent és capaç de percebre tots els aspectes del medi que serveixen per a prendre decisions. En cas contrari, direm que l'entorn és parcialment observable.*

En els entorns completament observables l'agent sempre sabrà en quin estat es troba. Els entorns parcialment observables poden ser deguts a imprecisions o ambigüitats en les percepcions. En aquests casos, l'agent no té la certesa de saber en quin estat es troba. Tot i això, un agent racional hauria de prendre les millors accions possibles amb la informació que disposa. En aquest treball tractarem amb entorns completament observables.

Definició 2.4. *Un problema de decisió seqüencial consisteix a buscar el conjunt ordenat d'accions òptimes. És a dir aquella seqüència que maximitza els valors esperats.*

Podem dir llavors que un agent racional que ha de resoldre un problema de decisió seqüencial ha d'aprendre com triar, en cada estat, l'acció que maximitza l'esperança de la mesura del seu rendiment.

Denotarem per s_0 l'estat en el qual comença un agent i ens referirem a ell per estat inicial, de forma genèrica $s_t \in S$ és l'estat en l'instant $t \in \mathbb{N}$, on S és el conjunt d'estats. De forma semblant, escriurem a_t per a l'acció que realitza l'agent quan es troba en l'estat s_t i el durà a s_{t+1}.

L'espai que es defineix per les transicions forma un graf, que pot contenir cicles, llavors, pot donar-se el cas que $S_i = S_j$ per a $i \neq j$, és a dir que després de realitzar unes quantes accions tornem a trobar-nos en la mateixa situació. Adonem-nos que no necessàriament un estat i el seu successor han de ser diferents, pot donar-se el cas que una acció no tingui cap repercussió i, per tant, l'estat continuï essent el mateix. Eventualment, es podrà considerar que l'acció "no fer res" està dintre del conjunt d'accions possibles i depenent de l'entorn en què ens trobem, l'estat serà o no el mateix després de realitzar-la.

2.2 Hipòtesi i cadenes de Markov

Passem a introduir ara alguns conceptes per tal de definir que és un procés de decisió de Markov. El primer concepte clau és la hipòtesi de Markov. Tal com podem trobar al llibre[7]:

Definició 2.5. *Un procés verifica la hipòtesi de Markov si el seu estat actual depèn només d'un conjunt finit d'estats que el precedeixen. Els processos que compleixen aquesta hipòtesi s'anomenen processos de Markov o cadenes de Markov.*

Els processos de Markov poden ser quasi tan complexos com vulguem, els que ens interessen per desenvolupar aquest tema són els més simples:

Definició 2.6. *Es diu que un procés o cadena de Markov és de primer ordre si l'estat actual només depèn de l'estat que el precedeix.*

En unes altres paraules, l'estat actual resumeix tota la informació rellevant per tal que els estats futurs siguin independents dels estats anteriors a l'actual.

2.3 Procés de decició de Markov

Aquestes últimes definicions ens permet introduir la definició de model de transició:

Definició 2.7. *Es denomina model de transició a l'especificació de les probabilitats resultants per a cada acció y estat possible. S'utilitza la notació T(s,a,s') per tal de denotar la probabilitat assolir l'estat resultant s' si es realitza l'acció a en l'estat s.*

Es considerarà que les transicions compleixen la hipòtesi de Markov i són de primer ordre, això vol dir que, la probabilitat d'assolir s' a partir de s només depèn de s i no de l'historial d'estats anteriors.

Lligant-ho amb els conceptes definits en l'apartat anterior, un agent que interactués amb un medi on poguéssim considerar un model de transició com el que acabem de descriure, implicaria que la probabilitat de transitar a un cert estat només depèn de l'estat actual i de l'acció que es realitza, serà llavors independent de tot el que ha passat anteriorment.

$$P(S_{t+1} = s' \mid s_t, a_t, s_{t-1}, a_{t-1}, ..., S_0, a_0) = P(S_{t+1} = s' \mid s_t, a_t) \tag{2.1}$$

$\forall s' \in S$ i tots els possibles estats i accions passades.

L'últim concepte que cal aclarir és com mesurem el rendiment d'un agent. Com podem veure al llibre [10], considerarem que l'agent rep una recompensa R(s,a,s') quan en l'estat s, fa l'acció a i acaba en l'estat s'. Per a cada tripla, el valor de la funció R podrà ser positiu o negatiu però sempre acotat. En alguns casos més senzills la recompensa podria ser independent de l'acció a i l'estat s' al que arriba.

Llavors, un agent racional voldrà maximitzar les recompenses que rep, veurem més endavant de quina forma. Ja tenim definits tots els conceptes necessaris per introduir la definició següent:

Definició 2.8. *Es denomina procés de decisió de Markov o MDP*[1] *a l'especificació d'un problema de decisió seqüencial per a un entorn completament observable, amb un model de transició de Markov. Aquests processos es defineixen per les quatre components següents:*

- *Conjunt d'estats* S
- *Conjunts d'accions* $A(s)$ $\forall s \in S$
- *Model de Transició:*
  $T : S \times A \times S \longrightarrow \mathbb{R}$ *definida per* $T(s, a, s') = P(S_{t+1} = s' \mid a_t = a, S_t = s)$
- *Funció de Recompensa:* $R : S \times A \times S \longrightarrow \mathbb{R}$

A més, si l'espai d'estats S *i els conjunts d'accions* $A(s)$ *són finits diem que és un procés de decisió de Markov finit.*

Per últim comentarem quina és la naturalesa de la solució d'un MDP. Sabem que una acció pot no tenir sempre el mateix resultat, per tant, la solució no pot ser una seqüència d'accions a realitzar, sinó que caldrà especificar quina acció ha de realitzar l'agent en cada possible estat en què es pugui trobar, per molt poc probable que sigui arribar-hi. D'aquest tipus de solució se'n diu política i la notem per π. L'acció recomanada per la política en l'estat s serà $\pi(s) \in A(s)$.

[1] De l'anglès: *Markov Decision Process*

2.4 Utilitats de les seqüencies

En aquest apartat profunditzarem en com avaluar les seqüències d'estats. Notem que depenent de l'entorn una seqüència pot ser o no finita. Hi haurà entorns que permetran seqüències infinites d'estats, en canvi, hi haurà d'altres que transcorregut un cert temps finalitzin o que quan l'agent arriba a un cert estat, que en diem terminal, es donen per acabat, en aquest cas podrem o no garantir que l'agent l'assoleix sempre.

En els entorns que finalitzen transcorregut un cert nombre N d'estats finalitzen, és a dir, a partir S_N ja no ens interessa el seu estudi, direm llavors que tenim un horitzó temporal finit. Aquesta limitació és causada per l'entorn o el problema que tractem. En cas contrari, direm que l'horitzó temporal és infinit, això no vol dir que necessàriament totes les seqüències d'accions hagin de ser infinites, podrà existir un estat terminal que quan l'agent l'assoleixi el procés es doni per finalitzat. De forma molt excepcional podem garantir que després de realitzar un màxim de M accions l'agent assoleix un estat terminal, el més comú serà que l'agent tinguí l'opció d'evitar de forma indefinida aquests estats. Val a dir que, no necessàriament tots els entorns han de comptar amb estats terminals, pot ser que ens interessi estudiar processos que perdurin de forma infinita.

De forma intuïtiva podem pensar que si l'agent està limitat pel temps i ho pot percebre, pot ser que modifiqui la seva conducta en funció del temps restant, aquest fet en complica força l'estudi, dona lloc a polítiques que denominem no estacionaries, en les quals l'acció pot dependre del temps. Per altra banda, si l'horitzó és infinit la millor acció que es pot realitzar en un determinat estat sempre serà la mateixa. Dona lloc llavors a polítiques estacionàries. Els horitzons infinits i les polítiques estacionàries són més controlables i són les que tractarem en aquest treball.

El que volem és comparar seqüències amb intenció de saber quina és la millor. El que farem és assignar un valor d'utilitat a cada una. Partirem de la funció recompensa, ja que aquesta és més senzilla o almenys depèn de menys variables. D'aquest tipus d'assignació se'n diu utilitat multiatribut i podem trobar més informació sobre el desenvolupament de la seva teoria al llibre [7]. Per tal de definir una forma U de fer aquesta assignació necessitarem definir una ordenació de preferència entre els estats. La hipòtesi més natural és que les preferències de l'agent respecte a les seqüències d'estats siguin estacionàries. Això vol dir el següent: si dues seqüències d'estats $[S_0, S_1, S_2, ...]$ i $[S'_0, S'_1, S'_2, ...]$ comencen des del mateix estat ($S_0 = S'_0$) llavors haurien d'estar ordenades de la mateixa manera que les seqüències $[S_1, S_2, ...]$ i $[S'_1, S'_2, ...]$. D'aquesta hipòtesi se'n dedueix que només existeixen dues formes d'assignar utilitats a les seqüències:

- Utilitat basada en recompenses additives:

$$U([S_0, S_1, S_2, ...]) = \sum_{t=0} R(S_t, a_t, S_{t+1}) \tag{2.2}$$

  Consisteix a assignar el valor de la suma de totes les recompenses que rep l'agent

- Utilitat basada en recompenses depreciatives:

$$U([S_0, S_1, S_2, ...]) = \sum_{t=0} \gamma^t R(S_t, a_t, S_{t+1}) \tag{2.3}$$

  Al factor γ l'anomenem taxa descompte on $\gamma \in (0, 1)$. En aquest apliquem un descompte a les recompenses més gran a mesura que les recompenses són més llunyanes en el temps. Representa la prioritat de l'agent per les recompenses actuals envers les futures.

