

RRT alapú tanítóminta-gyűjtés alkalmazásai a megerősítéses tanulásban

Szerző:
Péter István



Budapesti Műszaki és Gazdaságtudományi Egyetem
Közlekedés- és Járműirányítási Tanszék
Közlekedésmérnöki és Járműmérnöki Kar

Konzulens:
Kővári Bálint

OTDK dolgozat

2024. december 26.

Absztrakt

A megerősítéses tanulásban a mintahatékonyság kulcsfontosságú olyan helyzetekben, amikor a környezet felfedezése időigényes, vagy más okból nehezített, illetve gyors prototípus-tervezés esetén, amikor a kisebb méretű, az állapotteret jobban lefedő tanítóminta-halmaz előnyösebb. Jelen dolgozatomban a Rapidly exploring Random Trees (RRT) módszer alkalmazásait vizsgálom a tanítóminta-gyűjtés során, ahol a fa építésében használt metrikus tér létrehozásához bizonyos előre tanított modellek Q-terét használom. A véletlenszerű felfedezéssel történő mintagyűjtéssel összehasonlítva, kimutatom az RRT-alapú mintagyűjtés előnyét a konvergencia sebességét tekintve, ezen összehasonlítást több ismert megerősítéses tanulás környezetben és különféle DQN modellek tanításán keresztül bemutatva. Ezen felül vizsgálom az RRT hozzáadása során lehetséges különféle variációk és hiperparaméterek hatását a tanítás pontosságára és a konvergencia sebességére.

Abstract

In Reinforcement Learning, sample efficiency is a key consideration when working with environments where exploration might be too time-consuming or otherwise prohibitive, or when doing fast prototyping, where a smaller, but more diverse dataset can span the state space more efficiently. In this work, I inspect the possibility of using a Rapidly exploring Random Trees (RRT) approach to sample gathering. For the state/observation representation, I explore the usage of a pre-trained (but not necessarily a fully competent) model's normalized output Q space. For the comparison with random exploration, I use well-known Reinforcement Learning environments and a family of DQN training methods as a benchmark to show the advantage of the RRT sample gathering in speed of convergence. I also discuss the different possible variations and hyperparameters in the inclusion of RRT in the sampling process, and their influence upon the overall performance and speed of convergence.

Tartalomjegyzék

1. Bevezető	4
1.1. A téma motivációja	4
1.2. Kapcsolódó eredmények	5
1.3. Kontribúció	5
2. Algoritmusok	7
2.1. Megerősítéses mélytanulás: DQN	7
2.2. Mintahatékony faépítés: RRT	8
2.3. Optimális tanítóminta-gyűjtés RRT algoritmussal	9
2.4. Környezetek leírása	11
2.4.1. Cartpole	11
2.4.2. Acrobot	12
2.4.3. Mountain Car	14
3. Kísérletek és eredmények	16
3.1. Kísérlet részletei és indokoltsága	16
3.2. Eredmények az egyes környezetekre	18
3.3. Az állapottér lefedése	19
3.4. Hiperparaméterek hatása	20
3.5. Konklúzió	22

1. fejezet

Bevezető

1.1. A téma motivációja

A mesterséges intelligencia területén, kimondottan pedig a megerősítéses tanulási (RL) problémáknál kulcsfontosságú a megfelelő felfedezési eljárás alkalmazása, ami által az ágens átfogó belső reprezentációt tud alkotni a környezet viszonyairól. Az új tudás beépítése és a meglévő tudás kiaknázása közti feszültség, az *exploration vs. exploitation* a megerősítéses tanulási módszerek legtöbbszörében valamilyen formában előjön, ilyenkor a nem minta-hatékony felfedezés nagy hátrányt jelent, mert az ágens erőforrást pazarol redundáns, kevés hozzáadott értékű információ gyűjtésére, ezalatt pedig viseli az elmaradt tudás-kiaknázásból fakadó jutalom-veszteség költségét. Számos népszerű RL algoritmus, köztük a jelen dolgozatban részletesebben tárgyalt *Deep Q-Network* (DQN) megközelítés a felfedezést a környezet állapotterében való véletlen bolyongással valósítja meg. Kezdetben, amikor az ágens semmilyen extra információval nem rendelkezik a környezettel kapcsolatban, ez segít elindulni, viszont ahogy a gyűjtött tudás bővül, megnyílik annak a lehetősége, hogy a felfedezés céltudatosan az eddig elhanyagolt területek feltárására, az ismeretlen állapottér-régiók minél hatékonyabb lefedésére koncentráljon, ezen célirányosság lehetősége azonban hiányzik a véletlen bolyongás módszeréből.

A *Rapidly-exploring Random Trees* (RRT) algoritmus megjelenése óta népszerű és széles körben alkalmazott, főleg az egyszerűsége és a módszer által biztosított optimális állapottér-lefedés miatt. A robotika területén különösen elterjedt, például a *motion planning* feladat tekintetében. Előnye, hogy gyorsan képes elérni távoli állapotokat, így alkalmas lehet a környezet átfogó felfedezésére. Jelen dolgozatban vizsgálom az algoritmus mintagyűjtésben való hasznosíthatóságát a DQN kontextusában, összehasonlítva a klasszikusan alkalmazott véletlenszerű mintagyűjtéssel, eredményekkel alátámasztva három népszerű RL környezetben. A vizsgálat során rámutatok, hogy megfelelően

alkalmazva, az RRT alapú mintagyűjtéssel jóval kevesebb adatból ugyanolyan, vagy jobb teljesítményű ágensek taníthatók az említett környezeteken, kihasználva az algoritmusban rejlő lehetőséget az információ-redundancia elkerülésére.

A téma aktuális és releváns, napjaink járműipari alkalmazásaiban egyre nagyobb teret nyer az intelligens adatfeldolgozás és irányítás, így a megerősítéses tanulás-alapú módszerek is, azonban a sokrétű szenzor-adatfolyamból érkező adatsokaságot kezelni és rendszerezni kell, amiben nagy segítséget tud nyújtani egy olyan módszer, ami által tömörebb reprezentációban eltárolható ugyanaz az információtartalom egy tanuló RL rendszer számára. A vizsgált környezetek és a faépítés jellege miatt a módszer alkalmas lehet szimulátoros környezetekbe való átültetésre, ahol így a kevesebb minta miatt kisebb számításigénnyel taníthatók az intelligens ágensek.

1.2. Kapcsolódó eredmények

Az RRT, illetve a trajektóriák hosszára optimalizált RRT* algoritmust [6] és [9] mutatja be. Az RRT megerősítéses tanulásban való alkalmazása számos publikációban felmerült, ezeket részben [1]-ben tekintik át: [5] a Planning for Policy Search (PPS) néven klasszikus irányítástechnikai módszerekkel, *Model Predictive Control* (MPC) alapú *local planner* segítségével valósít meg RRT minta-generálást, [3] a robotika területén vizsgál *motion planning* feladatokat, itt már neurális háló látja el a *local planner* szerepét, [4]-ban hosszú távú navigációs problémán vizsgálják az algoritmus alkalmazhatóságát, ezekben az esetekben azonban a faépítésnél használt távolságfüggvény többnyire a felhasználási területtől függően lett konstruálva.

