UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



REINFORCEMENT LEARNING - ALGORITMY

DIPLOMOVÁ PRÁCA

2021 Bc. Jana SKOKANOVÁ

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

REINFORCEMENT LEARNING - ALGORITMY

DIPLOMOVÁ PRÁCA

Študijný program:	Ekonomicko-finančná matematika a modelovanie
Študijný odbor:	1113 Matematika
Školiace pracovisko:	Katedra aplikovanej matematiky a štatistiky
Vedúci práce:	RNDr. Zuzana Chladná, Dr.

Bratislava 2021 **Bc. Jana SKOKANOVÁ**





Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jana Skokanová
Študijný program: ekonomicko-finančná matematika a modelovanie (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Reinforcement Learning - algoritmy
Reinforcement Learning Algorithms

Anotácia: Štandardné numerické techniky diskrétnej dynamickej optimalizácie zlyhávajú pri väčšom počte stavových a riadiacich premenných, ako aj v prípade prítomnosti náhodných premenných. Metodika Reinforcement learning (RL), niekedy nazývaná aj približné dynamické programovanie, sa snaží ponúknuť alternatívny prístup k riešeniu týchto problémov.

Vedúci: RNDr. Zuzana Chladná, Dr.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: prof. RNDr. Marek Fila, DrSc.

Dátum zadania: 17.12.2019

Dátum schválenia: 21.12.2019 prof. RNDr. Daniel Ševčovič, DrSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie Moja vďaka patrí vedúcej tejto diplomovej práce, RNDr. Zuzane Chladnej, Dr., za férový prístup, usmerňovanie pri písaní a vecné pripomienky. Touto cestou sa chcem zároveň poďakovať svojej rodine a priateľom za ich podporu. Špeciálna vďaka patrí môjmu otcovi Ing. Jozefovi Skokanovi a bratovi Mgr. Jozefovi Skokanovi za pomoc pri korektúre textovej časti práce.

Abstrakt v štátnom jazyku

SKOKANOVÁ, Jana: Reinforcement Learning - algoritmy [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: RNDr. Zuzana Chladná, Dr., Bratislava, 2021, 130 s.

Reiforcement learning (RL) zaraďujeme medzi metódy strojového učenia. Algoritmy reinforcement learningu možno použiť na riešenie komplexných úloh stochastického dynamického programovania. Táto práca poskytuje prehľad najpoužívanejších RL algoritmov: Iterovanie hodnôt, Iterovanie stratégií, metóda SARSA, Monte Carlo simulácie, metóda Q-učenie a algoritmus Dyna-Q. V praktickej časti práce vyriešime dve modelové úlohy s využitím predstavených techník. Jednotlivé algoritmy implementujeme v prostredí `Matlab`. V závere práce porovnáme jednotlivé prístupy a zhodnotíme ich využiteľnosť pri riešení úloh stochastického dynamického programovania.

Kľúčové slová: reinforcement learning, stochastické optimálne riadenie, markovovské rozhodovacie procesy, Bellmanova rovnica

Abstract

SKOKANOVÁ, Jana: Reinforcement Learning Algorithms [Diploma Thesis], Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: RNDr. Zuzana Chladná, Dr., Bratislava, 2021, 130 p.

We classify reiforcement learning (RL) algorithms as methods of machine learning. Reinforcement learning algorithms can be used to solve complex tasks of stochastic dynamic programming. This work provides an overview of the most used RL algorithms: Value Iteration, Policy Iteration, SARSA method, Monte Carlo simulations, Q-learning and Dyna-Q algorithm. In the practical part of the work we will solve two model tasks using the presented techniques. We implement individual algorithms in the `Matlab` environment. At the end of the work we will compare individual approaches and evaluate their usability in solving problems of stochastic dynamic programming.

Keywords: reinforcement learning, stochastic optimal control, Markov decision processes, Bellman's equation

Obsah

Zoznam použitých symbolov **9**

Úvod **10**

1 Reinforcement learning **12**

1.1 História reinforcement learningu 13

1.2 RL v teórii strojového učenia 14

1.3 Základné pojmy RL 15

1.3.1 Markovovské reťazce 16

1.3.2 Funkcie špecifické pre RL 18

1.3.3 Ohodnocovacie funkcie 19

2 Úloha o prechode robota v miestnosti **23**

2.1 Popis problému 23

2.2 Matematický model 24

2.3 Formulácia v štandardnom tvare úloh OR 26

3 Prehľad metód reinforcement learningu **32**

3.1 Základné delenie algoritmov RL 32

3.2 Dynamické programovanie 33

3.2.1 Bellmanova rovnica dynamického programovania 34

3.2.2 Iterovanie stratégií 36

3.2.3 Aplikácia metódy iterovania stratégií na vzorový problém 38

3.2.4 Iterovanie hodnôt 40

3.2.5 Aplikácia metódy iterovania hodnôt na vzorový problém 41

3.3 Metódy temporálnych diferencií 42

3.3.1 Princíp TD 43

3.3.2 Problém explorácie a exploitácie 43

3.3.3 Metóda SARSA 46

3.3.4 Aplikácia metódy SARSA na vzorový problém 48

3.4 Q-učenie 49

3.4.1 Aplikácia Q-učenia na vzorový problém 51

3.5 Metóda Monte Carlo 52
3.5.1 Aplikácia Monte Carlo metódy na vzorový problém 55
3.6 Dyna-Q 56
3.6.1 Aplikácia algoritmu Dyna-Q na vzorový problém 58

4 Aplikácia RL algoritmov na úlohu stochastického OR 61
4.1 Úloha o obchodnej prevádzke 61
4.2 Matematická formulácia úlohy o obchodnej prevádzke 62
4.3 Riešenie pomocou metódy stochastického OR 64
4.3.1 Aplikácia metódy stochastiského OR na úlohu o obchodnej prevádzke 67
4.4 Riešenie pomocou metód RL 67
4.5 Porovnanie použitých prístupov 69

Záver 72

Zoznam použitej literatúry 74

Príloha A 77

Príloha B 125

Zoznam použitých symbolov

MC: Monte Carlo

MRP: markovovský rozhodovací proces

OR: optimálne riadenie

RDP: rovnica dynamického programovania

RL: reinforcement learning

TD: temporálne diferencie

Úvod

Odbor strojového učenia je v posledných desaťročiach na výraznom vzostupe. Zároveň s ním sa do popredia dostal aj odbor reinforcement learningu ako jedna z jeho súčastí. Metódy reinforcement learningu sú založené na postupnom rozhodovaní, ktoré sa deje na základe nadobudnutých skúseností. Tieto metódy interagujú s prostredím, aby nadobúdali potrebné informácie o tom, aké odmeny plynú z vykonaných akcií.

Vychádzajúc z [23] vznikol odbor strojového učenia (spolu s reinforcement learningom) už v minulom storočí, ale po dlhú dobu čakal na svoje praktické využitie. V dnešnom svete má najmä reinforcement learning enormný potenciál na aplikáciu v mnohých odvetviach. Uvedúc pár príkladov, reinforcement learning je možné použiť v zdravotníctve pri diagnostikovaní chronických ochorení, akými sú napríklad cukrovka alebo schizofrénia. Ďalej je možné aplikovať metódy reinforcement learningu aj v školstve, a to konkrétne pri personalizácii výučbového procesu [25]. Samozrejme nemožno zabudnúť na jeho mnohé aplikácie v IT odvetví, najmä pri počítačových hrách. Vychádzajúc z [27] je možné pomocou reinforcement learning algoritmov vytvoriť bota, ktorý dokáže sám vyhrať v počítačovej hre.

Myšlienka reinforcement learningu vznikla spojením metódy pokus-omyl a metód optimálneho riadenia, najmä dynamického programovania. Cieľom tejto diplomovej práce je ukázať, že algoritmy reinforcement learningu ponúkajú alternatívny prístup k riešeniu problémov optimálneho riadenia. Ide najmä o také problémy, pri ktorých štandardné numerické techniky diskrétnej dynamickej optimalizácie zlyhávajú z dôvodu veľkého počtu stavových a riadiacich premenných. V tejto diplomovej práci predstavíme algoritmy reinforcement learningu, ktoré implementujeme na vzorový problém RL. Zároveň sa pokúsime vyriešiť zvolenú úlohu stochastického optimálneho riadenia pomocou tradičných metód optimálneho riadenia a aj pomocou RL algoritmov. Na záver porovnáme výhody a nevýhody oboch prístupov.

Diplomová práca bude členená do štyroch kapitol. V prvej kapitole sa zameriame na predstavenie reinforcement learningu ako vedeckého odvetvia. Definujeme jeho základné pojmy a princípy za pomoci teórie markovovských rozhodovacích procesov. Druhá kapitola bude venovaná špecifikovaniu problému, resp. konkrétnej úlohy typickej pre reinforcement learning. V tretej kapitole uvedieme všeobecnú kategorizáciu RL algoritmov a zároveň

popíšeme úlohu Bellmanovej rovnice dynamického programovania, ktorá je základným stavebným kameňom reinforcement learningu. Taktiež v tejto kapitole predstavíme vybrané RL algoritmy a implementujeme ich na problém popísaný v predchádzajúcej kapitole. Konkrétne pôjde o tieto algoritmy: *iterovanie hodnôt*, *iterovanie stratégií*, *Monte Carlo simulácie*, *algoritmus temporálnych diferencií SARSA*, *Q-učenie* a *algoritmus Dyna-Q*. V poslednej štvrtej kapitole aplikujeme algoritmy reinforcement learningu a metódy optimálneho riadenia na úlohu riešenú typicky tradičnými metódami stochastického optimálneho riadenia a zhodnotíme výhody a nevýhody všetkých zvolených prístupov. To nám umožní zodpovedať otázku, či sú, alebo nie sú reinforcement learning algoritmy budúcnosťou odboru optimálneho riadenia v prípadoch, keď zlyhávajú štandardné numerické techniky diskrétnej dynamickej optimalizácie.

1 Reinforcement learning

V úvode diplomovej práce predstavíme reinforcement learning (ďalej aj RL) ako modernú vedeckú disciplínu. Definujeme pojmy a popíšeme základné princípy, ktoré sa v tomto vedeckom odvetví používajú.

Vychádzajúc z [23] vieme, že metódy reinforcement learningu boli vo svojich začiatkoch inšpirované správaním sa živých organizmov, ich metódami učenia a následného správania sa podľa týchto nadobudnutých informácií a skúseností. Organizmy sa typicky učia tak, že interagujú s prostredím, v ktorom sa vyskytujú. Následne využívajú skúsenosti získané v tomto prostredí k vylepšeniu svojho doterajšieho správania. Ide teda o zmenu správania organizmov za účelom dosiahnutia konkrétneho cieľa (napr. naučenie sa nových pohybov, minimalizovanie svojho úsilia na získanie potravy a najmä dosiahnutie prežitia).

Ak sa pozrieme podrobnejšie na spôsob učenia sa organizmov, môžeme pozorovať, že ide o tzv. posilované učenie (t.j. reinforcement learning). Takýto spôsob učenia možno charakterizovať nasledovne. Organizmy poznajú cieľ, ktorý chcú dosiahnuť, nepoznajú však cestu ako sa k tomuto cieľu dostať. Spomínanú cestu vedúcu ku konkrétnemu cieľu môžeme označiť ako postupnosť správnych akcií. Na to, aby bol organizmus schopný naučiť sa správnu (teda optimálnu) postupnosť akcií, je potrebné, aby mal tento organizmus možnosť získať spätnú väzbu z prostredia, prislúchajúcu vykonanej akcii. Na ilustráciu uvedieme niekoľko príkladov. Jednoduchým príkladom sú mravce, ktoré majú na výber dve rôzne cesty k mŕtvej húsenici (potrave). Ak mravce dostanú od prostredia spätnú väzbu a vyhodnotia obe cesty ako rovnako bezpečné, nakoniec si zvolia kratšiu cestu. Metóda posilovaného učenia ich teda priviedla k riešeniu, ktoré minimalizuje ich úsilie. Ďalším príkladom je učenie sa šteniatok, akým spôsobom sa správať, aby im ich majiteľ vyhovel a šiel sa s nimi prejsť. Po určitom čase psíky zistia, ktorá sekvencia ich dovedie k vysnívanej prechádzke (ich konkrétnemu cieľu) a súčasne minimalizuje ich úsilie. Posledným príkladom spätnej väzby, ktorý uvedieme, možno pozorovať u organizmov s rozvinutou nervovou sústavou (napríklad u ľudí). Hormóny v organizme sú schopné pozitívne, respektíve negatívne stimulovať mozog. Z dlhodobého hľadiska tak ľudia majú šancu naučiť správať sa tak (t.j. vyberať si také postupnosti akcií), aby sa im vyplavovali najmä pozitívne pôsobiace hormóny a boli tak šťastní.

1.1 História reinforcement learningu

Reinforcement learning možno považovať za relatívne mladé vedecké odvetvie, keďže sa dostal do povedomia vedeckej obce až v osemdesiatych rokov dvadsiateho storočia. Vychádzajúc z [23] je RL z historického hľadiska spojením dvoch, resp. troch, významných prúdov, ktoré sa pred zlúčením do reinforcement learningu vyvíjali v minulosti nezávisle od seba. Prvý zo spomínaných troch prúdov predstavuje tzv. učenie sa pomocou metódy pokus-omyl. Tento typ učenia sa prvýkrát objavil v oblasti psychológie učenia sa u zvierat. Učenie sa pomocou metódy pokus-omyl sa taktiež objavilo v prvých prácach týkajúcich sa umelej inteligencie. Druhým prúdom, ktorý výrazne ovplyvnil RL, je optimálne riadenie zahŕňajúce riešenie rôznych problémov pomocou hodnotových funkcií a dynamického programovania. Je dôležité poznamenať, že metódy optimálneho riadenia dokážu v značnej miere fungovať bez akéhokoľvek učenia sa. Spojenie a premiešanie spomenutých dvoch oblastí, ktoré sa na prvý pohľad javia ako zdanlivo nezávislé, postavilo základ technikám reinforcement learningu.

V roku 1978 Richard Sutton [22] nadviazal na ideu A. Harryho Klopfa [11], ktorý ako prvý použil pojem „zovšeobecnený reinforcement", týkajúcu sa teórie učenia sa u zvierat. Popísal v nej akési pravidlá učenia sa, ktoré sú založené na sledovaní zmien postupných predikcií v čase. Neskôr Richard Sutton a Andrew Barto spolu túto teóriu ešte vylepšili a zároveň vyvinuli psychologický model podmieňovania založený na učení sa pomocou tzv. temporálnych diferencií [2], [3]. Táto práca bola odrazovým mostíkom pre mnoho ďalších modelov vychádzajúcich z rovnakej teórie. Niektoré z neurovedeckých modelov [1], [10] vytvorených v týchto rokoch sa dajú taktiež veľmi dobre interpretovať v reči učenia sa pomocou temporálnych diferencií, i keď z historického hľadiska nie je doložené prepojenie týchto neurovedeckých modelov s prvým psychologickým modelom Suttona a Bartoa.

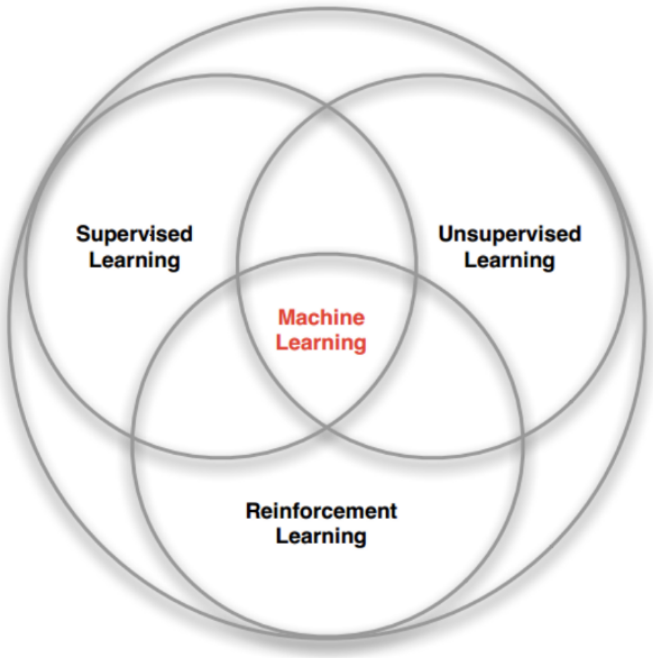
Teória o temporálnych diferenciách a teória optimálneho riadenia boli finálne spojené v práci Chrisa Watkinsa z 1989 [26], ktorá popisovala rozvoj tzv. *Q-učenia*. Táto práca napokon zlúčila a rozšírila predošlé práce zo všetkých vyššie spomenutých prúdov. I keď je nutné poznamenať, že v čase vydania Watkinsonovej práce boli už určité teórie reinforcement learningu vo výraznom rozmachu. Išlo najmä o oblasť strojového učenia, konkrétne v štúdiách o umelej inteligencii, ale taktiež RL značne prekvital v aplikáciách týkajúcich sa neurónových sietí. Od roku 1989 je tak rozvoj poľa reinforcement learningu skutočne

enormný.

V súčasnosti možno reinforcement learning charakterizovať ako časť strojového učenia, teda vedy, ktorej cieľom je naučiť počítače (respektíve softvér vykonávajúci akcie) konať správne bez explicitného naprogramovania. Vychádzajúc z [24] reinforcement learning rovnako ako mnoho odvetví končiacich sa príponou $-ing$ predstavuje triedu metód riešení vhodných, resp. efektívnych, pri riešení určitej skupiny problémov. Zároveň ide o odvetvie, ktoré popisuje tieto problémy a metódy ich riešenia.

1.2 RL v teórii strojového učenia

Vychádzajúc z [24] rozlišujeme v teórii *machine learningu* (alebo aj strojového učenia) dve hlavné skupiny algoritmov. A to tzv. *supervised learning* (t.j. učenie sa s učiteľom) a tzv. *unsupervised learning* (t.j. učenie sa bez učiteľa). Metódy RL tvoria však samostatnú skupinu metód, keďže ich nie je možné zaradiť ani do jednej z vyššie spomenutých skupín algoritmov. Na Obrázku č.1 pochádzajúceho z [16] možno sledovať postavenie RL v teórii strojového učenia.



Obr. 1: Pozícia RL v teórii strojového učenia

Rozdiel medzi RL a supervised learningom je najmä v tom, že pri reinforcement learningu absentuje tzv. *trénovacia množina*. Ide o množinu, ktorej prvkami sú páry

vstupov a výstupov. Pri supervised learningu je rozhodovací proces založený na znalosti trénovacej množiny a na následnom hľadaní podobností medzi pozorovanými množinami a trénovacou množinou. Rozhodovací proces pri reinforcement learningu je založený čisto na skúsenostiach získaných interagovaním s prostredím. A teda možno poznamenať, že algoritmus nemá k dispozícii žiadne príklady „správneho správania sa", t.j. správnych akcií.

Metódy unsupervised learningu možno charakterizovať ako podskupinu algoritmov strojového učenia, ktorá sa snaží nájsť určité skryté podobné vlastnosti v neoznačených dátach. Jedinečnosť algoritmov RL spočíva v princípe učenia sa na základe maximalizácie odmien, resp. minimalizácie celkových nákladov, ktoré učiaci sa organizmus získava po vykonaní prípustných akcií.

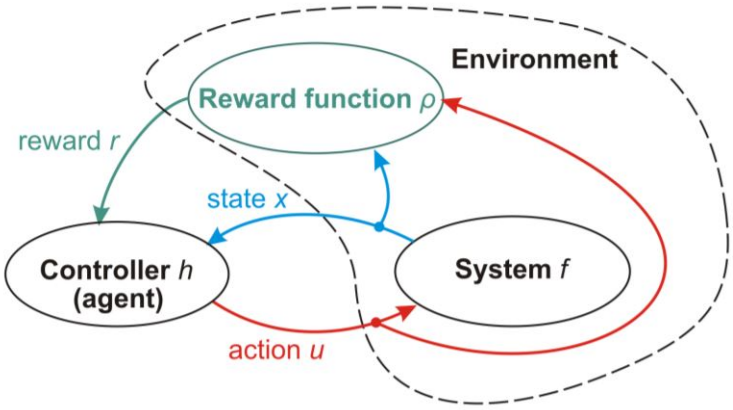
Poznamenajme, že v teórii reinforcement learningu sa používa vo väčšej miere pri optimalizácii maximalizácia celkových odmien. Na druhej strane, v teórii optimálneho riadenia sa častokrát stretávame aj s minimalizáciou konkrétnych hodnôt. Pre účely tejto diplomovej práce budeme používať pri optimalizačných úlohách formuláciu postavenú na maximalizácii celkových odmien.

1.3 Základné pojmy RL

V tejto podkapitole si predstavíme základné pojmy z oblasti reinforcement learningu. Mnohé z nich sú prebraté z iných vedných odvetví, na ktorých bol reinforcement learning postavený. Ide najmä o teóriu optimálneho riadenia a markovovských reťazcov.

Pri reinforcement learningu je typickým problémom proces učenia sa ako robiť postupné rozhodnutia, tzv. *akcie*, aby bol optimalizovaný celkový (kumulatívny) výkon neznámeho systému. Typická štruktúra RL obsahuje dva neustále integarujúce prvky. Ide konkrétne o tzv. *agenta* a o *prostredie* (environment). Pojmom agent rozumieme jednotku (organizmus), ktorá vykonáva akcie, učí sa a rozhoduje o tom, aká akcia bude zvolená. Za prostredie sa považuje neznámy sledovaný systém (napr. stroj alebo herné rozhranie), súbor všetkých jeho fyzikálnych veličín a taktiež parametrov, ktoré majú akýkoľvek vplyv na agenta, či už jeho rozhodovanie alebo dosiahnutie cieľa. Je dôležité poznamenať, že po vykonaní akcie agentom, sledovaný systém naviac poskytne informáciu o stave, v ktorom sa momentálne nachádza, a taktiež o odmene plynúcej z akcie vykonanej agentom.

Poznamenajme, že špecifikom reinforcement learningu v strojovom učení je skutočnosť, že dopredu nie je známe, aké akcie sú správne. Jedinou informáciou o vykonaných akciách sú odmeny, ktoré daný sledovaný systém agentovi poskytuje. Vizualizácia typického systému RL je na Obrázku č.2 pochádzajúceho z [5].



Obr. 2: Vizualizácia typického modelu RL

Okrem už spomínaných pojmov ako agent, akcia a prostedie vystupujú v typickej úlohe reinforcement learningu ďalšie dôležité pojmy ako funkcia odmien a stratégia. Pred ich predstavením je však potrebné uviesť viacero poznatkov z teórie markovovských reťazcov, keďže mnohé typy úloh RL sú postavené na teoretickom základe markovovského rozhodovacieho procesu.

1.3.1 Markovovské reťazce

Markovovský rozhodovací proces, ktorý predstavíme v tejto časti diplomovej práce, predstavuje matematický nástroj stavaný na popisovanie deterministických a stochastických rozhodovacích procesov, ktoré sledujeme v diskrétnych časových krokoch $i = 0, 1, 2, \ldots$. V každom kroku i je proces v konkrétnom stave x_i. Tento stav môže zmeniť agent tým, že vykoná akciu u_i. Je však potrebné si uvedomiť, že proces v časovom kroku $i+1$ po vykonaní akcie u_i prejde do stavu x_{i+1} na základe tzv. *pravdepodobnosti prechodu*. Práve pri pochopení prechodu sledovaného systému medzi jeho možnými stavmi sa ukazuje teória markovovských reťazcov ako kľúčová.

Poznamenajme, že z hľadiska teórie markovovských reťazcov budeme stav systému x považovať za náhodnú premennú. Ďalej označme X množinu všetkých stavov, v ktorých

sa môže sledovaný systém nachádzať.

Už spomínanú akciu vykonanú agentom vtedy, keď sa sledovaný systém nachádza v stave x_i, budeme označovať u_i. Ďalej označme množinu všetkých akcií, ktoré môže agent vykonať, ako U.

Postupnosť stavov, medzi ktorými sa v čase prejde buď pomocou akcie vykonanej agentom, alebo vďaka charakteru sledovaného systému, možno chápať ako náhodný proces. V teórii reinforcement learningu naviac sledujeme špecifické náhodné procesy, a to tzv. *reťazce*, ktoré je možné definovať nasledujúcim spôsobom.

Definícia 1.1. *([8]) Náhodný proces, ktorý nadobúda hodnoty z konečnej alebo spočítateľnej množiny stavov X, sa nazýva reťazec.*

Konkrétne je na vybudovanie teórie reinforcement learningu potrebné sledovať tzv. *markovovské reťazce*. T.j. reťazce, ktoré majú tzv. *Markovovu vlastnosť*. To znamená, že ide o reťazce, ktorých prechod zo stavu x_i do stavu x_{i+1} závisí iba od súčasného stavu x_i a v ňom vykonanej akcie u_i. Všetky informácie potrebné k prechodu do nového stavu sú obsiahnuté v informácii o súčasnom stave x_i.

Definícia 1.2. *([8]) Reťazec $(X_i)_{i=0}^{\infty}$ má Markovovu vlastnosť, ak pre $\forall i \in N \quad \forall x_0, x_1, ..., x_i \in \boldsymbol{X}$: $P(X_{i+1} = x_{i+1}|X_i = x_i, ..., X_1 = x_1, X_0 = x_0) = P(X_{i+1} = x_{i+1}|X_i = x_i)$.*

Ďalším potrebných pojmom z teórie markovovských reťazcov je tzv. *pravdepodobnosť prechodu medzi dvoma stavmi reťazca*, ktorú možno definovať nasledujúcim spôsobom.

Definícia 1.3. *([8]) Hodnotu $p_{jk}(i, i+1) = P(X_{i+1} = k|X_i = j)$ budeme nazývať pravdepodnosť prechodu zo stavu j v kroku i do stavu k v kroku i+1. Markovovský reťazec sa naviac nazýva homogénny, ak hodnoty $p_{jk} = p_{jk}(i, i+1)$ nezávisia od hodnoty i.*

Pre potreby tejto diplomovej práce budeme predpokladať, že všetky spomínané markovovské reťazce sú homogénne.

Kompaktný zápis pravdepodobností prechodu predstavuje tzv. *matica prechodu*, ktorú možno definovať nasledovne.

Definícia 1.4. *([8]) Matica prechodu markovovského reťazca $(X_i)_{i=0}^{\infty}$ je $P = (p_{jk})$, kde p_{jk} predstavuje pravdepodobnosť prechodu zo stavu j do stavu k.*

Ďalším pojmom súvisiacim s pravdepodobnosťou prechodu je tzv. *funkcia prechodu*, ktorá má v prípade reinforcement learningu špecifický tvar. A to z dôvodu, že prechody medzi stavmi reťazca (a teda sledovaného systému) sa nedejú iba náhodne (vďaka charakteru systému), ale aj vďaka akciám agenta. Funkciu prechodu zo stavu x_i do stavu x_{i+1} za vykonania akcie u_i budeme označovať $f(x_i, u_i, x_{i+1})$. Táto funkcia má špecifický tvar pre každú konkrétnu úlohu. Poznamenajme, že pre funkciu prechodu f platí, že $f : X \times U \times X \to [0, 1]$.

1.3.2 Funkcie špecifické pre RL

Dôležitou súčasťou každej úlohy riešiteľnej pomocou RL je tzv. *funkcia odmien* (reward function). Pomocou tejto funkcie dostáva agent spätnú väzbu hovoriacu o veľkosti odmeny, ktorá plynie z vykonanej akcie. Funkciu odmien pri prechode systému zo stavu x_i do stavu x_{i+1} za vykonania akcie u_i budeme označovať pre potreby tejto diplomovej práce ako $r(x_i, u_i, x_{i+1})$. Podobne ako funkcia prechodu je aj funkcia odmien špecifická pre každý sledovaný systém. Zároveň platí, že $r : X \times U \times X \to \mathbb{R}$.

Keďže sme popísali funkciu odmien, tak je možné definovať tzv. *markovovský rozhodovací proces*, skrátene MRP, porozumenie ktorého je kľúčové pre prácu s algoritmami reinforcement learningu.

Definícia 1.5. *([23]) Markovovský rozhodovací proces definujeme ako usporiadanú štvoricu (X, U, f(.), r(.)), kde:*

- *X je množina všetkých stavov,*
- *U je množina všetkých akcií,*
- $f(x_i, u_i, x_{i+1}) \in [0, 1]$ *je funkciu prechodu pre všetky dvojice stavov* (x_i, x_{i+1})*, kde* $x_i, x_{i+1} \in X$ *za vykonania akcie* $u_i \in U$*,*
- $r(x_i, u_i, x_{i+1}) \in \mathbb{R}$ *je funkcia odmien získaných pri prechode zo stavu* x_i *do stavu* x_{i+1} *vykonaním akcie* $u_i \in U$*.*

Posledným, no o to dôležitejším pojmom, v teórii reinforcement learningu je pojem tzv. *stratégie* (policy). Tá je analógiou pre množinu riadení z teórie optimálneho riadenia. Vychádzajúc z [7] množina riadení predstavuje množinu konkrétnych akcií, ktoré sa majú

vykonať v daných časových krokoch. Stratégia predstavuje súhrn informácií hovoriacich o tom, aká akcia sa vykoná v danom stave. Podľa [5] stratégiu možno popísať ako funkciu, ktorá danému stavu x_i priraďuje akciu u_i. V tejto diplomovej práci ju budeme označovať h. Môžeme teda písať, že $h : X \to U$. Teda

$$u_i = h(x_i). \tag{1}$$

Poznamenajme, že typicky stratégia obsahuje jeden údaj pre každý dosiahnuteľný (neterminálny) stav. A to z dôvodu, že určuje, aká akcia sa má vykonať v danom stave.

Pri úlohách MRP je typickým cieľom nájsť optimálnu stratégiu $h^*(x)$ takým spôsobom, aby bol maximalizovaný dlhodobý kumulatívny súčet odmien, či už pri nekonečnom chode MRP alebo pri dosiahnutí konečného (terminálneho) stavu x_{term} (niekedy označovaného ako cieľ).

Pojem *dlhodobý kumulatívny súčet odmien* by sa ponúkalo definovať ako sumu všetkých odmien získaných chodom MRP. Avšak v prípade nekonečného procesu by tento výraz skonvergoval do nekonečna a jeho maximalizácia by teda nedávala zmysel. Je teda potrebné zaviesť pojem tzv. *dlhodobého kumulatívneho súčtu očakávaných odmien*. Vychádzajúc z [5] označíme dlhodobý kumulatívny súčet očakávaných odmien so začiatkom v diskrétnom časovom kroku i ako R_i:

$$\begin{aligned} R_i &= r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + ... \\ &= \sum_{k=0}^{T} \gamma^k r(x_{i+k}, u_{i+k}, x_{i+k+1}), \end{aligned} \tag{2}$$

kde $\gamma \in (0, 1]$ predstavuje diskontný faktor. Pre nekonečné procesy, kde $T = \infty$ je potrebné zvoliť diskontný faktor $\gamma < 1$. Tak bude zaručená konvergencia nekonečnej sumy.

1.3.3 Ohodnocovacie funkcie

Pomocou dlhodobého kumulatívneho súčtu očakávaných odmien definovaného pomocou rovnice (2) sme schopní kvantifikovať vzdialenosť sledovanej stratégie $h(x)$ od optimálnej stratégie $h^*(x)$. Vychádzajúc z [23] sú na túto kvantifikáciu vhodné tzv. *ohodnocovacie funkcie*. Konkrétne ide o tzv. *V-funkciu* (z anglického value function), ktorú budeme

označovať $V^h(x)$ a tzv. *Q-funkciu* (z anglického Q-function), ktorú budeme označovať $Q^h(x, u)$.

- **V-funkcia**:

  Definícia 1.6. *([23]) V-funkciu $V^h(x) : X \to \mathbb{R}$ definujeme ako funkciu, ktorá priraďuje stavu $x \in X$ dlhodobý kumulatívny súčet očakávaných odmien pri aplikovaní stratégie $h(x)$. Teda*

$$\begin{aligned} V^h(x) &= \mathbb{E}_h \left[R_i | x_i = x \right] \\ &= \mathbb{E}_h \left[\sum_{k=0}^{T} \gamma^k r(x_{i+k}, u_{i+k}, x_{i+k+1}) \middle| x_i = x \right]. \end{aligned} \tag{3}$$

  Poznamenajme, že na diskontný faktor γ je možné v tomto prípade nahliadnuť ako na určitú neistotu získaných odmien v budúcich diskrétnych časoch (resp. v krokoch, ktoré sa ešte len odohrajú v budúcnosti).

