ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická
Katedra kybernetiky

Metody generování strategií pro stochastické hry s nulovým součtem

Bakalářská práce

Studijní program: Otevřená informatika (bakalářský)
Studijní obor: Informatika a počítačové vědy

Vedoucí práce: Mgr. Branislav Bošanský, Ph.D.

Marek Kryška

Praha 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Marek K r y š k a

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Metody generování strategií pro stochastické hry s nulovým součtem

Pokyny pro vypracování:

Dvouhráčové nekonečné stochastické hry, ve kterých jsou ohodnocení pouze v terminálních stavech, jsou důležitou třídou her s řadou potenciálních aplikací. Bohužel v současnosti neexistují efektivní a v praxi použitelné algoritmy pro výpočet optimálních strategií pro tuto třídu her. Standardní algoritmy, které využívají iterativní výpočet hodnot/strategií, mohou v nejhorším případě potřebovat až dvojitě exponenciální počet iterací.
Cílem studenta je proto (1) prozkoumat možnosti využití inkrementálního rozšiřovaní prostoru strategií pro řešení stochastických her a adaptace přístupů úspěšných v konečných hrách, (2) experimentálně porovnat nově navržené algoritmy se standardními algoritmy pro řešení nekonečných stochastických modelů na sadě konkrétních her.

Seznam odborné literatury:

[1] Kristoffer Arnsfelt Hansen, Rasmus IbsenJensen, and Peter Bro Miltersen. "The complexity of solving reachability games using value and strategy iteration." Computer Science–Theory and Applications. Springer Berlin Heidelberg, 2011. 7790.
[2] Branislav Bosansky, Viliam Lisy, Jiri Cermak, Roman Vitek, and Michal Pechoucek: Using Doubleoracle Method and Serialized AlphaBeta Search for Pruning in Simultaneous Moves Games. In Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI). 2013.
[3] Yoav Shoham, and Kevin LeytonBrown. Multiagent systems: Algorithmic, gametheoretic, and logical foundations. Cambridge University Press, 2009.

Vedoucí bakalářské práce: Mgr. Branislav Bošanský, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 14. 1. 2015

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne: .....

.....
Podpis autora práce

Poděkování

Děkuji Mgr. Branislavu Bošanskému, Ph.D za jeho aktivní pomoc a železnou trpělivost při tvorbě této bakalářské práce. Jeho vedení a připomínky mi velmi usnadnily její přípravu.

Abstrakt

Teorie her se v dnešní době promítá do stálé většího počtu disciplín. Lze její pomocí řešit jak samotné společenské hry, od níž je název teorie her odvozen, tak především ekonomické, hospodářské nebo vojenské cíle. Problematika zasahuje do každého rozhodovacího procesu, kde se účastníci snaží nějak predikovat chování ostatních vlivů - jiných hráčů a maximalizovat užitek svůj nebo nějaké skupiny. Ovšem mnohé převedení reálných případů do matematického modelu a případných dalších speciálních forem teorie her s sebou nese spoustu problémů. Kromě neuchopitelnosti skutečného světa do konečného počtu akcí a stavů je to hlavně problém rozsáhlosti a velikosti dat. S rostoucí velikostí hry roste i čas jejího zpracování, což může být kritické, pokud nám záleží na přesném výsledku v reálném čase.

Klíčová slova

stochastická nekonečná hra, Nashovo eqilibrium, hodnota hry, iterační algoritmus

Abstract

Game Theory is today reflected in a growing number of disciplines. With it we can solved the classic board games from which exists the current name of the game theory, but mainly economic, military or economic targets. The issue affects every decision-making process, where participants try to somehow predict the behavior of other influences - other players and maximize some profit. However, many real cases carries in a mathematical model or any other special forms of game theory a lot of problems. In addition to the difficult transfer of the real world into a finite number of events and conditions, it is mainly a problem of the breadth and size of data. With the growing size of the game grows the time of processing, which is critical if we depend on accurate results in real time.

Keywords

stochastic infinite game, Nash equlibrium, game value, iteration algorithm

Obsah

Kapitola 1

Úvod

Vypočtení Nashova equliberia v nekonečné stochastické hře tvoří v teorii her velmi obtížně řešitelný problém. Nejlepší algoritmy pro jeho vyřešení dokázaly konvergovat pouze za omezených podmínek a na malých datech.
Potenciál využití tohoto typu hry je ovšem obrovský. Například patrolování hlídek, tj. hra obránců a útočníků, kdy nemáme jasný začátek nebo konec hry, což může být například ochrana serverů před hackerským útokem nebo reálné lodě hlídkující před piráty. Stejně tak to mohou být klasické stolní hry, které standardně pracují s velkou sadou dat. Jak nám mnohé pokusy ukazují, daří se občas vyřešit stochastickou hru, například algoritmus pro přibližné řešení strategií equlibria k tříhráčovému Texas hold'em [11].
Zajímají nás především takové hry, které mají své utility v terminálních stavech. To někdy vede k tomu, že hodnota equliberia může díky vnitřním cyklům hry konvergovat velmi pomalu.
V této práci se pokusím ukázat, zda existuje nějaká možnost, jak zmenšit prostor hry a tím pomoci k rychlejší konvergenci iteračních algoritmů. Nejprve zavedu potřebné definice z teorie her, abych mohl ukázat nejznámější algoritmy pro řešení Nashova equlibria. V dalších kapitolách zavedu nový algoritmus, vycházející z principu Double Oracle [10] algoritmu a provedu s ním experimenty na vlastní sadě nagenerovaných her.

Kapitola 2

Teorie her

Pro začátek je třeba definovat základní pojmy teorie her z [3], především pak jistou podmnožinu tohoto rozsáhlého oboru, kterou budeme v dalších částech práce používat. Pro naše účely se budeme zabývat striktní podmnožinou her a to takových, která jsou v s nulovým součtem, jsou stochastické [4], nekonečné (budeme je zapisovat do soutavy matic) a mají pouze dva hráče.

2.1 Hráč

Hráč je entita, která rozhoduje jaké akce ve hře bude hrát. U hráče je důležitá jeho racionálnost, tj. jakým způsobem akce volí, případně podle jakých strategií. Jakožto racionálního hráče definujeme takového, který volí akce strategií tak, aby maximalizoval svůj zisk.
V tomto textu ho budeme značit jako *Hráč i*, kde i je index hráče.