A jelen dolgozatban vizsgálthoz hasonló módszert mutat be [1], itt sávtartási feladatokra alkalmazott DQN háló ([7]) teljesítményét vizsgálják különböző mintagyűjtési eljárások mellett. Az RRT fa építése során a legközelebbi csúcsok meghatározására alkalmazott távolságmetrika az állapothoz tartozó Q-értékek vektorainak euklideszi távolsága.

1.3. Kontribúció

A bemutatott összehasonlítás alapötlete [1]-ből származik, sávtartási probléma helyett három reprezentatív kiválasztott, népszerű megerősítéses tanulás környezetben vizsgálom az RRT alapú mintagyűjtés előnyét az egyszerű véletlen felfedezéssel szemben.

Emellett a jelen dolgozatban új megvilágításba helyeztem a minták halmaza fölött értelmezett távolságfüggvényt, amit az RRT fa építése során használ az algoritmus.

Ebben az újszerű megközelítésben az egyes állapotokhoz a DQN háló által becsült Q-vektorból már a *softmax* függvény segítségével valószínűségi eloszlást képezek, és az ezek közötti koszinusz-távolságot (*cosine similarity*) tekintem mérvadónak. Tehát míg a softmax-Q vektort a hivatkozott cikkben csak a faépítés *rollout* fázisában a következő akció kiválasztáshoz alkalmazzák, én beépítem a legközelebbi szomszédos csúcs keresésének szakaszába is az euklideszi távolság helyett.

Az egyes mintákhoz rendelt softmax-Q vektor az állapot *döntési profilját* jellemzi, amikor az ágens a hosszú távú jutalom maximalizálása szempontjából dönt a következő lépésről. A (kontrasztív tanulásban használt) koszinusz távolság ezen állapotrepresentációk viszonyát vizsgálja egy *kontrasztív* metrikus térben. A kapott eredmények visszaigazolják, hogy ez a megközelítés különböző környezetekben is képes jó teljesítményt nyújtani.

Ezen felül vizsgálom a faépítés hiperparamétereinek hatását a mintákon tanított ágens teljesítményére, illetve megkísérem vizuálisan és kvantitatívan jellemezni az RRT állapot-lefedését, összehasonlítva a véletlen bolyongás alapúval, bizonyítva az előbbi optimális mintahatékonyságát.

2. fejezet

Algoritmusok

2.1. Megerősítéses mélytanulás: DQN

A DQN (Deep Q-Network) módszer a Q-tanulás mély neurális hálókra szabott adaptációja, amit [7]-ben mutattak be. Az algoritmus célja egy adott markovi környezetben (*Markov Decision Process*, vagy MDP) az optimális Q -függvény becslése mély hálókkal, vagyis

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

amit az optimális π^* *policy* szolgáltat. A módszer a Q^* -függvénynek a Bellman-egyenlet alapján átalakított rekurzív alakját használja tanulásra:

$$Q^*(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

Az ágens a neurális háló becslését ($Q(s, a, \theta)$, ahol θ a háló paramétereit jelöli) használja arra, hogy minden állapotban a γ -val diszkontált várható teljes jövőbeli jutalom maximális értéke szerint válasszon beavatkozást (akciót).

A háló tanítása során adott (s, a, r, s') tanítómintára a háló s bemenetéhez az

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

értéket írjuk elő a becsült Q -vektor a -hoz tartozó kimenetére, az eltérést büntető *loss* függvény az átlagos négyzetes eltérés, vagyis

$$L(\theta) = \mathbb{E}_{s,a,r,s'} \left[(y - Q(s, a; \theta))^2 \right]$$

A [7] jelentős kontribúciója a korábbi, mély megerősítéses tanulással történő hasonló próbálkozások stabilitási problémáinak megoldása. Ehhez többek között az *ex-*

perience replay módszert alkalmazták, amivel korábbi állapot-átmenetekből véletlen mintavételezéssel választanak tanító *batch*-et, mert a tapasztalatok szerint az egy trajektórián belüli, egymásutáni, így erősen korrelált átmenetekkel való tanítás instabilitáshoz vezet. Hasonlóan javulást értek el külön *target network* használatával (a fenti képletekben θ^- jelöli, és a célváltozó meghatározásában használjuk), amit csak periodikusan frissítenek, ezzel szintén a háló konvergenciáját javítva a tanulás során, hiszen egy időre stabilizálja az elvárt célváltozók eloszlását.

A szerzők a módszer hatékonyságát 49 különböző, Atari 2600 szimulátoros környezeten kiértékelve támasztották alá. Az algoritmus 1-ben van összefoglalva.

1. Algorithm A DQN algoritmus sematikus leírása, [7] alapján.

Input: Q-network function $Q(s, a, \theta)$, MDP environment E

Output: θ model parameters

$\theta \leftarrow$ random weights

$\theta^- \leftarrow \theta$

Initialize replay memory D

for each episode **do**

 Initialize starting state s_0

for each step in episode **do**

 With probability ϵ , $a \leftarrow$ random action

 Otherwise $a \leftarrow \arg \max_a Q(s, a; \theta)$

 Execute action a , observe reward r and next state s' in E

$D \leftarrow D \cup (s, a, r, s')$

$(s, a, r, s') \leftarrow \text{RandomSample}(D)$ ▷ Minibatch mintavételezése

$y \leftarrow r + \gamma \max_{a'} Q(s', a'; \theta^-)$

 Perform a gradient descent step on $(y - Q(s, a; \theta))^2$

 Every C steps, update target network $\theta^- \leftarrow \theta$

Return θ

2.2. Mintahatékony faépítés: RRT

Az RRT (Rapidly-exploring Random Trees) egy [6]-ben bemutatott, állapotteret fa-struktúrában felfedező egyszerű algoritmus, amely a *Voronoi bias* nevű jelenséget kiaknázva alakítja ki a tér aszimptotikusan optimális lefedését. A felfedezés hatékonysága miatt a megjelenése óta gyakran használják útkeresésre különféle robotikai feladatoknál, főleg mozgástervezésre komplex dinamikájú rendszerekben. Az algoritmusnak létezik egy továbbfejlesztett, [9]-ban bemutatott RRT* nevű verziója is, amely az állapottér hatékony lefedése mellett a trajektóriák optimalitását is számításba veszi, törekedve az állapottér pontjai közt a legrövidebb utak megtalálására, de a jelen

dolgozatban az eredeti módszert adaptáljuk a megerősítéses tanulás területére.

Az eredeti algoritmus 2-ben található vázlatosan összefoglalva. A `Steer()` függvényt általában egy *planner* szolgáltatja, a keresési tér dinamikája ismeretében, ez a jelen dolgozat kontextusában a DQN háló által definiált Q-függvény, mint valószínűségi eloszlás.