  Keďže je našou úlohou maximalizovať dlhodobý kumulatívny súčet odmien, tak pre každú neoptimálnu stratégiu $h(x)$ platí nasledovné

$$V^h(x) < V^{h^*}(x). \tag{4}$$

  To znamená, že pre optimálnu stratégiu $h^*(x)$ bude hodnota V-funkcie $V^{h^*}(x)$ v stave x vždy väčšia. Zároveň platí, že čím je hodnota funkcie $V^h(x)$ v sledovanom stave x väčšia, tým je pre sledovanú stratégiu lepšie sa nachádzať v danom stave, keďže celkový kumulatívny súčet očakávaných odmien je v tomto stave väčší.

  Pomocou funkcie $V^h(x)$ je taktiež možné definovať riešenie markovovského rozhodovacieho procesu ako nájdenie optimálnej stratégie $h^*(x)$. To znamená stratégie, pre ktorú budú hodnoty V-funkcie pre všetky stavy maximálne. A teda:

$$V^*(x) = \max_{h(.)} V^h(x). \tag{5}$$

  Optimálna stratégia je potom definovaná nasledovne:

$$h^*(x) = arg \max_{h(.)} V^h(x). \tag{6}$$

Poznamenajme, že pre zložitejšie systémy, resp. procesy, je nájdenie riešenia MRP vo všeobecnosti veľmi obtiažne. Zároveň je dôležité uvedomiť si, že je nutné maximalizovať celkový dlhodobý kumulatívny súčet odmien, nie iba samotnú odmenu v jednom kroku. V určitých krokoch totižto môže akcia dávajúca vyššiu okamžitú odmenu zaviesť agenta do takej situácie, v ktorej budú dlhodobo odmeny záporné (t.j. budú predstavovať náklady).

- **Q-funkcia**:

  Na rozdiel od V-funkcie, Q-funkcia neohodnocuje iba stavy, ale ohodnocuje vykonanie akcie u v stave x za aplikovania stratégie $h(x)$. T.j. do argumentu Q-funkcie vstupujú páry pozostávajúce z jedného stavu a jednej akcie. Pre potreby tejto diplomovej práce budeme používať skrátené označenie *pár stav-akcia*.

  Definícia 1.7. *([23]) Q-funkciu $Q^h(x,u) : X \times U \to \mathbb{R}$ definujeme ako funkciu, ktorá priraďuje páru stav-akcia (x,u), kde $x \in X$, $u \in U$, dlhodobý kumulatívny súčet očakávaných odmien za aplikovania stratégie $h(x)$. Teda*

$$\begin{aligned} Q^h(x,u) &= \mathbb{E}_h\left[R_i | x_i = x, u_i = u\right] \\ &= \mathbb{E}_h\left[\sum_{k=0}^{T} \gamma^k r(x_{i+k}, u_{i+k}, x_{i+k+1}) \middle| x_i = x, u_i = u\right]. \end{aligned} \tag{7}$$

  Pre Q-funkciu existujú analogické vzťahy k vzťahom (5), (6) týkajúcich sa V-funkcie. A to z dôvodu, že riešenie MRP je možné definovať aj podľa Q-funkcie ako nájdenie optimálnej stratégie $h^*(x)$, a to konkrétne nasledujúcim spôsobom:

$$Q^*(x,u) = \max_{h(.)} Q^h(x,u). \tag{8}$$

  Optimálna stratégia je potom definovaná ako:

$$h^*(x) = arg \max_{h(.)} Q^h(x,u). \tag{9}$$

Všimnime si, že medzi V-funkciou a Q-funkciou existuje vzťah. Stačí si uvedomiť, že ak na úlohu aplikujeme stratégiu $h(x)$, tak platí, že

$$Q^h(x, h(x)) = V^h(x). \tag{10}$$

Vzťah 10 nám umožňuje na Q-funkciu nahliadnuť ako na určité rozšírenie V-funkcie. Použitím V-funkcie na každý stav $x \in X$ sa uchová iba jedna hodnota. Na druhej strane, použitím Q-funkcie pre daný stav $x \in X$ sa naviac uchovajú hodnoty pre všetky možné akcie. To sa ukazuje ako veľmi praktické pri metódach, ktoré nie sú založené na priamom nasledovaní konkrétnej stratégie, ale práve na dynamickej zmene stratégie.

2 Úloha o prechode robota v miestnosti

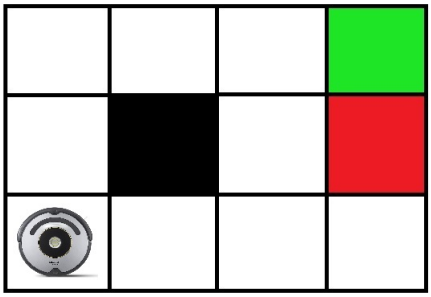
V tejto kapitole si predstavíme úlohu o prechode robota v miestnosti. Ide o úlohu stochastického dynamického programovania, ktorá sa štandardne rieši technikami RL. Zvolenú vzorovú úlohu naformulujeme v súlade s metodikou predstavenou v Kapitole 1. V závere kapitoly sa pokúsime o formuláciu danej úlohy v štandardnom tvare úloh optimálneho riadenia (viď[7]).

2.1 Popis problému

Úlohu, ktorú sme zvolili za vzorovú, sme prevzali z [21] a následne mierne modifikovali. Problém prechodu robota možno popísať nasledovne:

Uvažujme robota, ktorý sa pohybuje v miestnosti obdĺžnikového tvaru. Túto miestnosť je možné rozdeliť na 4 x 3 = 12 dlaždíc (4 - vodorovne, 3 - zvisle) tvoriacich mriežku. Každá dlaždica prislúcha konkrétnej pozícii, v ktorej sa môže robot nachádzať. Na pozícii [2, 2] sa v miestnosti nachádza nosník a teda robot nie je schopný na túto pozíciu „vkročiť". Zároveň pozícia [2, 4] predstavuje otvor v podlahe, ktorým sa začína schodisko. Posledným špecifikom sledovanej miestnosti je nabíjacia stanica pre robota, ktorá je umiestnená na pozícii [1, 4]. Otvor na schodisko rovnako ako nabíjacia stanica predstavujú pre robota miesto, z ktorého sa nie je schopný v krátkom čase pohnúť. V tejto úlohe budú preto tieto pozície chápané ako konečné stavy. V tejto úlohe budeme predpokladať, že robot je spustený do chodu na pozícii [3, 1] a jeho cieľom je dostať sa do nabíjacej stanice. Popísaná miestnosť je zobrazená na Obrázku č.3. Robot je schopný pohybovať sa štyrmi smermi, a to hore, dole, vpravo a vľavo. Robot má však softvérovú chybu. Ak zvolíme určitý smer, ktorým sa má robot vydať, pôjde týmto smerom iba s 80%-nou pravdepodobnosťou a s pravdepodobnosťou 10% sa pohne vpravo, respektíve vľavo od nami zvoleného smeru. To znamená, že ak zvolíme napríklad smer hore, tak robot s pravdepobnosťou 80% pôjde hore a s pravdepodobnoťou 10% doprava, resp. doľava. Ak by sme zvolili smer vpravo, tak robot sa s pravdepobnosťou 80% pohne vpravo a s 10%-nou pravdepodobnosťou zvolí smer hore, poťažmo dole. Analogicky sa bude správať pri zvolení smerov dole, respektíve vľavo. Poznamenajme, že pri každom pokuse o pohyb má robot energetické náklady vo výške 0,07. V prípade, že dosiahne nabíjacu stanicu, plne sa dobije a jeho odmena bude 1. Naopak v

prípade, že stúpi na dlaždicu prislúchajúcu schodisku, jeho odmena bude -1.



Obr. 3: Prostredie, v ktorom sa pohybuje robot (zelená dlaždica = nabíjacia stanica, červená dlaždica = otvor v podlahe, čierna dlaždica = nosník).

Poznamenajme, že ak sa bude robot nachádzať napríklad na pozícii [1, 2] a zvolíme akciu hore, tak s pravdepodobnosťou 80% robot „narazí" na stenu miestnosti. To znamená, že robot sa s pravdepodobnosťou 80% nepohne. Jeho náklady v danom čase však budú *0,07*, keďže vynaložil energiu na pohyb.

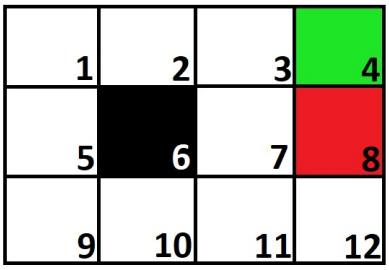
V tejto úlohe sa budeme snažiť maximalizovať odmenu plynúcu z pohybu robota, respektíve minimalizovať náklady robota na pohyb, berúc do úvahy odmeny plynúce z príchodu do nabíjacej stanice, resp. penalizáciu v prípade spadnutia do otvoru v podlahe.

2.2 Matematický model

Teraz sa pokúsime naformulovať problém o prechode robota s využitím metodiky predstavenej v Časti 1.3. Na určenie pozície robota v miestnosti nebudeme používať súradnicovú sústavu, ale jednotlivým dlaždiciam priradíme čísla od *1* do *12*. Priradené číslo jednoznačne identifikuje polohu robota v miestnosti. Konkrétny spôsob označenia dlaždíc je zobrazený na Obrázku č.4.

Prejdime teraz k samotnej formulácii úlohy. V nasledujúcich bodoch si popíšeme jednotlivé funkcie a premenné vstupujúce do modelu:

- robot sa môže nachádzať na 12-tich pozíciách, t.j. množina stavov je $X = \{1, ..., 12\}$,
- na začiatku sa robot nachádza na pozícii *9*, t.j. počiatočný stav je $x_0 = 9$,



Obr. 4: Grafické znázornenie identifikácie polohy robota v miestnosti

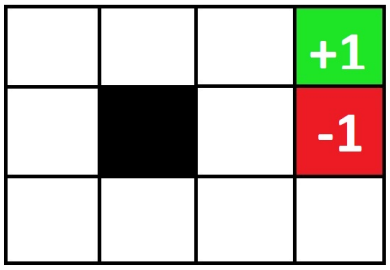
- stavy *4* a *8* sú pre robota terminálnymi (t.j. konečnými) stavmi,
- robot môže vykonať 4 akcie: H - hore, D - dole, P - vpravo, L - vľavo. Množina akcií (riadení) je $U = \{H, D, P, L\}$,
- robot volí smer, ktorým sa vyberie, potom ako sme mu zadali konkrétny smer, s pravdepodobnosťami, ktoré sú uvedené v nasledujúcej matici T:

$$T = \begin{bmatrix} 0,8 & 0 & 0,1 & 0,1 \\ 0 & 0,8 & 0,1 & 0,1 \\ 0,1 & 0,1 & 0,8 & 0 \\ 0,1 & 0,1 & 0 & 0,8 \end{bmatrix}.$$

  Výsledný smer, ktorým sa robot pohne, závisí od zvolenej akcie a náhody. Prvok t_{ij} v matici T označuje pravdepodobnosť, že sa robot pohne smerom j, ak zvolená akcia bola i. Poznamenajme, že v matici T prvý riadok prislúcha smeru H, druhý D, tretí P a štvrtý riadok prislúcha smeru L. So stĺpcami je to analogicky. Zároveň poznamenajme, že matica T predstavuje maticu prechodu popísanú v Časti 1.3.1, preto súčet prvkov každého riadku je rovný 1,
- energetické náklady na pohyb robota sú *0,07* (aj v prípade, že sa robot pokúsi pohnúť, ale narazí na stenu, alebo na nosník na pozícii *6*, a v konečnom dôsledku sa nepohne), teda $r_{pohyb} = -0,07$,
- dosiahnutím stavu (dlaždice) *4* získa robot odmenu *+1*, teda $r_{stav4} = 1$,
- dosiahnutím stavu (dlaždice) *8* získa robot odmenu *-1*, teda $r_{stav8} = -1$,

- úlohou je maximalizovať odmenu plynúcu z pohybu robota berúc do úvahy nedokonalosť v jeho pohybe.

Na Obrázku č.5 sú zobrazené odmeny prislúchajúce daným dlaždiciam (pozíciam) v sledovanej miestnosti. Poznamenajme, že odmeny týkajúce sa pohybu robota po miestnosti (t.j. náklady na pohyb) nie sú v tomto obrázku zobrazené.



Obr. 5: Odmeny prislúchajúce konkrétnym dlaždiciam v sledovanej miestnosti

2.3 Formulácia v štandardnom tvare úloh OR

Keďže diplomová práca je zameraná na porovnanie metód dynamického programovania (metódy OR) a algoritmov reinforcement learningu, pokúsime sa naformulovať úlohu o prechode robota v miestnosti (popísanú v Časti 2.1) v štandardnom tvare OR (viď[7]).

Úloha o prechode robota v miestnosti má ako úloha OR nasledujúci tvar:

$$
\begin{aligned}
\max_{h(.)} \quad & \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r(x_i, u_i, z_i)\right], \\
x_{i+1} &= f(x_i, u_i), \\
z_{i+1} &= z_i + g(x_i), \\
x_0 &= 9, \\
z_0 &= 0, \\
x_\infty &\in \{4, 8\}, \\
x_i &\in \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12\}, \\
u_i &\in \{H, D, P, L\},
\end{aligned}
\tag{11}
$$

kde $h(.)$ predstavuje stratégiu tak ako je popísaná v Časti 1.3.2. Možno ju chápať ako postupnosť akcií u_i, keďže platí, že $h(x_i) = u_i$.

Premenná z_i (viď (11)) predstavuje pomocnú premennú, ktorá hovorí o tom, v koľkom časovom kroku i v poradí sa nachádza robot v terminálnom stave. Na začiatku má táto premenná hodnotu 0, keďže počiatočný stav $x_0 = 9$, t.j. nie je terminálny. Funkciu $g(x_i)$ využívanú pre aktualizovanie hodnoty premennej z_i definujeme nasledovne:

$$g(x_i) = \begin{cases} 1 & , \text{ak } x_i = 4 \vee x_i = 8, \\ 0 & , \text{inak.} \end{cases} \tag{12}$$

Funkciu odmien $r(x_i, u_i, z_i)$ (viď (11)) definujeme nasledovne:

$$r(x_i, u_i, z_i) = \begin{cases} 0 & , \text{ak } x_i = x_0 \vee z_i > 1, \\ -0,07 + 1 & , \text{ak } x_i = 4 \wedge z_i \leq 1, \\ -0,07 - 1 & , \text{ak } x_i = 8 \wedge z_i \leq 1, \\ -0,07 & , \text{inak.} \end{cases} \tag{13}$$

Poznamenajme, že funkcia odmien definovaná vzťahom (13) predstavuje špeciálny tvar funkcie odmien, pri ktorom hodnota odmeny nezáleží na akcii u_i vykonanej v stave x_i a ani na nasledujúcom stave x_{i+1}. Takýto tvar funkcie odmien sme mohli skonštruovať na základe toho, že náklady robota na pokus o presun sú v každom prípade *0,07*.

Funkciu vystupujúcu v stavovej rovnici v zápise (11) definujeme nasledujúcim spôsobom:

$$f(x_i, H) = \begin{cases} x_i & , \text{ak } x_i \in \{4, 8\} \text{ s pravdepodobnosťou 1,} \\ & , \text{ak } x_i = 1 \text{ s pravdepodobnosťou 0,9,} \\ & , \text{ak } x_i \in \{2, 3, 10\} \text{ s pravdepodobnosťou 0,8,} \\ & , \text{ak } x_i = 5 \text{ s pravdepodobnosťou 0,2,} \\ & , \text{ak } x_i \in \{7, 9, 12\} \text{ s pravdepodobnosťou 0,1,} \\ x_i - 4 & , \text{ak } x_i \in \{5, 7, 9, 11, 12\} \text{ s pravdepodobnosťou 0,8,} \\ x_i + 1 & , \text{ak } x_i \in \{1, 2, 3, 7, 9, 10, 11\} \text{ s pravdepodobnosťou 0,1,} \\ x_i - 1 & , \text{ak } x_i \in \{2, 3, 10, 11, 12\} \text{ s pravdepodobnosťou 0,1,} \end{cases} \tag{14}$$

$$f(x_i, D) = \begin{cases} x_i & , \text{ak } x_i \in \{4, 8\} \text{ s pravdepodobnosťou 1,} \\ & , \text{ak } x_i \in \{9, 12\} \text{ s pravdepodobnosťou 0,9,} \\ & , \text{ak } x_i \in \{2, 10, 11\} \text{ s pravdepodobnosťou 0,8,} \\ & , \text{ak } x_i = 5 \text{ s pravdepodobnosťou 0,2,} \\ & , \text{ak } x_i \in \{1, 7\} \text{ s pravdepodobnosťou 0,1,} \\ x_i + 4 & , \text{ak } x_i \in \{1, 3, 5, 7\} \text{ s pravdepodobnosťou 0,8,} \\ x_i + 1 & , \text{ak } x_i \in \{1, 2, 3, 7, 9, 10, 11\} \text{ s pravdepodobnosťou 0,1,} \\ x_i - 1 & , \text{ak } x_i \in \{2, 3, 10, 11, 12\} \text{ s pravdepodobnosťou 0,1,} \end{cases} \tag{15}$$

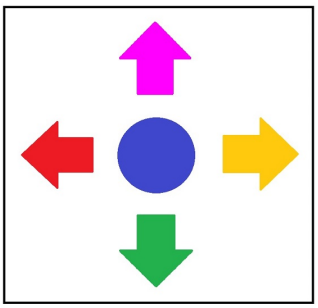
$$f(x_i, P) = \begin{cases} x_i & , \text{ak } x_i \in \{4, 8\} \text{ s pravdepodobnosťou 1,} \\ & , \text{ak } x_i = 12 \text{ s pravdepodobnosťou 0,9,} \\ & , \text{ak } x_i = 5 \text{ s pravdepodobnosťou 0,8,} \\ & , \text{ak } x_i \in \{2, 10\} \text{ s pravdepodobnosťou 0,2,} \\ & , \text{ak } x_i \in \{1, 3, 9, 11\} \text{ s pravdepodobnosťou 0,1,} \\ x_i - 4 & , \text{ak } x_i \in \{5, 7, 9, 11, 12\} \text{ s pravdepodobnosťou 0,1,} \\ x_i + 4 & , \text{ak } x_i \in \{1, 3, 5, 7\} \text{ s pravdepodobnosťou 0,1,} \\ x_i + 1 & , \text{ak } x_i \in \{1, 2, 3, 7, 9, 10, 11\} \text{ s pravdepodobnosťou 0,8,} \end{cases} \tag{16}$$

$$f(x_i, L) = \begin{cases} x_i & , \text{ak } x_i \in \{4, 8\} \text{ s pravdepodobnosťou 1,} \\ & , \text{ak } x_i \in \{1, 9\} \text{ s pravdepodobnosťou 0,9,} \\ & , \text{ak } x_i \in \{5, 7\} \text{ s pravdepodobnosťou 0,8,} \\ & , \text{ak } x_i \in \{2, 10\} \text{ s pravdepodobnosťou 0,2,} \\ & , \text{ak } x_i \in \{3, 11, 12\} \text{ s pravdepodobnosťou 0,1,} \\ x_i - 4 & , \text{ak } x_i \in \{5, 7, 9, 11, 12\} \text{ s pravdepodobnosťou 0,1,} \\ x_i + 4 & , \text{ak } x_i \in \{1, 3, 5, 7\} \text{ s pravdepodobnosťou 0,1,} \\ x_i - 1 & , \text{ak } x_i \in \{2, 3, 10, 11, 12\} \text{ s pravdepodobnosťou 0,8.} \end{cases} \tag{17}$$

Pre lepšie pochopenie funkcie $f(x_i, u_i)$ sú v nasledujúcej časti vizualizované pravdepodobnosti prechodu pri daných akciách H, D, P, L pomocou obrázkov. Na každom z týchto obrázkov (t.j. Obrázok č.7 - č.10) predstavujú dané farby nasledujúce pravdepodobnosti prechodu:

- *modrá* - pravdepodobnosť, že sa robot pri zvolenej akcii u_i nepohne (t.j. $x_{i+1} = x_i$),
- *ružová* - pravdepodobnosť, že sa robot pri zvolenej akcii u_i pohne o jednu dlaždicu hore (t.j. $x_{i+1} = x_i - 4$),
- *zelená* - pravdepodobnosť, že sa robot pri zvolenej akcii u_i pohne o jednu dlaždicu dole (t.j. $x_{i+1} = x_i + 4$),
- *žltá* - pravdepodobnosť, že sa robot pri zvolenej akcii u_i pohne o jednu dlaždicu vpravo (t.j. $x_{i+1} = x_i + 1$),
- *červená* - pravdepodobnosť, že sa robot pri zvolenej akcii u_i pohne o jednu dlaždicu vľavo (t.j. $x_{i+1} = x_i - 1$).

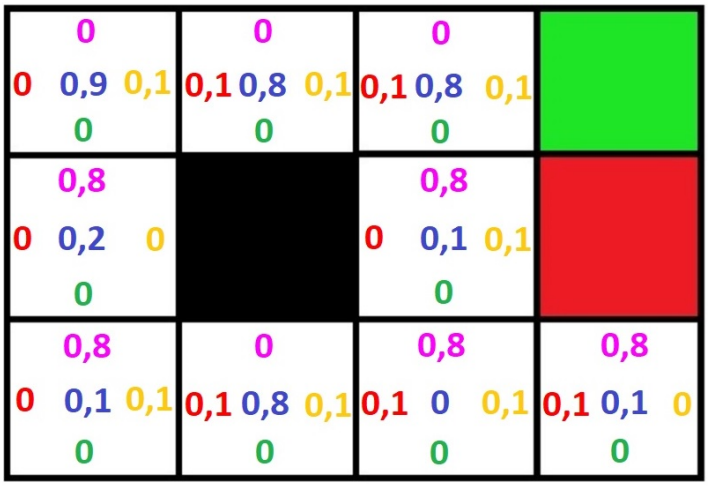
Na Obrázku č.6 sú zobrazené vyššie popísané farby prislúchajúce daným smerom.



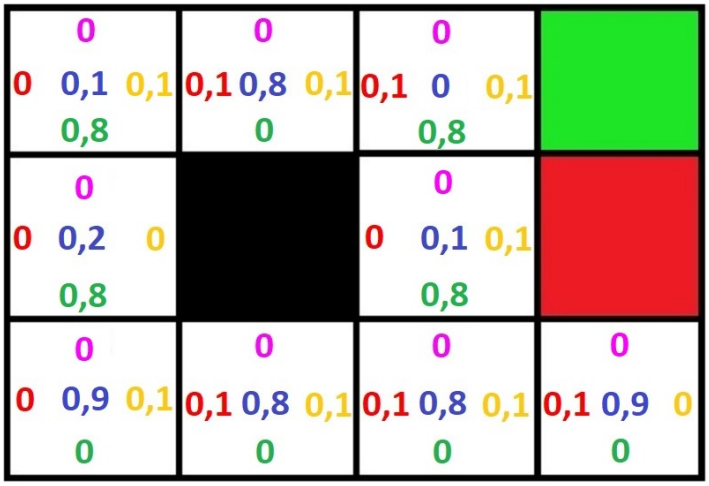
Obr. 6: Farby prislúchajúce konkrétnym smerom vo vizualizácii pravdepodobností prechodu pre úlohu popísanú v (11)

Na Obrázku č.7 sú zobrazené pravdepodobnosti prechodu, ak agent vykoná akciu H - *hore*, t.j. $u_i = H$.

Na nasledujúcom obrázku, teda Obrázku č.8, sú zobrazené pravdepodobnosti prechodu, ak agent vykoná akciu D - *dole*, t.j. $u_i = D$.



Obr. 7: Vizualizácia pravdepodobností prechodu pre úlohu popísanú v (11), ak agent zvolí akciu *H - hore*

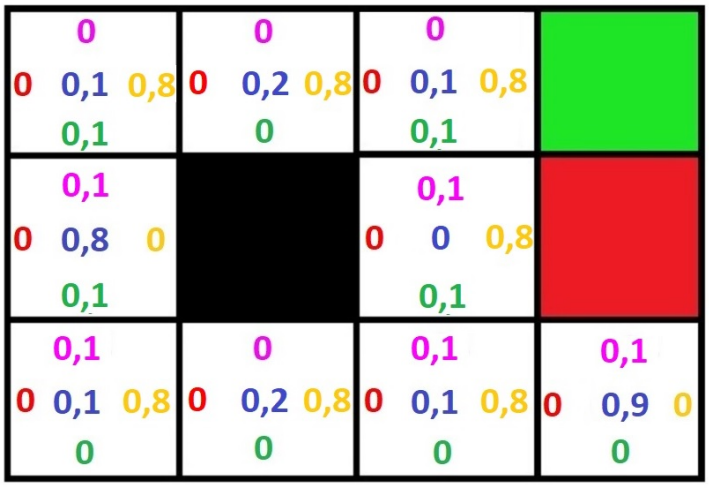


Obr. 8: Vizualizácia pravdepodobností prechodu pre úlohu popísanú v (11), ak agent zvolí akciu *D - dole*

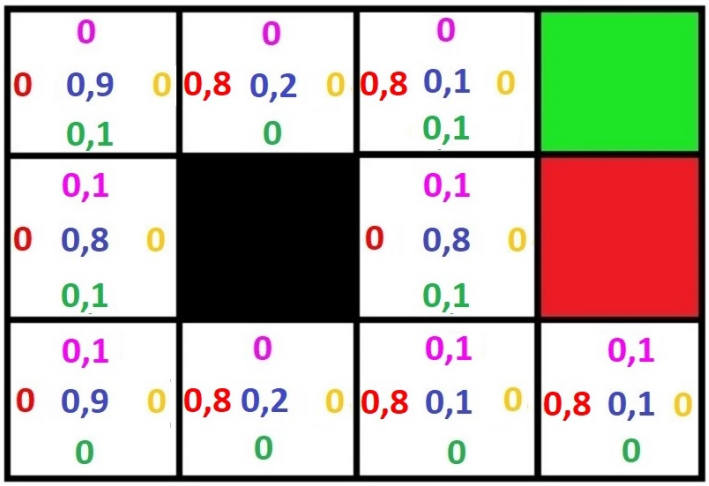
Na ďalšom obrázku, teda Obrázku č.9, sú zobrazené pravdepodobnosti prechodu, ak agent vykoná akciu P - *vpravo*, t.j. $u_i = P$.

Na finálnom obrázku, teda Obrázku č.10, sú zobrazené pravdepodobnosti prechodu, ak agent vykoná akciu L - *vľavo*, t.j. $u_i = L$.

Poznamenajme, že na Obrázkoch č.7 - č.10 na dlaždiciach č.4 a č.8 (t.j. zelená dlaždica reprezentujúca nabíjaciu stanicu a červená dlaždica prislúchajúca schodiku) nie sú uvedené pravdepodobnosti prechodu z dôvodu, že stav 4 rovnako ako stav 8 ($x_i = 4$, $x_i = 8$) sú



Obr. 9: Vizualizácia pravdepodobností prechodu pre úlohu popísanú v (11), ak agent zvolí akciu *P - vpravo*



Obr. 10: Vizualizácia pravdepodobností prechodu pre úlohu popísanú v (11), ak agent zvolí akciu *L - vľavo*

konečné (terminálne) stavy. To znamená, že pravdepodobnosť, že sa robot z takého stavu dostane je 0.

Diskontný faktor v zápise (11) reprezentuje parameter γ. Poznamenajme, že v prípade takto zapísanej úlohy, t.j. v tvare úlohy na nekonečnom časovom horizonte, je potrebné, aby sa pri riešení pomocou metód OR diskontný faktor volil ako $\gamma < 1$. A to z dôvodu, aby nedošlo k divergovaniu maximalizovanej sumy.

3 Prehľad metód reinforcement learningu

V tejto kapitole uvedieme prehľad a všeobecnú klasifikáciu metód (resp. algoritmov) RL, ktoré boli vytvorené na riešenie problémov popísateľných pomocou markovovského rozhodovacieho procesu. Zároveň predstavíme šesť konkrétnych vybraných RL algoritmov, uvedieme ich schémy a následne tieto metódy implementujeme na vzorový RL problém popísaný v Kapitole 2.

3.1 Základné delenie algoritmov RL

Vychádzajúc z [23] existujú dve základné kategórie, do ktorých môžeme reinforcement learning algoritmy rozdeliť. Prvým dôležitým parametrom RL algoritmu je to, v akom rozsahu pozná agent prostredie, v ktorom vykonáva akcie. Z matematického hľadiska je možné popísať tento ukazovateľ ako informáciu o modeli. Druhým parametrom je čas, v ktorom agent interaguje so systémom. Na základe uvedených kritérií môžeme algoritmy RL rozdeliť do dvoch nasledujúcich skupín:

- **Kategórie algoritmov RL na základe množstva informácie o modeli, ktorý sa v nich využíva**:

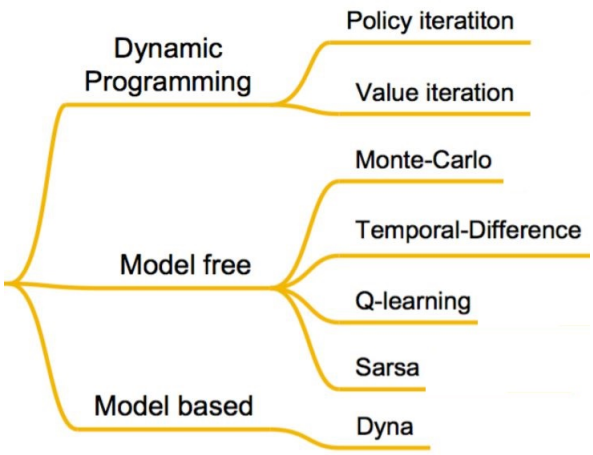
  1. *Algoritmy založené na modeli* (model-based algoritmy) - funkcia prechodu f a funkcia odmien r sú vopred známe,
  2. *Algoritmy bez informácie o modeli* (model-free algoritmy) - funkcia prechodu f a funkcia odmien r nie sú vopred známe,
  3. *Algoritmy, ktoré sa postupne model učia* (model-learning algoritmy) - funkcia prechodu f a funkcia odmien r sa získavajú postupne z dát.

  Poznamenajme, že medzi algoritmy založené na modeli patria aj vhodne modifikované algoritmy dynamického programovania (skrátene DP) z teórie optimálneho riadenia, i keď v mnohých publikáciach týkajúcich sa RL tvoria algoritmy DP samostatnú kategóriu (viď [27]).

- **Kategórie algoritmov RL na základe času, kedy agent interaguje so systémom**:

1. *Offline algoritmy* - algoritmy, ktoré zbiehajú vopred,
2. *Online algoritmy* - algoritmy, ktoré zbiehajú zároveň so systémom.

Na Obrázku č.11 pochádzajúceho z [27] sú uvedené konkrétne reinforcement learning algoritmy zaradené do vyššie spomenutých kategórií.



Obr. 11: Delenie RL algoritmov na riešenie MRP

3.2 Dynamické programovanie

Najprv popíšeme aplikovanie metód dynamického programovania. Je nutné poznamenať, že tieto metódy boli z historického hľadiska vyvinuté na riešenie MRP skôr ako RL algoritmy. Metódy DP sú dobrým stavebným kameňom pre pochopenie samotných algoritmov reinforcement learningu, keďže RL metódy sú odvodené práve z metód dynamického programovania. Metódy DP sú špecifické tým, že k nájdeniu riešenia potrebujú presný popis rozhodovacieho procesu. To znamená, že je potrebné, aby boli známe všetky pravdepodobnosti prechodu.

Poznamenajme, že metódy reinforcement learningu vychádzajú z Bellmanovou rovnicou dynamického programovania (skrátene RDP).

Ako sme už uviedli, z hľadiska kategorizácie uvedenej v Časti 3.1 je možné metódy dynamického programovania klasifikovať ako algoritmy založené na modeli, keďže k riešeniu potrebujú poznať presný model sledovaného systému. Zaraďujeme ich taktiež medzi offline algoritmy, teda optimálna stratégia je získaná vopred z modelu, nie z dát získaných

interakciou s prostredím. V tejto diplomovej práci sa zameriame na dve najznámejšie metódy DP určené na riešenie MRP, a to tzv. *iterovanie stratégií* (policy iteration) a tzv. *iterovanie hodnôt* (value iteration).

3.2.1 Bellmanova rovnica dynamického programovania

Jedným zo základných konceptov dynamického programovania je tzv. *Bellmanov princíp optimality* uvedený v [7]. Jeho dôležitou súčasťou je už spomínaná Bellmanova rovnica dynamického programovania. V tejto časti ukážeme, že Bellmanova RDP je vhodná (a platí) aj pre prácu s Q-funkciou, resp. s V-funkciou. Poznamenajme, že teóriu Bellmanovej RDP predstavíme pre stochastickú úlohu, kde náhodnou premennou je kvôli náhodnosti spôsobenej prostredím stav x (stavová premenná) a tým pádom aj veľkosť odmien $r(x, u, x')$.