2.2 Stav hry

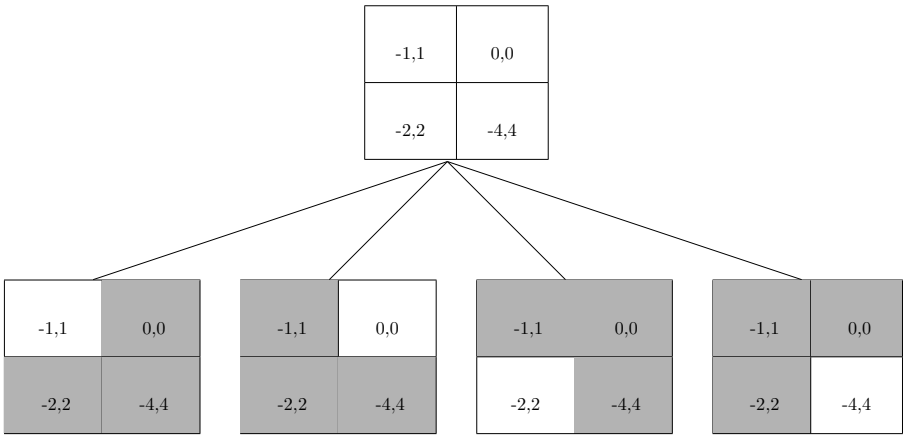
Je určitá situace ve hře. Hráči v ní vyberou své akce a poté se podle zahraných akcí hra přesouvá do dalšího stavu. Speciálními stavy ve hře jsou stavy koncové, resp. terminální. To jsou stavy ve kterých hra končí, kdy se nelze dostat do žádného dalšího stavu a hráči získávají nějakou výhru danou utilitní funkcí. Ostatním stavům říkáme přechodné a slouží k přechodu do jiných předchodných nebo koncových stavů. Dalším speciálním stavem je stav počáteční, kde hra začíná.
V tomto textu budeme stav hry reprezentovat jako matici, kde každé políčko udává kombinaci dvou akcí jednotlivých hráčů. Více v kapitole 3 a na **Obrázku 3.2**.

2.3 Hra

Je soubor pravidel a podmínek, za jakých mohou jednotliví hráči volit akce a v jakém pořadí, které pak vedou k nějakému výsledku. Hru lze zapsat jak maticově, tak i formou stromu, kde každý uzel (v matici její jedna hrana) přiřazuje danému hráči jeho akce.

U her v našem případě předpokládáme, že každý vyplacený zisk za dohrání hry jendomu hráči je na úkor ostatních hráčů. To znamená, že při zahrání všech strategií všech hráčů dává suma vyplacených výher dohromady nulu. Takové hry definujeme jako hry s *nulovým součtem.*

Pro ilustraci je na **Obrázku 2.1** uvedena hra s nulovým součtem, kde po dohrání první akce hráči okamžitě dostávají hodnotu utility. Graf znázorňuje všechny možnosti, jak mohou hráči své akce volit. Bílá pole určují, která akce byla zvolena.



Obrázek 2.1: Příklad hry s nulovým součtem ve formě grafu

Tato hra je převedená do maticové formy na **Obrázku 2.2**. Dimenze matice přiřazuje hráčům počet jejich akcí, tady konkrétně pro Hráče 1 akce $a_{i,j}$ a Hráči 2 $b_{i,j}$.

Hra: 1	$b_{1,1}$	$b_{1,2}$
$a_{1,1}$	-1,1	0,0
$a_{1,2}$	-2,2	-4,4

Obrázek 2.2: Příklad hry s nulovým součtem v maticové formě

Zde nazýváme matici hrou. Je to speciální případ pro jednostavovou hru. Jako hru však budeme dále v textu reprezentovat soustavu matic. Tedy jedna matice značí stav hry.

2.4 Hra v normální formě

Nejčastějším vyjádřením hry je hra v tzv. normálové formě. Jde o přesný popis uzavřené množiny všech akcí pro každého hráče hry. U hry v normálním tvaru se předpokládá, že hráči volí své akce zároveň a nemají přehled o tazích protihráčů [3].

Definice 1 *Konečná n-hráčová hra A je v normální formě, tj. trojice (N, A, u) pokud:*

- *N je konečná množina n-hráčů, idexované* $i = \{1, 2, ..., n\}$
- $A = A_1 \times ... \times A_n$, A_i *je konečná množina akcí dostupných pro hráče i*
- $u = u_1 \times ... \times u_n$, *kde* $u_i \mapsto \Re$ *je utilitní (výplatní) funkce pro hráče i*

Definice 2 *Stochastickou n-hráčovou hru A nazýváme nekonečnou, jestliže nemáme zaručenou konvergenci strategií všech hráčů v reálném čase.*

To znamená, že hra je sice popsána konečně mnoho stavy, ale jejich vzájemné propojení může vytvářet cykly a tím zvyšovat časovou složitost řešení hry.
[4] Pro generování nekonečné hry můžeme využít například *konečný automat.*

2.5 Determinismus a stochasticita

Pojmem stochasticita (náhodnost) resp. stochastická hra rozumíme hru takovou, kde figuruje prvek náhody. Každý souhrn akcí vede na pravděpodobnostní rozložení možných následných cílů (stavů). V teorii her nazýváme stochastické hry často hrami nekonečnými se stochastickými přechody [6].
Naopak determinismus (předurčenost), případně deterministická hra, je hra taková, kde souhrn akcí všech hráčů vede přesně k jednomu stavu ve hře. Deterministická hra je tedy v podstatě hra stochastická, kde víme se stoprocentní pravděpodobností návaznost stavů.

2.6 Utilita

Utilita neboli zisk ohodnocuje danou hru. Může být získávána jak po jednotlivých kolech, tak po dohrání celé hry. Nejčastěji se reprezentuje formou tzv. *výplatní matice.* Ta určuje, jakou hodnotu daný hráč pro příslušné kolo získá. Druhým způsobem je reprezentace formou *utilitní funkce*, která každé kombinací akcí přiřazuje hodnotu výhry. Podle těchto hodnot pak hráč tvoří svou strategii, aby takovýto zisk maximalizoval.
V našem případě budeme uvažovat, že zisk je pouze v koncových stavech. Jelikož se jedná o hry s nulovým součtem, hodnota výhry jednoho hráče se bude rovnat ztrátě hráče poraženého.

2.7 Strategie

2.7.1 Strategie pro hry v normální formě

Strategie definuje akce, které hráč zvolí pro stav hry. Pokud hráč volí pouze jednu akci pro každý stav, říká se se takové startegii *ryzí startegie.* Jestliže jsou akce

náhodné (podle nějaké distribuce pravděpodobnosti), nazýváme takovou startegii *smíšenou startegií*. Množině strategií přes všechny akce říkáme *profil dané strategie*. Strategie může být založená na různých předpovědích. Ty určujeme podle toho, jak budou protihráči hrát, což může záviset na nějaké vnější informaci nebo rozhodnutí hráčů v předchozích kolech hry (minulost hry)[3].