2. Algorithm Az eredeti RRT algoritmus vázlata, [6]-ból átvéve.

Input: Initial state x_{init} , Sample distribution X , expected no. of samples K

Output: T exploration tree

Initialize tree T with root at x_{init}

while less than K samples in T **do**

$x_{\text{rand}} \leftarrow \text{RandomSample}(X)$

$x_{\text{near}} \leftarrow \text{NearestNeighbor}(T, x_{\text{rand}})$

 ▷ A legközelebbi, fában levő csúcs

$a_{\text{steer}} \leftarrow \text{Steer}(x_{\text{near}}, x_{\text{rand}})$

 ▷ A véletlen pont irányába mutató akció

$x_{\text{new}} \leftarrow \text{Step}(x_{\text{near}}, a_{\text{steer}})$

 ▷ Lépés a rendszer dinamikája szerint

$T \leftarrow T \cup (a_{\text{steer}}, x_{\text{new}})$

 ▷ Él és csúcs hozzáadása a fához

Return T

2.3. Optimális tanítóminta-gyűjtés RRT algoritmus-sal

A munkám során az RRT algoritmust adaptáltam a vizsgált környezetekre és DQN módszerrel tanított ágensekre. Az algoritmus fő ötlete [1]-ban van vázolva, amitől én például a Q-értékek leképezése (*softmax*) és a minták közti távolságfüggvény tekintetében (*cosine similarity*) térek el, illetve több környezetre kiterjesztem, hiperparaméter-vizsgálattal.

A mintákat tároló fát (általános esetben erdőt) olyan kezdeti állapottal (állapotokkal) inicializálom, amit a környezet által implementált `reset()` metódus ad vissza (ezek általában a kezdeti állapot kis környezetéből véletlenszerűen generált vektorok). Ezek lesznek az erdő fájnak gyökerei.

A tanítóminták gyűjtése során először az adott probléma állapotterében mintavételezek egy véletlen pontot (a jelen dolgozatban vizsgált környezetek mindegyike esetén ez egy vektor, amely komponensei egymástól függetlenek, és adott intervallum fölötti egyenletes eloszlásból származnak), és megkeresem az ehhez tartozó legközelebbi pontot a fában. A távolságot egy referencia DQN háló által definiált, állapotok fölött értelmezett, (*softmax* függvénnyel normalizált) Q-függvény kimenetei között számolt koszinusz-távolsággal (*cosine similarity*) értelmezem. A dolgozatom egyik fő állítása,

hogy a [1]-ban használt euklideszi távolságmérika mellett ez a távolságfüggvény is képes optimális teljesítményt produkálni az állapottér lefedése szempontjából.

A legközelebbi fa-csomópont megtalálása után a probléma környezetét az adott állapotba állítom, majd a *rollout* fázisban K (a jelen esetben alapértelmezetten 5) egymás utáni lépésben továbblépek olyan akciókkal, amiket mindig az aktuális állapothoz tartozó (referencia háló által szolgáltatott, softmax utáni) Q -vektor, mint valószínűségi eloszlás szerint mintavételezek. Ha a lépéssorozat során terminális állapotba kerül a környezet, korábban megszakad a *rollout*.

Az adaptált algoritmus, ami a [1] módosított változata, 3-ban látható vázlatosan összefoglalva.

3. Algorithm A jelen dolgozathoz adaptált RRT alapú mintagyűjtés.

Input: $Q(s)$ vector-valued function with softmax, MDP environment E , N no. of samples, K rollout steps

Output: T tree of samples

distance $\leftarrow$ cosine similarity

Initialize T with seed states from E

$\triangleright$ Az RRT erdő gyökérállapotai

while less than N samples in T **do**

$s_{\text{random}} \leftarrow \text{RandomState}(E)$

$s_{\text{closest}} \leftarrow \arg \max_s \text{distance}(Q(s), Q(s_{\text{random}}))$

$\triangleright$ Legközelebbi fa-csúcs

for K steps **do**

$\triangleright$ Rollout

 Set state s_{closest} in E

$q_{\text{closest}} \leftarrow Q(s_{\text{closest}})$

$a \leftarrow \text{RandomSample}(q_{\text{closest}})$

$\triangleright$ Akció választása

 Execute action a , observe reward r and next state s' in E

$T \leftarrow T \cup (s, a, r, s')$

$\triangleright$ Állapot-átmenet hozzáadása a fához

if episode is terminated **then**

break

$s_{\text{closest}} \leftarrow s'$

Return T

A konkrét implementáció során számos optimalizációval éltem, ugyanis bizonyos környezetek és mintaszámok esetén (pl. *Mountain Car* és az ahhoz szükséges 200000 minta gyűjtése) így is meglehetősen lassú az eljárás futásideje. Például, ha a *rollout* során egy ponton már érintett állapotba kerül az ágens, csak a még nem választott akciók közül "sorsoljuk" a következő lépést, illetve a már bejárt csúcsok (állapotok) esetén számon tartjuk a már választott akciók számát, így a telített állapotokat (a fában már minden akcióját kimerítettük) előre elkerüljük. Az explicit duplikáció-ellenőrzés nélkül a módszer gyakran gyűjtött "belterjes" mintákat és trajektóriákat, ami miatt szub-optimális állapottér-lefedést produkált az eljárás.

2.4. Környezetek leírása

A kísérleteim során olyan környezetekre koncentráltam, amelyek megfigyelési és állapottere folytonos, alacsony dimenziószámú vektorokkal leírható, a beavatkozások halmaza pedig diszkrét és véges. Az előbbi feltétel megtartja az állapottér általános jellegét, nem korlátozódva diszkrét állapotterekre és az azon értelmezhető dinamikára, utóbbi pedig a döntési folyamat fa-jellegét erősíti, ahol a fa csúcsainak véges gyerek-csúcsszáma megfeleltethető a döntések halmazának.

A fenti szempontok alapján a megerősítéses tanulás területén nagy népszerűségnek örvendő *Cartpole*, *Acrobot* és *Mountain Car* környezeteket választottam, amelyek implementációja megtalálható a *Gymnasium* szoftvercsomag *Classic Control* környezet-csoportjában.

2.4.1. Cartpole

A *Cartpole* környezetben (amit [2]-ben mutatnak be részletesebben) az ágens célja egy kocsira/csúszó talpra szerelt rúd minél tovább tartó egyensúlyozása (vizualizáció a 2.1 ábrán). A kocsí vízszintes síkon mozoghat balra vagy jobbra, a súrlódás elhanyagolható, míg a rúd a rendszer tehetetlenségétől függően mozog. Ezen felül a környezet kis intenzitású, véletlenszerű behatásokkal próbálja kilengetni a rudat, amit az ágensnek egy ellentétes hatású beavatkozással kell ellensúlyoznia.

A feladat aránylag könnyű, jól használható megerősítéses tanulási algoritmusok *benchmark* tesztelésére.

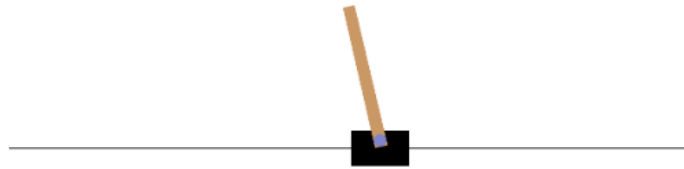
Az ágens megfigyelése a négy dimenziós

$$\mathbf{v} = \begin{bmatrix} x \\ x' \\ \theta \\ \theta' \end{bmatrix}$$

vektor, és kétféle beavatkozással élhet: a 0 adott erővel balra, az 1 pedig jobbra tolja a kocsit (a rudat és a szögét tehát csak indirekt módon, a csatlakozáson és a rendszer tehetetlenségén keresztül befolyásolja).