V prvom kroku uvedieme alternatívny tvar Q-funkcie ku tvaru uvedenému v (7). A to konkrétne tvar Q-funkcie, pri ktorej sa v stave x_0 vykonala akcia u_0 a následne sa aplikovala stratégia h. A teda:

$$Q^h(x_0, u_0) = E_{x_1}\left[r(x_0, u_0, x_1) + \gamma R_1)\right], \tag{18}$$

kde R_1 pochádza zo vzťahu (2).

Bellmanovu RDP pre takýto tvar Q-funkcie odvodíme nasledujúcim spôsobom:

$$\begin{aligned} Q^h(x_0, u_0) &= E_{x_1}\left[r(x_0, u_0, x_1) + \gamma R^h(x_1)\right] \\ &= E_{x_1}\left[r(x_0, u_0, x_1) + \gamma E_{x_2}\left[r(x_1, h(x_1), x_2) + \gamma R^h(x_2)\right]\right] \\ &= E_{x_1}\left[r(x_0, u_0, x_1) + \gamma Q^h(x_1, h(x_1))\right]. \end{aligned} \tag{19}$$

V prvom kroku sme pomocou rozpísania dlhodobého kumulatívneho súčtu očakávaných odmien dokázali vyjadriť Q-funkciu $Q^h(x_0, u_0)$ pomocou Q-funkcie $Q^h(x_1, h(x_1))$. Možno teda jednoducho nahliadnuť, že analogicky dokážeme vyjadriť Q-funkciu $Q^h(x_1, u_1)$ pomocou Q-funkcie $Q^h(x_2, h(x_2))$. A teda:

$$Q^h(x_1, u_1) = E_{x_2}\left[r(x_1, u_1, x_2) + \gamma Q^h(x_2, h(x_2))\right]. \tag{20}$$

Použitím princípu matematickej indukcie sme tak schopní dostať rovnicu (založenú na princípe Bellmanovej RDP) v nasledujúcom tvare:

$$Q^h(x,u) = E_{x'}\left[r(x,u,x') + \gamma Q^h(x', h(x'))\right]. \tag{21}$$

Pre aplikovanie tejto rovnice na konkrétne typy príkladov je však vhodné ju používať v upravenom tvare. A to:

$$\begin{aligned} Q^h(x,u) &= E_{x'}\left[r(x,u,x') + \gamma Q^h(x', h(x'))\right] \\ &= \sum_{x'} f(x,u,x')\left[r(x,u,x') + \gamma Q^h(x', h(x'))\right], \end{aligned} \tag{22}$$

kde $f(x,u,x')$ predstavuje funkciu prechodu a $r(x,u,x')$ funkciu odmien ako sme ich definovali v Podkapitole 1.3.1.

Bellmanova RDP má analogický tvar aj pre V-funkciu. A to konkrétne:

$$V^h(x) = \sum_{x'} f(x,u,x')\left[r(x,u,x') + \gamma V^h(x')\right]. \tag{23}$$

Na nájdenie optimálneho riešenia teda stačí použiť vyššie spomenutý Bellmanov princíp optimality, podľa ktorého nezáleží na predchádzajúcich akciách, ale iba na budúcich akciách, pri ktorých je potrebné sledovať optimálnu stratégiu. Optimálna Q-funkcia (resp. V-funkcia) je teda daná nasledujúcim spôsobom:

$$\begin{aligned} Q^*(x,u) &= \max_{h(.)} Q^h(x,u) \\ &= \sum_{x'} f(x,u,x')\left[r(x,u,x') + \gamma \max_{u'} Q^*(x',u')\right], \end{aligned} \tag{24}$$

$$\begin{aligned} V^*(x) &= \max_{u \in U(x)} Q^{h^*}(x,u) \\ &= \max_{u \in U(x)} \sum_{x'} f(x,u,x')\left[r(x,u,x') + \gamma V^*(x')\right], \end{aligned} \tag{25}$$

kde $Q^{h^*}(x,u)$ predstavuje hodnoty Q-funkcie za aplikovania optimálnej stratégie h^* pre všetky páry stav-akcia.

Je dôležité poznamenať, že pre metódy DP je spomínaná znalosť systému (resp. modelu) nutná práve kvôli použitiu vzťahov (24) a (25). Na nich je možné pozorovať to, že výber optimálnej akcie u v terajšom stav x je závislý na výbere optimálne akcie v nasledujúcom stav x'. Bez znalosti modelu by nebolo možné zistiť, aký bude konkrétne nasledujúci stav x' a ani odmena $r(x, u, x')$, keďže ako sme už spomínali vyššie, stav rovnako ako veľkosť odmien možno považovať za náhodné.

3.2.2 Iterovanie stratégií

Ako sme už spomínali v predchádzajúcej časti práce, jednou z metód DP je metóda iterovania stratégií. Vychádzajúc z [19] a [20] táto metóda pozostáva z dvoch samostatných procesov: tzv. *vyhodnotenie stratégie* (z anglického policy evaluation) a tzv. *vylepšenie stratégie* (z anglického policy improvement).

Pri tejto metóde sa najprv zvolí počiatočná stratégia $h_0(x)$ pre každý stav $x \in X$. Zvolená počiatočná stratégia musí byť asymptoticky stabilná.

Definícia 3.1. *([18]) Za asymptoticky stabilný bod označujeme taký bod, pre ktorý platí, že ak ho zvolíme za počiatočný bod, tak sledovaný systém skonverguje do stabilného bodu.*

Následne sú pomocou Bellmanovej rovnice dynamického programovania v tvare (22) zistené príslušné hodnoty $Q^{h_l}(x, h_l(x))$ pre všetky stavy $x \in X$, kde index l predstavuje poradie stratégie. Tým je ukončený proces vyhodnotenia danej stratégie. Vďaka stabilite stratégie sú tieto hodnoty Q-funkcie konečné.

V ďalšom kroku je potrebné vykonať vylepšenie sledovanej stratégie. Proces vylepšenia stratégie spočíva v tom, že pre každý stav $x \in X$ je nájdená taká akcia u', pre ktorú platí, že:

$$u' = arg \max_{u \in U(x)} Q^{h_l}(x, u), \tag{26}$$

kde množina $U(x)$ predstavuje množinu možných akcií vykonateľných v stave x. Po ukončení tohto kroku je nájdená nová stratégia $h_{l+1}(x)$. Vychádzajúc z [23] vieme, že stratégie $h_{l+1}(x)$, ktoré sú získané takouto iteráciou stratégií, za určitých podmienok konvergujú k optimálnej stratégii $h^*(x)$.

Algoritmus iterovania stratégií má nasledujúcu schému:

Algoritmus 1: *Offline iterovanie stratégií*

Vstupné dáta: model sledovaného systému, počiatočná stratégia $h_0(x)$, diskontný faktor γ.

Cyklus: pre $i = 0, 1, 2, ...$

- **Vyhodnotenie stratégie:** Pre všetky stavy $x \in X$ a všetky akcie $u \in U$ (využijúc index i) nájdeme podľa vzťahu (22) hodnoty Q-funkcie $Q^{h_l}(x, u)$ za aplikovania (resp. nasledovania) stratégie $h_l(x)$:

$$Q^{h_l}(x_i, u_i) = r(x_i, u_i, x_{i+1}) + \gamma Q^{h_l}(x_{i+1}, u_{i+1}).$$

- **Vylepšenie stratégie:** Nájdeme novú stratégiu h_{l+1} nasledujúcim spôsobom:

$$h_{l+1}(x) = arg \max_{h(.)} Q^{h_l}(x, h(x)).$$

- Opakujeme kroky *Vyhodnotenie stratégie* a *Vylepšenie stratégie* dokiaľ neplatí, že $h_l \approx h_{l+1}$. To znamená, dokým hodnoty stratégie neskonvergujú.

Výsledok: Optimálna stratégia $h^*(x)$.

Poznamenajme, že krok *Vyhodnotenia stratégie* použitím Bellmanovej RDP tak ako je uvedený v Algoritme 1 je častokrát výpočtovo a pamäťovo náročný. A to z dôvodu, že sa v ňom rieši sústava rovníc. Podľa [23] však vieme, že Bellmanova RDP je rovnica s pevným bodom. To pre nás znamená, že krok vyhodnotenia stratégie možno vykonať aj iteračne. Na začiatku je zvolené počiatočné ohodnotenie $Q_0^{h_l}(x, h_l(x))$. V každej iterácii sa v krokoch $j = 0, 1, 2, ...$ vykoná ohodnotenie Q-funkcie $Q_j^{h_l}(x, h_l(x))$, ktoré konverguje k hľadanej hodnote $Q^{h_l}(x, h_l(x))$. Je dôležité poznamenať, že konvergencia k hľadanej hodnote $Q^{h_l}(x, h_l(x))$ je garantovaná asymptoticky, t.j. pre $j \to \infty$. Krok vyhodnotenia stratégie môže byť teda nahradený nasledujúcim vzťahom:

$$Q_{j+1}(x_i, u_i) = r(x_i, u_i, x_{i+1}) + \gamma Q_j(x_{i+1}, u_{i+1}) \qquad pre \quad j = 1, 2, 3, ... \tag{27}$$

Vzťah (27) budeme označovať ako tzv. *iteračné vyhodnotenie stratégie.*

Pre praktické účely môže byť algoritmus zastavený práve vtedy, keď

$$||Q_{j+1}^{h_l} - Q_j^{h_l}|| \leq \epsilon, \tag{28}$$

kde j predstavuje index iterácie vyhodnocovania sledovanej stratégie h_l. Stratégiu h_l teda považujeme za vyhodnotenú vtedy, keď sú hodnoty Q-funkcie pri dvoch za sebou idúcich iteráciach vyhodnocovania sledovanej stratégie zhodné, resp. blízko seba v dostatočnom rozsahu.

3.2.3 Aplikácia metódy iterovania stratégií na vzorový problém

Metódu iterovania stratégií, tak ako je popísaná v Časti 3.2.2, sme aplikovali na vzorový problém o pohybe robota v miestnosti popísaný v Kapitole 2. Poznamenajme, že všetky predstavené výpočtové algoritmy boli na zvolený problém implementované autorkou v systéme `Matlab` a sú uvedené v Prílohe A. Pre aplikovanie Algoritmu 1 sme zvolili nasledujúce konkrétne hodnoty parametrov:

- počiatočná stratégia je $h_0(x) = \{H, H, H, -, H, -, H, -, H, H, H, H\}$, t.j. robot volí akciu *Hore* bez ohľadu na stav, v ktorom sa nachádza. Poznamenajme, že táto počiatočná stratégia môže byť zvolená ľubovoľne, môžeme ju aj priamo generovať z ľubovoľného rozdelenia. My sme zvolili presnú počiatočnú stratégiu h_0 z dôvodu urýchlenia programu,

- pri implementovaní tohto algoritmu sme používali metódu iteračného vyhodnotenie stratégie,

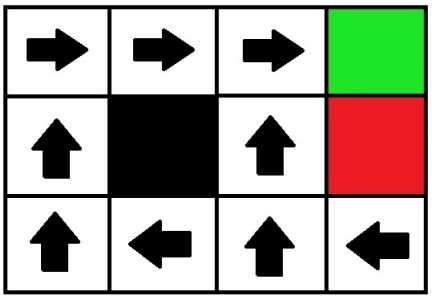
- hodnota diskontného faktora je $\gamma = 1$ (t.j. hodnoty Q-funkcie nie sú v čase diskontované),

- počiatočná hodnota Q-funkcie pri každom iteračnom vyhodnotení stratégie (t.j. pre $j = 0$) je $Q_0^{h_l}(x, u) = 0$. Poznamenajme, že počiatočné hodnoty Q-funkcie sme zvolili pre všetky páry stav-akcia rovnaké z dôvodu, aby nebol žiaden pár stav-akcia zvýhodnený a zároveň sme túto rovnakú hodnotu Q-funkcie zvolili ako 0, aby bolo ľahšie odčítateľné, pre ktoré páry stav-akcia sú hodnoty Q-funkcie pozitívne a pre ktoré sú negatívne,

- model (t.j. funkcia prechodu a funkcia odmien), ktorý je použitý v programe, je popísaný v Časti 2.3,
- požadovaná presnosť (pre zastavenie iterovania Q-funkcie pri iteračnom vyhodnocovaní každej stratégie) je $\epsilon = 10^{-3}$.

Aplikujúc metódu iterovania stratégií sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{P, P, P, -, H, -, H, -, H, L, H, L\}$, jej vizualizáciu v sledovanej miestnosti (mriežke) možno sledovať na Obrázku č.12,



Obr. 12: Vizualizácia riešenia vzorového problému pomocou metódy iterovania stratégií

- počas algoritmu bolo pred nájdením optimálnej stratégie vyhodnotených 5 rôznych stratégií (vrátane stratégie $h_0(x)$ a optimálnej stratégie $h^*(x)$), t.j. $l = 5$, a to konkrétne:
  1. $h_0(x) = \{H, H, H, -, H, -, H, -, H, H, H, H\}$,
  2. $h_1(x) = \{P, P, P, -, H, -, P, -, P, P, P, H\}$,
  3. $h_2(x) = \{P, P, P, -, H, -, H, -, H, L, H, H\}$,
  4. $h_3(x) = \{P, P, P, -, H, -, H, -, H, L, L, L\}$,
  5. $h_4(x) = \{P, P, P, -, H, -, H, -, H, L, H, L\}$,
- celkový čas výpočtu bol $t \approx 0,433\ sek$.

3.2.4 Iterovanie hodnôt

Druhou z metód dynamického programovania je tzv. *iterovanie hodnôt*. Vychádzajúc z [21] a [20] táto metóda spočíva v priamom nájdení optimálnej ohodnocovacej funkcie. Z hodnôt optimálnej ohodnocovacej funkcie sme vo finálnom kroku algoritmu schopní získať informáciu o optimálnej stratégii $h^*(x)$, ktorá je v konečnom dôsledku cieľom algoritmu.

V počiatočnom kroku sú pre všetky páry stav-akcia zvolené ľubovoľné hodnoty Q-funkcie, t.j. $Q_0(x, u)$. Následne je opäť využitá Bellmanova rovnica dynamického programovania (22), pomocou ktorej sú získané nové hodnoty Q-funkcie nasledovným spôsobom:

$$Q_{j+1}(x_i, u_i) = r(x_i, u_i, x_{i+1}) + \gamma \max_{u_{i+1}} Q_j(x_{i+1}, u_{i+1}). \tag{29}$$

Túto rovnicu je možné zapísať aj za využitia V-funkcie. A teda:

$$Q_{j+1}(x_i, u_i) = r(x_i, u_i, x_{i+1}) + \gamma V_j(x_{i+1}). \tag{30}$$

Pre účely tejto diplomovej práce budeme pracovať s vyššie uvedenou rovnicou zo vzťahu (29).

Vychádzajúc z [23] možno konštatovať, že hodnoty Q-funkcie $Q_{j+1}(x, u)$ konvergujú k optimálnej hodnote $Q^*(x, u)$. V poslednom kroku metódy iterovania hodnôt je potrebné nájsť optimálnu stratégiu $h^*(x)$. To vykonáme pomocou vzťahu (9).

Algoritmus iterovania hodnôt má teda nasledujúci tvar:

Algoritmus 2: *Offline iterovanie hodnôt*

Vstupné dáta: model sledovaného systému, počiatočné hodnoty $Q_0(x, u)$, diskontný faktor γ.

Cyklus: pre $j = 0, 1, 2, ...$

- **Vylepšenie ohodnotenia:** Pre všetky páry stav-akcia (využijúc index i) nájdeme hodnoty Q-funkcie $Q_{j+1}(x_i, u_i)$ využitím vzťahu:

$$Q_{j+1}(x_i, u_i) = r(x_i, u_i, x_{i+1}) + \gamma \max_{u_{i+1}} Q_j(x_{i+1}, u_{i+1}).$$

- Opakujeme krok vylepšenie ohodnotenia dokiaľ neplatí, že $Q_j \approx Q_{j+1}$. To znamená, dokým hodnoty Q-funkcie neskonvergujú.

Nájdenie optimálnej stratégie: Vieme, že platí $Q_{j+1}(x,u) = Q^*(x,u)$. Preto možno určiť optimálnu stratégiu $h^*(x)$ nasledujúcim spôsobom:

$$h^*(x) = arg \max_{u \in U(x)} Q^*(x,u),$$

kde $U(x)$ predstavuje množinu možných akcií vykonateľných v stave x.

Výsledok: Optimálna stratégia $h^*(x)$.

Pre praktické účely môže byť algoritmus zastavený práve vtedy, keď

$$||Q_{j+1} - Q_j|| \leq \epsilon, \tag{31}$$

analogicky k zastaveniu algoritmu iterovania stratégií popísaného vzťahom (28).

3.2.5 Aplikácia metódy iterovania hodnôt na vzorový problém

Metódu iterovania hodnôt, tak ako je popísaná v Časti 3.2.4, sme taktiež aplikovali na vzorový problém o pohybe robota v miestnosti popísaný v Kapitole 2. Pre aplikovanie Algoritmu 2 sme zvolili nasledujúce konkrétne hodnoty parametrov:

- hodnota diskontného faktora je $\gamma = 1$ (t.j. hodnoty Q-funkcie nie sú v čase diskontované),
- počiatočné hodnoty Q-funkcie (t.j. pre $j = 0$) je $Q_0(x,u) = 0$. Poznamenajme, že počiatočné hodnoty Q-funkcie sme volili s rovnakou myšlienkou ako v Časti 3.2.3,
- model (t.j. funkcia prechodu a funkcia odmien), ktorý je použitý v programe, je popísaný v Časti 2.3,
- požadovaná presnosť (pre zastavenie iterovania Q-funkcie) je $\epsilon = 10^{-3}$.

Aplikujúc metódu iterovania hodnôt sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{P, P, P, -, H, -, H, -, H, L, H, L\}$, (t.j. výsledok je zhodný s výsledkom v Časti 3.2.3),
- konvergencia nastala po $j = 21$ iteráciach (vrátane nultej iterácie). Postupné hodnoty Q-funkcie pre dané akcie sú uvedené v Prílohe B,
- celkový čas výpočtu bol $t \approx 0,343\ sek$.

3.3 Metódy temporálnych diferencií

Metódy dynamického programovania so sebou nesú signifikantnú nevýhodu, ktorou je nutná znalosť presného modelu sledovaného systému. Zároveň metódy DP sú offline metódami, teda zbierajú vopred (t.j. nie zároveň so systémom) a sú schopné vypočítať optimálne riešenie daného problému len na základe vopred známych informácií, a teda už spomínaného modelu systému. V tejto podkapitole predstavíme jednu zo základných myšlienok reinforcement learningu, a to využitie tzv. *temporálnych diferencií* (skrátene TD). Tieto metódy možno kategorizovať (vzhľadom na delenie uvedené v Časti 3.1) ako algoritmy bez informácie o modeli. Dôvodom je fakt, že agent sa pri týchto metódach učí o sledovanom systéme postupne (t.j. presný model nie je na začiatku známy). Optimálna stratégia je teda zistená iba na základe dát pozorovaných z prostredia. Zároveň metódy TD spadajú medzi online algoritmy. To znamená, že algoritmy sa „učia" v každom časovom kroku. Vďaka tomu je možné nájsť optimálne riešenie (resp. stratégiu), ktoré bude dostatočne adaptabilné.

Je dôležité poznamenať, že s využitím temporálnych diferencií sme schopní implementovať metódu iterovania stratégií rovnako ako metódu iterovania hodnôt ako online metódy. V tejto podkapitole sa zameriame konkrétne na metódy *SARSA* a *Q-učenie*, i keď v teórii reinforcement learningu sa môžeme stretnúť aj s metódami *TD(0), TD(1)* a $TD(\lambda)$ ([4], [17]). Poznamenajme, že v tejto časti diplomovej práce budeme pracovať s ohodnotením pomocou Q-funkcie, keďže používanie V-funkcie v tomto prípade so sebou prináša určité obmedzenia a nepraktickosť.

3.3.1 Princíp TD

Znalosť presného modelu systému pri metódach DP bola nutná najmä pre to, aby sme boli schopní nájsť hodnoty Q-funkcie $Q^h(x,u)$ pre všetky stavy $x \in X$, respektíve aby sme boli schopní vyhodnotiť danú stratégiu $h(x)$. Ako sme spomínali vyššie, metódy temporálnych diferencií pracujú s pozorovanými (nameranými) dátami a nepoužívajú presný model systému. Aby bol algoritmus založený na TD schopný naučiť sa hodnoty Q-funkcie (resp. aby bol schopný vyhodnotiť stratégiu) je potrebné, aby v stave x bola vykonaná akcia u.

Myšlienka metód temporálnych diferencií je teda nasledovná. Namiesto znalosti presných hodnôt funkcie $Q^h(x,u)$ pre všetky páry stav-akcia, stačí, ak budeme uchovávať informáciu iba o odhadoch týchto hodnôt. Hodnoty Q-funkcie pozostávajúcej z odhadov budeme označovať ako $Q^{\sim h}(x,u)$. Po vykonaní konkrétnej akcie u v danom stave x dostaneme vždy informáciu o odmene $r(x,u,x')$ a informáciu o hodnote funkcie $Q^{\sim h}(x',h(x'))$. V ďalšom kroku je na základe informácií o novozískaných hodnotách aktualizovaná pôvodná hodnota Q-funkcie založenej na odhade $Q^{\sim h}(x,u)$. Pri tomto aktualizovaní je dôležitý pojem tzv. *TD-chyby* (z anglického temporal difference error).

Definícia 3.2. *() TD-chybu v časovom kroku i, označme δ_i, definujeme ako rozdiel medzi predpokladanou hodnotou Q-funkcie založenej na odhadoch a jej novonadobudnutou hodnotou. Teda:*

$$\delta_i = [r(x_i,u_i,x_{i+1}) + \gamma Q^{\sim h}(x_{i+1},u_{i+1})] - Q^{\sim h}(x_i,u_i). \tag{32}$$

Vzťah (32) sa dá odvodiť zo vzťahu (22) pre $\delta_i = 0$. Vychádzajúc z [6], [23] vieme, že aktualizácia hodnôt Q-funkcie na základe temporálnych diferencií konverguje k presnej hodnote funkcie $Q^h(x,u)$. V takomto prípade bude teda platiť, že $Q^{\sim h}(x,u) = Q^h(x,u)$ a zároveň bude platiť to, že veľkosť TD-chyby je nulová, resp. blízka nule. V nasledujúcich častiach tejto kapitoly upustíme kvôli prehľadnosti od používania značenia $Q^{\sim h}(.)$.

3.3.2 Problém explorácie a exploitácie

Pri využití algoritmov, ktoré neobsahujú informáciu o modeli, vzniká problém v literatúre označovaný pojmom *rovnováha medzi exploráciou a exploitáciou* (z anglického exploration

vs. exploitation dilemma). Za exploráciu, respektíve objavovanie, sa považuje proces, pri ktorom agent skúša rôzne akcie, aby našiel neznáme odhady hodnôt $Q^h(x, u)$, prípadne aby tieto odhady vylepšil pre všetky páry stav-akcia. Agent tak získa nové skúsenosti (resp. informácie z prostredia), ktoré môžu viesť k nájdeniu nových (alebo lepších) riešení. Na druhej strane, exploitácia, respektíve využitie, je proces, pri ktorom sa využíva znalosť hodnôt $Q^h(x, u)$ pre maximalizáciu dlhodobého kumulatívneho súčtu očakávaných odmien. Ide o výber tzv. *greedy* (chamtivej) akcie vzhľadom k ohodnoteniu. Poznamenajme, že pri metódach dynamického programovania sú všetky hodnoty $Q^h(x, u)$ vypočítané na základe známeho modelu, preto sa v týchto metódach používa výhradne exploitácia. Pre metódy založené na temporálnych diferenciách je nutné tieto hodnoty nájsť priamym experimentom.

Nájsť rovnováhu medzi týmito dvoma procesmi je kľúčové pre metódy TD, rovnako ako aj iné metódy, ktoré nepracujú s plnou informáciou o modeli. V prípade, že by bola miera explorácie príliš vysoká, mohlo by sa stať, že by agent v dostatočnej miere nevyužil získané hodnoty. Konvergencia k optimálnemu riešeniu by tak mohla byť príliš pomalá. Mohlo by sa dokonca stať, že by k nej nemuselo dôjsť. Na druhej strane, v prípade že by si agent vyberal výhradne exploitáciu, mohlo by dôjsť k situácii, že by nájdené riešenie konvergovalo iba k tzv. *suboptimálnemu riešeniu* (vychádzajúc z [13]). Čo by bolo spôsobené tým, že by agent nemal dostatočnú informáciu, resp. dostatočné znalosti o prostredí.

Najčastejšie sa problém explorácie a exploitácie rieši použitím mechanizmu náhodnej selekcie. Jeho podstatou je to, že namiesto exploitačnej akcie sa s určitou pravdepodobnosťou vyberie iná (t.j. exploračná) akcia. Tým pádom sa stratégia určitým spôsobom rozšíri. Konkrétne namiesto presného mapovania zo stavu na akciu (tak že sa vyberie optimálna akcia vzhľadom na hodnotu Q-funkcie) sa do algoritmu pridá možnosť náhodného výberu.

Vychádzajúc z [23] sú najrozšírenejšie nasledujúce typy selekcie:

- **ϵ-greedy selekcia:** Najčastejším riešením pre výber náhodnej akcie je tzv. *ϵ-greedy selekcia*. Funguje na princípe toho, že s pravdepodobnosťou $(1-\epsilon)$ sa vyberie exploitačná akcia (greedy vzhľadom k ohodnoteniu) a s pravdepodobnosťou ϵ sa vyberie exploračná (t.j. náhodná) akcia. Teda:

$$u = \begin{cases} argmax_u Q(x,u) & \text{s pravdepodobnosťou } (1-\epsilon), \\ u \in U(x) & \text{s pravdepodobnosťou } \epsilon. \end{cases}$$

Poznamenajme, že pre praktické účely nezvykne mať parameter ϵ počas priebehu celého algoritmu konštantnú hodnotu, ale zvykne sa vyvíjať v závislosti od času (resp. časového kroku) i. Z toho dôvodu bude v neskorších častiach práce vystupovať označenie ϵ_{index}. Zvyčajne sa tento parameter volí na začiatku vysoký a postupne sa jeho veľkosť približuje nule. A to z dôvodu, že na začiatku algoritmu máme preskúmaných málo párov stav-akcia, ale čím viac takýchto explorácií vykonáme, tým menej je potrebné explorovať v priebehu algoritmu. Je jednoduché odčítať, že pred ukončením algoritmu už máme dostatočnú informáciu o všetkých pároch stav-akcia a teda úroveň explorácie nepotrebuje byť vysoká (jej hodnoty sa môžu pohybovať blízko pri nule). Zároveň poznamenajme, že keďže parameter ϵ_{index} prislúcha pravdepodobnosti, tak $\epsilon_{index} \in [0,1]$.

- **Softmax selekcia:** Druhým najpoužívanejším typom selekcie je tzv. *softmax selekcia*. Tá pracuje na veľmi podobnom princípe ako ϵ-greedy selekcia. Líši sa však tým, že možné exploračné akcie sa negenerujú pomocou rovnomerného rozdelenia. Pravdepodobnosť vybratia (resp. použitia) akcie u_i záleží od hodnoty Q-funkcie, ktorá sa na ňu vzťahuje. Tým pádom sú uprednostňované akcie, ktoré majú vyššiu hodnotu Q-funkcie. Vychádzajúc z [5] sa typicky pri sofmax selekcii využíva tzv. *Boltzmannovo rozdelenie*:

$$u_i = u \in U(x_i) \quad s\ pravdepodobnosťou\ \frac{exp(-Q(x_i,u_i)/\tau)}{\sum_u \exp(-Q(x_i,u)/\tau)}. \tag{33}$$

Úloha menovateľa vo vzťahu (33) je to, aby sa súčet pravdepodobností všetkých vykonateľných akcií pre daný stav rovnal 1. Parameter τ je pri takomto type rozdelenia nazývaný *teplota* a platí, že $\tau \in (0,\infty)$. V prípade, že hodnota tohto parametra je blízko nuly, tak je táto selekcia ekvivalentná so základnou greedy selekciou. Avšak v prípade, že sú hodnoty τ vysoké, je pravdepodobnosť selekcie pre každú akciu $u \in U(x_i)$ rovnaká, t.j. pravdepodobnosti pochádzajú z rovnomerného rozdelenia.

Poznamenajme, že zvolenie konkrétneho druhu selekcie závisí najmä od typu riešeného problému. Výhodou použitia ϵ-greedy selekcie je jednoduchosť implementovania a taktiež jednoduchosť pri voľbe parametra. Je dôležité poznamenať, že rovnaká pravdepodobnosť zvolenia pre všetky vykonateľné akcie môže byť v určitých prípadoch aj nevýhodou. Napríklad v prípade, kedy by opakovaná voľba zlej akcie mohla mať deštruktívny vplyv na agenta, resp. na systém. V takomto prípade je výhodnejšie použiť v algoritme softmax selekciu.

3.3.3 Metóda SARSA

Algoritmus SARSA možno považovať za online implementáciu metódy iterovania stratégií. Názov tohto algoritmu je odvodený od názvov prvkov, ktoré sa pri tejto metóde využívajú na aktualizovanie hodnôt párov stav-akcia. V terminológii reinforcement learningu ide o tieto prvky (po poradí): **S**tate(i), **A**ction(i), **R**eward(i), **S**tate(i+1), **A**ction(i+1). Zložiac dokopy počiatočné písmená použitých prvkov (z angličtiny) dostávame názov *SARSA*. Pri popisovaní tohto algoritmu budeme vychádzať z už odprezentovanej offline metódy iterovania stratégií. A to z dôvodu, že algoritmus SARSA obsahuje procesy vyhodnocovania stratégie a vylepšenia stratégie rovnako ako algoritmus iterovania stratégií. Poznamenajme však, že pri metóde SARSA tieto procesy nie sú veľmi jednoducho odčítateľné.

Namiesto toho, aby sme explicitne zvolili počiatočnú stratégiu $h_0(x)$, tak na začiatku zvolíme (ľubovoľným spôsobom) samotné hodnoty funkcie $Q_0(x, u)$. Práve pomocou týchto hodnôt sa potom spätne volia akcie (t.j. určuje sa stratégia). Poznamenajme, že akcie sa vyberajú pomocou greedy selekcie. Následne sú hodnoty Q-funkcie (a teda aj stratégie) vyhodnocované iteračne pomocou nasledujúceho vzťahu:

$$Q(x_i, u_i) = Q(x_i, u_i) + \alpha[r(x_i, u_i, x_{i+1}) + \gamma Q(x_{i+1}, u_{i+1}) - Q(x_i, u_i)], \tag{34}$$

kde sú akcie u_i a u_{i+1} v každom kroku i vyberané pomocou greedy selekcie za aktuálneho ohodnotenia (za použitia mechanizmu náhodnej selekcie uvedenom v Časti 3.3.2).

Vo vzťahu (34) vystupuje parameter α, ktorý sa nazýva tzv. *miera učenia*. Udáva mieru hovoriacu o tom, koľko novozískanej informácie (t.j. TD-chyba) zmení starú (už známu) informáciu (t.j. aktuálny odhad hodnôt Q-funkcie). Ako už bolo spomínané, tak parameter α predstavuje mieru, takže platí, že $\alpha \in (0, 1]$. Poznamenajme, že tento parameter v

priebehu algoritmu typicky klesá. To znamená, že na začiatku sa algoritmus „učí"vo veľkej miere. Čím má však algoritmus viac informácií, tým je učenie sa menej potrebné a to zodpovedá zmenšenej miere učenia α. Z toho dôvodu bude v neskorších častiach práce vystupovať miera učenia pod označením α_{index}.

Je možné všimnúť si fakt, že vo vzťahu (34) sa aktualizujú hodnoty $Q(x_i, u_i)$ na základe $Q(x_{i+1}, u_{i+1})$. To znamená, že tieto aktualizácie sa vykonávajú rovnako ako pri metóde offline iterovania stratégií (vzťah (22)).

Inak povedané, proces vyhodnocovania stratégie sa vyskytuje v každom kroku i, keď sa postupne získavajú lepšie odhady hodnôt funkcie $Q(x_i, u_i)$ (t.j. keď sa iteračným spôsobom vyhodnocuje stratégia). Vďaka takémuto spresňovaniu odhadov hodnôt Q-funkcie a za pomoci náhodných akcií je algoritmus schopný nájsť akcie s vyššími hodnotami Q-funkcie. Možno teda povedať, že vylepšenie stratégie nastáva práve vtedy, keď v danom stave x_i nájdeme takú akciu u_i', pre ktorú je hodnota $Q(x_i, u_i')$ vyššia ako pri pôvodnej akcii u_i (keďže akcie sú vyberané greedy selekciou).