2.7.2 Strategie pro stochastické hry

Strategie pro stochastické hry je tedy strategií smíšenou. Můžeme ji reprezentovat jako vektor čísel mezi 0 až 1 o velikosti počtu akcí pro danou herní matici. Obecně ji definujeme jako:

Definice 3 *Nechť je množna (N, A, u) hra v normální formě a pro každou množinu X nechť je* $\Pi(X)$ *množina všech pravděpodobnostních rozložení nad množinou X. Pak množina smíšených strategií pro hráče i je* $S_i = \Pi(A_i)$.

Definice 4 *Stochastickou strategií jednoho hráče myslíme množinu smíšených strategií* $\{s_i^1, s_i^2, ..., s_i^k\}$, *kde i je index hráče a k je počet stavů hry.*

2.7.3 Nejlepší reakce - optimální strategie

Pokud víme, jaké akce budou protihráči volit (například předpokládáme, že jsou plně inteligentní a hrají své optimální strategie), jsme schopni dopočítat ryzí strategii takovou, že maximalizuje prospěch při takto zahraných profilech strategií. Taková strategie se nazývá *nejlepší reakcí* na množinu strategií protihráčů [9].

2.8 Nashovo equlibrium

Množinu strategií nazýváme Nashovým equlibriem, [9] pokud každá strategie π_i je nejlepší možnou reakcí na všechny strategie π_j, kde i, j jsou indexy hráčů, N je jejich počet a platí $j \neq i$, $j, i \in \{1, 2, .., N\}$. V našem případě dvou protihráčů je N = 2. Strategie každého hráče je tedy nejlepší možnou reakcí na všechny ostatní strategie zbylých hráčů, tudíž každý hráč hraje svoji optimální strategii. Očekávaný zisk prvního hráče se nazývá *hodnota hry* a značíme ji ν^*. Ačkoliv může na jedné hře existovat více Nashových equlibrií, hodnota hry pro tento optimální případ zůstává pokaždé stejná (toto nám zaručuje předpoklad, že se pohybujeme ve hrách s nulovým součtem).

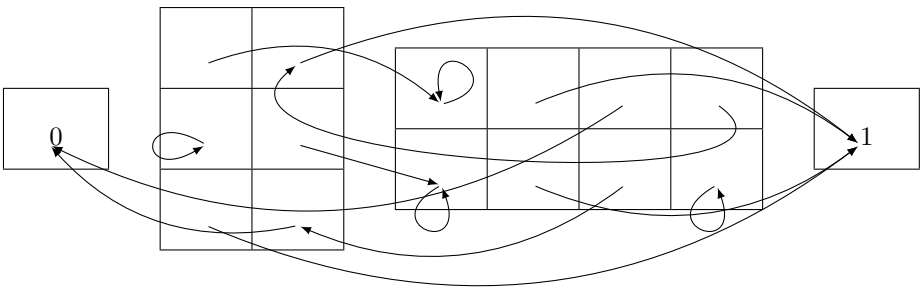
Kapitola 3

Iterační algoritmy, výpočet strategií a hodnota hry

V této sekci si zavedeme dva nejznámější algoritmy pro výpočet hodnoty hry ν^* a hodnot strategií [6] [5] Nashova equlibria. Pro názornost zavádíme náhodně vygenerovanou hru, na které si ukážeme, jak algoritmy fungují.

Definice 5 *Mějme hru zadanou tak, že má pouze deterministické přechody mezi stavy, dva hráče a velikost o N stavech. Přidejme ke hře další dva terminální stavy:* ***Stav 0*** *a* ***Stav N+1****, kde N+1 stav určuje výhru prvního hráče a stav 0 výhru druhého hráče.*

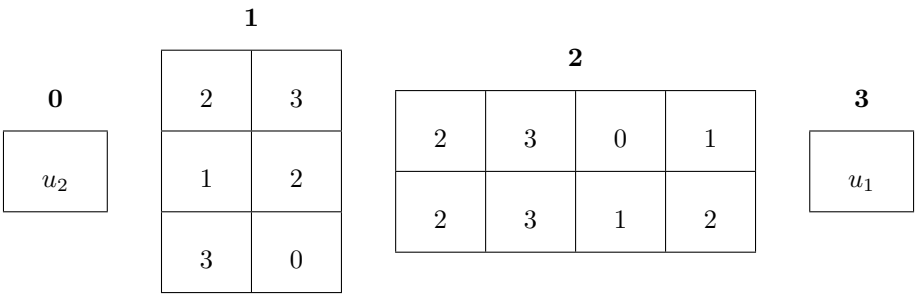
Uvažujme konkrétně zadanou hru z **Obrázku 3.1**. K ohodnocení terminálních stavů zvolíme vhodnou konstantu, tedy v našem případě budou výsledky z intervalu *<0,1>*. Zbylé stavy, jež nemají zapsanou žádnou hodnotu, jsou přechodné do stavu, kam směřuje šipka. Hráč 1 volí řádkové akce a Hráč 2 volí akce sloupcové. Hráč 1 má tedy u prvního stavu k dispozici tři akce a Hráč 2 má například u druhého stavu k dispozici akce čtyři.



Obrázek 3.1: Schema hry

Pokud třeba Hráč 1 zvolí v první stavu svojí druhou akci a Hráč 2 zvolí akci první, přechod ze stavu jedna je zpět do stejného stavu. Kdyby ale Hráč 1 zvolil svojí první akci ve stavu jedna, dostali bychom se do terminálního stavu, hra by skončila a Hráč 1 by vyhrál hru. Takto hráči volí své akce, aby maximalizovali šanci probojovat se ke svým terminálním cílům skrze všechny stavy hry.

Pro přehlednost budeme hru zapisovat maticemi přechodu z **Obrázek 3.2**, kde hodnota buňky matice určuje stav, do kterého se hráči při zahrání příslušných akcí dostanou. Hodnoty utilit koncových stavů zvolíme $u_2 = 0$ a $u_1 = 1$, ale aby nedošlo záměně za přechod, do matice je nepíšeme.