Az epizód során az ágens +1 jutalmat kap mindaddig, amíg a rudat állva és középben tudja tartani, pontosabban amíg a rúd szöge és a kocsí pozíciója a 2.1 táblázatban felsorolt szélsőértékek között marad. A rúd túlzott kilengése hatására az epizódnak vége szakad, de negatív (vagy +1-től különböző) jutalom nem generálódik, így az ágens az időtartam maximalizálására törekszik a jutalmak összegén keresztül.

Az epizód 500 lépés után mindenképp véget ér, így ez az ideális jutalom egy



2.1. ábra. A *Cartpole* környezet vizualizációja. A cél a sík felületen csúszó talpra szerelt fordított inga függőlegesen tartása a talp mozgatásával.

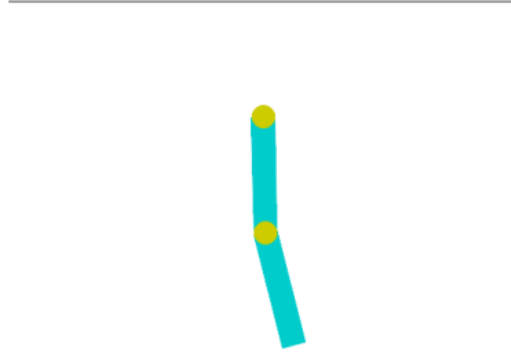
	x	x'	θ	θ'
min	-2.4	$-\infty$	-0.2095	$-\infty$
max	2.4	∞	0.2095	∞

2.1. táblázat. A *Cartpole* állapottere: a fordított inga talppontja (x) és szöge (θ), illetve ezek sebességei. A nem-terminális állapotok a táblázatban felsorolt alsó/felső korlátok közötti tartományból kerülnek ki.

optimálisan eljáró ágens esetén.

2.4.2. Acrobot

Az *Acrobot* környezetben az ágens egy felfüggesztésről lógó csuklós ingarendszert irányít, a cél pedig az inga végpontjának adott magasság fölé emelése (ld. 2.2 ábra, a környezet leírása megtalálható [10]-ben is). A probléma diszkrét változatában az ágens a csuklópontban, az inga két szakaszának találkozásánál tud hatni, fix mennyiségű pozitív vagy negatív, esetleg 0 forgatónyomatékot gyakorolva rá, erre a három beavatkozásra képes.



2.2. ábra. Az *Acrobot* környezet vizualizációja. A cél a szürke vonal fölé juttatni a csuklós inga végpontját, az ágens csak a csuklópontnál (középen) hathat a rendszerre.

Az ágens megfigyelési tere 6 dimenziós, a

$$\mathbf{v} = \begin{bmatrix} \cos(\theta_1) \\ \sin(\theta_1) \\ \cos(\theta_2) \\ \sin(\theta_2) \\ \theta'_1 \\ \theta'_2 \end{bmatrix}$$

vektort "látja", ahol θ_1 és θ_2 a felfüggesztési pontban és a csuklópontban levő szögek, θ'_1 és θ'_2 pedig a szögsebességeket jelöli. A végpont magassága

$$h = -\cos(\theta_1) - \cos(\theta_1 + \theta_2),$$

ami akkor éri el a cél-állapotot, ha nagyobb 1-nél. A környezet implementációjában a szögsebességeknek alsó- és felső korlátot szabnak, ezek a 2.2 táblázatban vannak összefoglalva.

A probléma nehézségét egyrészt a rendszer dinamikájának komplexitása, másrészt a ritka jutalom adja. Az *Acrobot* környezetben ritka a nemnegatív visszacsatolás, ugyanis a célállapot elérése előtt az ágens csak -1 jutalmat kap, a cél teljesítésekor pedig 0 jutalommal leáll az epizód. Emiatt sok felfedezést igényel akár egyetlen ilyen

	θ_1	θ_2	θ'_1	θ'_2
min	$-\pi$	$-\pi$	-4π	-9π
max	π	π	4π	9π

2.2. táblázat. Az *Acrobot* környezet belső állapottere: θ_1 , θ_2 csuklósögek, és ezek θ'_1 , θ'_2 szögsebességei. Az ágens nem figyeli meg közvetlenül a két szöget, csak azok $\sin$ és $\cos$ szögfüggvényeit, így a megfigyelési vektor 6 dimenziós.

jutalmat adó tapasztalat "beszerzése" is.

A *Gymnasium* legfrissebb implementációjában a környezet 500 lépés után akkor is leáll, ha nem lett elérve a cél, ez viszont felülírható konfigurációs paraméterekkel, az én kísérleteimben 1000 lépésig mehet el az ágens a környezetben siker nélkül, ezzel nagyobb esélyt adva a véletlenszerű mintagyűjtésnél egy szerencsés konfiguráció epizódon belüli megtalálására.

2.4.3. Mountain Car

A *Mountain Car* problémát [8]-ben vezetik be mint nemlineáris, alacsony dimenziószáma ellenére kihívást jelentő MDP környezet. Az implementációk közül én a folytonos állapotterű, diszkrét beavatkozási halmazzal rendelkező verziót választottam. A környezetben modellezett autó egy dombos terepen helyezkedik el (ld. 2.3 ábra), az ágens csak az x -tengelyre vetített pozícióját (x) és a sebességét (v) ismeri minden pillanatban.

A rendszer állapota időlépésenként, az alábbi dinamika szerint frissül:

$$v_{t+1} = v_t + (a - 1) \cdot f - \cos(3 \cdot x_t)g$$

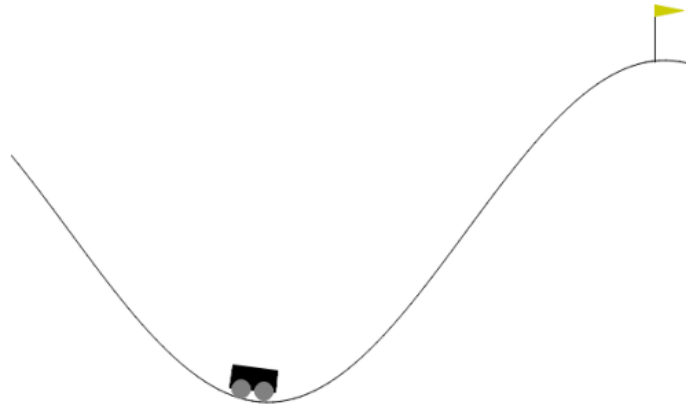
$$x_{t+1} = x_t + v_{t+1}$$

Ahol $a \in \{0, 1, 2\}$ a beavatkozást jelöli (0 és 2 a balra- és jobbra gyorsulás, 1 esetén nem gyorsul), $f = 0.001$ és $g = 0.0025$ a kifejtett erő, illetve a gravitációs gyorsulás. Az x és v értékek a 2.3 táblázatban feltüntetett korlátok között lehetnek.

	x	v
min	-1.2	-0.07
max	0.6	0.07

2.3. táblázat. A *Mountain Car* probléma megfigyelt állapottere, illetve az egyes változók alsó- és felső korlátai.