Algoritmus SARSA má teda nasledovný tvar:

Algoritmus 3: *SARSA (online iterovanie stratégií s použitím ϵ-greedy selekcie)*

Vstupné dáta: počiatočné hodnotenie $Q(x, u)$, diskontný faktor γ, miera učenia α_0, parameter ϵ_0 vystupujúci v ϵ-greedy selekcii.

Cyklus: pre $l = 1, 2, ...$

- **Inicializovanie:** Vygenerujeme počiatočný stav x_0 a vyberieme akciu, ktorá sa má v tomto stave vykonať, a to nasledujúcim spôsobom:

$$u_0 = \begin{cases} argmax_u Q(x_0, u) & \text{s pravdepodobnosťou } (1 - \epsilon_0), \\ u \in U(x_0) & \text{s pravdepodobnosťou } \epsilon_0. \end{cases}$$

- **Cyklus:** pre $i = 0, 1, 2, ...$

  Pre každé i vykonáme nasledujúce úkony:

  ☐ Pozorujúc hodnoty x_{i+1} a $r(x_i, u_i, x_{i+1})$ vyberieme akciu nasledujúcim spôsobom:

$$u_{i+1} = \begin{cases} argmax_{u'} Q(x_{i+1}, u') & \text{s pravdepodobnosťou } (1 - \epsilon_l), \\ u \in U(x_{i+1}) & \text{s pravdepodobnosťou } \epsilon_l. \end{cases}$$

- ☐ Aktualizujeme hodnoty Q-funkcie:

$$Q(x_i, u_i) = Q(x_i, u_i) + \alpha_l[r(x_i, u_i, x_{i+1}) + \gamma Q(x_{i+1}, u_{i+1}) - Q(x_i, u_i)].$$

- ☐ Opakujeme kroky výberu akcie a aktualizácie hodnôt, pokým sa algoritmus nedostane do jedného z koncových (terminálnych) stavov.

Nájdenie optimálnej stratégie: Nájdeme optimálnu stratégiu h^* nasledujúcim spôsobom:

$$h^*(x) = arg \max_u Q(x, u).$$

Výsledok: Optimálna stratégia $h^*(x)$.

3.3.4 Aplikácia metódy SARSA na vzorový problém

Metódu SARSA (t.j. online iterovanie stratégií s použitím ϵ-greedy selekcie), tak ako je popísaná v Časti 3.3.3, sme aplikovali na vzorový problém o pohybe robota v miestnosti popísaný v Kapitole 2. Pre aplikovanie Algoritmu 3 sme zvolili nasledujúce konkrétne hodnoty vstupných parametrov:

- počiatočná hodnota Q-funkcie je $Q(x, u) = 0$. Poznamenajme, že počiatočné hodnoty Q-funkcie sme volili s rovnakou myšlienkou ako v Časti 3.2.3,
- počiatočný stav x_0 na začiatku každej l-tej iterácie je generovaný z rovnomerného rozdelenia zo všetkých neterminálnych stavov,
- hodnota diskontného faktora je $\gamma = 1$ (t.j. hodnoty Q-funkcie nie sú v čase diskontované),
- cyklus sme vykonali $l = 30000$-krát,
- počiatočná hodnota miery učenia $\alpha_0 = 0,08$,

- na konci každej l-tej iterácie cyklu sme aktualizovali mieru učenia α pomocou vzťahu $\alpha_{l+1} = 0,9999 * \alpha_l$. To znamená, že finálny rozsah tejto premennej je *0,08* (na začiatku) - *0,00398* (na konci). Poznamenajme, že tento rozsah sme zvolili z dôvodu, že sa nám zdal dostatočný z hľadiska potrebného učenia sa. Na začiatku sa algoritmus „učí"vo väčšej miere ($\alpha_0 = 0,08$) a na konci sa v značnej miere využívajú už nadobudnuté informácie a „učí"sa v minimálnej miere ($\alpha_{30000} = 0,00398$),

- počiatočná hodnota parametra vystupujúceho v ϵ-greedy selekcii je $\epsilon_0 = 0,8$,

- na konci každej l-tej iterácie cyklu sme aktualizovali hodnotu používanú pri ϵ-greedy selekcii pomocou vzťahu $\epsilon_{l+1} = 0,9999 * \epsilon_l$. To znamená, že finálny rozsah tejto premennej je *0,8* (na začiatku) - *0,0398* (na konci). Poznamenajme, že tento rozsah sme zvolili z dôvodu, že sa nám zdal dostatočný z hľadiska explorácie a exploitácie. Na začiatku algoritmus objavuje vo veľkej miere ($\epsilon_0 = 0,8$) a na konci sa v značnej miere využívajú už nadobudnuté informácie a objavuje sa v minimálnej miere ($\epsilon_{30000} = 0,0398$).

Je dôležité poznamenať, že pri nami vytvorenej aplikácii algoritmu SARSA sme parameter počet iterácií pri ohodnotení Q-funkcie l zvolili tak, aby tento algoritmus skonvergoval. Tomu sme prispôsobili aj rozsah parametra ϵ a α. I keď z teoretického hľadiska (vychádzajúc z [5], [17]) by bolo potrebné vykonať nekonečne veľa iterácii, v našom prípade algoritmus skonverguje už pri zvolenom počte $l = 30000$ iterácií.
Aplikujúc metódu SARSA sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{P, P, P, -, H, -, H, -, H, L, H, L\}$, (t.j. výsledok je zhodný s výsledkom v Časti 3.2.3),

- celkový čas výpočtu bol $t \approx 6,544\ sek$.

3.4 Q-učenie

V tejto podkapitole popíšeme algoritmus s názvom *Q-učenie*, ktorý je vo svojej podstate pozmenený algoritmus iterovania hodnôt za použitia temporálnych diferencií. Pri metóde Q-učenia sú na začiatku, rovnako ako pri algoritme SARSA, zvolené ľubovoľné hodnoty

Q-funkcie. Vylepšenie tohto ohodnotenia sa v ďalších fázach algoritmu deje podľa nasledujúceho vzťahu:

$$Q(x_i, u_i) = Q(x_i, u_i) + \alpha[r(x_i, u_i, x_{i+1}) + \gamma \max_{u'} Q(x_{i+1}, u') - Q(x_i, u_i)], \qquad (35)$$

kde je akcia u_i vybraná opäť pomocou greedy selekcie. Výrazným rozdielom medzi Q-učením a metódou SARSA je použitie hodnoty $\max_{u'} Q(x_{i+1}, u')$ namiesto hodnoty $Q(x_{i+1}, u_{i+1})$. Možno povedať, že Q-učenie nevylepšuje ohodnotenie Q-funkcie na základe akcie použitej v stave x_{i+1}, ale na základe odhadovanej akcie (s maximálnou hodnotou Q-funkcie), ktorá bude použitá. Takýto spôsob vylepšovania ohodnotenia je akousi obdobou offline metódy iterovania hodnôt (vzťah (29)).

Algoritmus takto popísanej metódy Q-učenia má nasledujúcu štruktúru:

Algoritmus 4: *Q-učenie (online iterovanie hodnôt s použitím ϵ-greedy selekcie)*

Vstupné dáta: počiatočné hodnotenie $Q(x, u)$, diskontný faktor γ, miera učenia α_0, parameter ϵ_0 vystupujúci v ϵ-greedy selekcii.

Cyklus: pre $l = 1, 2, ...$

- **Inicializovanie:** Vygenerujeme počiatočný stav x_0.
- **Cyklus:** pre $i = 0, 1, 2, ...$

  Pre každé i vykonáme nasledujúce úkony:

  □ V každom stave x_i zvolíme akciu u_i ako:

$$u_i = \begin{cases} argmax_u Q(x_i, u) & \text{s pravdepodobnosťou } (1 - \epsilon_l), \\ u \in U(x_i) & \text{s pravdepodobnosťou } \epsilon_l. \end{cases}$$

  □ Pozorujúc hodnoty x_{i+1} a $r(x_i, u_i, x_{i+1})$ vykonáme aktualizáciu hodnôt Q-funkcie $Q(x_i, u_i)$ nasledujúcim spôsobom:

$$Q(x_i, u_i) = Q(x_i, u_i) + \alpha_l[r(x_i, u_i, x_{i+1}) + \gamma \max_{u'} Q(x_{i+1}, u') - Q(x_i, u_i)].$$

□ Opakujeme kroky výberu akcie a aktualizácie hodnôt, pokým sa algoritmus nedostane do jedného z koncových (terminálnych) stavov.

Nájdenie optimálnej stratégie: Nájdeme optimálnu stratégiu $h^*(x)$ nasledujúcim spôsobom:

$$h^*(x) = arg\max_u Q(x,u).$$

Výsledok: Optimálna stratégia $h^*(x)$.

3.4.1 Aplikácia Q-učenia na vzorový problém

Metódu Q-učenia (t.j. online iterovanie hodnôt s použitím ϵ-greedy selekcie), tak ako je popísaná v Časti 3.4, sme aplikovali na vzorový problém o pohybe robota v miestnosti popísaný v Kapitole 2. Pre aplikovanie Algoritmu 4 sme zvolili nasledujúce konkrétne hodnoty vstupných parametrov:

- počiatočná hodnota Q-funkcie je $Q(x,u) = 0$. Poznamenajme, že počiatočné hodnoty Q-funkcie sme volili s rovnakou myšlienkou ako v Časti 3.2.3,
- počiatočný stav x_0 na začiatku každej l-tej iterácie je generovaný z rovnomerného rozdelenia zo všetkých neterminálnych stavov,
- hodnota diskontného faktora je $\gamma = 1$ (t.j. hodnoty Q-funkcie nie sú v čase diskontované),
- cyklus sme vykonali $l = 30000$-krát,
- počiatočná hodnota miery učenia $\alpha_0 = 0,08$,
- na konci každej l-tej iterácie cyklu sme aktualizovali mieru učenia α pomocou vzťahu $\alpha_{l+1} = 0,9999 * \alpha_l$. To znamená, že finálny rozsah tejto premennej je *0,08* (na začiatku) - *0,00398* (na konci). Poznamenajme, že tento rozsah sme zvolili z dôvodu, že sa nám zdal dostatočný z hľadiska potrebného učenia sa. Na začiatku sa algoritmus „učí"vo väčšej miere ($\alpha_0 = 0,08$) a na konci sa v značnej miere využívajú už nadobudnuté informácie a „učí"sa v minimálnej miere ($\alpha_{30000} = 0,00398$),

- počiatočná hodnota parametra vystupujúceho v ϵ-greedy selekcii je $\epsilon_0 = 0,8$,
- na konci každej l-tej iterácie cyklu sme aktualizovali hodnotu používanú pri ϵ-greedy selekcii pomocou vzťahu $\epsilon_{l+1} = 0,9999 * \epsilon_l$. To znamená, že finálny rozsah tejto premennej je *0,8* (na začiatku) - *0,0398* (na konci). Poznamenajme, že tento rozsah sme zvolili z dôvodu, že sa nám zdal dostatočný z hľadiska explorácie a exploitácie. Na začiatku algoritmus objavuje vo veľkej miere ($\epsilon_0 = 0,8$) a na konci sa v značnej miere využívajú už nadobudnuté informácie a objavuje sa v minimálnej miere ($\epsilon_{30000} = 0,0398$).

Poznamenajme, že pri nami vytvorenom programe založenom na algoritme Q-učenia sme volili hodnoty parametrov l, α_l a ϵ_l s rovnakou myšlienkou ako v Časti 3.3.4.
Aplikujúc metódu Q-učenia sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{P, P, P, -, H, -, H, -, H, L, H, L\}$, (t.j. výsledok je zhodný s výsledkom v Časti 3.2.3),
- celkový čas výpočtu bol $t \approx 6,168\ sek$.

3.5 Metóda Monte Carlo

Ako sme už spomínali v predchádzajúcich častiach, pri aplikovaní metód DP je potrebné poznať presný model sledovaného systému (t.j. funkciu prechodu a funkciu odmien), čo však v mnohých prípadoch nie je realistické. V tejto časti práce predstavíme ďalšiu metódu, resp. prístup, ktorý pre svoje správne fungovanie nepotrebuje kompletnú informáciu o modeli sledovaného systém. A to tzv. *metódu Monte Carlo simulácií* (skrátene MC simulácie), ktorá sa využíva v mnohých odvetviach matematiky. Tento prístup môžeme kategorizovať vzhľadom na delenie RL algoritmov uvedené v Podkapitole 3.1 ako algoritmus bez informácie o modeli. Agent sa obdobne ako pri metódach založených na temporálnych diferenciách učí postupne z interakcií s prostredím. Optimálna stratégia je teda zistená výlučne na základe dát z týchto interakcií. To znamená, že metóda Monte Carlo simulácií je aplikovateľná aj na príklady a problémy, pri ktorých na začiatku nepoznáme každý prvok Markovovského rozhodovacieho procesu. Vzhľadom na delenie RL algoritmov na základe času, kedy agent interaguje so systémom možno metódu Monte Carlo

označiť za online algoritmus. A to z dôvodu, že algoritmus zbieha a teda sa „učí"zároveň so zbiehaním systému.

Monte Carlo metóda je založená na odhadovaní ohodnocovacích funkcií z už spomínaných interakcií s prostredím. Dáta o prostredí získavame pri tomto prístupe konkrétne zo simulovaných epizód, ktoré začínajú v náhodnom stave. Každá simulovaná epizóda sa riadi podľa zvolenej stratégie. Je dôležité poznamenať, že pri simulácii epizód je potrebné, aby tieto epizódy končili. To je typicky dosiahnuteľné tým, že budeme simulovať epizódu až pokým sa systém nedostane do terminálneho stavu (t.j. do stavu, z ktorého nie je možné odísť). V prípade, že riešime úlohu, ktorá v množine stavov neobsahuje žiaden terminálny stav, tak je nutné, aby sme simulovali epizódu s dostatočnou dĺžkou. Pre potreby tejto práce budeme prednastavenú maximálnu dĺžku simulovaných epizód označovať K. Poznamenajme, že určenie dostatočnej dĺžky je špecifické pre každú úlohu, resp. typ problému.

Pri každej simulácii epizódy si „zapamätáme"stavy, v ktorých bol systém počas danej epizódy, akcie vykonané v daných stavoch a taktiež odmeny, ktoré plynuli z vykonaných akcií v daných stavoch. Po ukončení každej simulovanej epizódy odhadneme hodnoty Q-funkcie pre každý pár stav-akcia, ktorý sa v danej epizóde vyskytol. Tento odhad sa deje tak, že na epizódu nahliadneme od konca a za informácie o odmenách v danej epizóde pomocou diskontovania zistíme hodnotu Q-funkcie v danom stave za vykonania danej akcie pri postupovaní podľa určenej stratégie. Vychádzajúc z [5] bude mať $Q^h(x_k, u_k)$ v k-tom kroku simulovanej epizódy nasledujúci tvar:

$$Q^h(x_k, u_k) = \sum_{j=k}^{K-1} \gamma^{j-k} r(x_j, u_j, x_{j+1}). \tag{36}$$

Po vykonaní dostatočne veľa epizód (počet epizód budeme označovať pre potreby tejto diplomovej práce ako N) pri danej stratégií je možné spriemerovaním hodnôt Q-funkcie pre dané páry stav-akcia vytvoriť finálne odhady Q-funkcie pri danej stratégii h, ktoré majú nasledujúci tvar:

$$Q^h(x_0, u_0) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=0}^{K_i-1} \gamma^j r_i(x_j, u_j, x_{j+1}), \tag{37}$$

kde $r_i(x_j, u_j, x_{j+1})$ predstavuje hodnotu odmeny z i-tej epizódy, ak sa v stave x_j vy-

konanala akcia u_j a systém sa tak dostal do stavu x_{j+1}.

Po získaní finálnych odhadov Q-funkcie pri aplikovaní danej stratégie h možno aplikovať greedy selekciu a nájsť tak lepšiu stratégiu. Tento proces je potrebné vykonať dostatočne veľakrát, pričom sa správny počet jeho vykonania líši od úlohy k úlohe.

Je dôležité poznamenať, že pri Monte Carlo metóde nie je garantované, že počas simulovaných epizód sa systém dostane do každého stavu. Preto za účelom navštívenia čo najviac možných stavov aplikujeme opäť ϵ-greedy výber akcií počas postupovania pomocou danej stratégie h (podobne ako sme to robili v Časti 3.3 pri metódach založených na temporálnych diferenciách).

Algoritmus Monte Carlo simulácií má teda nasledovný tvar:

Algoritmus 5: *Monte Carlo simulácie*

Vstupné dáta: počiatočná stratégia h_0, diskontný faktor γ, parameter ϵ_0 vystupujúci v ϵ-greedy selekcii.

Cyklus: pre $l = 0, 1, 2, ...$

Pre každé l vykonáme nasledujúce úkony:

- Nastavíme hodnoty Q-funkcie $Q(x, u) = 0$, hodnoty akumulátora $A(x, u) = 0$ a hodnoty počítadla $C(x, u) = 0$.
- Vykonáme N epizód, pričom si zaznamenáme navštívené stavy, akcie vykonané v týchto stavoch a z nich plynúce odmeny, pričom akciu u v každom stave x vyberáme nasledujúcim spôsobom:

$$u = \begin{cases} h_l(x) & \text{s pravdepodobnosťou } (1 - \epsilon_p), \\ u \in U(x) & \text{s pravdepodobnosťou } \epsilon_p. \end{cases}$$

- **Cyklus:** pre $k = 1, 2, ...$ pre každú i-tu epizódu vykonáme nasledovné:
  - ☐ $A(x_k, u_k) = A(x_k, u_k) + \sum_{j=k}^{K_i - 1} \gamma^{j-k} r_i(x_j, u_j, x_{j+1})$,
  - ☐ $C(x_k, u_k) = C(x_k, u_k) + 1$.

- Aktualizujeme hodnoty funkcie Q-funkcie $Q(x, u)$:

$$Q(x, u) = \frac{A(x, u)}{C(x, u)}.$$

- Nájdeme novú stratégiu h_{l+1} nasledujúcim spôsobom:

$$h_{l+1}(x) = arg \max_u Q(x, u).$$

Výsledok: Optimálna stratégia $h^*(x)$.

Poznamenajme, že index p vystupujúci pri parametri ϵ v Algoritme 5 predstavuje celkové množstvo vykonaných simulácií, t.j. $p = l \times N$.

Zároveň poznamenajme, že maximálna hodnota indexu k pre danú i-tu epizódu je rovná dĺžke tejto epizódy.

3.5.1 Aplikácia Monte Carlo metódy na vzorový problém

Metódu Monte Carlo simulácií, tak ako je popísaná v Časti 3.5, sme aplikovali na vzorový problém o pohybe robota v miestnosti popísaný v Kapitole 2. Pre aplikovanie Algoritmu 5 sme zvolili nasledujúce konkrétne hodnoty vstupných parametrov:

- počiatočná stratégia je $h_0(x) = \{D, L, P, -, D, -, D, -, P, P, P, H\}$. Táto počiatočná stratégia bola zvolená s ideou, aby sa v každom stave pohol robot tak, aby „nenarazil"do steny. Poznamenajme však, že takýto spôsob stratégie nie je nutný a počiatočná stratégia môže byť ľubovoľná (možno ju v programe aj priamo generovať zo zvoleného rozdelenia). My sme zvolili presnú počiatočnú stratégiu h_0 z dôvodu urýchlenia programu,

- hodnota diskontného faktora je $\gamma = 1$ (t.j. hodnoty Q-funkcie nie sú v čase diskontované),

- počet stratégií, ktoré sme iteračne ohodnocovali je $l = 10$,

- počet simulácií pri každom l-tom iteračnom ohodnocovaní je $N = 30000$,

- počiatočná hodnota parametra vystupujúceho v ϵ-greedy selekcii je $\epsilon_0 = 0,8$,
- po každej simulácii sme aktualizovali hodnotu používanú pri ϵ-greedy selekcii pomocou vzťahu $\epsilon_{p+1} = 0,99999 * \epsilon_p$. To znamená, že finálny rozsah tejto premennej je *0,8* (na začiatku) - *0,0398* (na konci). Poznamenajme, že tento rozsah sme zvolili z dôvodu, že sa nám zdal dostatočný z hľadiska explorácie a exploitácie. Na začiatku algoritmus objavuje vo veľkej miere ($\epsilon_0 = 0,8$) a na konci sa v značnej miere využívajú už nadobudnuté informácie a objavuje sa v minimálnej miere ($\epsilon_{300000} = 0,0398$).

Je dôležité poznamenať, že pri nami vytvorených MC simuláciách parametre počet stratégií l, ktoré sme iteračne ohodnocovali, a počet simulácií pri každom takomto iteračnom ohodnotení N sme zvolili tak, aby algoritmus MC simulácií skonvergoval. Tomu sme prispôsobili aj rozsah parametra ϵ. I keď z teoretického hľadiska (vychádzajúc z [5], [15]) by bolo potrebné iteračne ohodnotiť nekonečne veľa stratégií, v našom prípade algoritmus skonverguje už pri zvolenom počte $l = 10$ stratégií.
Aplikujúc metódu Monte Carlo simulácií sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{P, P, P, -, H, -, H, -, H, L, H, L\}$ (t.j. výsledok je zhodný s výsledkom v Časti 3.2.3),
- celkový čas výpočtu bol $t \approx 64,452\ sek$.

3.6 Dyna-Q

V predchádzajúcich častiach diplomovej práce sme popísali najprv metódy DP (Podkapitoly 3.2.2 a 3.2.4) a potom sme sa zamerali na odprezentovanie algoritmov, ktoré pre svoje fungovanie nepotrebujú poznať model prostredia (Podkapitoly 3.3, 3.4 a 3.5). V tejto časti sa zameriame na metódu RL, ktorá je založená na práci s modelom prostredia. A to tzv. algoritmus *Dyna-Q*. Poznamenajme, že pri tomto prístupe je buď model prostredia dopredu známy, alebo sa ho agent postupne učí. Pre potreby tejto práce sa zameriame na prístup Dyna-Q v prípade, že je model prostredia vopred známy. Z toho dôvodu môžeme metódu Dyna-Q zaradiť podľa kategorizácie RL algoritmov z Časti 3.1 do skupiny algoritmov, ktoré sú založené na modeli. V prípade, že by sa agent postupne

učil sledovaný model interakciami s prostredím, tak by sme algoritmus Dyna-Q zaradili medzi algoritmy, ktoré sa postupne model učia.

Vychádzajúc z [20], [23] bol algoritmus Dyna-Q vyvinutý s myšlienkou rýchlejšej konvergencie algoritmu Q-učenia. Je dôležité poznamenať, že i keď metóda Q-učenia patrí medzi algoritmy bez informácie o modeli, tak dala základ prístupu Dyna-Q. Metódu Dyna-Q možno teda označiť za určité rozšírenie algoritmu Q-učenia. Rozšírenie spočíva konkrétne v tom, že po vykonaní akcie u v stave x a následnom aktualizovaní hodnôt Q-funkcie umelo nasimulujeme aktualizovanie hodnôt Q-funkcie z informácií, ktoré už poznáme. A to takým spôsobom, že n-krát náhodne vygenerujeme pár stav-akcia (neterminálny stav), ktorý sa v i-tej epizóde už objavil a pre tento pár stav-akcia pozorujúc nasledujúci stav a z neho plynúcu odmenu vykonáme aktualizáciu hodnôt Q-funkcie spôsobom uvedeným vo vzťahu (35). To algoritmu Dyna-Q umožní vyťažiť lepšie odhady Q-funkcie (a tým sa priblížiť skôr ku konvergencii) z menšieho množstva informácií. Poznamenajme, že podľa [23] je takéto umelé simulovanie odhadov Q-funkcie výpočtovo a časovo menej náročné ako cykly algoritmu Q-učenia tak ako sú uvedené v Časti 3.4.

Algoritmus Dyna-Q má teda nasledovný tvar:

Algoritmus 6: *Dyna-Q*

Vstupné dáta: počiatočné hodnotenie $Q(x, u)$, model sledovaného systému, diskontný faktor γ, miera učenia α_0, parameter ϵ_0 vystupujúci v ϵ-greedy selekcii.

Cyklus: pre $l = 1, 2, ...$

- **Inicializovanie:** Vygenerujeme počiatočný stav x_0.
- **Cyklus:** pre $i = 0, 1, 2, ...$

  Pre každé i vykonáme nasledujúce úkony:

  □ V stave x_i zvolíme akciu pomocou:

$$u_i = \begin{cases} argmax_u Q(x_i, u) & \text{s pravdepodobnosťou } (1 - \epsilon_l), \\ u \in U(x_i) & \text{s pravdepodobnosťou } \epsilon_l. \end{cases}$$

☐ Pozorujúc hodnoty x_{i+1} a $r(x_i, u_i, x_{i+1})$ vykonáme aktualizáciu hodnôt Q-funkcie $Q(x_i, u_i)$ nasledujúcim spôsobom:

$$Q(x_i, u_i) = Q(x_i, u_i) + \alpha_l[r(x_i, u_i, x_{i+1}) + \gamma \max_{u'} Q(x_{i+1}, u') - Q(x_i, u_i)].$$

☐ **Cyklus:** pre $n = 1, 2, ...$

Pre každé n vykonáme nasledujúce úkony:

- ∗ Náhodne vygenerujeme z rovnomerného rozdelenia pár stav-akcia (neterminálny stav), ktorý už bol pozorovaný v danej epizóde, t.j. $x \in \{x_0, x_1, ..., x_i\}$, resp. $u \in \{u_0, u_1, ..., u_i\}$.
- ∗ Pozorujúc hodnoty x' (stav, do ktorého sa dostaneme zo stavu x aplikovaním akcie u) a $r(x, u, x')$ vykonáme aktualizáciu hodnôt Q-funkcie $Q(x, u)$ nasledujúcim spôsobom:

$$Q(x, u) = Q(x, u) + \alpha_l[r(x, u, x') + \gamma \max_{u'} Q(x', u') - Q(x, u)].$$

☐ Opakujeme vyššie uvedené kroky výberu akcie, aktualizácie hodnôt a cyklus, pokým sa algoritmus nedostane do jedného z koncových (terminálnych) stavov.

Nájdenie optimálnej stratégie: Nájdeme optimálnu stratégiu $h^*(x)$ nasledujúcim spôsobom:

$$h^*(x) = arg \max_u Q(x, u).$$

Výsledok: Optimálna stratégia $h^*(x)$.

3.6.1 Aplikácia algoritmu Dyna-Q na vzorový problém

Metódu Dyna-Q, tak ako je popísaná v Časti 3.6, sme taktiež aplikovali na vzorový problém o pohybe robota v miestnosti popísaný v Kapitole 2. Pre aplikovanie Algoritmu 6 sme zvolili nasledujúce konkrétne hodnoty vstupných parametrov:

- počiatočná hodnota Q-funkcie je $Q(x, u) = 0$. Poznamenajme, že počiatočné hodnoty Q-funkcie sme volili s rovnakou myšlienkou ako v Časti 3.2.3,
- model (t.j. funkcia prechodu a funkcia odmien), ktorý je použitý v programe, je popísaný v Časti 2.3,
- počiatočný stav x_0 na začiatku každej l-tej iterácie je generovaný z rovnomerného rozdelenia zo všetkých neterminálnych stavov,
- hodnota diskontného faktora je $\gamma = 1$ (t.j. hodnoty Q-funkcie nie sú v čase diskontované),
- hlavný cyklus sme vykonali $l = 300$-krát,
- vnútorný cyklus sme vykonali $n = 100$-krát,
- počiatočná hodnota miery učenia $\alpha_0 = 0,08$,
- na konci každej l-tej iterácie cyklu sme aktualizovali mieru učenia α pomocou vzťahu $\alpha_{l+1} = 0,99 * \alpha_l$. To znamená, že finálny rozsah tejto premennej je *0,08* (na začiatku) - *0,00392* (na konci). Poznamenajme, že tento rozsah sme zvolili z dôvodu, že sa nám zdal dostatočný z hľadiska potrebného učenia sa. Na začiatku sa algoritmus „učí" vo väčšej miere ($\alpha_0 = 0,08$) a na konci sa v značnej miere využívajú už nadobudnuté informácie a „učí" sa v minimálnej miere ($\alpha_{300} = 0,00392$),
- počiatočná hodnota parametra vystupujúceho v ϵ-greedy selekcii je $\epsilon_0 = 0,8$,
- na konci každej l-tej iterácie cyklu sme aktualizovali hodnotu používanú pri ϵ-greedy selekcii pomocou vzťahu $\epsilon_{l+1} = 0,99 * \epsilon_l$. To znamená, že finálny rozsah tejto premennej je *0,8* (na začiatku) - *0,0392* (na konci). Poznamenajme, že tento rozsah sme zvolili z dôvodu, že sa nám zdal dostatočný z hľadiska explorácie a exploitácie. Na začiatku algoritmus objavuje vo veľkej miere ($\epsilon_0 = 0,8$) a na konci sa v značnej miere využívajú už nadobudnuté informácie a objavuje sa v minimálnej miere ($\epsilon_{300} = 0,0392$).

Poznamenajme, že v nami vytvorenom programe založenom na aplikácii Dyna-Q algoritmu sme volili hodnoty parametrov l, n, α_l a ϵ_l s rovnakou myšlienkou ako sme volili parametre v Časti 3.3.4.

Aplikujúc metódu Dyna-Q sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{P, P, P, -, H, -, H, -, H, L, H, L\}$, (t.j. výsledok je zhodný s výsledkom v Časti 3.2.3),
- celkový čas výpočtu bol $t \approx 5,678\ sek$.

4 Aplikácia RL algoritmov na úlohu stochastického OR

V poslednej časti práce aplikujeme algoritmy reinforcement learningu (predstavené v Kapitole 3) na problém typicky riešiteľný tradičnými metódami stochastického optimálneho riadenia. Zároveň v tejto kapitole porovnáme vhodnosť, resp. výhody a nevýhody, všetkých použitých prístupov k riešeniu, t.j. reinforcement learning algoritmov rovnako ako metód OR. To nám poskytne možnosť nahliadnuť na to, v akej miere sa dajú RL algoritmy využiť pri riešení stochastických diskrétnych úloh optimálneho riadenia s ekonomickým zameraním.

4.1 Úloha o obchodnej prevádzke

Za vhodnú úlohu stochastického optimálneho riadenia sme zvolili úlohu týkajúcu sa obchodnej prevádzky pochádzajúcu z [14]. Jej presné znenie má nasledujúci tvar:

Uvažujme nasledujúci systém. Malá obchodná prevádzka má dvoch zamestnancov. Jeden zo zamestnancov vždy obsluhuje zákazníkov pri pokladni. Druhý zamestnanec okrem obsluhy zákazníkov pri pokladni vykonáva aj iné práce. Predpokladajme, že v obchodnej prevádzke sa môže naraz vyskytnúť 0-6 zákazníkov. V prípade, že v obchodnej prevádzke je 6 zákazníkov, nie je možné, aby tam vstúpil akýkoľvek ďalší prichádzajúci zákazník. Majiteľ prevádzky sa rozhoduje s ohľadom na maximalizáciu svojho zisku, pri akom počte zákazníkov v obchode bude obsluhovať zákazníkov pri pokladni len jeden zamestnanec a pri akom počte zákazníkov budú obsluhovať pri pokladni zákazníkov obaja zamestnanci. Predpokladajme, že na začiatku sledovaného obdobia nie je v obchodnej prevádzke žiaden zákazník.

V prípade, že zákazníkov pri pokladni obsluhuje iba jeden zamestnanec, má sledovaný systém nasledujúcu dynamiku. V každej časovej jednotke, ak je v obchode 1-5 zákazníkov, tak sa s rovnakou pravdepodobnosťou môžu udiať nasledujúce udalosti:

- *do obchodu vstúpi nový zákazník,*
- *nič sa nestane,*

- *zákazník je vybavený a odíde z prevádzky.*

Ak v prevádzke nie je žiaden zákazník, tak sa s pravdepodobnosťou $\frac{2}{3}$ nič nestane a s pravdepodobnosťou $\frac{1}{3}$ do prevádzky vstúpi nový zákazník. V prípade, že je v prevádzke 6 zákazníkov, tak sa s pravdepodobnosťou $\frac{2}{3}$ nič nestane a s pravdepodobnosťou $\frac{1}{3}$ je zákazník vybavený a odíde z prevádzky.