Una taxa de descompte γ és equivalent a una taxa d'interès del $(1/\gamma) - 1$, una altra forma de pensar-ho. Notem que si considerem $\gamma = 1$ en la segona obtindríem la primera. Però, val la pena mantenir-les diferenciades, ja que no compleixen les mateixes propietats.

Hem vist anteriorment que podien existir seqüències infinites i per tal de poder comparar-les necessitem que totes les utilitats siguin finites. Si tenim la certesa que transcorregut un cer temps T podem assegurar que l'agent assoleix un estat terminal i per tant donem per acabat el problema, podem considerar les utilitats additives, ja que la seva suma serà finita. De forma general no es consideren aquestes utilitats ja que en els problemes no solem tenim aquestes garanties. A més una taxa de descompte encoratja l'agent a realitza les accions el més aviat possible.Considerant utilitats additives ens trobarem amb utilitats infinites perquè estem considerant el sumatori finit d'una sèrie que no necessàriament tendeix a zero. Per tant, en un entorn amb horitzó infinit on no tenim certes condicions, necessitem considerar les recompenses depreciatives:

Proposició 2.9. *Sigui U una forma d'assignar utilitats a seqüències d'estats basada en recompenses depreciatives amb un factor descompte $\gamma \in (0, 1)$ i totes les recompenses estan fitades per R_{max}, llavors la utilitat d'una seqüència $[S_0, S_1, S_2, ...]$ infinita d'estats és finita.*

Demostració. Utilitzant la fórmula de les sumes geomètriques, veurem que la utilitat està acotada:

$$U([S_0, S_1, S_2, ...]) = \sum_{t=0}^{\infty} \gamma^t R(S_t, a_i, S_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma} < \infty \tag{2.4}$$

□

D'aquí en endavant considerarem les utilitats depreciatives tot i que en algun cas concret podrà considerar-se $\gamma = 1$.

2.5 Valor d'un estat i funció Q-valor

En aquest apartat definirem el valor d'un estat seguint una política i el Q-valor associat de triar una acció en un cert estat.

EL valor V d'un estat es defineix a partir de la utilitat U sobre seqüències d'estats i dependrà d'una política π que s'executa. Aquest valor serà la suma esperada de les següents recompenses de forma depreciativa si l'agent duu a terme les accions seguint la política π partint de l'estat actual. Remarcar que U assigna una utilitat a una seqüència concreta, mentre V és el valor esperat de recompensa. Sigui S_t l'estat després d'actuar t vegades sota la política π, tenim:

$$V^{\pi}(s) = E\left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) \mid \pi, s_t = s\right] \tag{2.5}$$

De forma semblant assignarem un valor a realitzar l'acció a l'estat s sota la política π com el valor esperat si comencem a s, fem l'acció a i després actuem sota la política π:

$$Q^{\pi}(s, a) = E\left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) \mid \pi, s_t = s, a_t = a\right] \tag{2.6}$$

D'aquesta funció en diem Q-valors(que prové de Quality) sota la política π.
Aquestes dues funcions tenen un lligam entre elles. Es pot veure que $V^{\pi}(s) = Q^{\pi}(s, a)$ per a $a = \pi(s)$ i es poden reformular introduint el model de transició en el càlcul de l'esperança:

$$Q^{\pi}(s, a) = \sum_{s' \in S'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi}(s')\right] \tag{2.7}$$

On $S' \subset S$ el subconjunt de tots els possibles estats successors de l'estat s. Per tant:

$$V^{\pi}(s) = \sum_{s' \in S'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi}(s')\right] \quad \text{per } a = \pi(s) \tag{2.8}$$

Com podem veure hi ha una relació recursiva directa entre la utilitat un estat i la dels seus successors.

2.6 Política òptima

Com ja hem comentat la solució d'un MDP és una política, però donat un problema podem configurar diverses polítiques. La política que ens interessa trobar és la que maximitza la utilitat esperada de les possibles seqüències.

Definició 2.10. *Anomenarem política òptima a la política en la qual les possibles seqüències tenen una major utilitat esperada d'entre totes les polítiques possibles.*

La política òptima es denota per π^*. L'agent percebrà l'estat actual s i llavors realitzarà l'acció $\pi^*(s)$. Això el converteix explícitament en un agent racional pel fet que realitzarà les accions que maximitzen la utilitat esperada.

Com hem vist en l'apartat anterior podem definir les funcions d'utilitat d'un estat i Q-valor per a totes les polítiques. En concret per la política òptima tenim

que $V^{\pi^*}(s) = \max_{a \in A(s)} Q^{\pi^*}(s,a)$ perquè l'acció escollida serà la que maximitza el valor utilitat. La funció utilitat d'un estat es pot escriure com:

$$V^{\pi^*}(s) = \max_{a \in A(s)} \sum_{s' \in S'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right] \tag{2.9}$$

Això ens permet descriure la política en funció del Q-valors o de les utilitats dels estats:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} Q^{\pi^*}(s,a) = \operatorname*{argmax}_{a \in A(s)} \sum_{s' \in S'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi^*}(s') \right] \tag{2.10}$$

3 Algorismes d'aprenentatge

L'objectiu d'aquest capítol és explicar dos algorismes per trobar la política òptima. El primer algorisme ens permet resoldre un problema MDP del tipus descrit en el capítol anterior en els casos on coneixem prèviament el model de transició i la funció recompensa. Tot seguit, veurem un algorisme per tal que un agent aprengui de forma autònoma quina acció ha de realitzar basant-se en la informació obtinguda de les seves experiències passades alhora que recopila les dades necessàries per prendre millors decisions en un futur, amb la finalitat d'aconseguir aprendre la política òptima.

3.1 Iteració de valors

Considerem, seguint la definició 2.8, un MDP format per (S,A,T,γ,R), on S és el conjunt dels N estats possibles, A el conjunt d'accions i A(s) el conjunt d'accions possibles a l'estat s, T el model de transició γ és el factor de descompte i R(s,a) la funció recompensa. Per tal de solucionar aquest problema hem de trobar la política òptima $\pi*$. La idea darrere l'algorisme d'iteració de valors consisteix a calcular la utilitat V(s) de tots els estats per tal de determinar quina és la millor acció que es pot realitzar per cada un dels estats.

Per tal de calcular la utilitat partirem de l'equació 2.9 que sabem que compleixen tots els valors d'utilitat per a la política òptima. Aquesta equació s'anomena equació de Bellman en honor al seu descobridor Richard Bellman. Aquesta equació és la base de l'algorisme d'iteració de valors per tal de resoldre un MDP.

Si tenim N estats possibles podem escriure N equacions de Bellman, una per cada estat. Aquest sistema amb N equacions té també N incògnites que són les utilitats V(s) de cada un dels estats s. Resolent aquest sistema d'equacions obtindrem, llavors, el que busquem. El principal problema ara és que aquest sistema d'equacions no és lineal, perquè apareix l'operació del màxim. Una solució a aquest inconvenient és fer iteracions per anar aproximant els resultats. Prenent uns valors inicials V(s) de forma arbitrària. Prenent aquests valors seleccionats s'aconsegueixen uns nous valors d'utilitat que ens serviran per calcular la següent aproximació. Repetint el procés de forma successiva.

Sigui $V_k(s)$ el valor d'utilitat de l'estat s a la iteració k, la denominada actualització de Bellman queda de la següent manera:

$$V_{k+1}(s) \longleftarrow \max_{a \in A(s)} \sum_{s' \in S'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right] \tag{3.1}$$

Iterant de forma indefinida està garantit trobar un punt d'equilibri on a cada iteració estem calculant de forma repetida els mateixos valors. Aquest algorisme convergeix per a qualssevol valors inicials que escollim. Es força estàndard triar tots els $V_0 = 0$. Una altra opció és escollir una política π qualsevol, calcular els valors d'utilitat per a tots els estats utilitzant l'equació 2.8 i utilitzar-los com a primera aproximació. Aquesta segona opció pot beneficiar la rapidesa de la convergència si

tenim alguna pista que unes accions són millors que altres. Vegem a continuació l'algorisme:

Algorithm 1 Iteració de Valors per al calcul de la política òptima

Inicialitzar V de forma arbitrària, per exemple: V(s) = 0 $\forall s$
while $\Delta > \epsilon(1-\gamma)/\gamma$ **do**
  $\Delta \leftarrow 0$
  for $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in A(s)} \sum_{s' \in S'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
end while
for $s \in S$ **do**
  $\pi(s) \leftarrow \text{argmax}_{a \in A(s)} \sum_{s' \in S'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi^*}(s')\right]$
end for
return π

Un cop aturem les iteracions i obtenim tots els valors d'utilitat harem d'utilitzar la igualtat dreta de l'equació 2.10 per a tots els estats per tal de trobar així la política òptima.