A környezet által támasztott kihívás abban áll, hogy mindig csak -1 jutalmat ad, az *Acrobot*-tal ellentétben még a célt elérve sem kap az ágens pozitív visszacsatolást,



2.3. ábra. A *Mountain Car* környezet vizualizációja. A cél a zászlóval jelölt pontra feljuttatni az autót, amit csak úgy lehet megvalósítani, ha az autó lendületet gyűjt.

csupán véget ér az epizód, emiatt pedig a cél a büntetett időtartam minimalizálása, az $x = 0.5$ pozíción található dombcsúcs minél gyorsabban való elérésével. Az optimális policy követésével az autó előbb többször oda-vissza kell gyorsuljon a mögötte, illetve előtte levő domb között, folyamatosan gyűjtve a lendületet, a helyzeti és mozgási energiát, amivel az előtte levő csúcsot el tudja érni.

3. fejezet

Kísérletek és eredmények

3.1. Kísérlet részletei és indokoltsága

A munkám során az RRT alapú felfedezés és mintakiválasztás előnyeit vizsgáltam, így ehhez olyan kísérleteket folytattam, ahol egyéb befolyásoló tényezők kiszűrésével tudtam összehasonlítani a módszer hatásait a megerősítéses tanulás konvergenciájára.

A klasszikus DQN algoritmust követve, a mintagyűjtés a tanulással párhuzamban zajlik, az *exploration vs. exploitation* egyensúlyt az ϵ paraméter befolyásolja, viszont akár a háló "döntése" szerint, akár véletlenszerűen léptünk, az (s, a, s', r) átmenet eltárolódik egy fix méretű FIFO (*first in, first out*) tárban, ahonnan a következő *batch*-et mintavételezzük. Ez azt eredményezi, hogy tanulás során a tapasztalatokat gyűjtő buffer eloszlása, részben pont az épp tanuló háló hatására, változik. Ez az összehasonlítást annyiban nehezíti, hogy a vizsgált kimenetel (a tanulás konvergenciája) részben autoregresszív módon függ önmagától is (a változó tanítóhalmazon keresztül), nem csak az általam összehasonlítani kívánt módszer megválasztásától. Épp ezért módosítottam az algoritmust, időben szétválasztva az adatgyűjtés és tanulás fázisait, így az előbbi nem befolyásolja az utóbbi.

A DQN-ben eredetileg epizódonként, $\epsilon(t)$ valószínűséggel (ahol t az aktuális epizód) alkalmazott véletlenszerű felfedezés az én esetemben már teljes egészében a tanítás megkezdése előtt lefut, ahogy értelemszerűen az RRT fa felépítése is, így gyűjt össze mindkét módszer fix számú mintát, a folytatás pedig mindkét esetben ugyanaz: azonos architektúrájú, véletlenszerűen inicializált DQN hálót tanítunk azonos számú lépésen keresztül. Egy tanító lépés a háló egyetlen, tanító *batch* alapján történő frissítésének felel meg (részben a már említett módosítás miatt használom a *lépés* kifejezést az *epizód* helyett). Ezután validációs lépés következik, ahol az aktuális DQN háló adott környezetben egy epizódot játszik, és az így elért össz-jutalmat rögzítjük. A tanítás végén a háló legjobb validációs jutalmat elérő állapotát vesszük figyelembe és mentjük

el (*checkpointing*).

A környezetek tekintetében a következő változtatást eszközöltem: mivel az RRT alapvetően fa-, de általánosabban véve erdőstruktúrát feltételez, ahol a gyökerek száma befolyásolja az adatstruktúra topológiáját (például: trajektóriák átlagos hossza N db. tanítómintából alkotott erdőben), ezért a kísérlet elején beállítottam az erdő gyökereinek számát, legyen ez K (alapértelmezetten 1), a tanítókönyezetben pedig az alapvetően sztochasztikus `reset()` műveletet úgy módosítottam, hogy K db. előre generált (de véletlenszerű) állapot valamelyikét állítsa be minden hívásra (véletlen, egyenletes eloszlás szerint). Ez tehát azt akadályozza meg, hogy a felfedezés/adatgyűjtés során terminális állapotokba érve, a `reset()` hívás hatására egy teljesen új gyökérrel/fa-kezdeménnyel bővüljön az erdő, ami hosszú távon az átlagos fa-méret (és így az átlagos trajektória-hossz) csökkenésével járhat, így rontva a tanulás hatékonyságát. A konzisztencia miatt a véletlenszerű felfedezéssel adatgyűjtésnél is pontosan ugyanígy inicializáljuk a környezetet, ugyanazon K db. gyökérállapottal. Fontos kiemelni, hogy ezt a "cinkelt" `reset()` függvényt csak az adatgyűjtés során alkalmazom, tanítás és validáció alatt hagyományos módon működik, így elkerülve azt, hogy (bizonyos értelemben) "tanítóhalmazon validáljunk".

Egy másik, környezetekre vonatkozó módosítást az indokol, hogy az adatgyűjtés és tanítás éles szétválasztásával az ágens nehezebben tanulja meg a problémát, ez főleg a ritka jutalmat adó környezetek, mint az *Acrobot* és a *Mountain Car* esetén jelentkezik. Mindkét esetben az érdemi jutalmat adó állapotokra nehéz rátalálni tisztán véletlen kereséssel, ennek hiányában viszont az ágens semmilyen, optimális *policy*-ra vonatkozó útbaigazítást sem kap. Ezt a nehezítést *reward shaping* alkalmazásával ellensúlyozom. *Acrobot* környezetben a hagyományos jutalomhoz hozzáadom a kar magasságát 0.01-gyel súlyozva, *Mountain Car* környezetben a helyzeti és mozgási energia összegét 10-es súllyal (ez azonban a *Gymnasium*-os implementációban 0.0025-re definiált gravitációs állandó és a hasonlóan alacsony maximális sebesség miatt maximum 0.05 nagyságrendű érték, az egyébkénti -1 jutalomhoz adódik hozzá), míg a *Cart Pole* környezet jutalmán, a probléma egyszerűsége miatt, nem változtatok. Itt fontos kiemelni, hogy a korábban említett validációs jutalmat is a *reward shaping* szerint számolom, viszont a kimentett legjobb modellekre a hagyományos jutalmak mellett is végzek validációt, így alátámasztva azt, hogy a "könnyített" problémán tanított ágens az eredeti jutalom szerint is betanul, illetve az RRT alapú adatgyűjtés előnye a módosított és a módosítatlan jutalom viszonylatában is megfigyelhető.

A fenti változtatások tehát azt alapozzák meg, hogy az összehasonlítás a lehető leginkább izoláltan vizsgálja az RRT-alapú mintagyűjtés és a véletlenszerű mintavételezés közti eltérést, egyenlő feltételeket teremtve a két módszer kontextusát tekintve.

3.2. Eredmények az egyes környezetekre

Mindegyik környezetre lefuttattam egy *referencia* tanítást, ami a véletlenszerű felfedezéssel gyűjtött mintákból, általában nagy mintaszám mellett tanulja meg a problémát. A problémák jellegének és nehézségének különbözősége miatt értelemszerűen az egyes környezetekre más-más hiperparaméter-kombinációk bizonyultak megfelelőnek egy elegendően jó tanításhoz. A neurális háló minden esetben két rétegű, rétegenként 256 rejtett neuronnal, *ReLU* aktivációs függvényvel, *Adam* optimalizálógöggvényvel tanítva. A tanítások konfigurációi (mintaszám, batch méret, bátorsági tényező) és a legjobb validációs eredménye a 3.1 táblázatban láthatók, továbbá az is, hogy a probléma nehézségével arányosan kell nagyobb mintaszám, lépésszám, illetve batch méret az adekvát eredmény eléréséhez.