V prípade, že pri pokladni obsluhujú zákazníkov obaja zamestnanci, tak má sledovaný systém nasledujúcu dynamiku. V každej jednotke času, ak je v obchode 1-5 zákazníkov, tak sa môžu udiať nasledujúce udalosti:

- *do obchodu vstúpi nový zákazník s pravdepodobnosťou $\frac{1}{4}$,*
- *nič sa nestane s pravdepodobnosťou $\frac{1}{4}$,*
- *zákazník je vybavený a odíde z prevádzky s pravdepodobnosťou $\frac{1}{2}$.*

Ak v prevádzke nie je žiaden zákazník, tak sa s pravdepodobnosťou $\frac{2}{3}$ nič nestane a s pravdepodobnosťou $\frac{1}{3}$ do prevádzky vstúpi nový zákazník. V prípade, že je v prevádzke 6 zákazníkov, tak sa s pravdepodobnosťou $\frac{1}{3}$ nič nestane a s pravdepodobnosťou $\frac{2}{3}$ je zákazník vybavený a odíde z prevádzky.

Pre majiteľa prevádzky predstavuje vybavenie každého zákazníka zisk (odmenu) 1. V prípade, že pri pokladni vybavujú zákazníkov obaja zamestnanci, tak pre majiteľa prevádzky to v každej časovej jednotke predstavuje stratu 0,2 (resp. zisk -0,2), pretože druhý zamestnanec nemôže vykonávať iné práce v prevádzke. Majiteľ pri svojich úvahách diskontuje svoj zisk v budúcich časových jednotkách pomocou diskontného faktora s hodnotou 0,9.

4.2 Matematická formulácia úlohy o obchodnej prevádzke

Teraz úlohu popísanú v Podkapitole 4.1 naformulujeme s využitím metodiky uvedenej v Časti 1.3. Z hľadiska teórie optimálneho riadenia a reinforcement learningu môžeme teda povedať, že:

- v obchodnej prevádzke sa môže nachádzať 0-6 zákazníkov, t.j. množina stavov je $X = \{0, 1, ..., 6\}$,

- majiteľ môže vykonať 2 akcie, a to *akcia 1* - pri pokladni obsluhuje zákazníkov 1 zamestnanec, *akcia 2* - pri pokladni obsluhujú zákazníkov 2 zamestnanci. Množina akcií (riadení) je $U = \{1, 2\}$,

- na začiatku sledovaného obdobia nie je v obchodnej prevádzke žiaden zákazník, t.j. počiatočný stav je $x_0 = 0$,

- v prípade aplikovania *akcie 1* má systém dynamiku, ktorú popíšeme pomocou matice P_1:

$$P_1 = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} \end{bmatrix}.$$

  Matica P_1 rozmeru 7×7 reprezentuje maticu prechodu medzi stavmi prislúchajúcimi počtom zákazníkov v obchodnej prevádzke. Poznamenajme, že i-ty riadok (resp. stĺpec) matice P_1 prislúcha počtu *i-1* zákazníkov v obchode. Zároveň poznamenajme, že keďže matica P_1 je maticou prechodu popísanú v Časti 1.3.1, tak súčet prvkov každého riadku je rovný 1,

- v prípade aplikovania *akcie 2* má sledovaný systém dynamiku, ktorú popíšeme pomocou matice P_2:

$$P_2 = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix}.$$

Význam prvkov matice P_2 je analogický ako pri matici P_1,

- v prípade aplikovania *akcie 1* definujeme maticu odmien R_1 (rozmeru 7×7) nasledovne:

$$R_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Poznamejnajme, že i-ty riadok (resp. stĺpec) matice R_1 prislúcha počtu *i-1* zákazníkov v obchode (t.j. odbobne ako pri maticiach prechodu P_1 a P_2),

- v prípade aplikovania *akcie 2* definujeme maticu odmien R_2 nasledovne:

$$R_2 = \begin{bmatrix} -0,2 & -0,2 & -0,2 & -0,2 & -0,2 & -0,2 & -0,2 \\ 0,8 & -0,2 & -0,2 & -0,2 & -0,2 & -0,2 & -0,2 \\ -0,2 & 0,8 & -0,2 & -0,2 & -0,2 & -0,2 & -0,2 \\ -0,2 & -0,2 & 0,8 & -0,2 & -0,2 & -0,2 & -0,2 \\ -0,2 & -0,2 & -0,2 & 0,8 & -0,2 & -0,2 & -0,2 \\ -0,2 & -0,2 & -0,2 & -0,2 & 0,8 & -0,2 & -0,2 \\ -0,2 & -0,2 & -0,2 & -0,2 & -0,2 & 0,8 & -0,2 \end{bmatrix}.$$

Význam prvkov matice R_2 je analogický ako pri matici R_1,

- majiteľ obchodnej prevádzky diskontuje svoje budúce zisky pomocou diskontného faktora *0,9*, t.j. $\gamma = 0,9$.

4.3 Riešenie pomocou metódy stochastického OR

Keďže cieľom tejto diplomovej práce je porovnanie metód optimálneho riadenia a RL algoritmov, tak v tejto podkapitole najprv naformulujeme úlohu o obchodnej prevádzke

(popísanú v Časti 4.1) ako úlohu v štandardnom tvare úloh optimálneho riadenia a neskôr túto úlohu taktiež vyriešime pomocou numerickej metódy stochastického OR.

Úloha o obchodnej prevádzke má ako úloha OR nasledujúci tvar:

$$\begin{aligned} \max_{h(.)} \quad & \mathbb{E}\left[\sum_{i=0}^{T} \gamma^i r(u_i, z(u_i, y_i))\right], \\ x_{i+1} &= f(x_i, z(u_i, y_i)), \\ y_i &= x_i + 1, \\ x_0 &= 0, \\ x_i &\in \{0, 1, ..., 6\}, \\ u_i &\in \{1, 2\}, \end{aligned} \tag{38}$$

kde γ predstavuje diskontný faktor.

Funkciu $z(u_i, y_i)$ vystupujúcu v zápise (38), ktorá predstavuje náhodnosť v úlohe, definujeme nasledujúcim spôsobom. Pre lepšiu prehľadnosť najprv uvedieme zjednotený tvar funkcie $z(u_i, y_i)$ a následne popíšeme samostatne podfunkcie pre každú vykonateľnú akciu (t.j. $u_i \in \{1, 2\}$). A teda:

$$z(u_i, y_i) = \begin{cases} z(u_i, 1) & , \text{ak } y_i = 1, \\ z(u_i, poc) & , \text{ak } poc \in \{2, ..., 6\}, \\ z(u_i, 7) & , \text{ak } y_i = 7, \end{cases} \tag{39}$$

kde

$$z(u_i, 1) = \begin{cases} \text{nič sa nestalo} & , \text{s pravdepodobnosťou } P_{u_i}[1, 1], \\ \text{prišiel zákazník} & , \text{s pravdepodobnosťou } P_{u_i}[1, 2], \end{cases} \tag{40}$$

$$z(u_i, poc) = \begin{cases} \text{odišiel zákazník} & , \text{s pravdepodobnosťou } P_{u_i}[poc, poc - 1], \\ \text{nič sa nestalo} & , \text{s pravdepodobnosťou } P_{u_i}[poc, poc], \\ \text{prišiel zákazník} & , \text{s pravdepodobnosťou } P_{u_i}[poc, poc + 1], \end{cases} \tag{41}$$

$$z(u_i, 7) = \begin{cases} \text{odišiel zákazník} & , \text{s pravdepodobnosťou } P_{u_i}[7, 6], \\ \text{nič sa nestalo} & , \text{s pravdepodobnosťou } P_{u_i}[7, 7]. \end{cases} \tag{42}$$

Matice P_{u_i}, t.j. matice P_1 a P_2, predstavujú matice prechodu popísané v Časti 4.2.

Premennú y_i možno chápať ako pomocnú premennú popisujúcu index v maticiach prechodu P_{u_i} prislúchajúci k stavu x_i. A to z dôvodu, že situácia, kedy je v obchodnej prevádzke x_i zákazníkov, je reprezentovaná riadkom č. $x_i + 1$ v spomínaných maticiach prechodu.

Parameter T (viď (38)) reprezentujúci počet časových krokov bude zvolený v neskoršej časti ako $T = 24$, čo môže predstavovať 24 polhodín v prevádzke otvorenej 12 hodín. Poznamenajme však, že tento parameter by mohol byť zvolený ľubovoľne.

Funkciu odmien $r(u_i, z(u_i, y_i))$ (viď (38)) definujeme nasledovne (použijúc podobnú štruktúru ako pri definovaní funkcie $z(u_i, y_i)$, viď (39)):

$$r(u_i, z(u_i, y_i)) = \begin{cases} r(1, z(u_i, y_i)) & , \text{ak } u_i = 1, \\ r(2, z(u_i, y_i)) & , \text{ak } u_i = 2, \end{cases} \tag{43}$$

kde

$$r(1, z(u_i, y_i)) = \begin{cases} 1 & , \text{ak } z(u_i, y_i) = \text{ odišiel zákazník}, \\ 0 & , \text{inak}, \end{cases} \tag{44}$$

$$r(2, z(u_i, y_i)) = \begin{cases} 0,8 & , \text{ak } z(u_i, y_i) = \text{ odišiel zákazník}, \\ -0,2 & , \text{inak}. \end{cases} \tag{45}$$

Poznamenajme, že funkcia odmien definovaná vzťahom (43) nepredstavuje špeciálny tvar funkcie odmien. I keď to nie je odčítateľné na prvý pohľad, tak argumentami tejto funkcie je aj stav systému x_i (vďaka premennej y_i), aj akcia (resp. riadenie) u_i.

Funkciu $f(x_i, z(u_i, y_i))$ vystupujúca v stavovej rovnici v zápise (38) definujeme nasledovne:

$$f(x_i, z(u_i, y_i)) = \begin{cases} x_i - 1 & , \text{ak } z(u_i, y_i) = \text{ odišiel zákazník}, \\ x_i & , \text{ak } z(u_i, y_i) = \text{ nič sa nestalo}, \\ x_i + 1 & , \text{ak } z(u_i, y_i) = \text{ prišiel zákazník}. \end{cases} \tag{46}$$

4.3.1 Aplikácia metódy stochastického OR na úlohu o obchodnej prevádzke

Na úlohu o obchodnej prevádzke, tak ako je popísaná v Podkapitole 4.1, a zapísaná ako úloha OR v Časti 4.3 sme aplikovali numerickú metódu stochastického optimálneho riadenia. Poznamenajme, že sme vychádzali z [7], [12] a [18]. Použitý výpočtový algoritmus bol na úlohu o obchodnej prevádzke implementovaný autorkou v systéme `Matlab` a je uvedený v Prílohe A.

Aplikujúc zvolenú numerickú metódu stochastického optimálneho riadenia sme dospeli k nasledujúcim výsledkom:

- optimálne riadenie pre každý časový krok i je $v_i(x) = \{1, 1, 1, 1, 1, 1, 2\}$,
- celkový čas výpočtu bol $t \approx 0,036\ sek$.

Výsledné hodnoty hodnotovej funkcie $V_i(x)$ rovnako ako hodnoty optimálnych riadení $v_i(x)$ pre zvolený počet časových krokov ($T = 24$) sú uvedené v Prílohe B.

4.4 Riešenie pomocou metód RL

V tejto podkapitole aplikujeme na úlohu o obchodnej prevádzke (popísanú v Časti 4.1) všetky algoritmy reinforcement learningu, ktoré sme predstavili v Kapitole 3. Poznamenajme, že všetky výpočtové algoritmy v tejto časti práce boli na zvolený problém boli implementované autorkou v systéme `Matlab` a sú uvedené v Prílohe A. Zároveň poznamenajme, že na výpočet zvolenej úlohy o obchodnej prevádzke boli použité rovnaké hodnoty parametrov ako pri aplikáciach na úlohu o pohybe robota v miestnosti v Kapitole 3 až na prípad troch výnimiek. Zmeny v hodnotách parametrov oproti Kapitole 3 sú nasledovné:

- pri aplikovaní metódy iterovania stratégií a MC simulácií sme použili počiatočnú stratégiu $h_0(x) = \{2, 2, 2, 2, 2, 2, 2\}$, t.j. za ľubovoľného počtu zákazníkov v obchodnej prevádzke sú pri pokladni obaja zamestnanci,
- keďže v úlohe o obchodnej prevádzke nie je žiaden zo stavov terminálny, tak pri aplikovaní metód SARSA, Q-učenie, Monte Carlo simulácií a Dyna-Q bolo potrebné nastaviť dĺžku epizód. Za vhodnú dĺžku sme určili epizódy skladajúce sa z *250* stavov,

- pri všetkých vypracovaných RL algoritmoch sme použili hodnotu diskontného faktora $\gamma = 0,9$ (vychádzajúc zo zadania v Časti 4.1).

Aplikujúc metódu iterovania stratégií sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{1, 1, 1, 1, 1, 1, 2\}$,
- počas algoritmu boli pred nájdením optimálnej stratégie vyhodnotené 2 rôzne stratégie, a to počiatočná stratégia $h_0(x)$ a optimálna stratégia $h^*(x)$), t.j. $l = 2$, teda:
  1. $h_0(x) = \{2, 2, 2, 2, 2, 2, 2\}$,
  2. $h_1(x) = \{1, 1, 1, 1, 1, 1, 2\}$,
- celkový čas výpočtu bol $t \approx 0,0554$ *sek*.

Aplikujúc metódu iterovania hodnôt sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{1, 1, 1, 1, 1, 1, 2\}$,
- konvergencia nastala po $j = 65$ iteráciach (vrátane nultej iterácie). Postupné hodnoty Q-funkcie pre možné akcie sú uvedené v Prílohe B,
- celkový čas výpočtu bol $t \approx 0,0425$ *sek*.

Aplikujúc metódu SARSA sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{1, 1, 1, 1, 1, 1, 2\}$,
- celkový čas výpočtu bol $t \approx 21,682$ *sek*.

Aplikujúc metódu Q-učenia sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{1, 1, 1, 1, 1, 1, 2\}$,
- celkový čas výpočtu bol $t \approx 22,5$ *sek*.

Aplikujúc metódu Monte Carlo simulácií sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{1, 1, 1, 1, 1, 1, 2\}$,

- celkový čas výpočtu bol $t \approx 50,355\ sek$.

Aplikujúc metódu Dyna-Q sme dospeli k nasledujúcim výsledkom:

- optimálna stratégia je $h^*(x) = \{1, 1, 1, 1, 1, 1, 2\}$,
- celkový čas výpočtu bol $t \approx 18,47\ sek$.

Je jednoduché odčítať, že všetky RL algoritmy vyústili v rovnaký výsledok ako zvolená numerická metóda stochastického OR uvedená v Časti 4.3.1.

4.5 Porovnanie použitých prístupov

Nahliadnuc na aplikovanie numerickej metódy stochastického optimálneho riadenia (viď Časť 4.3.1) a na aplikovanie šiestich rôznych algoritmov reinforcement learningu (viď Časť 4.4) môžeme tieto odlišné prístupy porovnať.

Ako sme už uviedli na konci predchádzajúcej kapitoly, tak všetky z použitých RL algoritmov dali rovnaký (a teda správny) výsledok ako pri použití metódy stochastického OR. Táto informácia nie je prekvapujúca pri aplikovaní RL algoritmov iterovania hodnôt a iterovania stratégií, keďže tieto metódy predstavujú metódy dynamického programovania mierne modifikované na účely RL. Avšak v prípadoch algoritmov SARSA, Monte Carlo simulácií a Q-učenia nebolo dopredu jasné, či ich výsledok bude zhodný s výsledkom pri aplikovaní metódy stochastického OR. A to z dôvodu, že tieto tri algoritmy sa typicky používajú pri úlohách, ktoré neobsahujú v zadaní informáciu o modeli. Zároveň algoritmy SARSA, MC simulácie a Q-učenie k výslednej stratégií konvergujú a pri nesprávnej kalibrácii vstupných parametrov nemusia skonvergovať vôbec, resp. nemusia skonvergovať k správnej optimálnej stratégii. Posledným použitým RL algoritmom bol algoritmus Dyna-Q. Keďže sme ho používali vo verzii, kde medzi vstupné údaje patril aj model sledovaného systému, tak tento algoritmus mal predpoklad vyústiť do správneho výsledku (resp. správnej optimálnej stratégie).

Z hľadiska výpočtového času možno za najlepší prístup vyhodnotiť tradičnú numerickú metódu stochastického optimálneho riadenia. Za veľmi rýchle prístupy možno označiť taktiež iterovanie stratégií a iterovanie hodnôt. Dokonca v prípade, že by sme určili vyšší počet časových krokov k pri metóde stochastického OR, tak by algoritmy iterovania stratégií

a iterovania hodnôt boli z hľadiska výpočtového času lepšie ako numerická metóda OR. Na druhej strane, vidíme, že ostatné použité algoritmy RL v tejto štatistike výrazne zaostávajú. Je tomu tak najmä z dôvodu, že na zistenie odhadov Q-funkcie využívajú informácie zo simulovaných epizód, ktoré nie sú ukončované v terminálnych stavoch, ale dĺžka každej takejto epizódy je prednastavená, keďže množina stavov v úlohe o obchodnej prevádzke neobsahuje žiadne terminálne stavy. Ak by sme znížili prednastavenú dĺžku týchto epizód, poťažmo aj počet celkových simulácií, tak by sa výpočtový čas pri algoritmoch SARSA, MC simulácie, Q-učenia a Dyna-Q výrazne znížil. Avšak v takomto prípade by nebola zaručená konvergencia. Poznamenajme ešte, že algoritmus Dyna-Q je z hľadiska výpočtového času medzi skupinou algoritmov využívajúcich simulované epizódy najlepší, čo sa dalo očakávať vzhľadom na popis hlavnej myšlienky jeho tvorby popísaný v Podkapitole 3.6. Naopak najdlhší výpočet sa dial pri aplikovaní algoritmu Monte Carlo simulácií. A to z dôvodu, že pri tomto algoritme sa vylepšujú odhady Q-funkcie z kompletných (ukončených) epizód.

Ak by sme sa pozreli na vhodnosť algoritmov reinforcement learningu na úlohy stochastického optimálneho riadenia, tak jednoznačne najvhodnejšími sú algoritmy iterovania hodnôt, resp. iterovania stratégií. Dôvodom je, že tieto dva typy RL algoritmov sú postavené priamo na teórii dynamického programovania a teda priamo na úlohy, ktoré sú veľmi podobné úlohám diskrétneho optimálneho riadenia, či už stochastického, alebo deterministického. Takéto úlohy totižto obsahujú úplnú informáciu o modeli, ktorá je nevyhnutná pre možnosť použitia algoritmov iterovania stratégií a iterovania hodnôt. Za informáciu o modeli z hľadiska RL je možné označiť funkciu odmien a funkciu prechodu, ktoré z hľadiska OR korešpondujú s účelovou funkciou (keďže sa v týchto úlohách maximalizuje dlhodobý kumulatívny súčet odmien) a so stavovou rovnicou. Na druhej strane, ostatné aplikované algoritmy RL, t.j. SARSA, MC simulácie a Q-učenie, i keď v konečnom dôsledku skonvergovali k správnemu výsledku, tak z dôvodu, že sú stavané najmä na prípady s neúplnou informáciou o modeli, tak v prípade úloh optimálneho riadenia sú menej efektívne a viac výpočtovo a časovo náročné. Algoritmus Dyna-Q sme zaradili do kategórie algoritmov založených na modeli, ale tento algoritmus je pri úlohách OR menej efektívny ako algoritmy iterovania stratégií a iterovania hodnôt. Je tomu tak z dôvodu, že je určitým rozšírením algoritmu Q-učenia, ktorý zaraďujeme medzi RL algoritmy bez

informácie o modeli.

V tejto diplomovej práci sme sa venovali úlohám s malým počtom riadiacich, resp. stavových premenných. Preto vyššie uvedené porovnanie sa týke práve takýchto úloh optimálneho riadenia.

Avšak vychádzajúc z [18] by sa dalo povedať vo všeobecnosti, že v prípade úlohy, kde by množina stavov bola veľmi veľká, resp. obsahovala veľa prvkov, tak by sa algoritmy iterovania stratégií a iterovania hodnôt ukázali ako efektívnejší variant od tradičných metód stochastického optimálneho riadenia. Poznamenajme, že to či je vhodnejšie aplikovať iterovanie stratégií alebo iterovanie hodnôt, záleží na danej úlohe. Dokonca možno povedať, že v prípade rozsiahlej množiny stavov by sa ukázali ako efektvínejšie aj všetky ostatné uvedené reinforcement learning algoritmy. Dôvodom je fakt, že pri tradičných metódach OR by bolo potrebné použiť veľmi veľa časových krokov, t.j. $T >> 0$, aby sme sa dostali do kažgého stavu a tak vedeli, aké riadenie je pre daný stav vhodné. Práve v takejto situácii sú reinforcement algoritmy veľmi efektívne. Poskytujú totiž informáciu o tom, čo máme robiť vždy, keď sa nachádzame v danom stave. Na druhej strane, ak by úloha nebola autonómna, tak by bolo potrebné RL algoritmy modifikovať a v tomto prípade sa ako vhodnejšie ukazujú typické metódy optimálneho riadenia. Poznamenajme, že metódy optimálneho riadenia sa javia ako vhodnejšie aj pre úlohy, v ktorých parametre sú také, že hodnoty Q-funkcie, resp. odhady týchto hodnôt, veľmi pomaly konvergujú. V takomto prípade by dokonca mohli algoritmy bez informácie o modeli dávať nesprávny výsledok.

Nahliadnuc na úlohy, kde je naopak veľmi rozsiahla množina možných riadení (resp. akcií), tak sa ako vhodnejšie ukazujú metódy optimálneho riadenia a RL algoritmy SARSA, Monte Carlo simulácie, Q-učenie a Dyna-Q. Dôvodom je fakt, že pri algoritmoch iterovania hodnôt a iterovania simulácií je potrebné si pamätať hodnoty Q-funkcie pre všetky páry stav-akcia po dvoch po sebe idúcich časových krokoch (resp. iteráciach), čo zaberá veľa pamäte. Pri metódach OR stačí mať uloženú hodnotu pre najlepšie riadenie (resp. akciu) a pri ostatných spomenutých RL algoritmoch (t.j. SARSA, MC simulácie, Q-učenie a Dyna-Q) sú v pamäti uložené odhady hodnôt Q-funkcii pre všetky páry stav-akcia, ktoré sa postupne sledovanými epizódami aktualizujú, resp. vylepšujú.

Záver

V tejto diplomovej práce sme sa zaoberali algoritmami reinforcement learningu, ich popísaním a implementovaním na vzorovom príklade rovnako ako ich aplikáciou na úlohu, ktorá je typická pre vedný odbor stochastického optimálneho riadenia. Snažili sme sa ukázať, že RL algoritmy ponúkajú alternatívny prístup na riešenie určitého typu problémov optimálneho riadenia.

V prvej kapitole sme sa zamerali na predstavenie reinforcement learningu ako takého. Predstavili sme jeho prepojenie s teóriou diskrétnych markovovských reťazcov a uviedli definície, ktoré sú pre RL typické.

V druhej kapitole sme sa zaoberali úlohou o prechode robota v miestnosti, ktorú pokladáme za vhodnú z hľadiska implementovania reinforcement learning algoritmov. Túto úlohu sme zároveň v Kapitole 2 popísali metodikou predstavenou v Kapitole 1. Na záver tejto kapitoly sme naformulovali úlohu o prechode robota v miestnosti vo forme štandardnej úlohy optimálneho riadenia.

V Kapitole 3 sme uviedli prehľad vybraných metód RL. Na začiatku kapitoly sme čitateľovi ponúkli všeobecnú kategorizáciu RL algoritmov z viacerých pohľadov. V ďalšej časti sme Bellmanov princíp optimality, resp. Bellmanovu rovnice dynamického programovania, interpretovali v reči premenných a funkcií popísaných v Kapitole 1. Tým sme položili základný stavebný kameň pre odprezentovanie zvolených reinforcement learning algoritmov: iterovanie stratégií, iterovanie hodnôt, SARSA, Q-učenie, Monte Carlo simulácie a Dyna-Q. Zároveň sme po popísaní každej zo spomenutých metód poskytli nahliadnutie na implementáciu daného algoritmu na vzorový problém o prechode robota v miestnosti. Pri každej takejto aplikácii sme uviedli hodnoty parametrov, ktoré sme použili, rovnako ako aj myšlienky, s ktorými boli tieto hodnoty volené.

V poslednej kapitole sme sa zamerali na porovnanie metódy stochastického optimálneho riadenia a algoritmov reinforcement learningu. Toto porovnanie sme vykonali na príklade typickom pre stochastické OR, a to na tzv. úlohe o obchodnej prevádzke popísanej v Časti 4.1. Aj pre túto úlohu sme poskytli čitateľovi popis v reči metodiky uvedenej v Kapitole 1, rovnako ako pri úlohe o prechode robota v miestnosti v Kapitole 2. Najprv sme zvolenú úlohu o obchodnej prevádzke vyriešili pomocou numerickej metódy stochastického optimálneho riadenia a neskôr aj pomocou aplikovania všetkých vyššie

spomenutých algoritmov reinforcement learningu. Na záver tejto kapitoly sme ponúkli čitateľovi porovnanie použitých prístupov nahliadnuc na úlohy, ktoré sme v tejto diplomovej práci riešili. Zároveň sme uviedli vhodnosť týchto prístupov v špecifických problémoch vo všeobecnosti. Možno konštatovať, že RL algoritmy sú skutočne vhodnou alternatívou metód OR v určitých prípadoch, najmä v prípadoch, keď v úlohe figuruje veľké množstvo stavových premenných alebo zadanie úlohy neobsahuje kompletnú informáciu o účelovej funkcii alebo stavovej rovnici. Avšak na prípady, keď je účelová funkcia neautonómna, resp. riadenie je spojité, sú vhodnejšie tradičné metódy optimálneho riadenia, i keď sú aj tieto prípady riešiteľné pomocou RL algoritmov.

Prínosom tejto práce je predstavenie reinforcement learningu v oblasti optimálneho riadenia a zodpovedanie otázky, či je RL alternatívou pre určité typy úloh typické v OR. To sme vykonali v zmysle prehľadného spracovania a popisu zvolených RL algoritmov rovnako ako aj v ich implemetovaní na zvolené problémy, pomocou čoho sme v záverečnej fáze porovnali všetky použité prístupy a ohodnotili ich vhodnosť.

Zoznam použitej literatúry

[1] Barto, A. G., Sutton, R. S.: *A temporal-difference model of classical conditioning*, Proceedings of the Ninth Annual Conference of the Cognitive Science Society, pp. 355-378, 1987

[2] Barto, A. G., Sutton, R. S.: *Goal seeking components for adaptive intelligence: An initial assessment*, Air Force Wright Aeronautical Laboratories Technical Report AFWAL-TR-81-1070. Wright-Patterson Air Force Base, Ohio 45433, 1981

[3] Barto, A. G., Sutton, R. S.: *Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element*, Behavioral Brain Research 4:221-235, 1982

[4] Baijayanta, D.: *Temporal-Difference (TD) Learning*, dostupné na webe (28.4.2021), https://towardsdatascience.com/temporal-difference-learning-47b4a7205ca8

[5] Busoniu, L.: *Basic of reinforcement learning*, dostupné na webe (28.4.2021), http://busoniu.net/files/repository/tmlss18_handout.pdf

[6] Busoniu, L., Babuška, R., De Schutter, B., Ernst, D.: *Reinforcement learning and dynamic programming using function approximators*, CRC Press, 2010

[7] Halická, M., Brunovský, P., Jurča, P.: *Optimálne riadenie*, EPOS, Bratislava, 2009

[8] Jankovská, K., Kilianová S., Brunovský, P., Bokes, P.: *Markovovské reťazce a ich aplikácie*, EPOS, Bratislava, 2015

[9] Klein, D.: *Reinforcement learning*, dostupné na webe (28.4.2021), https://www.cis.upenn.edu/ cis519/fall2015/lectures/14_ReinforcementLearning.pdf

[10] Klopf, A. H.: *A Theoretical Neuronal Learning Mechanism That Predicts the Basic Categories of Classical Conditioning Phenomena*, Cellular Mechanisms of Conditioning and Behavioral Plasticity, pp. 213-217, 1988

[11] Klopf, A. H.: *Brain Function and Adaptive Systems - A Heterostatic Theory*, Air Force Cambridge Research Laboratories, Special Reports, No. 133, 1972

[12] Lauko, M.: *Stochastické diskrétne úlohy optimálneho riadenia s ohraničeniami na koncový stav*, dizertačná práca, Univerzita Komenského v Bratislave, 2013

[13] Lewis, F. L., Vrabie, D.: *Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control*, IEEE Circuits and Systems Magazine, March 2009, pp. 32-50

[14] Lopardo, G.: *Stochastic optimization*, dostupné na webe (28.4.2021), https://github.com/gianluigilopardo/Reinforcement-Learning

[15] Nutini, J.: *Monte Carlo Methods (Estimators, On-policy/Off-policy Learning)*, dostupné na webe (28.4.2021), https://www.cs.ubc.ca/labs/lci/mlrg/slides/mlrg_MonteCarlo.pdf

[16] Patchigolla, D.: *A gentle intro to Reinforcement Learning*, dostupné na webe (28.4.2021), https://medium.com/analytics-vidhya/a-gentle-intro-to-reinforcement-learning-4f925b42f31c

[17] Poole, D. L., Mackworth, A. K.: *Artificial Intelligence: Foundations of Computational Agents, 2nd Edition*, Cambridge University Press, 2017

[18] Powell, W. B.: *Reinforcement Learning and Stochastic Optimalization*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2019

[19] Shimkin, N.: *Lectures Notes: Reinforcement Learning – Basic Algorithms*, dostupné na webe (28.4.2021), https://webee.technion.ac.il/people/shimkin/LCS11/ch4_RL1.pdf

[20] Silver, D.: *Lectures on Reinforcement Learning*, dostupné na webe (28.4.2021), https://www.davidsilver.uk/teaching/

[21] Sturtevant, N.: *Introduction to Artificial Intelligence, Lecture 15: Reinforcement Learning*, dostupné na webe (28.4.2021), https://www.cs.du.edu/ sturtevant/s14-ai/Lecture15.pdf

[22] Sutton, R. S.: *Single channel theory: A neuronal theory of learning*, Brain Theory Newsletter 3, No. 3/4, pp. 72-75, 1978

[23] Sutton, R. S., Barto, A. G.: *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, Massachussets, 2014

[24] Szepesvári, C.: *Algorithms for Reinforcement Learning*, Morgan & Claypool Publishers, 2010

[25] Tovarňák, M.: *Reinforcement learning v personalizácii vzdelávacieho procesu*, diplomová práca, Technická univerzita v Košiciach, 2020

[26] Watkins, C. J. C. H.: *Learning from Delayed Rewards*, dizertačná práca, King's College, Cambridge, UK, 1989

[27] Wen, Z.: *Lecture: Reinforcement Learning*, dostupné na webe (28.4.2021), http://bicmr.pku.edu.cn/ wenzw/bigdata/lect-RL.pdf