Obrázek 3.2: Hra přepsaná na matici přechodu

3.1 Lineární program pro optimální strategie

Nashovo equlibrium jsme schopni najít pomocí následujícího lineárního programu [7]. Ten vychází z lineárních programů pro jednotlivé hráče, kde se oba hráči N_i, kde $i \in \{1, 2\}$ snaží maximalizovat svojí hodnotu hry U_i^*. Jelikož se pohybujeme ve hrách s nulovým součtem, víme $U_1^* = -U_2^*$. První hráč se tedy snaží maximalizovat U_1^* přes všechny akce a stretegii protihráče a druhý hráč usiluje o totéž s výjimkou toto, že U_1^* minimalizuje. Pokud lineární programy obou hráčů spojíme do jednoho, dostáváme:

Definice 6 *Lineární program pro optimální strategie obou hráčů je [3]: minimalizace U_1^* za podmínek*

$$\sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k + r_1^j = U_1^* \qquad \forall j \in A_1$$

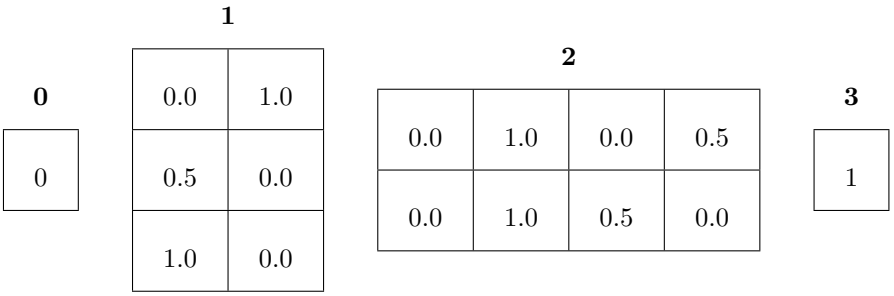
$$\sum_{k \in A_2} s_2^k = 1$$

$$s_2^k \geq 0 \qquad \forall k \in A_2$$

$$r_1^j \geq 0 \qquad \forall j \in A_1$$

3.1.1 Příklad výpočtu hodnoty hry a strategií

Pro výpočet hodnoty hry, potažmo strategií jednotlivých hráčů, bychom potřebovali znát ohodnocení jednotlivých akcí ve stavech hry. To se dá vypočítat z hodnotové iterace. To jak funguje si ukážeme později, proto si vypůjčíme třetí iteraci tohoto algoritmu, abychom na něm demonstrovali náš lineární program. Máme tedy stavy hry po třetím běhu programu pro hodnotovou iteraci z **Obrázku 3.2** ohodnocené takto:



Obrázek 3.3: Hodnota vektoru stavů při třetí iteraci algoritmu Value Iteration

Stav 1

Dosazením do definice dostáváme soustavu rovnic s podmínkou:

$\mathbf{min}\ U_1^*$
$0.0 \cdot s_2^0 + 1.0 \cdot s_2^1 + r_1^0 = U_1^*$
$0.5 \cdot s_2^0 + 0.0 \cdot s_2^1 + r_1^1 = U_1^*$
$1.0 \cdot s_2^0 + 0.0 \cdot s_2^1 + r_1^2 = U_1^*,$

kde suma s_2^k sčítá do jedničky a spolu se slackovými proměnnými r_1^j jsou rovny nebo větší nule.

Po vyřešení dostáváme:
$U_1^* = 0.5$
$s_2 = [0.5,\ 0.5]$
a snadno pak dopočítáme strategii prvního hráče:
$s_1 = [0.5,\ 0.0,\ 0.5]$

Tento příklad stavu je jednoduchý, lze ho vyřešit i úvahou. Pokud se podíváme na akce, které může zvolit sloupcový hráč (tj. *Hráč 2* v našem značení), je vidět, žě v obou sloupcích se objevuje vítězná hodnota pro oba hráče. Sloupcový hráč nemůže zablokovat řádkovému hráči přechod, kde se vyskytuje hodnota *1.0*. Je mu tedy jedno, kterou akci zvolí, protože v obou sloupcích je jeho vítězná hodnota *0.0*. Stejně tak hráč řádkový (tj. *Hráč 1*) vybírá mezi první a poslední akcí. Proto mají pro tyto řádky a sloupce jejich strategie hodnotu *0.5*. Z toho vyplývá i hodnota hry $U_1^* = 0.5$.

Stav 2

Analogicky bychom mohli dosadit do rovnic. Když se ale pozorně na hru podíváme, je zřejmé, že pokud sloupcový hráč zvolí svou první akci (přesun do svého výherního terminálního stavu), má výhru jistou, ať řádkový hráč hraje jakoukoliv strategii. Proto hodnota hry bude $U_1^* = 0.0$, sloupcový hráč bude hrát *ryzí strategii* $s_2 = [1.0, 0.0, 0.0, 0.0]$ a na strategii řádkového hráče nezáleží pro výpočet hodnoty hry.

3.2 Iterace hodnoty

První z iteračních algoritmů, nazvaný iterace hodnoty, pracuje na myšlence ohodnocování výsledku každých dvojic akcí podle hodnot hry stavu, do kterého podle zvolených akcí se hráči dostanou. Iterativní je tento algoritmus, jelikož opakovaně propočítává hodnoty jednotlivých stavů hry a končí, až pokud výsledky dvou po sobě jdoucích iterací mají rozdíl menší, než je ukončovací podmínka ε = *požadovaná přesnost* [8].

Data: Ukončovací podmínka přesnosti
Result: Hodnota hry
t := 0;
$v^0 := (0, ..., 0, 1)$; `// kde vektor` v^0 `je indexován 0, 1, ...,N, N+1`
while $|v_t - v_{t-1}| > \varepsilon$ **do**
    t := t+1;
    $v_0^t := 0$;
    $v_{N+1}^t := 1$;
    for $j \in \{1, 2, .., N\}$**do**
        $v_j^t := val(A_j(v^{t-1}))$;

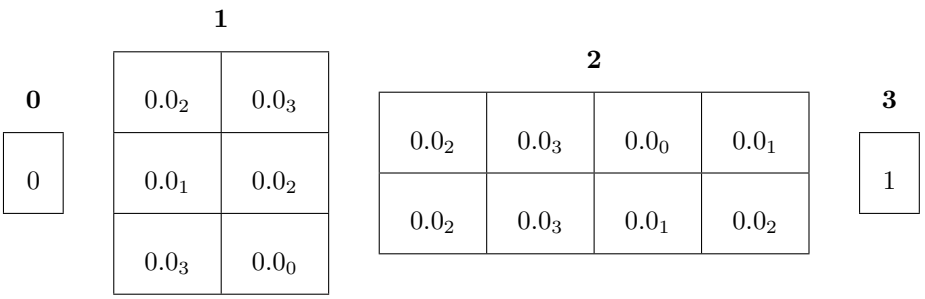
Algorithm 1: Iterace hodnoty [1]

Funkcí $val(A_j(v^{t-1}))$ rozumíme výpočet hodnoty hry. A_j zde reprezentuje množinu akcí pro j-tý stav hry A. Poté, co je spočtena hodnota hry pro všechny stavy, každá matice stavů ohodnotí své akce příslušnou hodnotou hry stavu, do kterého přechází. Toto ohodnocení je pak ještě přenásobeno pravděpodobností, jež určuje, zda se do stavu vůbec dostane.
V níže uvedených příkladech zavedeme ve stavech hry následující značení:
Hodnota v poli matice je hodnota utility a spodní index je stav do kterého se hráči po zahrání příslušných akcí dostanou.