Környezet	Minta#	Batch méret	lr.	Lépés#	R (max.)
Cart Pole	5 000	1 000	0.001	500	370.00
Acrobot	100 000	10 000	0.001	1 000	−78.95
Mountain Car	200 000	50 000	0.005	2 000	−116.64

3.1. táblázat. A referencia tanítások konfigurációja. A használt háló minden esetben 2 rejtett réteget tartalmaz, egyenként 256 neuronnal. Az lr. (*learning rate*) a bátorsági tényezőt, míg az R a legjobb háló-állapot (*checkpoint*) validációs jutalmát jelöli a *reward shaping* utáni jutalomfüggvényel számolva.

A továbbiakban a referencia háló legjobb validációs jutalomhoz tartozó állapotát mentettem el, és használtam az RRT-alapú mintagyűjtésben. A háló által megvalósított megfigyelés $\rightarrow$ Q-érték leképezés alapján az egyes minták távolságát a *softmax* függvényel normalizált Q-értékek koszinusz-távolságával (*cosine similarity*) határozta meg, ezzel a távolságfüggvényel futtatva a faépítést.

Az RRT módszerrel a referencia tanításnál jóval kevesebb (10 – 25%) mintát gyűjtöttem. Ezekkel a mintákkal tanítottam azonos architektúrájú (és a tanítás egyéb paramétereiben is megegyező) DQN hálókat, az eredmények a 3.2 táblázatban találhatók. A korábbi tanításokhoz hasonlóan itt is a legjobb validációs jutalomfüggvény értéke szerepel. Ezen túl feltűntettem a referenciához képest számolt különbséget a jutalomban (a negatív jutalmakat adó *Acrobot* és *Mountain Car* esetén azt vizsgáltam, a referenciához viszonyítva abszolút értékben mennyit csökkent, vagyis javult az elért eredmény).

Ezekből következik, hogy ezzel a módszerrel tanított háló képes kevesebb mintából hasonló, vagy jobb teljesítményt nyújtani a referenciánál. Továbbá a 3.3 táblázatban az is látható, hogy a referencia milyen eredményt érne el, ha ugyanannyi mintával futtatnánk, mint az RRT alapú tanításokat.

Környezet	Minta#	R (max.)	Különbség (%)
Cart Pole	1 000 (20%)	437.00	+18.10 %
Acrobot	10 000 (10%)	-62.83	+20.41 %
Mountain Car	50 000 (25%)	-88.86	+23.81 %

3.2. táblázat. Az RRT alapú felfedezéssel gyűjtött adatokon, a referencia tanítással azonos hálóval és tanítással (architektúra, learning rate, lépésszám, batch méret stb.) elért eredmények (legjobb validációs jutalom) és a referenciához viszonyított javulás.

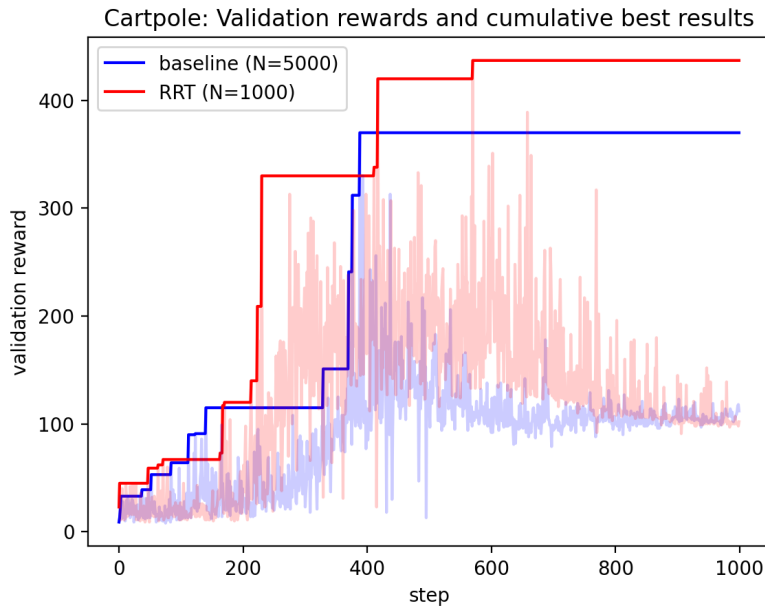
Környezet	Minta#	R (rand.)	R (RRT)	Különbség (%)
Cart Pole	1 000	266.00	437.00	+64.29%
Acrobot	10 000	-90.04	-62.83	+30.22%
Mountain Car	50 000	-130.09	-88.86	+31.69%

3.3. táblázat. Referencia tanítások eredményei, ha véletlen felfedezéssel (*rand.*) ugyanannyi mintát gyűjtenénk, mint az RRT alapú tanításoknál.

A tanulás során a konvergencia meglehetősen instabil volt mindkét módszer esetén. Ezt én főleg a DQN tanítás azon módosításának tulajdonítottam, hogy végig fix mintahalmazzal dolgoztunk, szemben a klasszikus módszer folyamatosan frissülő buffer-tartalmával, ami a konvergencia stabilitását segíti elő az ϵ *decay*-jel együtt alkalmazva, ugyanis ezáltal a tanulás során egyre inkább a háló aktuális állapota alapján gyűjtött minták kerülnek előtérbe (*exploitation*) a korábbi, túlsúlyban véletlen felfedezéssel gyűjtött mintákkal szemben. Az instabil konvergenciára a *Cartpole* probléma esetén mutat példát a 3.1 ábra, ahol egyszerre szerepel a lépésenkénti validációs pontosság, illetve az addig elért legjobb validációs pontosság grafikonja. Míg az utóbbi elvárható módon növekszik, az előbbi láthatóan mindkét tanításnál be-bezuhan.

3.3. Az állapotter lefedése

Az RRT algoritmus működése miatt megalapozott azt feltételezni, hogy az RRT-vel gyűjtött mintákkal azért tanul jobban a háló, mert ezek a minták hatékonyabban fedik le az állapotteret. A módszer, bár valójában a mintákat a háló Q-terébe leképezve dolgozza fel (így az RRT megkötései és garanciái is itt érvényesülnek), a problémák eredeti állapotterébe visszaképezve is képes optimális mértékben (vagy kevesebb mintával, de elegendő mértékben) lefedni a lényeges régiókat. A "kevesebb néha több" elv érvényesülésére a vizsgált három környezet közül a legjobb példát a *Mountain Car* esetén látjuk az 3.2 ábrán. Itt, bár az RRT fával jóval kevesebb (25%) mintát gyűjtöttünk, de ezek a minták jóval szélesebben lefedik az irányított autó fázisterét (a megfigyelt helyzet és sebesség), mégpedig ennél a problémánál a minél szélesebb felfedezés garantálja az optimum megtalálását (előbb képes kell legyen hátrafele felhajtani a mögötte



3.1. ábra. Lépésenkénti aktuális, illetve addigi legjobb validációs jutalom változása a *Cartpole* problémára a referenciaháló, illetve az RRT-adatokkal tanított háló tanítása során.