Príloha A

1 Aplikácie RL algoritmov na úlohu o prechode robota v miestnosti popísanej v Kapitole 2 v systéme Matlab

```
% FUNKCIA PRECHODU PRE VZOROVY PRIKLAD S MRIEZKOU

function[pst_prechodu_mriezka] = funkcia_prechodu_mriezka(stav_1, u, stav_2)
    % akcia 'Hore'
    if (stav_1 == stav_2+4 && u == 'H')
        pst_prechodu_mriezka = 0.8;
    elseif (stav_1 == stav_2 && u == 'H') && (stav_1 == 2 || stav_1 == 3 || stav_1 == 10)
        pst_prechodu_mriezka = 0.8;
    elseif (((stav_1 == stav_2+1 && stav_1 ~= 5 && stav_1 ~= 9) || (stav_1 == stav_2-1))
       && u == 'H')
        pst_prechodu_mriezka = 0.1;
    elseif (stav_1 == stav_2 && u == 'H') && (stav_1 == 1)
        pst_prechodu_mriezka = 0.9;
    elseif (stav_1 == stav_2 && u == 'H') && (stav_1 == 5)
        pst_prechodu_mriezka = 0.2;
    elseif (stav_1 == stav_2 && u == 'H') && (stav_1 == 7 || stav_1 == 9 || stav_1 == 12)
        pst_prechodu_mriezka = 0.1;
    else
        pst_prechodu_mriezka = 0;
    end

    % akcia 'Dole'
    if (stav_1 == stav_2-4 && u == 'D')
        pst_prechodu_mriezka = 0.8;
    elseif (stav_1 == stav_2 && u == 'D') && (stav_1 == 2 || stav_1 == 10 || stav_1 ==
       11)
        pst_prechodu_mriezka = 0.8;
    elseif (((stav_1 == stav_2+1 && stav_1 ~= 5 && stav_1 ~= 9) || (stav_1 == stav_2-1))
       && u == 'D')
        pst_prechodu_mriezka = 0.1;
    elseif (stav_1 == stav_2 && u == 'D') && (stav_1 == 9 || stav_1 == 12)
        pst_prechodu_mriezka = 0.9;
    elseif (stav_1 == stav_2 && u == 'D') && (stav_1 == 5)
        pst_prechodu_mriezka = 0.2;
    elseif (stav_1 == stav_2 && u == 'D') && (stav_1 == 1 || stav_1 == 7)
        pst_prechodu_mriezka = 0.1;
    end

    % akcia 'Pravo'
    if (stav_1 == stav_2-1 && u == 'P')
        pst_prechodu_mriezka = 0.8;
    elseif (stav_1 == stav_2 && u == 'P') && (stav_1 == 5)
```

```
        pst_prechodu_mriezka = 0.8;
    elseif (((stav_1 == stav_2+4) || (stav_1 == stav_2-4)) && u == 'P')
        pst_prechodu_mriezka = 0.1;
    elseif (stav_1 == stav_2 && u == 'P') && (stav_1 == 12)
        pst_prechodu_mriezka = 0.9;
    elseif (stav_1 == stav_2 && u == 'P') && (stav_1 == 2 || stav_1 == 10)
        pst_prechodu_mriezka = 0.2;
    elseif (stav_1 == stav_2 && u == 'P') && (stav_1 == 1 || stav_1 == 3 || stav_1 == 9
        || stav_1 == 11)
        pst_prechodu_mriezka = 0.1;
    end

    % akcia 'Lavo'
    if (stav_1 == stav_2+1 && stav_1 ~= 5 && stav_1 ~= 9 && u == 'L')
        pst_prechodu_mriezka = 0.8;
    elseif (stav_1 == stav_2 && u == 'L') && (stav_1 == 5 || stav_1 == 7)
        pst_prechodu_mriezka = 0.8;
    elseif (((stav_1 == stav_2+4) || (stav_1 == stav_2-4)) && u == 'L')
        pst_prechodu_mriezka = 0.1;
    elseif (stav_1 == stav_2 && u == 'L') && (stav_1 == 1 || stav_1 == 9)
        pst_prechodu_mriezka = 0.9;
    elseif (stav_1 == stav_2 && u == 'L') && (stav_1 == 2 || stav_1 == 10)
        pst_prechodu_mriezka = 0.2;
    elseif (stav_1 == stav_2 && u == 'L') && (stav_1 == 3 || stav_1 == 11 || stav_1 ==
        12)
        pst_prechodu_mriezka = 0.1;
    end
end
```

```
% FUNKCIA ODMIEN PRE VZOROVY PRIKLAD S MRIEZKOU

function[odmena] = funkcia_odmien_mriezka(stav_1, stav_2)
    cena_za_pohyb = -0.07;

    odmena = cena_za_pohyb;
end
```

```
% iterovanie hodnot (value iteration), dynamicke programovanie
% vzorova uloha s mriezkou

clc;
clear all;

tic;

% DEFINOVANIE ZADANYCH VELICIN
X = [1 2 3 4;
    5 6 7 8;
```

```
12      9  10  11  12];                          % mriezka stavov (matica)
13  X(2,2) = -Inf;                              % miesto, kde sa neda ist = stena
14
15  velkost_X = size(X);                        % velkost X
16
17  x = (1:12);                                 % vytvorenie vektora, ktory prislucha
18  x(6) = -Inf;                                % matici X, bude sa pouzivat neskor
19                                              % v programe
20
21  u = ['H', 'D', 'P', 'L'];                   % vektor riadeni
22
23  v = zeros(3, 4);                            % matica odmien (pozitivnych, aj negativnych)
24  v(1,4) = 1;                                 % uvedene len informativne
25  v(2,4) = -1;
26
27  gama = 1;                                   % diskontny faktor
28
29  %%%%
```

```
60          Q_novy_h(4) = 1;
61          Q_novy_h(8) = -1;
62          Q_novy_d = zeros(1, (velkost_X(1))*(velkost_X(2)));
63          Q_novy_d(4) = 1;
64          Q_novy_d(8) = -1;
65          Q_novy_p = zeros(1, (velkost_X(1))*(velkost_X(2)));
66          Q_novy_p(4) = 1;
67          Q_novy_p(8) = -1;
68          Q_novy_l = zeros(1, (velkost_X(1))*(velkost_X(2)));
69          Q_novy_l(4) = 1;
70          Q_novy_l(8) = -1;
71
72          Q_H = Q_novy_h;
73          Q_D = Q_novy_d;
74          Q_P = Q_novy_p;
75          Q_L = Q_novy_l;
76      else
77          Q_H = [Q_H; Q_novy_h];
78          Q_D = [Q_D; Q_novy_d];
79          Q_P = [Q_P; Q_novy_p];
80          Q_L = [Q_L; Q_novy_l];
81      end
82
83      % vynulovanie
84      Q_stary_h = Q_novy_h;
85      Q_novy_h = zeros(1, (velkost_X(1))*(velkost_X(2)));
86      Q_novy_h(4) = 1;
87      Q_novy_h(8) = -1;
88      Q_stary_d = Q_novy_d;
89      Q_novy_d = zeros(1, (velkost_X(1))*(velkost_X(2)));
90      Q_novy_d(4) = 1;
91      Q_novy_d(8) = -1;
92      Q_stary_p = Q_novy_p;
93      Q_novy_p = zeros(1, (velkost_X(1))*(velkost_X(2)));
94      Q_novy_p(4) = 1;
95      Q_novy_p(8) = -1;
96      Q_stary_l = Q_novy_l;
97      Q_novy_l = zeros(1, (velkost_X(1))*(velkost_X(2)));
98      Q_novy_l(4) = 1;
99      Q_novy_l(8) = -1;
100
101     % 'Hore' vetva
102     for a = 1:length(x)
103         if a ~= 4 && a ~= 6 && a ~= 8
104             for i = 1:velkost_X(1)
105                 for j = 1:velkost_X(2)
106                     if X(i,j) ~= -Inf
107                         if Q_stary_h(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i,j))
```

```
                , Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
108                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H', X
                        (i,j))*...
109                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X(i,j
                            )));
110             elseif Q_stary_d(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                    ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
111                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H', X
                        (i,j))*...
112                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X(i,j
                            )));
113             elseif Q_stary_p(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                    ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
114                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H', X
                        (i,j))*...
115                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X(i,j
                            )));
116             elseif Q_stary_l(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                    ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
117                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H', X
                        (i,j))*...
118                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X(i,j
                            )));
119             end
120         end
121     end
122 end
123 end
124
125 % 'Dole' vetva
126 for a = 1:length(x)
127     if a ~= 4 && a ~= 6 && a ~= 8
128         for i = 1:velkost_X(1)
129             for j = 1:velkost_X(2)
130                 if X(i,j) ~= -Inf
131                     if Q_stary_h(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i,j))
                            , Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
132                         Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D', X
                                (i,j))*...
133                             (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X(i,j
                                    )));
134                     elseif Q_stary_d(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                            ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
135                         Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D', X
                                (i,j))*...
136                             (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X(i,j
                                    )));
137
```

```
138                         elseif Q_stary_p(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
139                             Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D', X
                                    (i,j))*...
140                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X(i,j
                                        )));
141                         elseif Q_stary_l(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
142                             Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D', X
                                    (i,j))*...
143                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X(i,j
                                        )));
144                         end
145                     end
146                 end
147             end
148         end
149     end
150
151     % 'Pravo' vetva
152     for a = 1:length(x)
153         if a ~= 4 && a ~= 6 && a ~= 8
154             for i = 1:velkost_X(1)
155                 for j = 1:velkost_X(2)
156                     if X(i,j) ~= -Inf
157                         if Q_stary_h(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i,j))
                                , Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
158                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P', X
                                    (i,j))*...
159                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X(i,j
                                        )));
160                         elseif Q_stary_d(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
161                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P', X
                                    (i,j))*...
162                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X(i,j
                                        )));
163                         elseif Q_stary_p(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
164                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P', X
                                    (i,j))*...
165                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X(i,j
                                        )));
166                         elseif Q_stary_l(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
167                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P', X
                                    (i,j))*...
168                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X(i,j
```

```
                                        )));
169                                 end
170                             end
171                         end
172                     end
173                 end
174             end
175
176             % 'Lavo' vetva
177             for a = 1:length(x)
178                 if a ~= 4 && a ~= 6 && a ~= 8
179                     for i = 1:velkost_X(1)
180                         for j = 1:velkost_X(2)
181                             if X(i,j) ~= -Inf
182                                 if Q_stary_h(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i,j))
                                        , Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
183                                     Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L', X
                                            (i,j))*...
184                                         (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X(i,j
                                                )));
185                                 elseif Q_stary_d(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                        ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
186                                     Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L', X
                                            (i,j))*...
187                                         (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X(i,j
                                                )));
188                                 elseif Q_stary_p(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                        ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
189                                     Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L', X
                                            (i,j))*...
190                                         (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X(i,j
                                                )));
191                                 elseif Q_stary_l(X(i,j)) == max([Q_stary_h(X(i,j)), Q_stary_d(X(i
                                        ,j)), Q_stary_p(X(i,j)), Q_stary_l(X(i,j))])
192                                     Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L', X
                                            (i,j))*...
193                                         (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X(i,j
                                                )));
194                                 end
195                             end
196                         end
197                     end
198                 end
199             end
200 end
201
202 % inicializovanie vektora optimalnej strategie
203 pol_opt = ['a', 'a', 'a', 't', 'a', 'a', 'a', 't', 'a', 'a', 'a', 'a'];
```

```
204
205 % optimalna strategia
206 for i = 1:length(Q_stary_h)
207     if i ~= 4 && i ~= 6 && i ~= 8
208         if Q_novy_h(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)])
209             pol_opt(i) = 'H';
210         elseif Q_novy_d(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)])
211             pol_opt(i) = 'D';
212         elseif Q_novy_l(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)])
213             pol_opt(i) = 'L';
214         elseif Q_novy_p(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)])
215             pol_opt(i) = 'P';
216         end
217     end
218 end
219
220 pol_opt
221
222 toc;
```

```
1  % iterovanie strategii (policy iteration), dynamicke programovanie
2  % vzorova uloha s mriezkou
3
4  clc;
5  clear all;
6
7  tic;
8
9  % DEFINOVANIE ZADANYCH VELICIN
10 X = [1  2  3  4;
11     5  6  7  8;
12     9 10 11 12];                    % mriezka stavov (matica)
13 X(2,2) = -Inf;                      % miesto, kde sa neda ist = stena
14
15 velkost_X = size(X);                % velkost X
16
17 x = (1:12);                         % vytvorenie vektora, ktory prislucha
18 x(6) = -Inf;                        % matici X, bude sa pouzivat neskor
19                                     % v programe
20
21 u = ['H', 'D', 'P', 'L'];           % vektor riadeni
22
23 v = zeros(3, 4);                    % matica odmien (pozitivnych, aj negativnych)
24 v(1,4) = 1;                         % uvedene len informativne
25 v(2,4) = -1;
26
27 gama = 1;                           % diskontny faktor
```

```
28
29 %%%%
```

```
75              Q_novy_d(4) = 1;
76              Q_novy_d(8) = -1;
77              Q_novy_p = zeros(1, (velkost_X(1))*(velkost_X(2)));
78              Q_novy_p(4) = 1;
79              Q_novy_p(8) = -1;
80              Q_novy_l = zeros(1, (velkost_X(1))*(velkost_X(2)));
81              Q_novy_l(4) = 1;
82              Q_novy_l(8) = -1;
83
84              Q_H = Q_novy_h;
85              Q_D = Q_novy_d;
86              Q_P = Q_novy_p;
87              Q_L = Q_novy_l;
88          else
89              Q_H = [Q_H; Q_novy_h];
90              Q_D = [Q_D; Q_novy_d];
91              Q_P = [Q_P; Q_novy_p];
92              Q_L = [Q_L; Q_novy_l];
93          end
94
95          % vynulovanie
96          Q_stary_h = Q_novy_h;
97          Q_novy_h = zeros(1, (velkost_X(1))*(velkost_X(2)));
98          Q_novy_h(4) = 1;
99          Q_novy_h(8) = -1;
100         Q_stary_d = Q_novy_d;
101         Q_novy_d = zeros(1, (velkost_X(1))*(velkost_X(2)));
102         Q_novy_d(4) = 1;
103         Q_novy_d(8) = -1;
104         Q_stary_p = Q_novy_p;
105         Q_novy_p = zeros(1, (velkost_X(1))*(velkost_X(2)));
106         Q_novy_p(4) = 1;
107         Q_novy_p(8) = -1;
108         Q_stary_l = Q_novy_l;
109         Q_novy_l = zeros(1, (velkost_X(1))*(velkost_X(2)));
110         Q_novy_l(4) = 1;
111         Q_novy_l(8) = -1;
112
113         % 'Hore' vetva
114         for a = 1:length(x)
115             if a ~= 4 && a ~= 6 && a ~= 8
116                 for i = 1:velkost_X(1)
117                     for j = 1:velkost_X(2)
118                         if X(i,j) ~= -Inf
119                             if pol_nova(X(i,j)) == 'H' || pol_nova(X(i,j)) == 't'
120                                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H
                                    ', X(i,j))*...
121                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X
```

```
                                    (i,j)));
122                             elseif pol_nova(X(i,j)) == 'D'
123                                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H
                                        ', X(i,j))*...
124                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X
                                        (i,j)));
125                             elseif pol_nova(X(i,j)) == 'P'
126                                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H
                                        ', X(i,j))*...
127                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X
                                        (i,j)));
128                             elseif pol_nova(X(i,j)) == 'L'
129                                 Q_novy_h(a) = Q_novy_h(a) + funkcia_prechodu_mriezka(a,'H
                                        ', X(i,j))*...
130                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X
                                        (i,j)));
131                             end
132                         end
133                     end
134                 end
135             end
136         end
137
138         % 'Dole' vetva
139         for a = 1:length(x)
140             if a ~= 4 && a ~= 6 && a ~= 8
141                 for i = 1:velkost_X(1)
142                     for j = 1:velkost_X(2)
143                         if X(i,j) ~= -Inf
144                             if pol_nova(X(i,j)) == 'H' || pol_nova(X(i,j)) == 't'
145                                 Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D
                                        ', X(i,j))*...
146                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X
                                        (i,j)));
147                             elseif pol_nova(X(i,j)) == 'D'
148                                 Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D
                                        ', X(i,j))*...
149                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X
                                        (i,j)));
150                             elseif pol_nova(X(i,j)) == 'P'
151                                 Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D
                                        ', X(i,j))*...
152                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X
                                        (i,j)));
153                             elseif pol_nova(X(i,j)) == 'L'
154                                 Q_novy_d(a) = Q_novy_d(a) + funkcia_prechodu_mriezka(a,'D
                                        ', X(i,j))*...
155                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X
```

```
                                        (i,j)));
156                         end
157                     end
158                 end
159             end
160         end
161     end
162
163     % 'Pravo' vetva
164     for a = 1:length(x)
165         if a ~= 4 && a ~= 6 && a ~= 8
166             for i = 1:velkost_X(1)
167                 for j = 1:velkost_X(2)
168                     if X(i,j) ~= -Inf
169                         if pol_nova(X(i,j)) == 'H' || pol_nova(X(i,j)) == 't'
170                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P
                                ', X(i,j))*...
171                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X
                                    (i,j)));
172                         elseif pol_nova(X(i,j)) == 'D'
173                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P
                                ', X(i,j))*...
174                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X
                                    (i,j)));
175                         elseif pol_nova(X(i,j)) == 'P'
176                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P
                                ', X(i,j))*...
177                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X
                                    (i,j)));
178                         elseif pol_nova(X(i,j)) == 'L'
179                             Q_novy_p(a) = Q_novy_p(a) + funkcia_prechodu_mriezka(a,'P
                                ', X(i,j))*...
180                                 (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X
                                    (i,j)));
181                         end
182                     end
183                 end
184             end
185         end
186     end
187
188     % 'Lavo' vetva
189     for a = 1:length(x)
190         if a ~= 4 && a ~= 6 && a ~= 8
191             for i = 1:velkost_X(1)
192                 for j = 1:velkost_X(2)
193                     if X(i,j) ~= -Inf
194                         if pol_nova(X(i,j)) == 'H' || pol_nova(X(i,j)) == 't'
```

```
195                                 Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L
                                        ', X(i,j))*...
196                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_h(X
                                            (i,j)));
197                             elseif pol_nova(X(i,j)) == 'D'
198                                 Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L
                                        ', X(i,j))*...
199                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_d(X
                                            (i,j)));
200                             elseif pol_nova(X(i,j)) == 'P'
201                                 Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L
                                        ', X(i,j))*...
202                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_p(X
                                            (i,j)));
203                             elseif pol_nova(X(i,j)) == 'L'
204                                 Q_novy_l(a) = Q_novy_l(a) + funkcia_prechodu_mriezka(a,'L
                                        ', X(i,j))*...
205                                     (funkcia_odmien_mriezka(a, X(i,j)) + gama*Q_stary_l(X
                                            (i,j)));
206                             end
207                         end
208                     end
209                 end
210             end
211         end
212     end
213
214     % nova strategia
215     pol_stara = pol_nova;
216     pol_nova = ['a', 'a', 'a', 't', 'a', 'a', 'a', 't', 'a', 'a', 'a', 'a'];
217     for i = 1:length(Q_stary_h)
218         if i ~= 4 && i ~= 6 && i ~= 8
219             if Q_novy_h(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)])
220                 pol_nova(i) = 'H';
221             elseif Q_novy_d(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)
                        ])
222                 pol_nova(i) = 'D';
223             elseif Q_novy_l(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)
                        ])
224                 pol_nova(i) = 'L';
225             elseif Q_novy_p(i) == max([Q_novy_h(i), Q_novy_d(i), Q_novy_p(i), Q_novy_l(i)
                        ])
226                 pol_nova(i) = 'P';
227             end
228         end
229     end
230     pol_stara
231     pol_nova
```

```
end

pol_opt = pol_nova

toc;
```

```
    % FUNKCIA PRECHODU PRE VZOROVY PRIKLAD S MRIEZKOU

function[stav_2] = prechod_mriezka(stav_1, u)
    % URCENIE TOHO, AKA AKCIA SA V SKUTOCNOSTI VYKONA
    % matica pravdepodobnosti prechodu pre dane akcie
    P = [0.8 0 0.1 0.1;
         0 0.8 0.1 0.1;
         0.1 0.1 0.8 0;
         0.1 0.1 0 0.8];
    skutocna_akcia = randsample(1:4, 1, true, P(u,:));

    % akcia 'Hore'
    if skutocna_akcia == 1 && (stav_1 == 1 || stav_1 == 2 || stav_1 == 3 || stav_1 == 10)
        stav_2 = stav_1;
    elseif skutocna_akcia == 1 && (stav_1 ~= 1 && stav_1 ~= 2 && stav_1 ~= 3 && stav_1 ~=
        10)
        stav_2 = stav_1-4;
    end

    % akcia 'Dole'
    if skutocna_akcia == 2 && (stav_1 == 2 || stav_1 == 9 || stav_1 == 10 || stav_1 == 11
        || stav_1 == 12)
        stav_2 = stav_1;
    elseif skutocna_akcia == 2 && (stav_1 ~= 2 && stav_1 ~= 9 && stav_1 ~= 10 && stav_1
        ~= 11 || stav_1 ~= 12)
        stav_2 = stav_1+4;
    end

    % akcia 'Pravo'
    if skutocna_akcia == 3 && (stav_1 == 5 || stav_1 == 12)
        stav_2 = stav_1;
    elseif skutocna_akcia == 3 && (stav_1 ~= 5 && stav_1 ~= 12)
        stav_2 = stav_1+1;
    end

    % akcia 'Lavo'
    if skutocna_akcia == 4 && (stav_1 == 1 || stav_1 == 5 || stav_1 == 7 || stav_1 == 9)
        stav_2 = stav_1;
    elseif skutocna_akcia == 4 && (stav_1 ~= 1 && stav_1 ~= 5 && stav_1 ~= 7 && stav_1 ~=
        9)
        stav_2 = stav_1-1;
    end
```

```
end
```

```
% FUNKCIA ODMIEN PRE VZOROVY PRIKLAD S MRIEZKOU

function[odmena] = odmeny_mriezka(stav_1, stav_2)
    cena_za_pohyb = -0.07;

    if stav_2 == 4
        odmena = 1 + cena_za_pohyb;
    elseif stav_2 == 8
        odmena = -1 + cena_za_pohyb;
    else
        odmena = cena_za_pohyb;
    end
end
```

```
% Temporal differences, TD SARSA
% vzorova uloha s mriezkou

clc;
clear all;

tic;

% DEFINOVANIE ZADANYCH VELICIN
X = [1 2 3 4;
    5 6 7 8;
    9 10 11 12];                    % mriezka stavov (matica)
X(2,2) = -Inf;                      % miesto, kde sa neda ist = stena

u = ['H', 'D', 'P', 'L'];           % vektor riadeni
                                    % len informativne
                                    % v programe pouzite ako [1, 2, 3, 4]

gama = 1;                           % diskontny faktor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%    Temporal differences metoda SARSA
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% exploration - exploitation rozdelenie pravdepodobnosti
% epsilon
epsilon = 0.8;

% learning rate
alfa = 0.08;
```

```
33
34 % inicializovanie matice prisluchajucim danym akciam v danom stave
35 % na zaciatku nastavene na 0 (12 moznych stavov x 4 mozne akcie)
36 Q = zeros(12,4);
37
38 for l = 1:30000
39         % vygenerovanie prveho stavu
40         s_pov = randi([1,9]);
41         if s_pov == 1 || s_pov == 2 || s_pov == 3
42             aktualny_stav = s_pov;
43         elseif s_pov == 4
44             aktualny_stav = 5;
45         elseif s_pov == 5
46             aktualny_stav = 7;
47         elseif s_pov == 6
48             aktualny_stav = 9;
49         elseif s_pov == 7
50             aktualny_stav = 10;
51         elseif s_pov == 8
52             aktualny_stav = 11;
53         elseif s_pov == 9
54             aktualny_stav = 12;
55         end
56
57         % vygenerovanie akcie pomocou eps-greedy strategy
58         if(rand > epsilon)
59             if Q(aktualny_stav, 1) == max([Q(aktualny_stav, 1), Q(aktualny_stav, 2), Q(
                   aktualny_stav, 3), Q(aktualny_stav, 4)])
60                 aktualna_akcia = 1;
61             elseif Q(aktualny_stav, 2) == max([Q(aktualny_stav, 1), Q(aktualny_stav, 2),
                   Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
62                 aktualna_akcia = 2;
63             elseif Q(aktualny_stav, 3) == max([Q(aktualny_stav, 1), Q(aktualny_stav, 2),
                   Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
64                 aktualna_akcia = 3;
65             elseif Q(aktualny_stav, 4) == max([Q(aktualny_stav, 1), Q(aktualny_stav, 2),
                   Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
66                 aktualna_akcia = 4;
67             end
68         else
69             aktualna_akcia = randi([1 4],1);
70         end
71
72         % inicializovanie retazca stavov
73         epizoda_stavy = aktualny_stav;
74
75         % aplikovanie strategie
76         while aktualny_stav ~= 4 && aktualny_stav ~= 8
```

```
77              nasledujuci_stav = prechod_mriezka(aktualny_stav, aktualna_akcia);
78
79              % epsilon-greedy strategy
80              if(rand > epsilon)
81                  if Q(nasledujuci_stav, 1) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
82                      nasledujuca_akcia = 1;
83                  elseif Q(nasledujuci_stav, 2) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
84                      nasledujuca_akcia = 2;
85                  elseif Q(nasledujuci_stav, 3) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
86                      nasledujuca_akcia = 3;
87                  elseif Q(nasledujuci_stav, 4) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
88                      nasledujuca_akcia = 4;
89                  end
90              else
91                  nasledujuca_akcia = randi([1 4],1);
92              end
93
94              epizoda_stavy = [epizoda_stavy, nasledujuci_stav];
95
96              % vyhodnotenie Q-funkcie
97              if nasledujuci_stav == 4 || nasledujuci_stav == 8
98                  Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) + ...
99                      alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
100                     (gama * 0) - Q(aktualny_stav, aktualna_akcia));
101             else
102                 Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) + ...
103                     alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
104                     (gama * Q(nasledujuci_stav, nasledujuca_akcia)) - Q(aktualny_stav,
                            aktualna_akcia));
105             end
106
107             aktualny_stav = nasledujuci_stav;
108             aktualna_akcia = nasledujuca_akcia;
109         end
110
111         % zmensenie epsilonu, pretoze uz staci "objavovat" v mensom rozsahu
112         epsilon = 0.9999*epsilon;
113
114         % zmensenie miery ucenia, pretoze uz sa staci "ucit" v mensom rozsahu
115         alfa = 0.9999*alfa;
```

```
end

% optimalna strategia
pol_nova = zeros(1, 12);
for o = 1:12
    if o ~= 4 && o ~= 6 && o ~= 8
        if Q(o, 1) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
            pol_nova(o) = 1;
        elseif Q(o, 2) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
            pol_nova(o) = 2;
        elseif Q(o, 3) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
            pol_nova(o) = 3;
        elseif Q(o, 4) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
            pol_nova(o) = 4;
        end
    end
end

pol_opt = pol_nova

toc;
```

```
% FUNKCIA VYNOSOV PRE VZOROVY PRIKLAD S MRIEZKOU PRI MONTE CARLO METODE
% IDE O VYNOSY PRE EPIZODU OD DANEHO BODU

function[vynos] = vynosy_MC_mriezka(epizoda_stavy)
    cena_za_pohyb = -0.07;                  % cena za pohyb
    gama = 1;                               % diskontny faktor
    vynos = 0;

    % otocenie vektora, pretoze vynosy budeme pocitat od konca
    otocene_stavy = flip(epizoda_stavy);

    % vyhodnotenie zvysku vynosov
    for i = 1:length(otocene_stavy)
        if otocene_stavy(i) == 4
            vynos = vynos+1;
        elseif otocene_stavy(i) == 8
            vynos = vynos-1;
        else
            vynos = cena_za_pohyb + gama*vynos;
        end
    end
end
```

```
% Monte Carlo metoda, with exploring start
% vzorova uloha s mriezkou