3.2.1 Příklad iterace hodnoty

V pseudokódu algoritmu vidíme, že vektor v^0 má délku rovnu počtu stavů hry. Hodnoty jednotlivých v_i^0 určují hodnotu hry daného stavu, tedy první a poslední číslo vektoru v^0 určuje hodnotu hry terminálních stavů. To je triviálně hodnota terminálních stavů. Ostatním stavům nastavíme implicitně nuly. Každý z prvků vektoru je vlastně jedna matice. Terminální stavy mají rozměr 1x1 a ostatní přechodné podle našeho příkladu 3x2 a 2x4. Zvolíme ukončovací podmínku například takto: $\varepsilon = 0.01$, tím v praxi říkáme, že nám záleží pouze na prvních dvou desetinných místech výsledku.

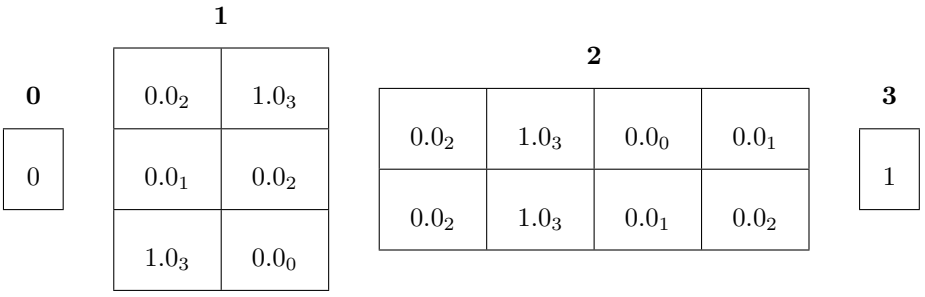
v^1 = (0.0, 0.0, 0.0, 1.0)



Obrázek 3.4: Algoritmus iterace hodnoty, t = 1

Začínáme iterovat. Za každou pozici vektoru v_i^t, kde t je počítadlo iterace, postupně počítáme hodnotu hry daného stavu podle lineárního programu. Poté přiřazujeme hodnoty her k jednotlivým akcím.

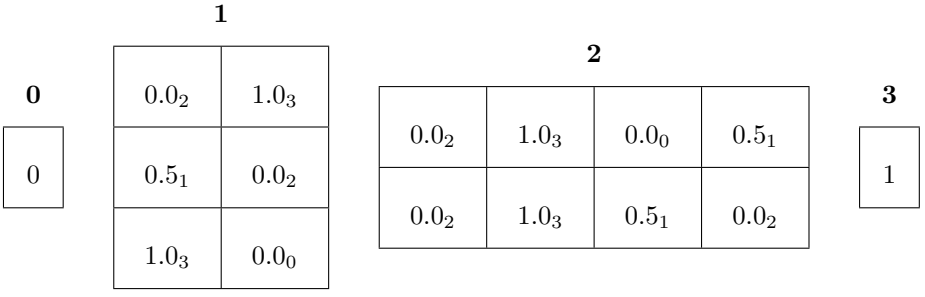
v^2 = (0.0, 0.5, 0.0, 1.0)



Obrázek 3.5: Algoritmus iterace hodnoty, t = 2

Algoritmus tak vlastně propisuje hodnoty *"od konce grafu"*. V první iteraci se objeví pouze hodnoty u stavů přechodných k stavům terminálním a až později se propíší i hodnoty dál, podle toho jak jsou navázané přechody mezi stavy.

v^3 = (0.0, 0.5, 0.0, 1.0)



Obrázek 3.6: Algoritmus iterace hodnoty, t = 3

Vidíme, že tento příklad je skutečně triviální. Algoritmus skončí po pouhých třech iteracích. Ukončovací konstanta $\varepsilon = 0.01$, tedy byla dokonce nadsazená. Jakákoliv na začátku zadaná vyšší přesnost výsledek neovlivní, protože další iterace nemění hodnoty her pro jednotlivé stavy. Hodnota hry je $U_1^* = 0.5$.

3.3 Iterace strategie

Druhý z iteračních algoritmů, tedy iterace strategií, je vlastně rozšířením hodnotové iterace. Znovu tu postupně dopočítáváme hodnoty her a přiřazujeme utility daným kombinacím akcí. Zároveň se však kontroluje, jestli neexistuje pro hráče lepší tzv. *optimální strategie*. Z iterace hodnoty zjistíme strategie pro stavy hry (přenásobené současnými strategiemi hráčů) a podle nich optimalizujeme (pokud to jde) iterované startegie.
Iterace strategie zpravidla konverguje velmi rychle na malých datech. Naopak na větších hrách je rychlost konvergence výrazně pomalejší než u iterace hodnoty. To je zapříčiněno tím, že se musí vyřešit velký prostor lineárních rovnic. Algoritmus ukončujeme (stejně jako u hodnotové iterace), pokud rozdíl ohodnocených stavů u dvou po sobě jdoucích iterací je menší než podmínka přesnosti ε [8].

Data: Ukončovací podmínka přesnosti
Result: Optimální strategie hráčů hry
t := 1;
s_1^1 := počáteční strategie pro Hráče 1;
while *true* **do**
  s_2^t :=optimální strategie na s_1^t;
  for $i \in$*{0, 1, 2,..,N, N+1}***do**
    $v_i^t := \mu_i(s_1^t, s_2^t)$;
  t := t+1;
  for $i \in \{1, 2, .., N\}$**do**
    if $val(A_i(v^{t-1})) > v^{t-1}$ **then**
      $s_{1,i}^t := maxmin(A_i(v^{t-1}))$;
    else
      $x_{1,i}^t := x_{1,i}^{t-1}$;

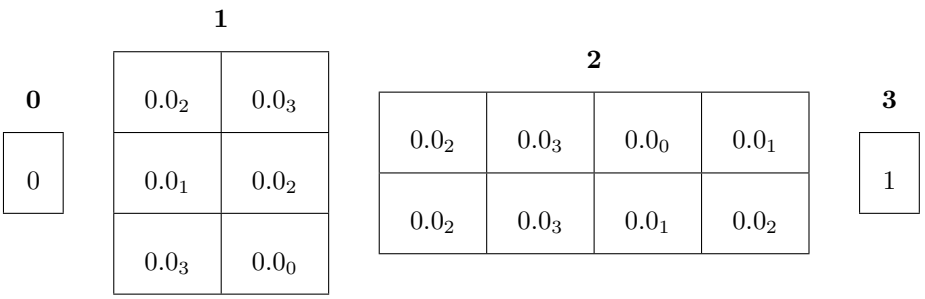
Algorithm 2: Iterace strategie [1]

Funkce $\mu_i(s_1^t, s_2^t)$ počítá násobek strategií s_1^t a s_2^t vůči danému ohodnocení stavu, tedy jednoltlivým přechodům dává pravděpodobnost, že se uskuteční. Procedura $maxmin(A_i(v^{t-1}))$ přiřazuje Hráči 1 jeho maximalizující strategii vypočtenou z lineárnícho programu.