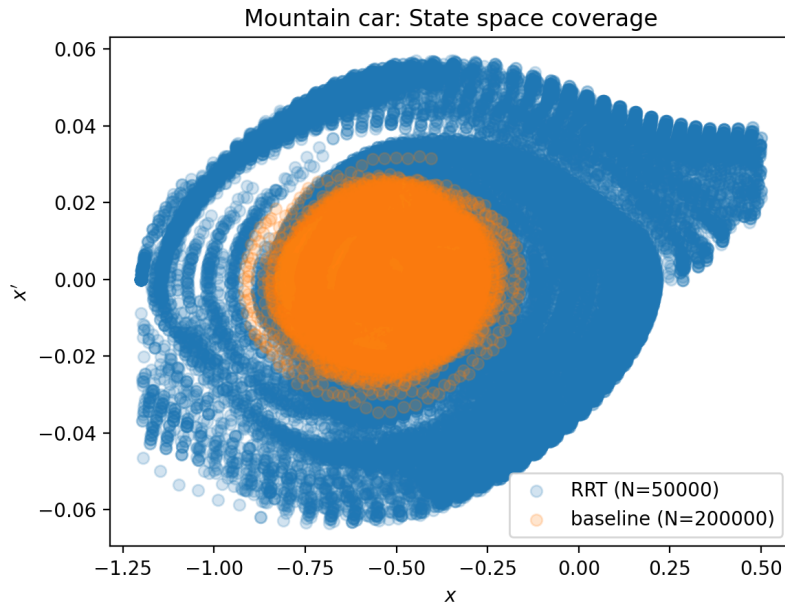
levő dombra, hogy kellő lendületet szerezzen a célul kitűzött, előtte levő dombcsúcs eléréséhez).

Hasonló összehasonlítás az *Acrobot* környezet esetén már kevésbé látványos (3.3, itt már az azonos mintaszám melletti mintagyűjtést hasonlítom össze), de az ágens eredményei ettől függetlenül jobb tanítóminta-eloszlásról árulkodnak.

Szintén az *Acrobot* környezet sajátossága, hogy a három vizsgált probléma közül az egyetlen, ahol nem csak egyféle jutalmat kapunk, így ebben az esetben vizsgálhatjuk a "nyerő" (0 jutalom) minták arányát a "büntetett" (-1 jutalom) mintákhoz képest. Ebben a tekintetben is az RRT alapú mintagyűjtés a sikeresebb, ugyanis míg a referencia adathalmazban 100 000 mintából 22-nél 0 jutalmat kapott az ágens, a tízszer kisebb RRT mintahalmaznál ez a szám 20, tehát az optimális Q -értékek tanulása szempontjából hasznos minták aránya nagyjából tízszeres.

3.4. Hiperparaméterek hatása

Az RRT faépítésben a *rollout* paraméterrel azt szabályozzuk, hogy egy iterációban az aktuálisan kiválasztott véletlenszerű pont irányába hány lépést tegyen a felfedezés. Nagyobb rollout paraméterrel inkább mélységi keresés irányába toljuk el az eljárást, míg alacsonyan tartva szerteágazóbb fát, de átlagosan rövidebb trajektóriákat kapunk. A



3.2. ábra. Az állapottér-lefedés összehasonlítása a *Mountain Car* probléma esetén. Az RRT módszerrel, negyedannyi mintából látványosan nagyobb területet fogunk át az állapottérből.

magas rollout értékkel ellensúlyozhatjuk azt, ha az RRT fa helyett RRT erdőt építünk, több gyökér-állapotból kiindulva, és az így egy fára eső kevesebb minta mellett megszeretnénk tartani a trajektóriák átlagos hosszát.

A korábban bemutatott RRT-alapú tanításoknál a rollout lépésszám értéke 5-re volt állítva, a paraméter változtatásának hatása az RRT tanítás eredményére a 3.4 táblázatban látható. Ez alapján elmondható, hogy az 1 általában túl alacsony lépésszám az optimális teljesítményhez, 5-re állítva többnyire jó eredményeket kapunk, a paraméter további növelésének hatása pedig problémafüggő (*Acrobot* esetén a lépésszám 40-re emelése sem ront a teljesítményen).

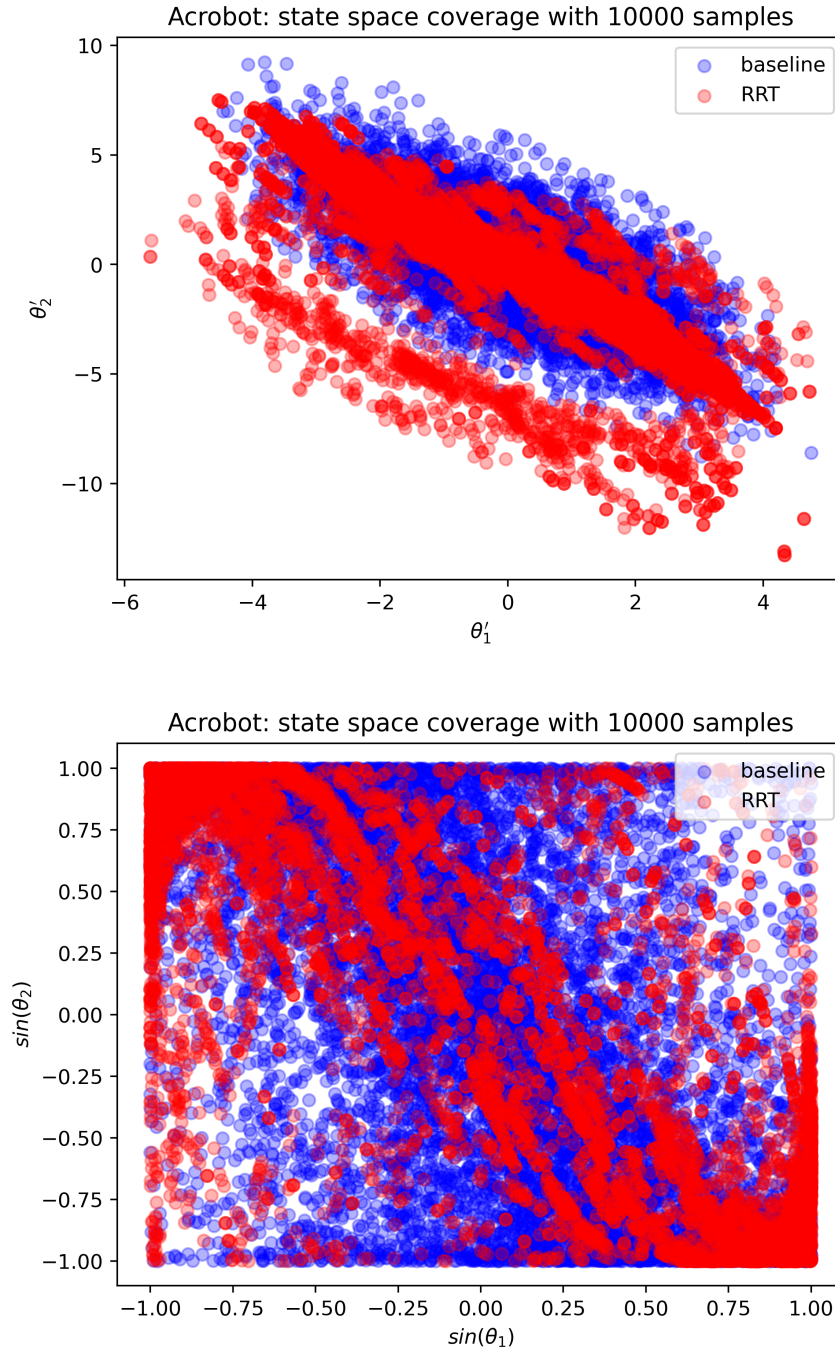
Környezet	Ref.	Rollout				
		$n = 1$	$n = 5$	$n = 10$	$n = 20$	$n = 40$
Cart Pole	370.00	388.00	437.00	373.00	421.00	74.00
Acrobot	-78.95	-107.28	-62.83	-73.00	-62.84	-61.84
Mountain Car	-116.64	-198.85	-88.86	-88.86	-149.62	-109.38

3.4. táblázat. A *rollout* lépésszám változtatásának hatása az RRT-alapú tanításon elért maximális jutalomra (összehasonlítva a referencia-tanítás jutalmával).