A la pràctica es pot veure que convergix més ràpidament la política òptima que als valors utilitat, de fet existeix una relació entre els errors d'aquestes dues convergències. Per tant, sigui V_k el vector amb els k-èsims valors utilitat podem triar un ϵ prou petit per aturar les iteracions quan la norma del màxim $||V_k - V_{k+1}|| < \epsilon(1-\gamma)/\gamma$.

3.2 Aprenentatge per reforç actiu

En aquest apartat explicarem un mètode perquè un agent que actua en el seu entorn aprengui la política òptima a través de les recompenses que va obtenint com a conseqüència de les seves accions. Diem que un agent és actiu quan ell mateix ha de decidir quines accions ha de realitzar durant l'aprenentatge i no té cap política preestablerta. Aquesta serà la principal dificultat que haurà de resoldre l'algorisme. L'agent no té informació sobre el model de transició ni sap quines recompenses rebrà per dur a terme cada acció. La idea és que ha d'anar recopilant d'alguna forma aquesta informació, si és necessari al llarg de diversos intents sobre el problema. Anomenarem episodi a cada un d'aquests intents des de l'estat inicial fins, si hi hagués, l'estat final. Ha d'utilitzar la informació que va recopilant per decidir quines accions realitzar. Tot plegat, amb la intenció d'haver-se trobat suficients vegades amb la mateixa situació per a aprendre quina és l'acció que més recompensa esperada li aporta en cada estat.

En comptes de calcular els valors d'utilitat dels estats, l'agent aprendrà quin és el Q-valor de cada parella (s,a) per cada estat i acció. És per això que aquest mètode d'aprenentatge s'anomena Q-aprenentatge, sovint s'utilitza el terme en anglès *Q-*

learning. Un cop els tinguem tots, esbrinar quina és la millor acció en un estat no és una tasca complicada.

A partir de l'equació 2.7 podem deduir que els Q-valors de totes les parelles sota la política òptima compleixen la següent igualtat:

$$Q^{\pi^*}(s,a) = \sum_{s' \in S'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a' \in A(s')} Q^{\pi^*}(s',a') \right] \tag{3.2}$$

Podríem utilitzar aquesta equació directament com una equació d'actualització per anar trobant aproximacions successives dels Q-valors, d'una forma semblant a la iteració de valors. L'inconvenient és que l'agent hauria d'aprendre el model, ja que a la igualtat anterior apareix el terme T(s,a,s').

Una solució implica realitzar el que es diu un aprenentatge de diferència temporal. La intenció principal d'aquests mètodes d'aprenentatge és definir primer les condicions que es compleixen localment quan l'estimació del Q-valor és correcta. Y llavors actualitzar les equacions que estenen els valors d'aquesta l'estimació.

Per dur a terme un a terme un aprenentatge de diferència temporal dels Q-valors podem utilitzar la següent actualització quan l'agent es troba en l'estat s, decideix fer l'acció a i transita a l'estat s':

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,a,s') + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)) \tag{3.3}$$

Notem que és una variació de l'equació 3.2 on apareix α, un nou paràmetre. Es diu que α és la taxa d'aprenentatge perquè introduïm la nova informació obtenim en realitzar una acció ponderada per aquest factor i ens quedem amb el valor teníem fins ara a raó de $1 - \alpha$.

La característica més important d'aquesta regla d'actualització és que no apareix el terme T(s,a,s') i, per tant, no ens fa falta conèixer el model per tal d'executar-la. Un altre punt rellevant és que només estem tenint en compte l'estat successor s' al qual l'agent acaba de transitar i no tots els estats als quals podia haver transitat. Llavors, tots els termes que apareixen seran coneguts en el moment de dur a terme una iteració.

Podríem pensar que tenir només l'estat successor que observem pot produir un canvi inapropiat en el valor de Q(s,a) quan es produeix una transició que és poc probable. De fet, aquestes transicions succeiran poques vegades i, per tant, el valor mitjà Q(s,a) anirà convergit al valor esperat si prenem un valor α prou petit. Ho farà d'una forma més lenta que si l'agent coneixes el model i els valors tindran unes fluctuacions més elevades, però els càlculs són molt més senzills a cada iteració i sobretot podem prescindir de l'aprenentatge del model. Aquests dos últims beneficis fan de l'algorisme Q-aprenentatge una eina molt potent i revolucionaria per a solucionar problemes que no consten d'un nombre d'estats massa gran.

Per tal de reduir les fluctuacions de les estimacions successives dels Q-valors i assegurar-ne la convergència podem canviar el valor fix de la taxa d'aprenentatge per una funció que decreixi en funció del nombre de vegades que hem visitat cada

estat. Caldrà llavors una taula de freqüències N(s) que anem incrementant cada cop que l'agent visita un estat. Sigui n el nombre de vegades que hem visitat un estat, es requereixen dues restriccions sobre la funció $\alpha(n)$ per assegurar la convergència:

- $\sum_{n=1}^{\infty} \alpha(n) = \infty$
- $\sum_{n=1}^{\infty} \alpha^2(n) < \infty$

La funció $\alpha(n) = 1/n$ compleix les condicions. Funcions inversament proporcionals a n són bones candidates.

Usar un valor α fix pot tenir avantatges quan el nostre objectiu final consisteix a trobar π^* i no hi ha un interès ferm en la convergència dels Q-valors. Hem d'anar amb compte, perquè la tria de valors petits endarrereixen l'aprenentatge, però, per altra banda, si triem valors massa grans correm el risc d'incórrer en fluctuacions grans del Q-valors que distorsionin π^* en alguna de les iteracions. Una altra característica important és com l'agent tria les següents accions que ha de realitzar mentre computa els Q-valors. Una primera aproximació podria ser utilitzar els Q-valors apresos fins al moment per tal d'escollir la millor acció amb la informació que tenim fins ara. Quan l'agent és a l'estat s pot triar l'acció que té el màxim Q-valor en aquest moment. Aquesta tria se'n diu explotació, ja que maximitza la recompensa esperada. D'un agent que només tria les accions d'aquesta forma se'n diu agent voraç. Aquests agents solen trobar una solució al problema però, es difícil que convergeixin a la política òptima, perquè un cop han trobat una seqüència d'accions que soluciona el problema o funciona mínimament la repeteixen amb variacions molt petites i es queden estancats. Algunes vegades convergeixen a polítiques força dolentes.

Un agent voraç considera que els Q-valors que han après són l'esperança real de prendre cada acció i no una aproximació. Si l'agent és capaç de millorar aquestes aproximacions podrà prendre millors decisions en un futur i potser trobar una política amb què obtingui millors recompenses esperades. Llavors un agent hauria de considerar provar seqüències que encara no ha intentat o que ha intentat poques vegades per veure si la informació que extreu serveix per millorar la política considerada òptima amb la informació extreta fins al moment. D'aquesta tria se'n diu exploració. Un agent no arriba gaire lluny si només explora, ja que sempre està considerant opcions noves i no explota les conegudes per tal d'acabar de refinar-les. Per tant, un agent hauria de dur a terme exploració i exploració.

La forma més senzilla de combinar l'exploració i l'explotació és considerar una probabilitat ϵ. Per decidir quina acció realitzar quan ens trobem en l'estat s escollirem amb una probabilitat $1 - \epsilon$ l'acció que té el màxim Q-valor de totes les disponibles. Per altra banda escollirem amb una probabilitat ϵ una acció de forma aleatòria. D'aquesta manera estarem explorant tot l'espai d'accions. Aquesta metodologia tot i que és la més senzilla d'executar te dos inconvenients. Continuarem explorant amb una probabilitat ϵ inclús quan ja tinguem la suficient informació. Podem solucionar aquest aspecte triant ϵ una funció que decreixi amb el nombre d'episodis. De forma que a partir de l'episodi M_ϵ deixem d'explorar triant accions de forma aleatòria i només explotem la informació que tenim. Un altre aspecte de

la tria aleatòria d'accions és que sovint fem accions que ja hem provat suficients vegades i ja podem determinar que no són gaire bones, o almenys no són les millors.