3.3.1 Příklad iterace strategie

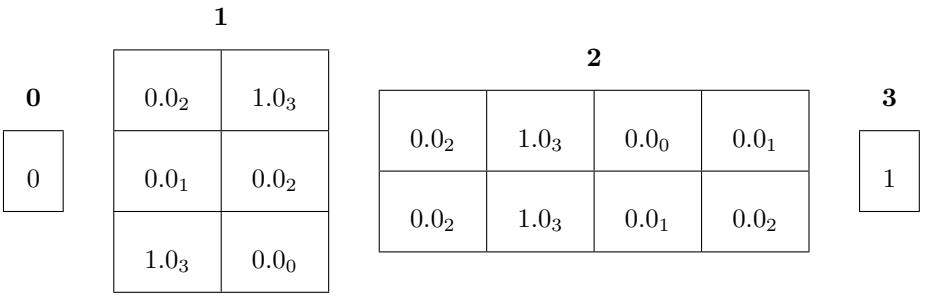
Inicializujeme si vektor v^0 stejně jako u iterace strategie. Akce, které vedou rovnou k terminálním stavům, dostanou příslušnou utilitu. Dále je potřeba nastavit si profily strategií jednotlivých hráčů. Hráči 1 nastavíme libovolnou počáteční strategii a Hráče 2 necháme spočítat hodnotu *nejlepší reakce* na strategii řádkového hráče. Hráč 1 zde začíná s ryzí strategií pro první akci. Pro srovnání s hodnotovou iterací volíme $\varepsilon = 0.01$.

Stav 1: s_1^1 = *[1.0, 0.0, 0.0]*
Stav 1: s_2^1 = *[1.0, 0.0]*
Stav 2: s_1^1 = *[1.0, 0.0]*
Stav 2: s_2^1 = *[1.0, 0.0, 0.0, 0.0]*



Obrázek 3.7: Algoritmus iterace strategie, t = 1

Stav 1: s_1^2 = *[1.0, 0.0, 0.0]*
Stav 1: s_2^2 = *[1.0, 0.0]*
Stav 2: s_1^2 = *[1.0, 0.0]*
Stav 2: s_2^2 = *[1.0, 0.0, 0.0, 0.0]*



Obrázek 3.8: Algoritmus iterace strategie, t = 2

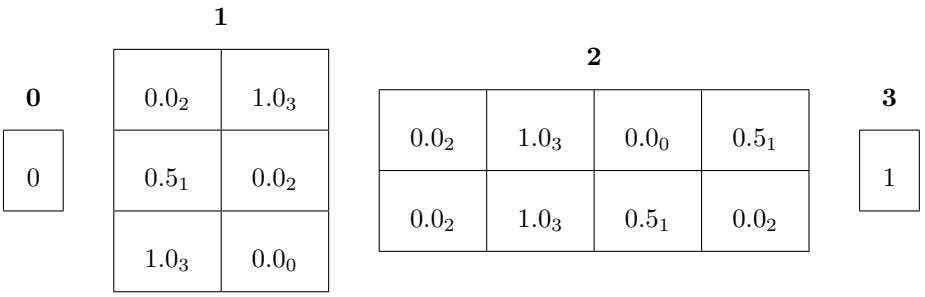
Algoritmus poté podle takto daných strategií přepočítá hodnoty pro všechny akce ve stavech hry a udělá z nich sumu $\mu_i(s_1^t, s_2^t)$. Tu pak porovnává s hodnotou hry vypočtenou lineárním programem. Pokud je hodnota hry vyšší než $\mu_i(s_1^t, s_2^t)$, nahradí se stávající strategie Hráče 1 strategií z lineárního programu.

Stav 1: s_1^3 = *[0.5, 0.0, 0.5]*
Stav 1: s_2^3 = *[1.0, 0.0]*
Stav 2: s_1^3 = *[1.0, 0.0]*
Stav 2: s_2^3 = *[1.0, 0.0, 0.0, 0.0]*

Vidíme, že strategie Hráče 1 se již přizpůsobila novému ohodnocení akcí prvního stavu. To je dáno tím, že hodnota funkce:

$$\mu_i(s_1^2, s_2^3) = \begin{bmatrix}1.0 & 0.0 & 0.0\end{bmatrix} \cdot \begin{bmatrix}0.0 & 1.0\\ 0.0 & 0.0\\ 1.0 & 0.0\end{bmatrix} \cdot \begin{bmatrix}1.0\\ 0.0\end{bmatrix} = 0$$

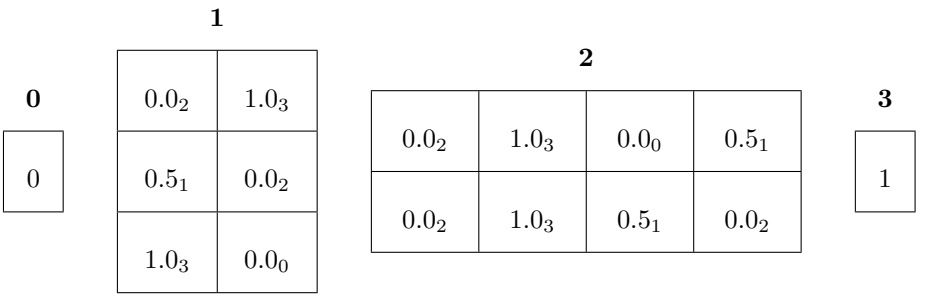
Ovšem hodnota hry prvního stavu je 0.5, což je lepší hodnota než 0. Vezmeme strategii z lineárního programu a dostaneme *[0.5, 0.0, 0.5]*, jakožto novou startegii s_1^3 pro Hráče 1.



Obrázek 3.9: Algoritmus iterace strategie, t = 3

A stejně tak na změnu strategie Hráče 2 zareagoval Hráč 1. Již teď vidíme, že následující iterace bude poslední, protože se hodnoty matic shodují s ohodnoceními poslední iterace hodnotového algoritmu.