```

```
4  clc;
5  clear all;
6
7  tic;
8
9  % DEFINOVANIE ZADANYCH VELICIN
10 X = [1 2 3 4;
11     5 6 7 8;
12     9 10 11 12];                    % mriezka stavov (matica)
13 X(2,2) = -Inf;                      % miesto, kde sa neda ist = stena
14
15 u = ['H', 'D', 'P', 'L'];           % vektor riadeni
16                                     % len informativne
17                                     % v programe pouzite ako [1, 2, 3, 4]
18
19 N = 30000;                          % pocet simulacii
20
21 %%%%
```

```
52              aktualny_stav = s_pov;
53          elseif s_pov == 4
54              aktualny_stav = 5;
55          elseif s_pov == 5
56              aktualny_stav = 7;
57          elseif s_pov == 6
58              aktualny_stav = 9;
59          elseif s_pov == 7
60              aktualny_stav = 10;
61          elseif s_pov == 8
62              aktualny_stav = 11;
63          elseif s_pov == 9
64              aktualny_stav = 12;
65          end
66
67          % vygenerovanie prvej akcie
68          aktualna_akcia = randi([1,4]);
69
70          % POSTUPOVANIE POMOCOU STRATEGIE, KTORU MAM PRAVE ZVOLENU
71          % inicializovanie retazca stavov
72          epizoda_stavy = aktualny_stav;
73
74          % inicializovanie retazca akcii
75          epizoda_akcie = aktualna_akcia;
76
77          % aplikovanie strategie
78          while aktualny_stav ~= 4 && aktualny_stav ~= 8
79              aktualny_stav = prechod_mriezka(aktualny_stav, aktualna_akcia);
80              epizoda_stavy = [epizoda_stavy, aktualny_stav];
81
82              % epsilon-greedy strategy
83              if(rand > epsilon)
84                  aktualna_akcia = pol_nova(aktualny_stav);
85              else
86                  aktualna_akcia = randi([1 4],1);
87              end
88              epizoda_akcie = [epizoda_akcie, aktualna_akcia];
89          end
90
91          % zmensenie epsilonu, pretoze uz staci "objavovat" v mensom rozsahu
92          epsilon = 0.99999*epsilon;
93
94          % zistenie matic A a C pre danu epizodu
95          for k = 1:(length(epizoda_stavy)-1)
96              A(epizoda_stavy(k), epizoda_akcie(k)) = A(epizoda_stavy(k), epizoda_akcie(k))
                    + ...
97                  vynosy_MC_mriezka(epizoda_stavy(k:length(epizoda_stavy)));
98              C(epizoda_stavy(k), epizoda_akcie(k)) = C(epizoda_stavy(k), epizoda_akcie(k))
```

```
                + 1;
        end

        % vyhodnotenie Q(x,u) po N simulaciach
        for riadok = 1:12
            if riadok ~= 4 && riadok ~= 6 && riadok ~= 8
                for stlpec = 1:4
                    if C(riadok, stlpec) ~= 0
                        Q(riadok, stlpec) = A(riadok, stlpec)/C(riadok, stlpec);
                    end
                end
            end
        end
    end

    % nova strategia
    for o = 1:12
        if o ~= 4 && o ~= 6 && o ~= 8
            if Q(o, 1) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
                pol_nova(o) = 1;
            elseif Q(o, 2) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
                pol_nova(o) = 2;
            elseif Q(o, 3) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
                pol_nova(o) = 3;
            elseif Q(o, 4) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
                pol_nova(o) = 4;
            end
        end
    end
end

pol_opt = pol_nova

toc;
```

```
% Q-learning (off policy)
% vzorova uloha s mriezkou

clc;
clear all;

tic;

% DEFINOVANIE ZADANYCH VELICIN
X = [1 2 3 4;
    5 6 7 8;
    9 10 11 12];                    % mriezka stavov (matica)
X(2,2) = -Inf;                      % miesto, kde sa neda ist = stena
```

```
14
15 u = ['H', 'D', 'P', 'L'];                % vektor riadeni
16                                          % len informativne
17                                          % v programe pouzite ako [1, 2, 3, 4]
18
19 gama = 1;                                % diskontny faktor
20
21 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
22 %
23 %                 Q-learning
24 %
25 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26
27 % exploration - exploitation rozdelenie pravdepodobnosti
28 % epsilon
29 epsilon = 0.8;
30
31 % learning rate
32 alfa = 0.08;
33
34 % inicializovanie matice prisluchajucim danym akciam v danom stave
35 % na zaciatku nastavene na 0 (12 moznych stavov x 4 mozne akcie)
36 Q = zeros(12,4);
37
38 for l = 1:30000
39         % vygenerovanie prveho stavu (z rovnomerneho rozdelenia)
40         s_pov = randi([1,9]);
41         if s_pov == 1 || s_pov == 2 || s_pov == 3
42             aktualny_stav = s_pov;
43         elseif s_pov == 4
44             aktualny_stav = 5;
45         elseif s_pov == 5
46             aktualny_stav = 7;
47         elseif s_pov == 6
48             aktualny_stav = 9;
49         elseif s_pov == 7
50             aktualny_stav = 10;
51         elseif s_pov == 8
52             aktualny_stav = 11;
53         elseif s_pov == 9
54             aktualny_stav = 12;
55         end
56
57         % inicializovanie retazca stavov
58         epizoda_stavy = aktualny_stav;
59
60         % aplikovanie strategie
61         while aktualny_stav ~= 4 && aktualny_stav ~= 8
```

```
62              % epsilon-greedy strategy
63              if(rand > epsilon)
64                  if Q(aktualny_stav, 1) == max([Q(aktualny_stav, 1), Q(aktualny_stav, 2),
                        Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
65                      aktualna_akcia = 1;
66                  elseif Q(aktualny_stav, 2) == max([Q(aktualny_stav, 1), Q(aktualny_stav,
                        2), Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
67                      aktualna_akcia = 2;
68                  elseif Q(aktualny_stav, 3) == max([Q(aktualny_stav, 1), Q(aktualny_stav,
                        2), Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
69                      aktualna_akcia = 3;
70                  elseif Q(aktualny_stav, 4) == max([Q(aktualny_stav, 1), Q(aktualny_stav,
                        2), Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
71                      aktualna_akcia = 4;
72                  end
73              else
74                  aktualna_akcia = randi([1 4],1);
75              end
76
77              nasledujuci_stav = prechod_mriezka(aktualny_stav, aktualna_akcia);
78
79              epizoda_stavy = [epizoda_stavy, nasledujuci_stav];
80
81              % vyhodnotenie Q-funkcie
82              if nasledujuci_stav == 4 || nasledujuci_stav == 8
83                  Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) + ...
84                      alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
85                      (gama * 0) - Q(aktualny_stav, aktualna_akcia));
86              else
87                  if Q(nasledujuci_stav, 1) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
88                      Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                            ...
89                          alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
90                          (gama * Q(nasledujuci_stav, 1)) - Q(aktualny_stav, aktualna_akcia
                                ));
91                  elseif Q(nasledujuci_stav, 2) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
92                      Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                            ...
93                          alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
94                          (gama * Q(nasledujuci_stav, 2)) - Q(aktualny_stav, aktualna_akcia
                                ));
95                  elseif Q(nasledujuci_stav, 3) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
```

```
96                      Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                            ...
97                          alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
98                          (gama * Q(nasledujuci_stav, 3)) - Q(aktualny_stav, aktualna_akcia
                                ));
99                  elseif Q(nasledujuci_stav, 4) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
100                     Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                            ...
101                         alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
102                         (gama * Q(nasledujuci_stav, 4)) - Q(aktualny_stav, aktualna_akcia
                                ));
103                 end
104             end
105
106             aktualny_stav = nasledujuci_stav;
107         end
108
109         % zmensenie epsilonu, pretoze uz staci "objavovat" v mensom rozsahu
110         epsilon = 0.9999*epsilon;
111
112         % zmensenie miery ucenia, pretoze uz sa staci "ucit" v mensom rozsahu
113         alfa = 0.9999*alfa;
114 end
115
116 % optimalna strategia
117 pol_nova = zeros(1, 12);
118 for o = 1:12
119     if o ~= 4 && o ~= 6 && o ~= 8
120         if Q(o, 1) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
121             pol_nova(o) = 1;
122         elseif Q(o, 2) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
123             pol_nova(o) = 2;
124         elseif Q(o, 3) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
125             pol_nova(o) = 3;
126         elseif Q(o, 4) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
127             pol_nova(o) = 4;
128         end
129     end
130 end
131
132 pol_opt = pol_nova
133
134 toc;
```

```
1 % Dyna-Q
```

```
2  % vzorova uloha s mriezkou
3
4  clc;
5  clear all;
6
7  tic;
8
9  % DEFINOVANIE ZADANYCH VELICIN
10 X = [1  2  3  4;
11      5  6  7  8;
12      9  10 11 12];                    % mriezka stavov (matica)
13 X(2,2) = -Inf;                        % miesto, kde sa neda ist = stena
14
15 u = ['H', 'D', 'P', 'L'];             % vektor riadeni
16                                       % len informativne
17                                       % v programe pouzite ako [1, 2, 3, 4]
18
19 gama = 1;                             % diskontny faktor
20
21 %%%%
```

```
50              aktualny_stav = 10;
51          elseif s_pov == 8
52              aktualny_stav = 11;
53          elseif s_pov == 9
54              aktualny_stav = 12;
55          end
56
57          % inicializovanie retazca stavov
58          epizoda_stavy = aktualny_stav;
59
60          % inicializovanie retazca akcii
61          epizoda_akcie = [];
62
63          % aplikovanie strategie
64          while aktualny_stav ~= 4 && aktualny_stav ~= 8
65              % epsilon-greedy strategy
66              if(rand > epsilon)
67                  if Q(aktualny_stav, 1) == max([Q(aktualny_stav, 1), Q(aktualny_stav, 2),
                        Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
68                      aktualna_akcia = 1;
69                  elseif Q(aktualny_stav, 2) == max([Q(aktualny_stav, 1), Q(aktualny_stav,
                        2), Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
70                      aktualna_akcia = 2;
71                  elseif Q(aktualny_stav, 3) == max([Q(aktualny_stav, 1), Q(aktualny_stav,
                        2), Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
72                      aktualna_akcia = 3;
73                  elseif Q(aktualny_stav, 4) == max([Q(aktualny_stav, 1), Q(aktualny_stav,
                        2), Q(aktualny_stav, 3), Q(aktualny_stav, 4)])
74                      aktualna_akcia = 4;
75                  end
76              else
77                  aktualna_akcia = randi([1 4],1);
78              end
79
80              nasledujuci_stav = prechod_mriezka(aktualny_stav, aktualna_akcia);
81
82              epizoda_stavy = [epizoda_stavy, nasledujuci_stav];
83              epizoda_akcie = [epizoda_akcie, aktualna_akcia];
84
85              % vyhodnotenie Q-funkcie
86              if nasledujuci_stav == 4 || nasledujuci_stav == 8
87                  Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) + ...
88                      alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
89                      (gama * 0) - Q(aktualny_stav, aktualna_akcia));
90              else
91                  if Q(nasledujuci_stav, 1) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                        ])
```

```
92                  Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                        ...
93                      alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
94                      (gama * Q(nasledujuci_stav, 1)) - Q(aktualny_stav, aktualna_akcia
                            ));
95              elseif Q(nasledujuci_stav, 2) == max([Q(nasledujuci_stav, 1), Q(
                    nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                    ])
96                  Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                        ...
97                      alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
98                      (gama * Q(nasledujuci_stav, 2)) - Q(aktualny_stav, aktualna_akcia
                            ));
99              elseif Q(nasledujuci_stav, 3) == max([Q(nasledujuci_stav, 1), Q(
                    nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                    ])
100                 Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                        ...
101                     alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
102                     (gama * Q(nasledujuci_stav, 3)) - Q(aktualny_stav, aktualna_akcia
                            ));
103             elseif Q(nasledujuci_stav, 4) == max([Q(nasledujuci_stav, 1), Q(
                    nasledujuci_stav, 2), Q(nasledujuci_stav, 3), Q(nasledujuci_stav, 4)
                    ])
104                 Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                        ...
105                     alfa * (odmeny_mriezka(aktualny_stav, nasledujuci_stav) + ...
106                     (gama * Q(nasledujuci_stav, 4)) - Q(aktualny_stav, aktualna_akcia
                            ));
107             end
108         end
109
110         for n = 1:100
111             % vygenerovanie nahodneho stavu a akcie, ktora sa v nom vykonala
112             % budeme generovat cez vygenerovanie indexu danej dvojice
113             % stav - akcia
114             index = randi([1,(length(epizoda_stavy)-1)]);
115
116             stav = epizoda_stavy(index);
117             akcia = epizoda_akcie(index);
118
119             dalsi_stav = prechod_mriezka(stav, akcia);
120
121             % vyhodnotenie Q-funkcie, netreba podmienku o terminalnych
122             % stavoch
123             if Q(dalsi_stav, 1) == max([Q(dalsi_stav, 1), Q(dalsi_stav, 2), Q(
                    dalsi_stav, 3), Q(dalsi_stav, 4)])
124                 Q(stav, akcia) = Q(stav, akcia) + alfa * (odmeny_mriezka(stav,
```

```
                        dalsi_stav) + ...
125                     (gama * Q(dalsi_stav, 1)) - Q(stav, akcia));
126                 elseif Q(dalsi_stav, 2) == max([Q(dalsi_stav, 1), Q(dalsi_stav, 2), Q(
                        dalsi_stav, 3), Q(dalsi_stav, 4)])
127                     Q(stav, akcia) = Q(stav, akcia) + alfa * (odmeny_mriezka(stav,
                        dalsi_stav) + ...
128                     (gama * Q(dalsi_stav, 2)) - Q(stav, akcia));
129                 elseif Q(dalsi_stav, 3) == max([Q(dalsi_stav, 1), Q(dalsi_stav, 2), Q(
                        dalsi_stav, 3), Q(dalsi_stav, 4)])
130                     Q(stav, akcia) = Q(stav, akcia) + alfa * (odmeny_mriezka(stav,
                        dalsi_stav) + ...
131                     (gama * Q(dalsi_stav, 3)) - Q(stav, akcia));
132                 elseif Q(dalsi_stav, 4) == max([Q(dalsi_stav, 1), Q(dalsi_stav, 2), Q(
                        dalsi_stav, 3), Q(dalsi_stav, 4)])
133                     Q(stav, akcia) = Q(stav, akcia) + alfa * (odmeny_mriezka(stav,
                        dalsi_stav) + ...
134                     (gama * Q(dalsi_stav, 4)) - Q(stav, akcia));
135                 end
136             end
137
138             aktualny_stav = nasledujuci_stav;
139         end
140
141         % zmensenie epsilonu, pretoze uz staci "objavovat" v mensom rozsahu
142         epsilon = 0.99*epsilon;
143
144         % zmensenie miery ucenia, pretoze uz sa staci "ucit" v mensom rozsahu
145         alfa = 0.99*alfa;
146 end
147
148 % optimalna strategia
149 pol_nova = zeros(1, 12);
150 for o = 1:12
151     if o ~= 4 && o ~= 6 && o ~= 8
152         if Q(o, 1) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
153             pol_nova(o) = 1;
154         elseif Q(o, 2) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
155             pol_nova(o) = 2;
156         elseif Q(o, 3) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
157             pol_nova(o) = 3;
158         elseif Q(o, 4) == max([Q(o, 1), Q(o, 2), Q(o, 3), Q(o, 4)])
159             pol_nova(o) = 4;
160         end
161     end
162 end
163
164 pol_opt = pol_nova
165
```

```
toc;
```

2 Aplikácia numerickej metódy stochastického OR na úlohu o obchodnej prevádzke popísanej v Kapitole 4 v systéme Matlab

```
% UCELOVA FUNKCIA PRE PRIKLAD O OBCHODNEJ PREVADZKE

    function[hodnota] = ucelova_funkcia_obchod(akcia, udalost)
    % urcenie hodnoty ucelovej funkcie
    if akcia == 1
        if udalost == 1
            hodnota = 1;
        elseif udalost == 2 || udalost == 3
            hodnota = 0;
        end
    elseif akcia == 2
        if udalost == 1
            hodnota = 0.8;
        elseif udalost == 2 || udalost == 3
            hodnota = -0.2;
        end
    end
end
```

```
% STAVOVA ROVNICA PRE PRIKLAD O OBCHODNEJ PREVADZKE

    function[stav_2] = stavova_rovnica_obchod(stav_1, udalost)
    % urcenie nasledujuceho stavu
    if udalost == 1
        stav_2 = stav_1 - 1;
    elseif udalost == 2
        stav_2 = stav_1;
    elseif udalost == 3
        stav_2 = stav_1 + 1;
    end
end
```

```
% PST PRECHODU PRE PRIKLAD O OBCHODNEJ PREVADZKE

function[pst_prechodu] = pst_prechodu_obchod(stav_1, u, stav_2)
    % '1 ZAMESTNANEC PRI POKLADNI'
    P1 = [2/3 1/3 0   0   0   0   0;
          1/3 1/3 1/3 0   0   0   0;
          0   1/3 1/3 1/3 0   0   0;
          0   0   1/3 1/3 1/3 0   0;
          0   0   0   1/3 1/3 1/3 0;
          0   0   0   0   1/3 1/3 1/3;
          0   0   0   0   0   1/3 2/3];
```

```
12
13      % '2 ZAMESTNANCI PRI POKLADNI'
14      P2 = [2/3 1/3 0    0    0    0    0;
15            1/2 1/4 1/4 0    0    0    0;
16            0   1/2 1/4 1/4 0    0    0;
17            0   0   1/2 1/4 1/4 0    0;
18            0   0   0   1/2 1/4 1/4 0;
19            0   0   0   0   1/2 1/4 1/4;
20            0   0   0   0   0   2/3 1/3];
21
22        % urcenie pravdepodobnosti prechodu
23        if stav_1 == 1 && stav_2 == 0
24            pst_prechodu = 0;
25        elseif stav_1 == 7 && stav_2 == 8
26            pst_prechodu = 0;
27        else
28            if u == 1
29                pst_prechodu = P1(stav_1, stav_2);
30            elseif u == 2
31                pst_prechodu = P2(stav_1, stav_2);
32            end
33        end
34  end
```

```
1  % numericke riesenie pomocou stochastickeho OR
2  % uloha o obchodnej prevadzke
3
4  clc;
5  clear all;
6
7  tic;
8
9  % DEFINOVANIE ZADANYCH VELICIN
10 x = (1:7);                % mnozina stavov
11
12 u = [1, 2];               % vektor riadeni
13
14 z = (1:3);                % vektor toho, co sa moze stat s prostredim, 1 - odisiel,
15                           % 2 - nic sa nestalo, 3 - prisiel
16
17 gama = 0.9;               % diskontny faktor
18
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 %
21 %              Stochasticke OR
22 %
23 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24
```

```
% pocet krokov
k = 24;

% inicializovanie hodnotovej funkcie
V = -inf*ones(k+1, length(x));

% inicializovanie spatnej vazby
v = zeros(k+1, length(x));

V(k+1, :) = 0;

for i = k:-1:1
    for prem = 1:length(x)
        for u = 1:2
            cislo = 0;
            for z = 1:3
                f = stavova_rovnica_obchod(prem, z);
                f0 = 0.9^i * ucelova_funkcia_obchod(u, z);
                if pst_prechodu_obchod(prem, u, f) ~= 0
                 cislo = cislo + pst_prechodu_obchod(prem, u, f)*(f0 + V(i+1,f));
                end
            end
            if cislo > V(i, prem)
                V(i, prem) = cislo;
                v(i, prem) = u;
            end
        end
    end
end

v

toc;
```

3 Aplikácie RL algoritmov na úlohu o obchodnej prevádzke popísanej v Kapitole 4 v systéme Matlab

```
% FUNKCIA PRECHODU PRE PRIKLAD O OBCHODNEJ PREVADZKE

function[pst_prechodu] = funkcia_prechodu_obchod(stav_1, u, stav_2)
    % '1 ZAMESTNANEC PRI POKLADNI'
    P1 = [2/3 1/3 0   0   0   0   0;
          1/3 1/3 1/3 0   0   0   0;
          0   1/3 1/3 1/3 0   0   0;
          0   0   1/3 1/3 1/3 0   0;
          0   0   0   1/3 1/3 1/3 0;
          0   0   0   0   1/3 1/3 1/3;
          0   0   0   0   0   1/3 2/3];
```

```
12
13      % '2 ZAMESTNANCI PRI POKLADNI'
14      P2 = [2/3  1/3  0    0    0    0    0;
15            1/2  1/4  1/4  0    0    0    0;
16            0    1/2  1/4  1/4  0    0    0;
17            0    0    1/2  1/4  1/4  0    0;
18            0    0    0    1/2  1/4  1/4  0;
19            0    0    0    0    1/2  1/4  1/4;
20            0    0    0    0    0    2/3  1/3];
21
22       % urcenie pravdepodobnosti prechodu
23       if u == 1
24           pst_prechodu = P1(stav_1, stav_2);
25       elseif u == 2
26           pst_prechodu = P2(stav_1, stav_2);
27       end
28  end
```

```
1   % FUNKCIA ODMIEN PRE PRIKLAD O OBCHODNEJ PREVADZKE
2
3   function[odmena] = funkcia_odmien_obchod(stav_1, u, stav_2)
4       % '1 ZAMESTNANEC PRI POKLADNI'
5       R1 = [0 0 0 0 0 0 0;
6             1 0 0 0 0 0 0;
7             0 1 0 0 0 0 0;
8             0 0 1 0 0 0 0;
9             0 0 0 1 0 0 0;
10            0 0 0 0 1 0 0;
11            0 0 0 0 0 1 0];
12
13      % '2 ZAMESTNANCI PRI POKLADNI'
14      R2 = [-0.2  -0.2  -0.2  -0.2  -0.2  -0.2  -0.2;
15             0.8  -0.2  -0.2  -0.2  -0.2  -0.2  -0.2;
16            -0.2   0.8  -0.2  -0.2  -0.2  -0.2  -0.2;
17            -0.2  -0.2   0.8  -0.2  -0.2  -0.2  -0.2;
18            -0.2  -0.2  -0.2   0.8  -0.2  -0.2  -0.2;
19            -0.2  -0.2  -0.2  -0.2   0.8  -0.2  -0.2;
20            -0.2  -0.2  -0.2  -0.2  -0.2   0.8  -0.2];
21
22      % urcenie odmeny
23      if u == 1
24          odmena = R1(stav_1, stav_2);
25      elseif u == 2
26          odmena = R2(stav_1, stav_2);
27      end
28  end
```

```
1   % iterovanie hodnot (value iteration), dynamicke programovanie
```

```
2  % uloha o obchodnej prevadzke
3
4  clc;
5  clear all;
6
7  tic;
8
9  % DEFINOVANIE ZADANYCH VELICIN
10 x = (1:7);                % mnozina stavov
11
12 u = [1, 2];               % vektor riadeni
13
14 gama = 0.9;               % diskontny faktor
15
16 %%%%
```

```
50
51      % vynulovanie
52      Q_stary_1 = Q_novy_1;
53      Q_novy_1 = zeros(1, length(x));
54      Q_stary_2 = Q_novy_2;
55      Q_novy_2 = zeros(1, length(x));
56
57      % vetva '1 zamestnanec pri pokladni'
58      for a = 1:length(x)
59          for b = 1:length(x)
60              if Q_stary_1(b) == max([Q_stary_1(b), Q_stary_2(b)])
61                  Q_novy_1(a) = Q_novy_1(a) + funkcia_prechodu_obchod(a, 1, b)*...
62                      (funkcia_odmien_obchod(a, 1, b) + gama*Q_stary_1(b));
63              elseif Q_stary_2(b) == max([Q_stary_1(b), Q_stary_2(b)])
64                  Q_novy_1(a) = Q_novy_1(a) + funkcia_prechodu_obchod(a, 1, b)*...
65                      (funkcia_odmien_obchod(a, 1, b) + gama*Q_stary_2(b));
66              end
67          end
68      end
69
70      % vetva '2 zamestnanci pri pokladni'
71      for a = 1:length(x)
72          for b = 1:length(x)
73              if Q_stary_1(b) == max([Q_stary_1(b), Q_stary_2(b)])
74                  Q_novy_2(a) = Q_novy_2(a) + funkcia_prechodu_obchod(a, 2, b)*...
75                      (funkcia_odmien_obchod(a, 2, b) + gama*Q_stary_1(b));
76              elseif Q_stary_2(b) == max([Q_stary_1(b), Q_stary_2(b)])
77                  Q_novy_2(a) = Q_novy_2(a) + funkcia_prechodu_obchod(a, 2, b)*...
78                      (funkcia_odmien_obchod(a, 2, b) + gama*Q_stary_2(b));
79              end
80          end
81      end
82  end
83
84  % inicializovanie vektora optimalnej strategie
85  pol_opt = -100*ones(1, length(x));
86
87  % optimalna strategia
88  for i = 1:length(x)
89      if Q_novy_1(i) == max([Q_novy_1(i), Q_novy_2(i)])
90          pol_opt(i) = 1;
91      elseif Q_novy_2(i) == max([Q_novy_1(i), Q_novy_2(i)])
92          pol_opt(i) = 2;
93      end
94  end
95
96  pol_opt
97
```

```
98 toc;
```

```
1  % iterovanie strategii (policy iteration), dynamicke programovanie
2  % uloha o obchodnej prevadzke
3
4  clc;
5  clear all;
6
7  tic;
8
9  % DEFINOVANIE ZADANYCH VELICIN
10 x = (1:7);               % mnozina stavov
11
12 u = [1, 2];              % vektor riadeni
13
14 gama = 0.9;              % diskontny faktor
15
16 %%%%
```

```
47      while norm(Q_stary_1 - Q_novy_1) > 10^(-3)
48          k = k+1;
49
50          % zapisanie do matic
51          if k == 1
52              Q_novy_1 = zeros(1, length(x));
53              Q_novy_2 = zeros(1, length(x));
54
55              Q_1 = Q_novy_1;
56              Q_2 = Q_novy_2;
57          else
58              Q_1 = [Q_1; Q_novy_1];
59              Q_2 = [Q_2; Q_novy_2];
60          end
61
62          % vynulovanie
63          Q_stary_1 = Q_novy_1;
64          Q_novy_1 = zeros(1, length(x));
65          Q_stary_2 = Q_novy_2;
66          Q_novy_2 = zeros(1, length(x));
67
68          % vetva '1 zamestnanec pri pokladni'
69          for a = 1:length(x)
70              for b = 1:length(x)
71                  if pol_nova(b) == 1
72                      Q_novy_1(a) = Q_novy_1(a) + funkcia_prechodu_obchod(a, 1, b)*...
73                          (funkcia_odmien_obchod(a, 1, b) + gama*Q_stary_1(b));
74                  elseif pol_nova(b) == 2
75                      Q_novy_1(a) = Q_novy_1(a) + funkcia_prechodu_obchod(a, 1, b)*...
76                          (funkcia_odmien_obchod(a, 1, b) + gama*Q_stary_2(b));
77                  end
78              end
79          end
80
81          % vetva '2 zamestnanci pri pokladni'
82          for a = 1:length(x)
83              for b = 1:length(x)
84                  if pol_nova(b) == 1
85                      Q_novy_2(a) = Q_novy_2(a) + funkcia_prechodu_obchod(a, 2, b)*...
86                          (funkcia_odmien_obchod(a, 2, b) + gama*Q_stary_1(b));
87                  elseif pol_nova(b) == 2
88                      Q_novy_2(a) = Q_novy_2(a) + funkcia_prechodu_obchod(a, 2, b)*...
89                          (funkcia_odmien_obchod(a, 2, b) + gama*Q_stary_2(b));
90                  end
91              end
92          end
93      end
94
```

```
    % nova strategia
    pol_stara = pol_nova;
    pol_nova = -100*ones(1, length(x));

    for i = 1:length(x)
        if Q_novy_1(i) == max([Q_novy_1(i), Q_novy_2(i)])
            pol_nova(i) = 1;
        elseif Q_novy_2(i) == max([Q_novy_1(i), Q_novy_2(i)])
            pol_nova(i) = 2;
        end
    end

    pol_stara
    pol_nova
end

pol_opt = pol_nova

toc;
```

```
% FUNKCIA PRECHODU PRE PRIKLAD O OBCHODNEJ PREVADZKE

function[stav_2] = prechod_obchod(stav_1, u)
    % '1 ZAMESTNANEC PRI POKLADNI'
    P1 = [2/3 1/3 0   0   0   0   0;
          1/3 1/3 1/3 0   0   0   0;
          0   1/3 1/3 1/3 0   0   0;
          0   0   1/3 1/3 1/3 0   0;
          0   0   0   1/3 1/3 1/3 0;
          0   0   0   0   1/3 1/3 1/3;
          0   0   0   0   0   1/3 2/3];

    % '2 ZAMESTNANCI PRI POKLADNI'
    P2 = [2/3 1/3 0   0   0   0   0;
          1/2 1/4 1/4 0   0   0   0;
          0   1/2 1/4 1/4 0   0   0;
          0   0   1/2 1/4 1/4 0   0;
          0   0   0   1/2 1/4 1/4 0;
          0   0   0   0   1/2 1/4 1/4;
          0   0   0   0   0   2/3 1/3];

    % pst_prichod = p(i,i+1), t.j. do obchodu vojde dalsi zakaznik
    if (u == 1 && stav_1 ~= 7)      % obchod nie je plny
        pst_prichod = P1(stav_1, stav_1+1);
    elseif (u == 1 && stav_1 == 7)
        pst_prichod = 0;
    elseif (u == 2 && stav_1 ~= 7)
        pst_prichod = P2(stav_1, stav_1+1);
```

```
    elseif (u == 2 && stav_1 == 7)
        pst_prichod = 0;
    end

    % pst_odchod = p(i,i-1), t.j. zakaznik v obchode je vybaveny a odide
    if (u == 1 && stav_1 ~= 1)      % obchod nie je prazdny
        pst_odchod = P1(stav_1, stav_1-1);
    elseif (u == 1 && stav_1 == 1)
        pst_odchod = 0;
    elseif (u == 2 && stav_1 ~= 1)
        pst_odchod = P2(stav_1, stav_1-1);
    elseif (u == 2 && stav_1 == 1)
        pst_odchod = 0;
    end

    % pst_ziadna_zmena = p(i,i), t.j. nic sa nestane
    if u == 1
        pst_ziadna_zmena = P1(stav_1,stav_1);
    elseif u == 2
        pst_ziadna_zmena = P2(stav_1,stav_1);
    end

    % SIMULACIA STAVU V OBCHODE
    cislo = rand(1);

    if (cislo <= pst_prichod  && stav_1 ~= 7)
        stav_2 = stav_1+1;
    end
    if (cislo > pst_prichod && cislo <= pst_prichod + pst_ziadna_zmena)
        stav_2 = stav_1;
    end
    if (cislo > pst_prichod + pst_ziadna_zmena && cislo <= 1 && stav_1 ~= 1)
        stav_2 = stav_1-1;
    end
end
```

```
% Temporal differences, TD SARSA (on-policy control)
% uloha o obchodnej prevadzke

clc;
clear all;

tic;

% DEFINOVANIE ZADANYCH VELICIN
x = (1:7);              % mnozina stavov

u = [1, 2];             % vektor riadeni
```

```
13
14 x_0 = 1;                  % na zaciatku je v obchode 0 zakaznikov
15
16 N = 30000;                % pocet simulacii
17
18 gama = 0.9;               % diskontny faktor
19
20 %%%%
```

```
61              if(rand > epsilon)
62                  if Q(nasledujuci_stav, 1) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2)])
63                      nasledujuca_akcia = 1;
64                  elseif Q(nasledujuci_stav, 2) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2)])
65                      nasledujuca_akcia = 2;
66                  end
67              else
68                  nasledujuca_akcia = randi(u, 1);
69              end
70
71              epizoda_stavy = [epizoda_stavy, nasledujuci_stav];
72
73              % vyhodnotenie Q-funkcie
74              Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) + ...
75                  alfa * (funkcia_odmien_obchod(aktualny_stav, aktualna_akcia,
                        nasledujuci_stav) + ...
76                  gama * Q(nasledujuci_stav, nasledujuca_akcia) - Q(aktualny_stav,
                        aktualna_akcia));
77
78              aktualny_stav = nasledujuci_stav;
79              aktualna_akcia = nasledujuca_akcia;
80          end
81
82          % zmensenie epsilonu a alfy, pretoze uz staci "objavovat" a "ucit sa" v mensom
                rozsahu
83          epsilon = 0.9999*epsilon;
84          alfa = 0.9999*alfa;
85  end
86
87  % optimalna strategia
88  pol_opt = -100*ones(1, length(x));
89  for o = 1:length(x)
90      if Q(o, 1) == max([Q(o, 1), Q(o, 2)])
91          pol_opt(o) = 1;
92      elseif Q(o, 2) == max([Q(o, 1), Q(o, 2)])
93          pol_opt(o) = 2;
94      end
95  end
96
97  pol_opt
98
99  toc;
```

```
1  % FUNKCIA VYNOSOV PRE PRIKLAD O OBCHODNEJ PREVADZKE PRI MONTE CARLO METODE
2  % IDE O VYNOSY PRE EPIZODU OD DANEHO BODU
3
```

```
function[vynosy] = vynosy_MC_obchod(epizoda_stavy, epizoda_akcie)
    gama = 0.9;              % diskontny faktor
    vynosy = 0;

    % otocenie vektora, pretoze vynosy budeme pocitat od konca
    otocene_stavy = flip(epizoda_stavy);
    otocene_akcie = flip(epizoda_akcie);

    % vyhodnotenie zvysku vynosov
    for i = 1:(length(otocene_stavy)-1)
        vynosy = funkcia_odmien_obchod(otocene_stavy(i+1), otocene_akcie(i+1), ...
            otocene_stavy(i)) + gama*vynosy;
    end
end
```

```
% Monte Carlo metoda, with exploring start
% uloha o obchodnej prevadzke

clc;
clear all;

tic;

% DEFINOVANIE ZADANYCH VELICIN
x = (1:7);               % mnozina stavov

u = [1, 2];              % vektor riadeni

gama = 0.9;              % diskontny faktor

x_0 = 1;                 % na zaciatku je v obchode 0 zakaznikov

N = 1000;                % pocet simulacii

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Monte Carlo metoda pomocou Q-funkcie
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% inicializovanie strategie, t.j. co robit v kazdom stave
% nahodne zvolena
pol_nova = 2*ones(1, length(x));

% exploration - exploitation rozdelenie pravdepodobnosti
% epsilon
epsilon = 0.8;

```

```
for l = 1:10
    % inicializovanie matice prisluchajucim danym akciam v danom stave
    % na zaciatku nastavene na 0 (7 moznych stavov x 2 mozne akcie)
    Q = zeros(length(x), length(u));

    % inicializovanie matice prisluchajucej nascitanym hodnotam Q(s,a),
    % t.j. akumulacnej matice
    A = zeros(length(x), length(u));

    % inicializovanie matice prisluchajucej poctu vyslytov dvojice Q(s,a),
    % t.j. poctovej (counter) matice
    C = zeros(length(x), length(u));

    for i = 1:N                    % cyklus simulacii
        % na zaciatku je v obchode 0 zakaznikov
        aktualny_stav = x_0;

        % vygenerovanie prvej akcie
        aktualna_akcia = randi(u);

        % POSTUPOVANIE POMOCOU STRATEGIE, KTORU MAM PRAVE ZVOLENU
        % inicializovanie retazca stavov
        epizoda_stavy = aktualny_stav;

        % inicializovanie retazca akcii
        epizoda_akcie = aktualna_akcia;

        % aplikovanie strategie
        % kedze nemame terminalny stav, tak len vykoname dostatocne vela
        % krokov
        for t = 1:250   % zatial nahodne zvolene cislo
            aktualny_stav = prechod_obchod(aktualny_stav, aktualna_akcia);
            epizoda_stavy = [epizoda_stavy, aktualny_stav];

            % epsilon-greedy strategy
            if(rand > epsilon)
                aktualna_akcia = pol_nova(aktualny_stav);
            else
                aktualna_akcia = randi(u, 1);
            end
            epizoda_akcie = [epizoda_akcie, aktualna_akcia];
        end

        % zmensenie epsilonu, pretoze uz staci "objavovat" v mensom rozsahu
        epsilon = 0.9999*epsilon;

        % zistenie matic A a C pre danu epizodu
        for k = 1:length(epizoda_stavy)
```

```
            A(epizoda_stavy(k), epizoda_akcie(k)) = A(epizoda_stavy(k), epizoda_akcie(k))
                + ...
                vynosy_MC_obchod(epizoda_stavy(k:length(epizoda_stavy)), epizoda_akcie(k:
                    length(epizoda_akcie)));
            C(epizoda_stavy(k), epizoda_akcie(k)) = C(epizoda_stavy(k), epizoda_akcie(k))
                + 1;
        end

        % vyhodnotenie Q(x,u) po N simulaciach
        for riadok = 1:length(x)
            for stlpec = 1:length(u)
                if C(riadok, stlpec) ~= 0
                    Q(riadok, stlpec) = A(riadok, stlpec)/C(riadok, stlpec);
                end
            end
        end
    end

    % nova strategia
    for o = 1:length(x)
        if Q(o, 1) == max([Q(o, 1), Q(o, 2)])
            pol_nova(o) = 1;
        elseif Q(o, 2) == max([Q(o, 1), Q(o, 2)])
            pol_nova(o) = 2;
        end
    end
end

pol_opt = pol_nova

toc;
```

```
% Q-learning (off policy)
% uloha o obchodnej prevadzke

clc;
clear all;

tic;

% DEFINOVANIE ZADANYCH VELICIN
x = (1:7);              % mnozina stavov

u = [1, 2];             % vektor riadeni

x_0 = 1;                % na zaciatku je v obchode 0 zakaznikov

N = 30000;              % pocet simulacii
```

```
17
18 gama = 0.9;               % diskontny faktor
19
20 %%%%
```

```
                2)])
64              Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) + ...
65                  alfa * (funkcia_odmien_obchod(aktualny_stav, aktualna_akcia,
                        nasledujuci_stav) + ...
66                  (gama * Q(nasledujuci_stav, 1)) - Q(aktualny_stav, aktualna_akcia));
67          elseif Q(nasledujuci_stav, 2) == max([Q(nasledujuci_stav, 1), Q(
                nasledujuci_stav, 2)])
68              Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) + ...
69                  alfa * (funkcia_odmien_obchod(aktualny_stav, aktualna_akcia,
                        nasledujuci_stav) + ...
70                  (gama * Q(nasledujuci_stav, 2)) - Q(aktualny_stav, aktualna_akcia));
71          end
72
73          aktualny_stav = nasledujuci_stav;
74      end
75
76      % zmensenie epsilonu, pretoze uz staci "objavovat" v mensom rozsahu
77      epsilon = 0.9999*epsilon;
78
79      % zmensenie miery ucenia, pretoze uz sa staci "ucit" v mensom rozsahu
80      alfa = 0.9999*alfa;
81  end
82
83  % optimalna strategia
84  pol_opt = -100*ones(1, length(x));
85  for o = 1:length(x)
86      if Q(o, 1) == max([Q(o, 1), Q(o, 2)])
87          pol_opt(o) = 1;
88      elseif Q(o, 2) == max([Q(o, 1), Q(o, 2)])
89          pol_opt(o) = 2;
90      end
91  end
92
93  pol_opt
94
95  toc;
```

```
1   % Dyna-Q
2   % uloha o obchodnej prevadzke
3
4   clc;
5   clear all;
6
7   tic;
8
9   % DEFINOVANIE ZADANYCH VELICIN
10  x = (1:7);              % mnozina stavov
11
```

```
12 u = [1, 2];              % vektor riadeni
13
14 x_0 = 1;                  % na zaciatku je v obchode 0 zakaznikov
15
16 N = 300;                  % pocet simulacii
17
18 gama = 0.9;               % diskontny faktor
19
20 %%%%
```

```
59              end
60
61              nasledujuci_stav = prechod_obchod(aktualny_stav, aktualna_akcia);
62
63              epizoda_stavy = [epizoda_stavy, nasledujuci_stav];
64              epizoda_akcie = [epizoda_akcie, aktualna_akcia];
65
66              % vyhodnotenie Q-funkcie
67                  if Q(nasledujuci_stav, 1) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2)])
68                      Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                             ...
69                          alfa * (funkcia_odmien_obchod(aktualny_stav, aktualna_akcia,
                                nasledujuci_stav) + ...
70                          (gama * Q(nasledujuci_stav, 1)) - Q(aktualny_stav, aktualna_akcia
                                ));
71                  elseif Q(nasledujuci_stav, 2) == max([Q(nasledujuci_stav, 1), Q(
                        nasledujuci_stav, 2)])
72                      Q(aktualny_stav, aktualna_akcia) = Q(aktualny_stav, aktualna_akcia) +
                             ...
73                          alfa * (funkcia_odmien_obchod(aktualny_stav, aktualna_akcia,
                                nasledujuci_stav) + ...
74                          (gama * Q(nasledujuci_stav, 2)) - Q(aktualny_stav, aktualna_akcia
                                ));
75                  end
76
77              for n = 1:100
78                  % vygenerovanie nahodneho stavu a akcie, ktora sa v nom vykonala
79                  % budeme generovat cez vygenerovanie indexu danej dvojice
80                  % stav - akcia
81                  index = randi([1,(length(epizoda_stavy)-1)]);
82
83                  stav = epizoda_stavy(index);
84                  akcia = epizoda_akcie(index);
85
86                  dalsi_stav = prechod_obchod(stav, akcia);
87
88                  % vyhodnotenie Q-funkcie, netreba podmienku o terminalnych
89                  % stavoch
90                  if Q(dalsi_stav, 1) == max([Q(dalsi_stav, 1), Q(dalsi_stav, 2)])
91                      Q(stav, akcia) = Q(stav, akcia) + alfa * (funkcia_odmien_obchod(stav,
                             akcia, dalsi_stav) + ...
92                          (gama * Q(dalsi_stav, 1)) - Q(stav, akcia));
93                  elseif Q(dalsi_stav, 2) == max([Q(dalsi_stav, 1), Q(dalsi_stav, 2)])
94                      Q(stav, akcia) = Q(stav, akcia) + alfa * (funkcia_odmien_obchod(stav,
                             akcia, dalsi_stav) + ...
95                          (gama * Q(dalsi_stav, 2)) - Q(stav, akcia));
96                  end
```

```
97              end
98
99              aktualny_stav = nasledujuci_stav;
100         end
101
102         % zmensenie epsilonu, pretoze uz staci "objavovat" v mensom rozsahu
103         epsilon = 0.99*epsilon;
104
105         % zmensenie miery ucenia, pretoze uz sa staci "ucit" v mensom rozsahu
106         alfa = 0.99*alfa;
107 end
108
109 % optimalna strategia
110 pol_opt = -100*ones(1, length(x));
111 for o = 1:length(x)
112     if Q(o, 1) == max([Q(o, 1), Q(o, 2)])
113         pol_opt(o) = 1;
114     elseif Q(o, 2) == max([Q(o, 1), Q(o, 2)])
115         pol_opt(o) = 2;
116     end
117 end
118
119 pol_opt
120
121 toc;
```

Príloha B

1 Hodnoty Q-funkcie v algoritme iterovania hodnôt pri aplikovaní akcie *H - hore* v konkrétnych iteráciach pre úlohu pohybujúceho sa robota v miestnosti

V každej tabuľke hodnôt prislúcha j-ty riadok j-tej iterácii a i-ty stĺpec i-temu stavu.