Una alternativa més complexa que l'anterior és considerar una funció d'exploració. Aquesta funció f(q,n) determina el compromís entre la voracitat i la curiositat, on q és l'estimació d'un Q-valor que tenim d'una parella (s,a) i n el nombre de vegades que s'ha realitzat l'acció a en l'estat s. En un estat s escollirem l'acció a de la manera següent:

$$a = \arg\max_{a' \in A(s)} f(Q(s,a'), N(s,a')) \tag{3.4}$$

on N(s,a') es la freqüència de la parella (s,a'). Una bona funció d'exploració ha de créixer si ho fa q i decreix quan n augmenta. A continuació donem dues propostes de funció d'exploració:

$$f(q,n) = \begin{cases} R^+ & n < N_e \\ \\ q & n \geq N_e \end{cases} \tag{3.5}$$

on R^+ és una estimació optimista de la millor recompensa que es pot obtenir en qualsevol estat i N_e un nombre natural fixat. Ens assegurem així explorar almenys N_e vegades cada parella (s,a).

$$f(q,n) = q + k/n \tag{3.6}$$

on k és una constant, amb aquesta funció donem un valor extra a les accions que s'han realitzat menys vegades.

Algorithm 2 Q-aprenentatge

Inicialitzar Q(s,a) de forma arbitrària, per exemple: Q(s,a) = 0 $\forall s, a$
for number of episodes **do**
  $s \leftarrow s_0$
  while s not terminal **do**
    $a \leftarrow \text{argmax}_{a' \in A(s)} f(Q(s,a'), N(s,a'))$
    Fer l'acció a, observar r,s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha(N(s))(R(s,a,s') + \gamma Q(s',a') - Q(s,a))$
  end while
end for

El mètode Q-aprenentatge és una forma diferent d'emmagatzemar la informació de la utilitat amb la propietat important: un agent que realitza un aprenentatge de diferència temporal per aprendre els Q-valors no necessita conèixer un model ni per aprendre ni per seleccionar les accions. Diem llavors que el Q-aprenentatge es un mètode lliure de model(*model-free*).

4 Aprenentatge amb múltiples criteris

Fins ara hem considerat entorns on un agent ha de realitzar una tasca tenint en compte un únic criteri. En entorns més realistes és força comú que s'hagin de satisfer múltiples objectius a la vegada, per exemple, assolir una fita el més ràpid possible mentre es manté el cost el més baix possible. Sovint no és possible optimitzar els dos objectius alhora. Podria ser que la forma més ràpida no fos la més barata o viceversa. En aquest capítol estendrem els conceptes que ja hem vist per tal de treballar amb entorns on es consideren recompenses associades a diferents motivacions. Entendrem perquè en aquests casos hi ha diverses polítiques òptimes i desenvoluparem un algorisme per tal d'aprendre-les totes. Podem trobar informació detallada sobre aquest tema en el llibre [6]

4.1 Generalització multiobjectiu

El primer concepte que cal expandir és la funció recompensa. Ara, en comptes de rebre un sol valor a cada transició, l'agent rebrà diversos valors en forma de vector. De forma general el vector tindrà una component associada a cada objectiu. Per tant, on teníem una funció real de recompensa R(s,a) ara considerarem una funció vectorial $\vec{R}(s,a) = [R_1(s,a), R_2(s,a), ..., R_d(s,a)]$. De forma semblant al cas més simple, cada component pot ser positiva o negativa i de diferents magnituds, però hauran d'estar acotades. Tot plegat, ha d'estar relacionada amb el valor que aporta la realització de cada acció a cadascun dels objectius.

Un agent podrà optimitzar una gran varietat de funcions d'agregació d'aquestes recompenses, la més senzilla que podem considerar és la suma ponderada. Triarem un vector pes $\vec{w} \in \mathbb{R}^d$ associat a les preferències de l'agent sobre les diferents recompenses. D'aquesta manera, per cada $\vec{w}$ podrem obtenir la següent funció recompensa sobre els reals:

$$R_{\vec{w}}(s,a) = \vec{w} \cdot \vec{R}(s,a) \tag{4.1}$$

Aquesta funció és real i acotada. Per tant, podrem executar els algorismes vists anteriorment per tal de trobar la seva política òptima $\pi_{\vec{w}}$. Llavors, podem calcular la política òptima per a cada $\vec{w}$. La política òptima resultant dependrà de forma directa de les preferències de l'agent sobre les diferents recompenses. Seran aquestes les que donaran en gran part la seva forma. No tots els vectors $\vec{w}$ tindran la mateixa política òptima associada, però pot donar-se el cas que un conjunt de vectors generin la mateixa.

Prenent un $\vec{w}$ fix i considerem tots els seus múltiples escalars. Cada múltiple genera diferents funcions $R_{\vec{w}}(s,a)$, però és fàcil veure que tots ells generaren la mateixa política òptima. Per tant, sense pèrdua de generalitat, podem considerar les classes de vectors, prenent com a representant els vectors que compleixen $\sum_i w_i = 1$.

Sabem que donat un vector $\vec{w}$ és possible calcular la seva política òptima associada, però aquest fet no succeeix de forma inversa. Per cada política no existeix un vector $\vec{w}$ tal que aquesta política és l'òptima associada. Per tal d'estudiar un cas

amb múltiples objectius ens interessa donar resposta a aquesta pregunta: quines polítiques val la pena considerar, quines són òptimes per algun vector $\vec{w}$? Ens interessa poder aprendre-les totes. Així donada qualsevol preferència serem capaços de donar solució al problema.

4.2 Algorisme d'iteració de valors Convex Hull

En aquest apartat explicarem com aprendre totes les possibles polítiques òptimes donat un problema on existeixen diversos criteris i també les definicions i conceptes necessaris per comprendre'l. Es tracta del *Convex hull value iteration algorithm* i el podem trobar a l'article [2], així com la seva demostració.

Definició del problema

Considerarem un MDP format per (S,A,T,γ,$\vec{R}$), on S és un conjunt finit amb N estats, A(s) el conjunt d'accions disponibles en l'estat $s \in S$, T el corresponent model de transició, $\gamma \in [0, 1)$ el factor de descompte i $\vec{R} : S \times A \longrightarrow \mathbb{R}^d$ és la funció vectorial de recompensa amb d components. Una política π assignarà una acció $a \in A(s)$ cada estat $s \in S$

La funció valor seguint una política d'accions π qualsevol i avaluada en un estat s_i serà llavors el vector:

$$\vec{V}^{\pi}(s_i) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \vec{R}(s_{i+k}, a_{i+k}) \mid \pi\right] \tag{4.2}$$

on l'esperança es calcula sobre el model de transició dels estats i la seqüència de recompenses que s'obté si s'executa la política π partint de l'estat s_i. De forma semblant el Q-valor de realitzar l'acció a en l'estat s serà el vector:

$$\vec{Q}^{\pi}(s, a) = \mathbb{E}\left[\vec{R}(s, a) + \gamma \vec{V}^{\pi}(s')\right] \tag{4.3}$$

on l'esperança es calcula sobre tots els possibles estats successors s' utilitzant la distribució de probabilitats inclosa en el model de transició T(s,a,s') = P(s'|s,a). Per tal de trobar quines són totes les polítiques òptimes considerarem el conjunt de vectors que poden prendre els Q-valors per a cada parella (s,a) per a totes les possibles polítiques.

Convex Hull

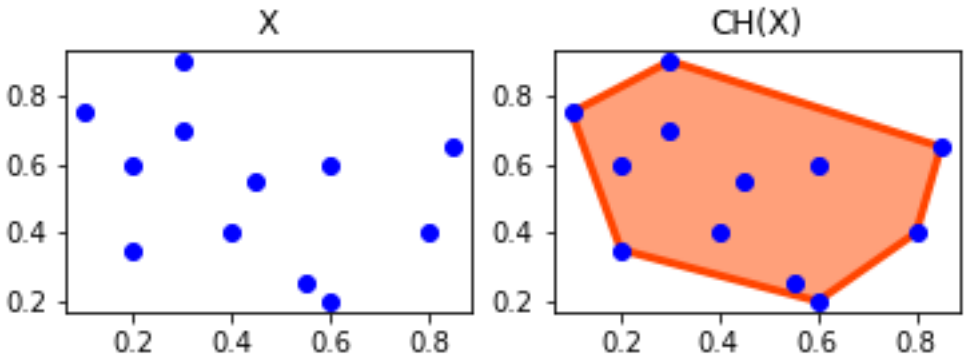
Un concepte important per entendre l'algorisme que veurem més tard és la definició d'Envolupant Convexa o en anglès *Convex Hull* sobre un conjunt un de punts:

Definició 4.1. *Donat X un conjunt de k punts $p_1, p_2, ..., p_k \in \mathbb{R}^n$ anomenem Convex Hull o Envolupant Convexa a la intersecció de tots els conjunts convexos que*

contenen X. Donat un conjunt X l'expressió de la seva Convex Hull és:

$$CH(X) = \left\{ \sum_{i=1}^{k} \alpha_i p_i \mid p_i \in X, \alpha_i \in \mathbb{R}, \alpha_i \geq 0, \sum_{i=1}^{k} \alpha_i = 1 \right\} \tag{4.4}$$

Exemple 4.2. Calculem la Convex Hull sobre el conjunt X format per 12 punts en el pla:
X = {(0.55, 0.25), (0.6, 0.2), (0.8, 0.4), (0.2,0.35), (0.85, 0.65), (0.3, 0.7), (0.2, 0.6), (0.4, 0.4), (0.6, 0.6), (0.3, 0.9), (0.45,0.55), (0.1,0.75)}



Notem que la Convex Hull d'un conjunt queda determinada de forma inequívoca pels seus vèrtexs en qualsevol dimensió. Per tal de desenvolupar l'algorisme el que ens interessen són els vèrtex d'una Convex Hull.