Stav 1: s_1^4 = *[0.5, 0.0, 0.5]*
Stav 1: s_2^4 = *[1.0, 0.0]*
Stav 2: s_1^4 = *[1.0, 0.0]*
Stav 2: s_2^4 = *[1.0, 0.0, 0.0, 0.0]*



Obrázek 3.10: Algoritmus iterace strategie, t = 4

Hodnota hry je po čtyřech iteracích $U_1^* = 0.5$, tedy zkonvergovala ke stejné hodnotě hry jako u hodnotové iterace.

3.4 Double Oracle

Do třetice si zavedeme Double Oracle algoritmus, jehož myšlenky a principy dále využijeme pro řešení problému rozšiřování matic stavů [2] [10].
Algoritmus pracuje na myšlence postupného přidávání akcí, které mohou ovlivnit hru. Vezmeme stav, který má ohodnocené kombinace akcí nějakou utilitní funkcí. Začínáme s malou podmnožinou akcí. Na ni vypočítáme hodnoty Nashova equlibria, tj. jejich *optimální strategie*. Pak každému hráči na tuto startegii necháme provést *nejlepší reakci* ve formě *ryzí strategie*. Akce, které hráči zvolili přidáme do množiny a tento proces opakujeme, dokud už nelze do množiny žádné akce přidávat. Další akce se nepřidají, jelikož rozšíření množiny akcí jednoho hráče nezmění optimální strategii druhého hráče, tedy tato akce nijak neovlivňuje hodnotu Nashova equlibria. Tímto způsobem se zbavíme akcí, o kterých víme, že hodnotu hry nezmění, tedy žádný z hráčů by ji nikdy při současném stavu hry nehrál.

Data: Část hry tj. její podhra := subGame
Result: Maximální podhra taková, že v ní hráči mají své akce postačující k optimální strategii

```
1 subGame := matice základní podhry ze vstupu;
2 NE := množina strategií pro všechny hráče na omezené hře -¿ vypočteno
  z lineárního programu;
3 while true do
4 |   if bestResponse(NE) ∈ subGame then
5 |   |   break ;          // pokud už podhra obsahuje všechny akce
  |   |   postačující k optimální strategii
6 |   update(subGame) ;       // přidání nových řádků a sloupců do
  |   herní matice podle nově spočítané optimální strategie
```

Algorithm 3: Double oracle algoritmus

3.4.1 Příklad Double Oracle algoritmu

Pro výpočet příkladu si vypůjčíme Stav 1 ze třetí iterace hodnoty z **Obrázku 3.6**. Jako první provedeme inicializaci algoritmu, tedy zvolíme podmnožinu akcí. Abychom se pokusili co nejvíce stav ořezat, necháme oběma hráčům pouze jednu a to první akci v prvním řádku a sloupci. Tato hra má triviální výsledek *optimálních strategií*.

Iterace 1

Optimální strategie Hráče 1 $s_1 = [1]$
Optimální strategie Hráče 2 $s_2 = [1]$

Nejlepší reakce Hráče 2 na Hráče 1 $br_1 = [1, 0]$
Nejlepší reakce Hráče 1 na Hráče 2 $br_2 = [0, 0, 1]$

Hráči 2 zcela vyhovuje výběr utility rovné nule a jeho *nejlepší reakce* se tím nemění. Hráč 1 však raději zvolí svou třetí akci s utilitou rovnou jedné, aby tím zvýšil hodnotu hry.

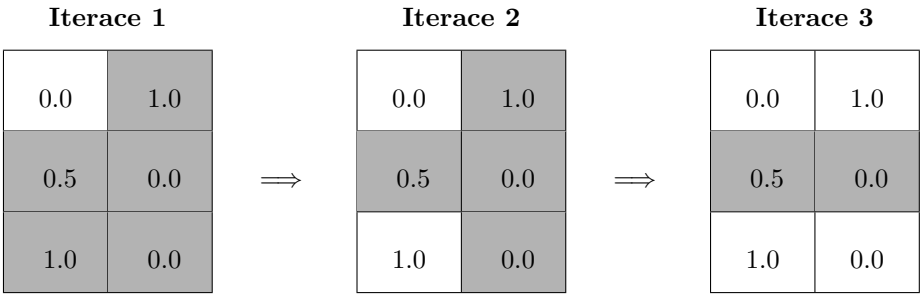
Iterace 2

Optimální strategie Hráče 1 $s_1 = [1]$
Optimální strategie Hráče 2 $s_2 = [0, 1]$
Nejlepší reakce Hráče 2 na Hráče 1 $br_1 = [0, 1]$
Nejlepší reakce Hráče 1 na Hráče 2 $br_2 = [0, 0, 1]$

Iterace 3

Optimální strategie Hráče 1 $s_1 = [0.5, 0.5]$
Optimální strategie Hráče 2 $s_2 = [0.5, 0.5]$
Nejlepší reakce Hráče 2 na Hráče 1 $br_1 = [1, 0]$
Nejlepší reakce Hráče 1 na Hráče 2 $br_2 = [1, 0, 0]$

Tento selektivní postup nám na tomto jednoduchém stavu demonstroval to, co bylo zřejmé již v iteraci hodnoty. Druhá akce řádkového hráče matice nese přebytečnou informaci o Stavu 1, tedy pro naše účely není potřebná k nalezení optimální strategie.

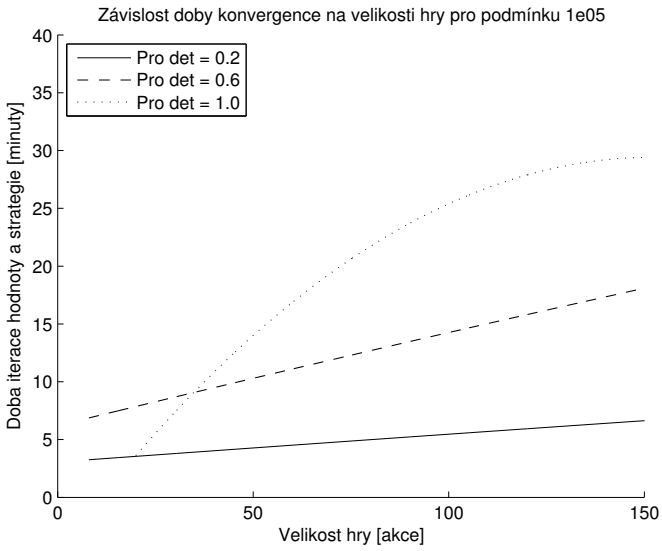


Obrázek 3.11: Algoritmus Double Oracle na Stavu 1 z Obrázku 3.8

Kapitola 4

Iterační algoritmy ve stochastických hrách v nekonečně krocích

Předpokládejme, že máme algoritmy optimálně naprogramované a k výpočtům používáme výkonný počítač. Jediný parametr, se kterým můžeme hýbat, je ukončovací podmínka. U té však požadujeme, co nejvyšší přesnost, tedy co nejmenší rozdíl ve zkonvergovaných hodnotách hry nebo startegie dvou po sobě jdoucích iteracích.