3.5. Konklúzió

Az eredményekből világosan látszik, hogy az RRT alapú mintagyűjtés jelentősen elősegíti a problémák állapotterének minta-hatékony lefedését.

Abból kiindulva, hogy az RRT fában gyűjtött minták alapján, egyébként azonosan kialakított feltételek mellett tanítható ugyanannyi lépésből, kevesebb mintából egy jobb háló, felveti annak a lehetőségét, hogy a jelen dolgozatban bemutatott módszert a klasszikus DQN tanításba is be lehessen építeni (tehát nem azon "mesterséges" körülmények között, ahogy a jelen dolgozatban például ki lett iktatva az új minták bufferbe helyezése a tanítás során). A bővítés tekinthető úgy, mint a felfedezés (*exploration*) fázis egy javított változata, de akár mint az *exploration* és *exploitation* közti átmenet (hiszen kihasználjuk a háló egy, idővel egyre jobb változatát adatgyűjtésre), amennyiben N epizódonként frissítve egy referencia hálót (például az előző N epizód legjobb validációs jutalmát produkáló háló), annak segítségével gyűjtünk további mintákat, akár az egyébkénti, ϵ valószínűséggel végrehajtott felfedező lépésként, vagy az mellett. Az eddigiekben bemutatottakból kiindulva megalapozottan sejthetjük, hogy egy ilyen kiegészítés hatására egy idő után az egyes újonnan bekerülő minták nagyobb hozzáadott értéket képviselnének, és gyorsabb konvergenciát biztosítanának. Természetesen számos konkrét módszertani, illetve stabilitási kérdés is felmerül, ami alapos vizsgálatot indokol.



3.3. ábra. Az *Acrobot* probléma állapotterének lefedése. Az ábrán késsel a véletlenszerű felfedezéssel, pirossal az RRT fa-alapú felfedezéssel gyűjtött minták találhatók.

Ábrák jegyzéke

2.1.	A <i>Cartpole</i> környezet vizualizációja. A cél a sík felületen csúszó talpra szerelt fordított inga függőlegesen tartása a talp mozgatásával.	12
2.2.	Az <i>Acrobot</i> környezet vizualizációja. A cél a szürke vonal fölé juttatni a csuklós inga végpontját, az ágens csak a csuklópontnál (középen) hathat a rendszerre.	13
2.3.	A <i>Mountain Car</i> környezet vizualizációja. A cél a zászlóval jelölt pont-ra feljuttatni az autót, amit csak úgy lehet megvalósítani, ha az autó lendületet gyűjt.	15
3.1.	Lépésenkénti aktuális, illetve addigi legjobb validációs jutalom változása a <i>Cartpole</i> problémára a referenciaháló, illetve az RRT-adatokkal tanított háló tanítása során.	20
3.2.	Az állapottér-lefedés összehasonlítása a <i>Mountain Car</i> probléma esetén. Az RRT módszerrel, negyedannyi mintából látványosan nagyobb területet fogunk át az állapottérből.	21
3.3.	Az <i>Acrobot</i> probléma állapotterének lefedése. Az ábrán késsel a véletlenszerű felfedezéssel, pirossal az RRT fa-alapú felfedezéssel gyűjtött minták találhatók.	23

Táblázatok jegyzéke

2.1.	A <i>Cartpole</i> állapottere: a fordított inga talppontja (x) és szöge (θ), illetve ezek sebességei. A nem-terminális állapotok a táblázatban felsorolt alsó/felső korlátok közötti tartományból kerülnek ki.	12
2.2.	Az <i>Acrobot</i> környezet belső állapottere: θ_1, θ_2 csuklósögek, és ezek θ'_1, θ'_2 szögsebességei. Az ágens nem figyeli meg közvetlenül a két szöget, csak azok $\sin$ és $\cos$ szögfüggvényeit, így a megfigyelési vektor 6 dimenziós. .	14
2.3.	A <i>Mountain Car</i> probléma megfigyelt állapottere, illetve az egyes változók alsó- és felső korlátai.	14
3.1.	A referencia tanítások konfigurációja. A használt háló minden esetben 2 rejtett réteget tartalmaz, egyenként 256 neuronnal. Az lr. (<i>learning rate</i>) a bátorsági tényezőt, míg az R a legjobb háló-állapot (<i>checkpoint</i>) validációs jutalmát jelöli a <i>reward shaping</i> utáni jutalomfüggvényel számolva.	18
3.2.	Az RRT alapú felfedezéssel gyűjtött adatokon, a referencia tanítással azonos hálóval és tanítással (architektúra, learning rate, lépésszám, batch méret stb.) elért eredmények (legjobb validációs jutalom) és a referenciához viszonyított javulás.	19
3.3.	Referencia tanítások eredményei, ha véletlen felfedezéssel (<i>rand.</i>) ugyanannyi mintát gyűjtenénk, mint az RRT alapú tanításoknál.	19
3.4.	A <i>rollout</i> lépésszám változtatásának hatása az RRT-alapú tanításon elért maximális jutalomra (összehasonlítva a referencia-tanítás jutalmával). .	21

Irodalomjegyzék

- [1] Kovari Balint, Angyal Balint Gergo, and Becsi Tamas. Deep reinforcement learning combined with rrt for trajectory tracking of autonomous vehicles. *Transportation Research Procedia*, 78:246–253, 2024. 25th Euro Working Group on Transportation Meeting.
- [2] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.
- [3] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia, and Aleksandra Faust. Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies, 2019.
- [4] Aleksandra Faust, Oscar Ramirez, Marek Fiser, Kenneth Oslund, Anthony Francis, James Davidson, and Lydia Tapia. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning, 2018.
- [5] Jakob Hollenstein, Erwan Renaudo, Matteo Saveriano, and Justus Piater. Improving the exploration of deep reinforcement learning in continuous domains using planning for policy search, 10 2020.
- [6] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.
- [7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmashan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.
- [8] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

- [9] Jauwairia Nasir, Fahad Islam, and Yasar Ayaz. Adaptive rapidly-exploring-random-tree-star (rrt*) -smart: Algorithm characteristics and behavior analysis in complex environments. *Asia-Pacific Journal of Information Technology and Multimedia*, 02:39–51, 12 2013.
- [10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.