Denotarem per $\overset{\circ}{Q}(s,a)$ els vèrtexs de la Convex Hull sobre tots els possibles vectors Q-valor que pot prendre si realitzem l'acció a en l'estat s. Aquests són els Q-valors maximals per algun pes $\vec{W}$. Per tal de construir l'algorisme definirem una sèrie d'operacions sobre aquests conjunts:

Definició 4.3. *Sigui $\overset{\circ}{Q}$ el conjunt dels vertex d'una Convex Hull definim l'operació de translació pel $\vec{u}$ i el producte escalar per $b \in \mathbb{R}$ com:*

$$\vec{u} + b\overset{\circ}{Q} \equiv \{\vec{u} + b\vec{q} : \vec{q} \in \overset{\circ}{Q}\} \tag{4.5}$$

Definició 4.4. *Siguin $\overset{\circ}{Q}$ i $\overset{\circ}{U}$ els conjunt dels vèrtex de dues Convex Hull definirem la suma de dues Convex Hulls com:*

$$\overset{\circ}{Q} + \overset{\circ}{U} \equiv CH(\{\vec{q} + \vec{u} : \vec{q} \in \overset{\circ}{Q}, \vec{u} \in \overset{\circ}{U}\}) \tag{4.6}$$

Algorisme

De forma semblant al que hem vist anteriorment (equació 4.3) podem definir una relació de recurrència entre cada un dels conjunts $\overset{\circ}{Q}(s,a)$ i els conjunts dels estats

successors. La relació queda de la següent manera:

$$\overset{\circ}{Q}(s,a) = \mathbb{E}\left[\vec{R}(s,a) + \gamma\ CH\left(\bigcup_{a'} \overset{\circ}{Q}(s',a') \mid s,a\right)\right] \tag{4.7}$$

On $a' \in A(s')$. Encara que el càlcul de l'esperança sobre Convex Hull pot semblar entranya es pot dividir, de la forma usual, en sumes i productes de la forma que hem definit anteriorment sobre els possibles estats successors s' i les probabilitats establertes al model de transició. Ja tenim tots els conceptes necessaris per introduir l'algoritme:

Algorithm 3 Convex Hull Value Iteration

Inicialitzar $\overset{\circ}{Q}(s,a)$ arbitràriament $\forall s, a$
while not convergeix **do**
  for all $s \in S, a \in A$ **do**
    $\overset{\circ}{Q}(s,a) \leftarrow \mathbb{E}\left[\vec{R}(s,a) + \gamma\ CH\left(\bigcup_{a'} \overset{\circ}{Q}(s',a') \mid s,a\right)\right]$
  end for
end while

Com es pot veure, en coptes d'anar actualitzant les recompenses màximes esperades amb descompte el que guardem és el conjunt de Q-valors que són maximals per algun $\vec{w}$.

Un cop hem executat l'algorisme trobar quina és la millor acció en un estat s per a un determinat vector és molt més senzill, es tracta de trobar l'acció amb el màxim Q-valor definit de la forma següent:

$$Q_{\vec{w}}(s,a) \equiv \max_{\vec{q} \in \overset{\circ}{Q}(s,a)} \vec{w} \cdot \vec{q} \tag{4.8}$$

Convergència

Per últim, cal aclarir quan l'algorisme convergeix per tal d'aturar-lo. L'algorisme ha convergit quan estem calculant les mateixes Convex Hull una vegada i una altra. Això significa que podem reutilitzar la informació de la iteració anterior.

La idea és fer un seguiment dels punts que pertanyen a cada Convex Hull. Si un nou punt no pertany la farem més gran, per contra, dels punts que ja pertanyien guardarem el vector apuntant a la direcció en la que són maximals. Tot seguit en la següent iteració podem comparar quan s'han mogut aquests vectors per tal de determinar si ja hem convergit.

5 Comportament ètic en un problema multiobjectiu

En aquest capítol començarem introduint els comportaments ètics en els problemes multiobjectiu. Enunciarem sota quines condicions un agent té un dilema ètic. Per acabar desenvoluparem un algorisme per garantir que un agent que aprèn mitjançant un algorisme d'aprenentatge per reforç aprèn a comportar-se de forma ètica davant els dilemes que es pugui trobar.

5.1 Formalització del problema

En aquest apartat donarem les bases formals per tal de considerar un problema de comportament ètic. Un agent que ha de complir una tasca de la forma òptima es troba en un problema d'aquest tipus si alhora l'ha de dur a terme de la forma més ètica possible. Aquests problemes es poden dividir en dues parts: especificació de les recompenses (transformar el coneixement ètic que tenim sobre l'entorn en recompenses per l'agent) i inserció ètica (per garantir que les recompenses garanteixen que l'agent es comporta de forma ètica). Un agent en aquesta situació rebrà recompenses amb dos criteris diferents. El primer, segons l'aportació de les seves accions a l'objectiu particular que ha de complir, el segon criteri anirà lligat al valor ètic que assignem a les seves accions.

Tal com podem veure a [8] avaluarem el comportament ètic des de dues perspectives. Per una banda, des d'una dimensió normativa que castiga el trencament dels requeriments normatius. D'altra banda, una dimensió avaluadora, que premia accions moralment lloables.

Definició 5.1. *Sigui $M = \langle S, A, (R_0, R_N + R_E), T\rangle$ un problema MDP multiobjectiu (MOMDP) on R_0 correspon a la recompensa associada l'objectiu individual de l'agent, diem que M és un MOMDP ètic si i només si:*

- *$R_N : S \times A \longrightarrow \mathbb{R}^-$ és una funció de recompensa normativa que penalitza trencar els requeriments normatius.*
- *$R_E : S \times A \longrightarrow \mathbb{R}^+$ és una funció de recompensa avaluadora, que recompensa de forma positiva realitzar accions definides com moralment lloables.*

Dividint en dos la funció de recompensa ètica ens permet evitar el problema que l'agent aprengui a maximitzar el nombre d'accions moralment positives alhora que realitza accions que trenquen les normes.

La noció de política ètica que utilitzarem serà la següent: una política ètica obeeix totes les normes mentre es comporta de la forma més lloable possible. Veiem una definició més formal:

Definició 5.2. *Sigui M un MOMDP ètic. Diem que la política π_* és una política ètica sobre M si i només si el seu vector $\vec{V^{\pi_*}} = (V_0^{\pi_*}, V_N^{\pi_*}, V_E^{\pi_*})$ és òptim per als seus objectius ètics. És a dir, compleixen:*

$$V_N^{\pi_*} = \max_{\pi} V_N^{\pi}$$

$$V_E^{\pi_*} = \max_{\pi} V_E^{\pi}$$

Les polítiques que no compleixen aquesta definició s'anomenen polítiques no ètiques.

Un cop ja tenim la definició de política ètica podem definir formalment les polítiques ètiques òptimes, que són les que voldrem que l'agent aprengui. Les polítiques ètiques òptimes corresponen a les polítiques que persegueixen l'objectiu individual mentre l'objectiu ètic es compleix. Diem que una política ètica és òptima si és ètica i maximitza les recompenses de l'objectiu individual, de manera formal:

Definició 5.3. *Donat un MMODP $M = \langle S, A, (R_0, R_N + R_E), T \rangle$, una política π_* és etico-òptima en M si i només si és maximal sobre tot el conjunt $\prod_e$ de polítiques òptimes:*

$$V_0^{\pi_*} = \max_{\pi \in \prod_e} V_0^{\pi}$$

Notem que llevat poden haver-hi diferents polítiques etico-òptimes totes comparteixen el mateix vector $\vec{V^*}$

En una inserció ètica volem transformar un problema ètic MOMDP en un problema amb un únic objectiu del tipus MDP, on l'agent aprendrà la seva política a través de la funció d'inserció (o, en Anglès, *embedding*) f_e. En concret si aquesta funció és lineal, diem que estem aplicant una inserció amb una ponderació lineal. Volem trobar una funció inserció que només permeti a l'agent aprendre polítiques etico-òptimes. En la seva forma més senzilla, aquesta funció inserció serà lineal:

$$f(\vec{V^{\pi}}) = \vec{w} \cdot \vec{V^{\pi}} = w_0 V_0^{\pi} + w_e (V_N^{\pi} + V_E^{\pi}) \tag{5.1}$$

on $\vec{w} = (w_0, w_e)$ el vector de pesos amb $w_0, w_e > 0$ per garantir que l'agent te en compte totes les recompenses.