Q_D | Q_H | Q_L | Q_P

21x12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	1	0	0	0	-1	0	0	0	0
2	-0.0700	-0.0700	0.0300	1	-0.0700	0	-0.1700	-1	-0.0700	-0.0700	-0.0700	-0.8700
3	-0.1400	-0.0600	0.6070	1	-0.1400	0	0.4070	-1	-0.1400	-0.1400	-0.1400	-0.8840
4	-0.1460	0.3956	0.7168	1	-0.2100	0	0.5075	-1	-0.2100	-0.2100	0.2276	-0.8980
5	0.2685	0.5787	0.7769	1	0.1296	0	0.5610	-1	-0.2800	-0.2362	0.2940	-0.8682
6	0.4297	0.6587	0.7970	1	0.3340	0	0.5787	-1	0.0127	-0.0125	0.3849	-0.8415
7	0.5337	0.6929	0.8053	1	0.4649	0	0.5860	-1	0.2164	0.1131	0.4173	-0.8251
8	0.5833	0.7073	0.8084	1	0.5342	0	0.5887	-1	0.3509	0.2124	0.4406	-0.8138
9	0.6083	0.7135	0.8096	1	0.5699	0	0.5898	-1	0.4243	0.2641	0.4507	-0.8081
10	0.6202	0.7161	0.8100	1	0.5874	0	0.5902	-1	0.4630	0.2945	0.4565	-0.8049
11	0.6258	0.7173	0.8102	1	0.5959	0	0.5903	-1	0.4822	0.3098	0.4592	-0.8033
12	0.6284	0.7178	0.8102	1	0.5999	0	0.5904	-1	0.4922	0.3220	0.4611	-0.8025
13	0.6296	0.7180	0.8103	1	0.6017	0	0.5904	-1	0.4981	0.3375	0.4632	-0.8020
14	0.6302	0.7181	0.8103	1	0.6026	0	0.5904	-1	0.5014	0.3476	0.4646	-0.8016
15	0.6304	0.7181	0.8103	1	0.6030	0	0.5904	-1	0.5031	0.3537	0.4655	-0.8013
16	0.6305	0.7181	0.8103	1	0.6031	0	0.5904	-1	0.5040	0.3571	0.4660	-0.8011
17	0.6306	0.7181	0.8103	1	0.6032	0	0.5904	-1	0.5044	0.3590	0.4663	-0.8009
18	0.6306	0.7181	0.8103	1	0.6032	0	0.5904	-1	0.5046	0.3600	0.4665	-0.8008
19	0.6306	0.7181	0.8103	1	0.6033	0	0.5904	-1	0.5047	0.3605	0.4666	-0.8008
20	0.6306	0.7181	0.8103	1	0.6033	0	0.5904	-1	0.5048	0.3608	0.4666	-0.8008
21	0.6306	0.7181	0.8103	1	0.6033	0	0.5904	-1	0.5048	0.3609	0.4666	-0.8008

2 Hodnoty Q-funkcie v algoritme iterovania hodnôt pri aplikovaní akcie *D - dole* v konkrétnych iteráciach pre úlohu pohybujúceho sa robota v miestnosti

Q_D | Q_H | Q_L | Q_P

21x12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	1	0	0	0	-1	0	0	0	0
2	-0.0700	-0.0700	0.0300	1	-0.0700	0	-0.1700	-1	-0.0700	-0.0700	-0.0700	-0.0700
3	-0.1400	-0.0600	-0.0330	1	-0.1400	0	-0.2330	-1	-0.1400	-0.1400	-0.1400	-0.1400
4	-0.1460	0.3956	0.4056	1	-0.2100	0	-0.2413	-1	-0.2100	-0.2100	-0.2100	-0.2100
5	-0.1411	0.5787	0.5027	1	-0.2800	0	0.0628	-1	-0.2800	-0.2362	0.0701	-0.2362
6	0.1553	0.6587	0.5532	1	-0.2681	0	0.1213	-1	-0.3150	-0.0125	0.1713	-0.0486
7	0.3329	0.6929	0.5701	1	0.0070	0	0.1958	-1	-0.0407	0.1131	0.2623	0.0264
8	0.4441	0.7073	0.5771	1	0.1961	0	0.2225	-1	0.1522	0.2124	0.3057	0.1017
9	0.5026	0.7135	0.5796	1	0.3176	0	0.2414	-1	0.2777	0.2641	0.3322	0.1345
10	0.5327	0.7161	0.5806	1	0.3834	0	0.2495	-1	0.3465	0.2945	0.3452	0.1554
11	0.5474	0.7173	0.5810	1	0.4179	0	0.2542	-1	0.3826	0.3098	0.3522	0.1652
12	0.5544	0.7178	0.5812	1	0.4350	0	0.2564	-1	0.4012	0.3220	0.3562	0.1705
13	0.5578	0.7180	0.5812	1	0.4437	0	0.2579	-1	0.4120	0.3375	0.3598	0.1732
14	0.5593	0.7181	0.5812	1	0.4488	0	0.2596	-1	0.4185	0.3476	0.3628	0.1750
15	0.5600	0.7181	0.5812	1	0.4516	0	0.2607	-1	0.4221	0.3537	0.3648	0.1768
16	0.5604	0.7181	0.5812	1	0.4531	0	0.2614	-1	0.4241	0.3571	0.3661	0.1781
17	0.5605	0.7181	0.5812	1	0.4538	0	0.2618	-1	0.4251	0.3590	0.3668	0.1789
18	0.5606	0.7181	0.5812	1	0.4542	0	0.2621	-1	0.4256	0.3600	0.3672	0.1794
19	0.5606	0.7181	0.5812	1	0.4543	0	0.2622	-1	0.4258	0.3605	0.3674	0.1797
20	0.5606	0.7181	0.5812	1	0.4544	0	0.2623	-1	0.4260	0.3608	0.3675	0.1798
21	0.5606	0.7181	0.5812	1	0.4545	0	0.2623	-1	0.4260	0.3609	0.3676	0.1799

3 Hodnoty Q-funkcie v algoritme iterovania hodnôt pri aplikovaní akcie *P - vpravo* v konkrétnych iteráciach pre úlohu pohybujúceho sa robota v miestnosti

Q_D | Q_H | Q_L | Q_P

21x12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	1	0	0	0	-1	0	0	0	0
2	-0.0700	-0.0700	0.7300	1	-0.0700	0	-0.8700	-1	-0.0700	-0.0700	-0.0700	-0.1700
3	-0.1400	0.5000	0.7960	1	-0.1400	0	-0.8040	-1	-0.1400	-0.1400	-0.1400	-0.2330
4	0.3020	0.6668	0.8503	1	-0.2100	0	-0.8044	-1	-0.2100	-0.2100	-0.1553	-0.2960
5	0.4726	0.7436	0.8658	1	-0.2288	0	-0.7622	-1	-0.2800	0.0701	-0.1645	-0.3590
6	0.5851	0.7713	0.8727	1	0.0529	0	-0.7540	-1	-0.0290	0.1792	0.0084	-0.1780
7	0.6390	0.7824	0.8751	1	0.2570	0	-0.7442	-1	0.1080	0.2738	0.0778	-0.1121
8	0.6663	0.7866	0.8761	1	0.3875	0	-0.7408	-1	0.2171	0.3186	0.1458	-0.0401
9	0.6793	0.7882	0.8765	1	0.4591	0	-0.7383	-1	0.2734	0.3462	0.1756	-0.0095
10	0.6855	0.7888	0.8766	1	0.4963	0	-0.7373	-1	0.3064	0.3598	0.1943	0.0103
11	0.6884	0.7891	0.8767	1	0.5148	0	-0.7367	-1	0.3229	0.3671	0.2031	0.0195
12	0.6897	0.7892	0.8767	1	0.5238	0	-0.7364	-1	0.3357	0.3718	0.2079	0.0246
13	0.6903	0.7892	0.8767	1	0.5281	0	-0.7362	-1	0.3514	0.3769	0.2103	0.0271
14	0.6906	0.7892	0.8767	1	0.5302	0	-0.7360	-1	0.3614	0.3809	0.2120	0.0287
15	0.6907	0.7892	0.8767	1	0.5312	0	-0.7359	-1	0.3675	0.3834	0.2136	0.0304
16	0.6907	0.7892	0.8767	1	0.5317	0	-0.7358	-1	0.3709	0.3850	0.2147	0.0315
17	0.6908	0.7892	0.8767	1	0.5320	0	-0.7357	-1	0.3727	0.3858	0.2155	0.0323
18	0.6908	0.7892	0.8767	1	0.5321	0	-0.7357	-1	0.3737	0.3863	0.2159	0.0328
19	0.6908	0.7892	0.8767	1	0.5321	0	-0.7357	-1	0.3742	0.3865	0.2161	0.0330
20	0.6908	0.7892	0.8767	1	0.5322	0	-0.7357	-1	0.3744	0.3867	0.2163	0.0332
21	0.6908	0.7892	0.8767	1	0.5322	0	-0.7357	-1	0.3746	0.3867	0.2164	0.0332

4 Hodnoty Q-funkcie v algoritme iterovania hodnôt pri aplikovaní akcie *L - vľavo* v konkrétnych iteráciach pre úlohu pohybujúceho sa robota v miestnosti

Q_D | Q_H | Q_L | Q_P

21x12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	1	0	0	0	-1	0	0	0	0
2	-0.0700	-0.0700	-0.0700	1	-0.0700	0	-0.0700	-1	-0.0700	-0.0700	-0.0700	-0.1700
3	-0.1400	-0.1400	-0.0600	1	-0.1400	0	-0.0600	-1	-0.1400	-0.1400	-0.1400	-0.2330
4	-0.2100	-0.0820	0.4503	1	-0.2100	0	0.3212	-1	-0.2100	-0.2100	-0.1553	-0.2960
5	0.1808	0.3050	0.5992	1	-0.2288	0	0.4438	-1	-0.2800	-0.2800	-0.1645	-0.0089
6	0.3683	0.4568	0.6676	1	0.0529	0	0.4948	-1	-0.3090	-0.2800	0.0716	0.0643
7	0.4900	0.5524	0.6922	1	0.2570	0	0.5187	-1	-0.0252	-0.0240	0.1697	0.1444
8	0.5516	0.5977	0.7020	1	0.3875	0	0.5281	-1	0.1713	0.1579	0.2494	0.1783
9	0.5831	0.6204	0.7058	1	0.4591	0	0.5326	-1	0.2993	0.2745	0.2878	0.2003
10	0.5984	0.6311	0.7072	1	0.4963	0	0.5345	-1	0.3688	0.3387	0.3110	0.2106
11	0.6057	0.6362	0.7077	1	0.5148	0	0.5354	-1	0.4054	0.3723	0.3225	0.2162
12	0.6091	0.6385	0.7080	1	0.5238	0	0.5358	-1	0.4236	0.3902	0.3328	0.2190
13	0.6107	0.6396	0.7080	1	0.5281	0	0.5361	-1	0.4329	0.4018	0.3473	0.2208
14	0.6114	0.6401	0.7081	1	0.5302	0	0.5363	-1	0.4385	0.4089	0.3568	0.2227
15	0.6118	0.6403	0.7081	1	0.5312	0	0.5365	-1	0.4415	0.4129	0.3626	0.2239
16	0.6119	0.6404	0.7081	1	0.5317	0	0.5365	-1	0.4431	0.4150	0.3659	0.2248
17	0.6120	0.6404	0.7081	1	0.5320	0	0.5366	-1	0.4439	0.4162	0.3677	0.2253
18	0.6120	0.6404	0.7081	1	0.5321	0	0.5366	-1	0.4443	0.4168	0.3686	0.2256
19	0.6120	0.6405	0.7081	1	0.5321	0	0.5366	-1	0.4445	0.4171	0.3691	0.2257
20	0.6120	0.6405	0.7081	1	0.5322	0	0.5367	-1	0.4446	0.4172	0.3693	0.2258
21	0.6120	0.6405	0.7081	1	0.5322	0	0.5367	-1	0.4446	0.4173	0.3695	0.2259

5 Výsledné hodnoty hodnotovej funkcie v numerickej metóde stochastického OR v konkrétnych krokoch pre úlohu o obchodnej prevádzke

V tabuľkách hodnôt prislúcha j-ty riadok j-temu kroku a i-ty stĺpec stavu i-*1*.

V v

25x7 double

	1	2	3	4	5	6	7
1	1.5097	2.0787	2.4073	2.6069	2.7458	2.8725	3.0303
2	1.3391	1.8509	2.1461	2.3250	2.4495	2.5630	2.7049
3	1.1856	1.6459	1.9110	2.0714	2.1827	2.2844	2.4119
4	1.0477	1.4616	1.6995	1.8430	1.9425	2.0335	2.1481
5	0.9236	1.2957	1.5092	1.6375	1.7263	1.8076	1.9106
6	0.8121	1.1466	1.3380	1.4526	1.5316	1.6042	1.6966
7	0.7119	1.0125	1.1839	1.2861	1.3563	1.4210	1.5039
8	0.6220	0.8919	1.0453	1.1362	1.1984	1.2559	1.3303
9	0.5411	0.7836	0.9206	1.0013	1.0562	1.1072	1.1739
10	0.4686	0.6862	0.8085	0.8798	0.9281	0.9732	1.0330
11	0.4036	0.5987	0.7076	0.7705	0.8127	0.8524	0.9059
12	0.3453	0.5202	0.6169	0.6720	0.7086	0.7435	0.7913
13	0.2930	0.4497	0.5354	0.5833	0.6149	0.6453	0.6880
14	0.2464	0.3864	0.4620	0.5035	0.5303	0.5567	0.5947
15	0.2047	0.3297	0.3961	0.4315	0.4540	0.4766	0.5105
16	0.1675	0.2790	0.3368	0.3666	0.3852	0.4044	0.4345
17	0.1345	0.2335	0.2835	0.3081	0.3230	0.3391	0.3658
18	0.1053	0.1929	0.2356	0.2553	0.2667	0.2801	0.3038
19	0.0796	0.1567	0.1924	0.2075	0.2158	0.2267	0.2477
20	0.0572	0.1245	0.1535	0.1642	0.1697	0.1785	0.1970
21	0.0379	0.0958	0.1182	0.1248	0.1279	0.1348	0.1511
22	0.0217	0.0702	0.0860	0.0890	0.0901	0.0953	0.1096
23	0.0089	0.0473	0.0561	0.0561	0.0561	0.0597	0.0715
24	0	0.0266	0.0266	0.0266	0.0266	0.0266	0.0372
25	0	0	0	0	0	0	0

6 Hodnoty optimálnych riadení v numerickej metóde stochastického OR v konkrétnych krokoch pre úlohu o obchodnej prevádzke

V v

25x7 double

	1	2	3	4	5	6	7
1	1	1	1	1	1	1	2
2	1	1	1	1	1	1	2
3	1	1	1	1	1	1	2
4	1	1	1	1	1	1	2
5	1	1	1	1	1	1	2
6	1	1	1	1	1	1	2
7	1	1	1	1	1	1	2
8	1	1	1	1	1	1	2
9	1	1	1	1	1	1	2
10	1	1	1	1	1	1	2
11	1	1	1	1	1	1	2
12	1	1	1	1	1	1	2
13	1	1	1	1	1	1	2
14	1	1	1	1	1	1	2
15	1	1	1	1	1	1	2
16	1	1	1	1	1	1	2
17	1	1	1	1	1	1	2
18	1	1	1	1	1	1	2
19	1	1	1	1	1	1	2
20	1	1	1	1	1	1	2
21	1	1	1	1	1	1	2
22	1	1	1	1	1	1	2
23	1	1	1	1	1	1	2
24	1	1	1	1	1	1	2
25	0	0	0	0	0	0	0

7 Hodnoty Q-funkcie v algoritme iterovania hodnôt pri aplikovaní akcie *1 - jeden zamestnanec pri pokladni* v konkrétnych iteráciach pre úlohu o obchodnej prevádzke

V každej tabuľke hodnôt prislúcha j-ty riadok j-tej iterácii a i-ty stĺpec i-temu stavu.

Q_1 Q_2

65x7 double

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333
3	0.1000	0.5333	0.6333	0.6333	0.6333	0.6733	0.7133
4	0.2200	0.7133	0.8733	0.9033	0.9153	0.9673	1.0193
5	0.3460	0.8753	1.0803	1.1409	1.1691	1.2319	1.2911
6	0.4702	1.0238	1.2623	1.3505	1.3959	1.4679	1.5314
7	0.5893	1.1602	1.4243	1.5359	1.5976	1.6785	1.7458
8	0.7016	1.2855	1.5695	1.7007	1.7770	1.8662	1.9370
9	0.8066	1.4003	1.7000	1.8475	1.9365	2.0334	2.1075
10	0.9041	1.5054	1.8177	1.9785	2.0786	2.1824	2.2595
11	0.9941	1.6015	1.9238	2.0958	2.2052	2.3150	2.3949
12	1.0769	1.6891	2.0197	2.2008	2.3181	2.4333	2.5156
13	1.1529	1.7690	2.1062	2.2949	2.4190	2.5386	2.6230
14	1.2224	1.8418	2.1844	2.3794	2.5091	2.6326	2.7188
15	1.2860	1.9079	2.2550	2.4552	2.5896	2.7164	2.8042
16	1.3440	1.9680	2.3188	2.5233	2.6617	2.7911	2.8803
17	1.3968	2.0225	2.3763	2.5844	2.7262	2.8579	2.9482
18	1.4448	2.0720	2.4283	2.6394	2.7839	2.9176	3.0088
19	1.4885	2.1169	2.4753	2.6888	2.8356	2.9709	3.0629
20	1.5282	2.1575	2.5176	2.7332	2.8819	3.0186	3.1112
21	1.5642	2.1943	2.5559	2.7732	2.9235	3.0613	3.1544
22	1.5968	2.2276	2.5903	2.8091	2.9607	3.0994	3.1931
23	1.6264	2.2578	2.6215	2.8414	2.9941	3.1336	3.2276
24	1.6532	2.2850	2.6495	2.8704	3.0241	3.1642	3.2585
25	1.6774	2.3096	2.6748	2.8965	3.0509	3.1916	3.2862

Q_1 Q_2

65x7 double

	1	2	3	4	5	6	7
26	1.6993	2.3319	2.6976	2.9200	3.0751	3.2162	3.3110
27	1.7192	2.3520	2.7182	2.9411	3.0967	3.2382	3.3332
28	1.7371	2.3701	2.7367	2.9601	3.1162	3.2580	3.3532
29	1.7533	2.3865	2.7534	2.9772	3.1336	3.2757	3.3710
30	1.7679	2.4013	2.7685	2.9926	3.1493	3.2916	3.3870
31	1.7812	2.4147	2.7821	3.0065	3.1634	3.3059	3.4014
32	1.7931	2.4267	2.7943	3.0189	3.1761	3.3187	3.4142
33	1.8039	2.4376	2.8053	3.0301	3.1874	3.3302	3.4258
34	1.8136	2.4473	2.8152	3.0402	3.1977	3.3405	3.4362
35	1.8224	2.4562	2.8242	3.0493	3.2068	3.3498	3.4455
36	1.8303	2.4641	2.8322	3.0574	3.2151	3.3581	3.4539
37	1.8374	2.4713	2.8395	3.0647	3.2225	3.3656	3.4614
38	1.8438	2.4778	2.8460	3.0714	3.2292	3.3723	3.4681
39	1.8496	2.4836	2.8519	3.0773	3.2352	3.3784	3.4742
40	1.8549	2.4889	2.8572	3.0826	3.2406	3.3838	3.4796
41	1.8596	2.4936	2.8619	3.0875	3.2454	3.3887	3.4845
42	1.8638	2.4979	2.8662	3.0918	3.2498	3.3931	3.4889
43	1.8677	2.5017	2.8701	3.0957	3.2537	3.3970	3.4929
44	1.8711	2.5052	2.8736	3.0992	3.2573	3.4005	3.4964
45	1.8742	2.5083	2.8767	3.1023	3.2604	3.4037	3.4996
46	1.8770	2.5111	2.8795	3.1052	3.2633	3.4066	3.5025
47	1.8795	2.5136	2.8821	3.1077	3.2658	3.4092	3.5051
48	1.8818	2.5159	2.8844	3.1100	3.2682	3.4115	3.5074
49	1.8839	2.5180	2.8864	3.1121	3.2702	3.4136	3.5095
50	1.8857	2.5198	2.8883	3.1140	3.2721	3.4154	3.5113

Q_1 Q_2

65x7 double

	1	2	3	4	5	6	7
50	1.8857	2.5198	2.8883	3.1140	3.2721	3.4154	3.5113
51	1.8874	2.5215	2.8899	3.1156	3.2738	3.4171	3.5130
52	1.8889	2.5230	2.8915	3.1171	3.2753	3.4187	3.5146
53	1.8902	2.5243	2.8928	3.1185	3.2767	3.4200	3.5159
54	1.8914	2.5255	2.8940	3.1197	3.2779	3.4212	3.5172
55	1.8925	2.5266	2.8951	3.1208	3.2790	3.4223	3.5183
56	1.8935	2.5276	2.8961	3.1218	3.2800	3.4233	3.5192
57	1.8944	2.5285	2.8970	3.1227	3.2809	3.4242	3.5201
58	1.8952	2.5293	2.8978	3.1235	3.2817	3.4250	3.5209
59	1.8959	2.5300	2.8985	3.1242	3.2824	3.4258	3.5217
60	1.8965	2.5307	2.8992	3.1249	3.2830	3.4264	3.5223
61	1.8971	2.5312	2.8997	3.1255	3.2836	3.4270	3.5229
62	1.8976	2.5318	2.9003	3.1260	3.2842	3.4275	3.5234
63	1.8981	2.5322	2.9007	3.1265	3.2846	3.4280	3.5239
64	1.8985	2.5327	2.9012	3.1269	3.2851	3.4284	3.5243
65	1.8989	2.5330	2.9015	3.1273	3.2854	3.4288	3.5247

8 Hodnoty Q-funkcie v algoritme iterovania hodnôt pri aplikovaní akcie *2 - obaja zamestnanci pri pokladni* v konkrétnych iteráciach pre úlohu o obchodnej prevádzke

Q_1 Q_2

65x7 double

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	-0.2000	0.3000	0.3000	0.3000	0.3000	0.3000	0.4667
3	-0.1000	0.4500	0.6000	0.6000	0.6000	0.6300	0.8067
4	0.0200	0.6075	0.8250	0.8700	0.8790	0.9180	1.1127
5	0.1460	0.7560	1.0207	1.1022	1.1301	1.1799	1.3809
6	0.2702	0.8957	1.1937	1.3059	1.3537	1.4140	1.6201
7	0.3893	1.0260	1.3486	1.4860	1.5521	1.6230	1.8334
8	0.5016	1.1467	1.4882	1.6460	1.7283	1.8091	2.0238
9	0.6066	1.2581	1.6143	1.7887	1.8850	1.9749	2.1935
10	0.7041	1.3606	1.7283	1.9164	2.0246	2.1225	2.3448
11	0.7941	1.4545	1.8316	2.0308	2.1490	2.2540	2.4795
12	0.8769	1.5405	1.9251	2.1334	2.2601	2.3711	2.5995
13	0.9529	1.6191	2.0097	2.2256	2.3594	2.4755	2.7065
14	1.0224	1.6907	2.0863	2.3084	2.4482	2.5687	2.8018
15	1.0860	1.7560	2.1556	2.3829	2.5276	2.6518	2.8867
16	1.1440	1.8153	2.2183	2.4498	2.5987	2.7260	2.9625
17	1.1968	1.8693	2.2751	2.5101	2.6624	2.7923	3.0301
18	1.2448	1.9183	2.3263	2.5642	2.7194	2.8516	3.0905
19	1.2885	1.9628	2.3727	2.6130	2.7706	2.9046	3.1444
20	1.3282	2.0031	2.4145	2.6569	2.8164	2.9520	3.1925
21	1.3642	2.0396	2.4523	2.6963	2.8576	2.9944	3.2356
22	1.3968	2.0727	2.4865	2.7319	2.8945	3.0323	3.2741
23	1.4264	2.1026	2.5173	2.7639	2.9276	3.0664	3.3085
24	1.4532	2.1297	2.5451	2.7926	2.9573	3.0968	3.3394
25	1.4774	2.1542	2.5702	2.8185	2.9840	3.1241	3.3670

Q_1 Q_2

65x7 double

	1	2	3	4	5	6	7
26	1.4993	2.1763	2.5929	2.8418	3.0080	3.1486	3.3917
27	1.5192	2.1963	2.6133	2.8628	3.0295	3.1706	3.4139
28	1.5371	2.2144	2.6317	2.8817	3.0489	3.1903	3.4338
29	1.5533	2.2307	2.6484	2.8987	3.0663	3.2079	3.4516
30	1.5679	2.2455	2.6633	2.9140	3.0819	3.2238	3.4676
31	1.5812	2.2588	2.6768	2.9278	3.0959	3.2380	3.4819
32	1.5931	2.2708	2.6890	2.9401	3.1085	3.2508	3.4948
33	1.6039	2.2816	2.7000	2.9513	3.1198	3.2623	3.5063
34	1.6136	2.2914	2.7099	2.9613	3.1300	3.2726	3.5167
35	1.6224	2.3002	2.7188	2.9704	3.1392	3.2818	3.5260
36	1.6303	2.3081	2.7268	2.9785	3.1474	3.2901	3.5343
37	1.6374	2.3153	2.7340	2.9858	3.1548	3.2976	3.5418
38	1.6438	2.3218	2.7405	2.9924	3.1615	3.3043	3.5486
39	1.6496	2.3276	2.7464	2.9983	3.1674	3.3103	3.5546
40	1.6549	2.3328	2.7517	3.0037	3.1728	3.3158	3.5601
41	1.6596	2.3376	2.7564	3.0085	3.1777	3.3206	3.5650
42	1.6638	2.3418	2.7607	3.0128	3.1820	3.3250	3.5694
43	1.6677	2.3456	2.7646	3.0167	3.1859	3.3290	3.5733
44	1.6711	2.3491	2.7681	3.0202	3.1895	3.3325	3.5769
45	1.6742	2.3522	2.7712	3.0233	3.1926	3.3357	3.5800
46	1.6770	2.3550	2.7740	3.0261	3.1955	3.3385	3.5829
47	1.6795	2.3576	2.7766	3.0287	3.1981	3.3411	3.5855
48	1.6818	2.3598	2.7788	3.0310	3.2004	3.3434	3.5878
49	1.6839	2.3619	2.7809	3.0331	3.2024	3.3455	3.5899
50	1.6857	2.3637	2.7828	3.0349	3.2043	3.3474	3.5918

Q_1 Q_2

65x7 double

	1	2	3	4	5	6	7
50	1.6857	2.3637	2.7828	3.0349	3.2043	3.3474	3.5918
51	1.6874	2.3654	2.7844	3.0366	3.2060	3.3491	3.5935
52	1.6889	2.3669	2.7859	3.0381	3.2075	3.3506	3.5950
53	1.6902	2.3682	2.7873	3.0395	3.2089	3.3520	3.5964
54	1.6914	2.3694	2.7885	3.0407	3.2101	3.3532	3.5976
55	1.6925	2.3705	2.7896	3.0418	3.2112	3.3543	3.5987
56	1.6935	2.3715	2.7906	3.0428	3.2122	3.3553	3.5997
57	1.6944	2.3724	2.7915	3.0436	3.2131	3.3562	3.6006
58	1.6952	2.3732	2.7923	3.0444	3.2139	3.3570	3.6014
59	1.6959	2.3739	2.7930	3.0452	3.2146	3.3577	3.6021
60	1.6965	2.3746	2.7936	3.0458	3.2152	3.3583	3.6028
61	1.6971	2.3752	2.7942	3.0464	3.2158	3.3589	3.6033
62	1.6976	2.3757	2.7947	3.0469	3.2163	3.3595	3.6039
63	1.6981	2.3761	2.7952	3.0474	3.2168	3.3599	3.6043
64	1.6985	2.3766	2.7956	3.0478	3.2172	3.3604	3.6048
65	1.6989	2.3770	2.7960	3.0482	3.2176	3.3607	3.6052