Problema 5.4. INSERCIÓ ÈTICA
Sigui $M = \langle S, A, (R_0, R_N + R_E), T \rangle$ un MOMDP ètic. Trobar $\vec{w}$ amb components positives de forma que totes les polítiques òptimes en el problema MDP $M' = \langle S, A, w_o R_0 + w_e (R_N + R_E), T \rangle$ són etico-òptimes en M.

5.2 Solució al problema d'inserció ètica

En aquesta secció explicarem com calcular el vector $\vec{w}$ per solucionar el problema d'inserció ètica. Per fer-ho utilitzarem l'algorisme ètic d'inserció que podem trobar a l'article [8]. L'algorisme es divideix en 3 passos: Càlcul de la Convex Hull; Extracció de dos vectors valor maximals i Obtenció del mínim w_e.

Càlcul de la Convex Hull

El primer que hem de fer és computar la Convex Hull per al nostre problema multiobjectiu. Un fet importat en aquest cas és que no necessitem calcular la Convex Hull sencera. Com ja sabem, la CH conté tots els vectors etico-òptims $\vec{V}^*$, necessaris per trobar $\vec{w}$. De fet, com el vector $\vec{V}^*$ és el mateix per totes les polítiques etico-òptimes, podem calcular qualsevol $P \subset CH$ que contingui almenys una política etico-òptima.

Podem convenir llavors buscar en l'espai de vectors que compleixen que són de la forma $\vec{w} = (1, w_e)$ amb $w_e > 0$. Això redueix força l'alt cost computacional.

Extracció de dos vectors valor maximals

Per saber quin és el vector valor en P que correspon a la política òptima hem de trobar el que maximitza la funció $(V_N + V_E)$ del MOMDP ètic. De manera formal obtenim el vector de la següent manera:

$$\vec{V}^*(s) = \underset{(V_0, V_N+V_E)\in P}{\arg\max} [V_N(s) + V_E(s)] \qquad \forall s \tag{5.2}$$

De forma semblant trobem $\vec{V'}^*$ el segon millor vector de la forma:

$$\vec{V'}^*(s) = \underset{(V_0, V_N+V_E)\in P\setminus\{V^*\}}{\arg\max} [V_N(s) + V_E(s)] \qquad \forall s \tag{5.3}$$

Que acumula la quantitat més gran de recompensa ètica si no considerem V^*.

Mínim w_e

Amb el següent teorema veurem que només necessitem comparar $\vec{V}^*$ i $\vec{V'}^*$ per trobar w_e:

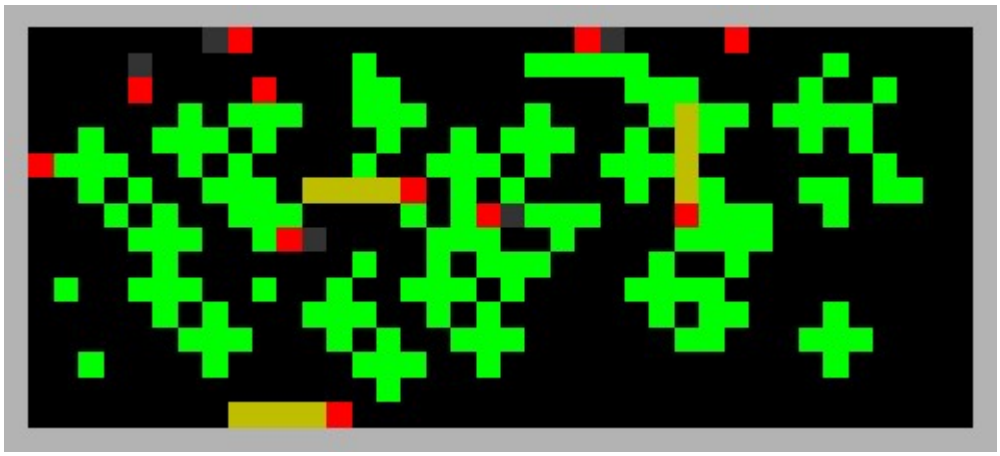
Teorema 5.5. *Sigui $M = \langle S, A, (R_0, R_N + R_E), T\rangle$ un MOMDP ètic on existeix almenys una política ètica. Sigui $P \subset CH(M)$ el subconjunt de la Convex Hull de M, limitada als vectors pes de la forma $\vec{w} = (1, w_e)$ amb $w_e > 0$. Considerant $\vec{V}^*$ la política etico-òptima i $\vec{V'}^*$ el segon millor vector valor, si per a un vector $\vec{w} = (1, w_e)$ tenim que $\vec{w} \cdot \vec{V}^* > \vec{w} \cdot \vec{V'}^*$ llavors $\vec{V}^*$ és l'única política $\vec{w}-$òptima de P.*

6 Commons Game

En aquest capítol explicarem amb detall en què consisteix l'entorn Commons Game proposat per DeepMain en l'article [5] i explicarem el problema ètic que hem creat sobre aquest entorn. Finalment, utilitzarem els coneixements detallats en aquest treball per resoldre el problema mitjançant la computació de l'algorisme ètic d'inserció sobre el problema. Per generar una inserció a un problema d'un únic objectiu de forma que totes les polítiques òptimes que un agent pugui aprendre siguin ètiques.

6.1 Definició de l'entorn

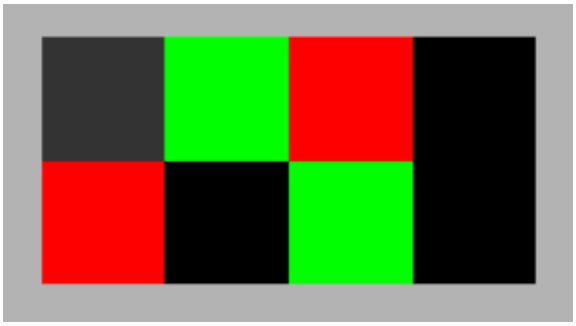
De forma semblant a la tragèdia dels béns comuns [4] tenim un entorn on uns agents han de recollir pomes en un tauler en forma de quadrícula. En l'entorn original les pomes es regeneren segons la quantitat de pomes que tenen a prop. Per tant, si els agents realitzen una estratègia agressiva recollint totes les pomes al seu abast, el nombre de pomes que es regeneren disminueix fins a l'extinció i, per tant, la quantitat de pomes que els agents pot recollir. Ens basarem en l'entorn original[3] que es troba implementat utilitzant pycolab de Python per al motor del joc.



En la imatge podem veure un fotograma d'aquest joc. Els agents (color vermell) poden moure's en les quatre direccions al llarg del taulell fins on es troben les pomes (color verd) per tal de recollir-les un cop arriben a una casella on hi ha una. En aquesta implementació els agents poden disparar un raig (color groc), si aquest raig toca un altre agent, aquest queda paralitzar durant un determinat nombre de torns. Com hem dit, si els agents recullen moltes pomes, la velocitat de regeneració disminueix, per això, els agents poden voler paralitzar els altres amb el raig per tal d'evitar que recullin pomes de forma desmesurada. Per apuntar amb el raig, l'agent pot rotar un quart de volta en el sentit de les agulles del rellotge o, si ho prefereix, pot fer una rotació en sentit contrari. Per tant, en cada estat un agent pot: moure's en alguna de les 4 direccions, rotar la seva orientació en algun dels dos sentits i disparar en la direcció que està apuntant.

6.2 Adaptació de l'entorn

El principal canvi de l'adaptació que hem decidit utilitzar consisteix a permetre als agents donar les pomes al que anomenem common pool, un fons comú on s'emmagatzemen els recursos d'ús compartit per tots els agents. Els agents no només poden donar pomes, sinó que també poden agafar pomes d'aquest fons quan ho considerin. Hem decidit eliminar l'acció de disparar el raig, ja que en aquesta versió de l'entorn no serà necessària. Lligat a això els agents ja no necessiten orientar-se per disparar i, per tant, ja no caldrà que realitzin les accions de rotació. Les pomes apareixen en unes caselles preestablertes. Quan una poma és collida la casella on era queda buida. A cada torn, hi ha una probabilitat que aparegui una nova poma en una de les destinades si aquesta està buida. És a dir, si ja ha sigut collida i no hi ha cap agent en aquesta posició. Treballarem amb dos agents i un taulell de dimensions de 4×2 caselles, considerarem les caselles com una matriu i utilitzarem les coordenades (col, fil) per referir-nos a les caselles amb $col \in 0, 1, 2, 3$ i $fil \in 0, 1$. Les pomes apareixen en les 4 caselles centrals. Per exemple en el següent mapa, els agents es troben a les caselles (2,0) i (0,1) respectivament i hi ha pomes en les caselles (1,0) i (2,1):



La simulació es desenvoluparà el llarg de 36 torns. Una vegada hagin transcorregut els torns, el joc tornarà a començar. Inicialment, l'Agent#0 es troba a la cantonada dreta superior (3,0) i l'Agent#1 es troba a la cantonada inferior esquerra (0,1). En cada torn, els dos agents realitzaran una acció de forma simultània, això els durà al següent estat. L'objectiu individual de cada agent és posseir el major nombre possible de pomes en finalitzar el joc, tot i que un agent en te prou per a sobreviure si al final d'una simulació posseeix 4 pomes.

Formalització dels estats

Passem ara a definir el conjunt S d'estats possibles i de quina forma l'agent els percep i els diferencia entre ells. Donat l'entorn de simulació que acabem de descriure es poden donar diverses disposicions. Cadascun dels 2 agents es pot trobar el 8 caselles diferents. En cada una de les 4 caselles centrals pot haver-hi o no pomes. Cada un dels agents pot posseir menys de 4 pomes, exactament 4 o més de 4. I per últim el common pool pot estar buit o contenir pomes. Tota aquesta informació la codificarem com una tupla, amb l'estructura següent:

(Posició dels agents, Posició de les pomes, Possessió de pomes, Common Pool)

- Posició dels agents = (casella Agent#0, casella Agent#1)
- Posició de les pomes = $\{(x, y) \in [0, 3] \times [0, 1] \mid casella(x, y) = poma\}$
- Possessió de pomes = (C_0, C_1) on:

$$C_i = \begin{cases} -1 & Pomes(Agent\#i) < 4 \\ 0 & Pomes(Agent\#i) = 4 \\ +1 & Pomes(Agent\#i) > 4 \end{cases}$$

- Common Pool = CP on CP = True si hi ha pomes o CP = False si està buit

Veiem com codificaríem l'estat inicial s_0 . Per diferenciar els agents hem decidit que l'Agent#0 serà de color blau i l'Agent#1 continuarà sent vermell, els agents comencen sense cap poma i el Common Pool comença buit:

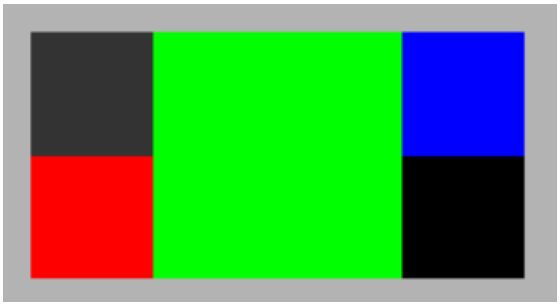


Figura 1: Estat inicial s_0

$$((3, 0), (0, 1)), ((1, 0), (2, 0), (1, 1), (2, 1)), (-1, -1), False)$$

Tot i que pot semblar un espai reduït, sota aquestes condicions disposem de 10656 estats possibles un nombre considerablement gran.

Conjunt d'accions i model de transició

Com ja hem avançat abans un agent podrà realitzar una de les següents accions en cada torn. Desplaçar-se en una de les 4 direccions cardinal si la casella adjacent corresponent, si es desplaça a una casella on hi ha una poma la seva recollida va implícita en aquest desplaçament. A més pot realitzar dues accions extres: donar una poma al common pool sempre que tingui almenys una poma en la seva possessió i agafar una poma del common pool quan hi hagi alguna disponible.

Pel que fa al model de Transició, nosaltres ens mirarem la simulació des del punt de vista de l'Agent#0 que serà l'agent que volem que aprengui una política ètica,

l'Agent#1 durà una política fixa. Donat un estat sempre realitzarà la mateixa acció. Aquesta política hem convingut que serà egoista, l'Agent#1 només mirarà pels seus interessos. Recollirà totes les pomes al seu abast, inclús agafarà pomes del common pool per tal d'obtenir el major nombre de pomes en finalitzar la simulació. Per tant donat un estat sempre sabrem quina acció realitzarà l'Agent#1, per tant, aquesta serà determinista, ja que sempre serà la mateixa.

El que depèn de l'atzar és la regeneració de pomes. No podem preveure on apareixeran. Per tal d'aproximar el model de transició i poder avançar hem aproximat aquestes probabilitats. El nombre d'estats successors depèn de quantes pomes falten, per exemple si en falten 2 es pot transitar a 4 estats diferents: Un en el que no apareix cap poma, un en el que apareixen les dues pomes i dos en els que apareix una sola poma, cada una respectivament. Hem decidit aproximar les probabilitats de forma que assignem la mateixa probabilitat de succeir a tots els possibles estats successors, en el cas que faltin dues pomes la probabilitat de cadascun dels 4 possibles estats successors seria 1/4. No seran les probabilitats reals, però és una aproximació prou bona per tal que els resultats obtinguts continuïn tenint sentit.

Donat un estat podem saber on transitarà l'Agent#2 de forma determinista, quin son els possibles estats successors en funció de les pomes que manquen i assignarlos una probabilitat. Tenim llavors la forma general del model de transició. En la implementació la funció "transition"a l'arxiu model.py genera tots els possibles estats successors.

6.3 Funcions recompensa

Com ja hem comentat l'objectiu individual de cada agent és acaparar el màxim nombre de pomes, veiem a continuació la funció de recompensa associada a l'objectiu individual:

$$R_0(s,a,s') = \begin{cases} +1 & \text{Casella Agent\#0(s')} = \text{poma} \\ +1 & \text{a} = \text{agafar poma del Common Pool} \\ -1 & \text{a} = \text{donar poma al Common Pool} \\ 0 & \textit{altrament} \end{cases} \tag{6.1}$$

Recollir una poma d'una casella o agafar-la del common pool apropa l'agent al seu objectiu. Mentre que donar una poma l'allunya d'ell.

Per una altra banda, tenim l'objectiu ètic, com hem vist a la definició 5.1 es subdivideix en dos blocs, les accions lloables i les que infringeixen les normes ètiques:

- Definiríem R_N una funció de recompensa normativa que penalitza l'agent per trencar les normes, en aquest cas voldrà dir que l'agent agafa pomes del

common pool quan ja en te suficient, és a dir l'agent posseeix 4 o més pomes:

$$R_N(s,a,s') = \begin{cases} -1 & \text{a = agafar poma del Common Pool and } C_0 \geq 0 \\ 0 & \textit{altrament} \end{cases} \tag{6.2}$$

- Definirem R_E una funció de recompensa avaluadora que premia les accions lloables. En aquest cas això voldrà dir donar una poma quan l'agent te estrictament més de 4 pomes i s'ho pot permetre sense quedar-se amb pomes insuficients.

$$R_E(s,a,s') = \begin{cases} -1 & \text{a = donar poma al Common Pool and } C_0 = 1 \\ 0 & \textit{altrament} \end{cases} \tag{6.3}$$

6.4 Solució problema d'inserció ètica al Commons Game

Tot plegat fa que l'Agent#0 es trobi en un MOMDP ètic com el definit en el problema 5.4 i que podem solucionar amb l'algorisme d'inserció ètica explicat en l'apartat 5.2, per tal de trobar w_e. En aquest apartat comentarem les dues funcions més importants de la nostra implementació d'aquest algorisme sobre aquest problema.

Parcial Convex Hull

Donat un conjunt de punts la següent funció calcula el subconjunt de la Convex Hull que ens interessa i en retorna els vèrtexs. Utilitzem la funció ConvexHull de scipy, podem trobar informació sobre el seu funcionament a l'article [1].

```
def partialHull(points):
    try:
        max0 = np.max(points[:, 0])
        max1 = np.max(points[:, 1])
        extreme = np.array([max0,max1])

        for p in points:
            c = p == extreme
            if c.all():
                return np.array([extreme])
        hull_points = np.append(points, np.array([extreme]), axis=0
                                            )
        fullHull = ConvexHull(hull_points, qhull_options='QG' + str
                                            (len(points)))
        return np.array([fullHull.points[x] for x in set(fullHull.
                                            simplices[fullHull.good].
                                            flatten())])
    except:
        return points
```

Notem que utilitzem un `try`, `except` a causa del fet que hi ha conjunts de punts sobre els quals no podem calcular la Convex Hull, per exemple, quan els punts estan alineats. Un altre possibilitat que genera un error és quan demanem el càlcul sobre 2 o menys punts. En aquest cas retornem els punts d'entrada.

Convex Hull Value Iratión

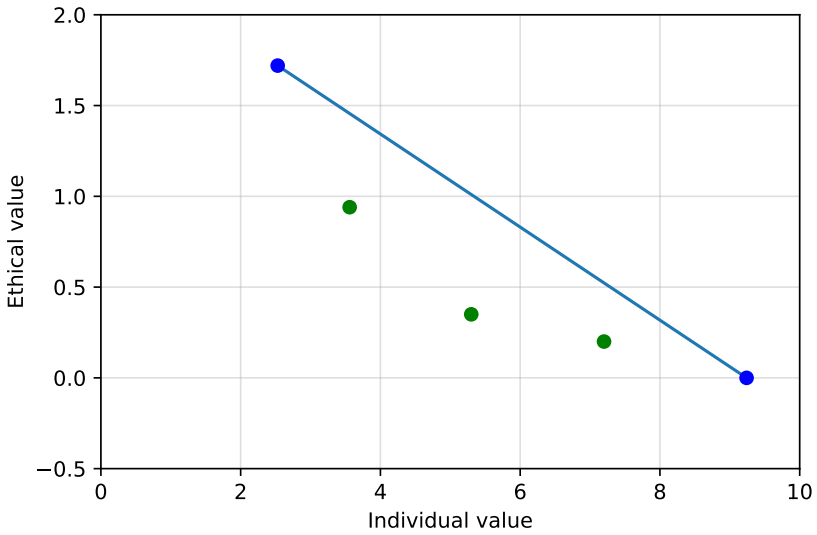
En l'arxiu hull.py podem trobar la funció per iterar la Convex Hull seguint l'algorisme 3. Iterem 20 vegades sobre totes les parelles estat-acció, per tal de convergir.

```
Q = hullSets()
all_states = generador_states()
for i in range(20):
    for state in all_states:
        for action in getLegalActions(0,state):
            succesors = transition(state, action)
            p = 1/len(succesors)
            q = None
            for ss in succesors:
                legal_actions = getLegalActions(0,ss)
                U = [ point for aa in legal_actions for point in Q[
                                                    (ss,aa)]]

                U = np.asarray(U)
                U = unique(U)
                if len(U) == 0:
                    sshull = np.array([p * reward(state, action)])
                else:
                    try:
                        sshull =  p * (reward(state, action) + 0.8
                                                              *
                                                              partialHull
                                                              (U))
                    except:
                        h = partialHull(U)


                if q is None:
                    q = sshull
                else:
                    q = sumHull(q,sshull)

            Q[(state,action)] = q
```

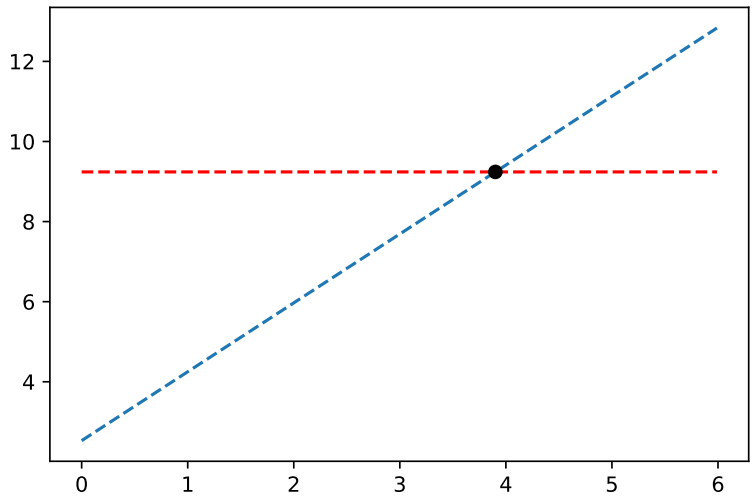
Per emmagatzemar els conjunts de vèrtex utilitzem la classe hullSets que podem trobar a l'arxiu utils.py, és una espècie de diccionari que per defecte retorna un array de numpy buit. La funció `unique` neteja el conjunt de punts repetits, ens quedem amb una testimoni de queda Q-valor. La funció getLegalActions ens retorna el conjunt A(s) donat un estat s. Per últim la funció generador_states genera totes les configuracions possibles d'estats en aquest entorn.

6.5 Resultats

Després d'executar l'algorisme Convex Hull Value Iratión sobre el problema, ens quedem amb els vèrtexs de les Convex Hulls asociades a Q-valors de l'estat inicial. Les unifiquem tots els punts per tal de crear el vector valor de l'estat inicial i calculem la Convex Hull parcial sobre aquest conjunt de punts obtenim el següent resultat:



Els vèrtexs de la convex hull parcial són els punts: (2.53,1.72) i (9.24,0) corresponents al primer i segon vector valor. Resolent la inequació proporcionada pel teorema 5.5 obtenim el següent resultat:



Podem garantir llavors que per a $w_e > 3,9011$ generem una inserció on l'agent només aprèn politípiques òptimes ètiques.

7 Conclusions

En aquest treball ens vam proposar conèixer la teoria necessària per arribar a comprendre els articles [8] i [9]. Aquesta tasca s'ha realitzat en els capítols 2,3,4 explicant des dels conceptes més bàsics per entendre totes les parts que apareixen, en l'aprenentatge per reforç, passant per la definició d'un procés de decisió de Markow. També hem explicat l'algorisme Q-aprenentatge, per tal que un agent aprengui sense necessitat de conèixer el model. En el capítol 4 hem introduït els problemes multiobjectiu i desenvolupat un algorisme que garanteix aprendre totes les possibles polítiques òptimes sobre la ponderació lineal dels diferents criteris.

Quan ja disposàvem de la base teòrica suficient, en el capítol 5 hem explicat el contingut dels articles. Introduint així els problemes ètics multiobjectiu i detallant un algorisme per crear una inserció ètica per tal de garantir que un agent que aprèn mitjançant Q-aprenentatge només pot aprendre polítiques ètiques.

En el capítol 6 hem posat en pràctica els coneixements assolits, adaptant el problema Common Game proposat per DeepMind per tal de crear un entorn on poder aplicar l'algorisme ètic d'inserció. Hem obtingut resultats i ja sabem com garantir que un agent que aprengui en aquest entorn aprengui polítiques ètiques.

Hem obtingut una conclusió que no buscàvem sobre l'entorn que hem creat: si fixem la política d'un agent com de forma egoista, que es comporta de forma no ètica, agafant pomes del common pool inclús quan no les necessita no donem la possibilitat a l'agent que aprèn a comportar-se de la mateixa manera, ja que no hi ha pomes en el pool fins que aquest en té suficients, que és l'únic que les aporta.

Ens hem adonat que la complexitat de l'algorisme ètic d'inserció és molt gran, per obtenir resultats amb un ordinador estàndard hem necessitat molt de temps d'execució. Això ens ha representat un impediment a l'hora de considerar taulells més grans. Que aquest algorisme necessiti un model fa que no puguem aprofitar avantatges com les del Q-aprenentatge on no ens fa falta. Per això recomanem utilitzar-lo en problemes on el model sigui fàcil de conèixer.

Concloem llavors que amb aquest algorisme s'obtenen uns resultats molt potents, ja que podem garantir un aprenentatge ètic.

Feina futura

Creiem que seria molt útil continuar estudiant sobre la matèria, ja que els dilemes ètics estan a l'ordre del dia en la intel·ligència artificial. Seria molt profitós treballar en un algorisme que garanteixi l'aprenentatge ètic sense la necessitat del coneixement previ del model, sense la necessitat de conèixer les probabilitats de transició.

Referències

[1] Documentation of scipy.spatial.convexhull. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.htmlscipy.spatial.ConvexHull. Scipy.

[2] Barrett, L., and Narayanan, S. Learning all optimal policies with multiple criteria. Proceedings of the 25th International Conference on Machine Learning (01 2008), 41–47.

[3] Cuervo, T. Openai gym implementation of the commons game. https://github.com/tiagoCuervo/CommonsGame. GitHub.

[4] Hardin, G. Tragèdia dels bens comuns. https://ca.wikipedia.org/wiki/Tragèdia_dels_comuns. Wikipedia.

[5] Perolat, J., Leibo, J., Zambaldi, V., Beattie, C., Tuyls, K., and Graepel, T. A multi-agent reinforcement learning model of common-pool resource appropriation. https://deepmind.com/research/publications/2019/multi-agent-reinforcement-learning-model-common-pool-resource-appropriation. DeepMind. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (pp. 464-473).

[6] Roijers, D. M., and Whiteson, S. *Multi-Objective Decision Making*. Morgan Claypool Puclishers, 2017.

[7] Rusell, S., and Norving, P. *Inteligencia Artificial Un Enfoque Moderno*. Pearson, 2003.

[8] Soto, M. R., Sanchez, M. L., and Aguilar, J. A. R. Guaranteeing the learning of ethical behaviour through multi-objective reinforcement learning.

[9] Soto, M. R., Sanchez, M. L., and Aguilar, J. A. R. A structural solution to sequential moral dilemmas.

[10] Sutton, R. S., and Barto, A. G. *Reinforment Learning An Introduction*. MIT Press, 1998.