Obrázek 4.1: Graf závislosti velikosti hry na čase konvergence obou iteračích algoritmů při fixní přesnosti $1 \cdot 10^{-5}$

4.1 Analýza problému

Jedním ze způsobů vedoucích ke zrychlení řešení stochastických nekonečných her je *ořezání* počtu stavů a akcí tak, abychom neztratili důležitou informaci o hře. V podstatě chceme využít principu Double Oracle algoritmu na všechny herní stavy. Jenže v příkladu ze sekce 3.4.1 jsme předpokládali znalost ohodnocení akcí. Ve skutečnosti tyto zkonvergované hodnoty známe až po dokončení běhu iterativních algoritmů. Rádi bychom tedy od nějaké malé podmnožny akcí a stavů rozšiřovali hru tak, aby přidané akce (případné přechody na nové stavy) měly smysl. Předpokládáme, že malá část hry (*podhra*) zkonverguje k hodnotám hry velmi rychle. Pokud do množiny akcí podhry přidáme akci a hodnota hry se nezmění, získáme tím informaci o tom, že tato akce nemění hodnotu optimální strategie obou hráčů. Naopak je žádoucí přidat všechny stavy, které hodnotu hry *vylepší*. Takové akce mají pro hru význam a vylepšují startegie hráčů.

4.2 Algoritmus rozšiřování podhry

Jak jsme již zmínili v úvodu kapitoly, je třeba nejdříve sestavit nějakou malou podhru, která má smysl. To jest taková množina stavů a akcí, že se hráči dokáží dostat od počátečního stavu ke stavu terminálnímu. Toto je problém hledání cesty v grafu, který vyřešíme procedurou *init()*. Pokud taková cesta neexistuje, pak hru nemá smysl osekávat, protože neobsahuje cesty k terminálním stavům. Tedy pouze cyklí mezi přechodnými stavy. Pro splnění podmínky stačí nalézt alespoň jednu cestu k terminálnímu stavu ze všech možných ve hře.
Dále budeme potřebovat frontu akcí Q. Pro tu platí, že neobsahuje množinu akcí A^* (definujeme ji jako akce, které už byly do podhry přidány). Pro *subGame* tedy platí $A^* \notin$ Q. Tím říkáme jen, že přidáváme stavy, které ještě podhra neobsahuje. Ve chvíli, kdy je přidán stav, provedeme kontrolu, zda se změnila hodnota iteračního algoritmu. Pokud ano, aktualizujeme funkcí *dosažitelnéStavy()* přechod do nově dostupných stavů. Pomocí této funkce ověříme, jestli existuje z nějaké nově přidané akce cesta do stavů, které nejsou v podhře. V případě rozšíření o nový stav se přidává první akce pro oba dva hráče. Jestliže všechny přidané akce nezmění hodnotu hry, algoritmus ukončujeme.

Data: G = Hra, e = Ukončovací podmínka pro iterační algoritmy
Result: Podhra vstupu se stejnou hodnotou hry

```
1  if init(G) existuje? then
2  |  subGame = init(G);
3  else
4  |  return "Podhra nemá řešení";
5  Nechť Q je fronta;
6  Nechť S jsou stavy, které podhra obsahuje;
7  gameValue^0 = iterativníAlgoritmus(subGame, e);
8  t := 1;
9  sameQ := true;
10 while true do
11 |  if Q.vyjmi je prázdná? then
   |  |  // jestliže fronta Q neobsahuje žádné další akce ke
   |  |     zpracování
12 |  |  Q.přidej(akce ∉ subGame);
13 |  |  if existuje dostupný stav X z množiny stavů S then
14 |  |  |  Q.přidej(akce ∈ X);
15 |  |  if sameQ == false then
16 |  |  |  break ;        // tzn. pokud se žádná z akcí nepřidala
17 |  |  else
18 |  |  |  sameQ = false
19 |  a := Q.vyjmi;
20 |  subGame.přidej(a);
21 |  if iterativníAlgoritmus(subGame, e) = gameValue^t then
22 |  |  subGame.vyjmi(a);
23 |  else
24 |  |  S.přidej(dosažitelnéStavy(subGame));
25 |  |  sameQ = true;
26 |  t := t+1;
27 |  gameValue^t := iterativníAlgoritmus(subGame, e);
```

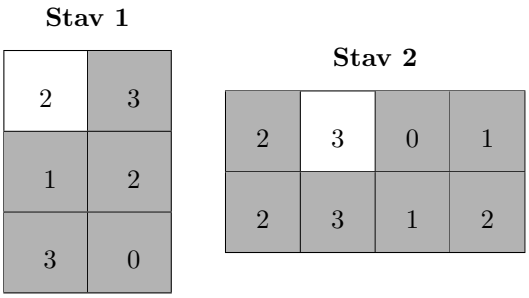
Algorithm 4: Rozšiřování podhry

V Double Oracle jsme využívali nejlepší reakci hráčů, abychom určili, které akce do herních stavů přidávat. Tato odpověď donutila hráče změnit svojí strategii v případě, že existovala lepší varianta. Jestliže jeden z hráčů změnil svojí strategii, znamená to změnu hodnoty hry v daném stavu. Tuto vlastnost využíváme právě v algoritmu rozšiřování podher.

4.2.1 Příklad na deterministické hře

Pro demonstraci algoritmu využijeme náš známý příklad z třetí kapitoly. Nejprve si inicializujeme nějakou cestu z počátečního do terminálního stavu. Hodnoty v maticích odpovídají stavu, do kterého se z daného pole dostaneme.

Nechť stav 0 a 3 je terminální a přechody matic jsou ohodnoceny pořadovým číslem stavu, pak dostáváme podhru o velikosti $\Omega = 2$:



Obrázek 4.2: Incializace podhry hry z Obrázku 3.2

Je zřejmé, že takovýchto počátečních podher můžeme najít více. Například druhá akce sloupcového hráče spolu s třetí akcí řádkového hráče dají podhru jen o velikost $\Omega = 1$. Obecně platí, že čím menší matice stavů vezmeme na začátku, tím lépe.

Sestavíme si tedy frontu v pořadí, jak jdou akce postupně ve stavech po řádcích. Označíme si její prvky trojčíslím ve formátu:
[Pořadí stavu hry, Index hráče, Pořadí akce hráče].

$\mathbf{Q} = \{[112], [113], [121], [212], [221], [223], [224]\}$

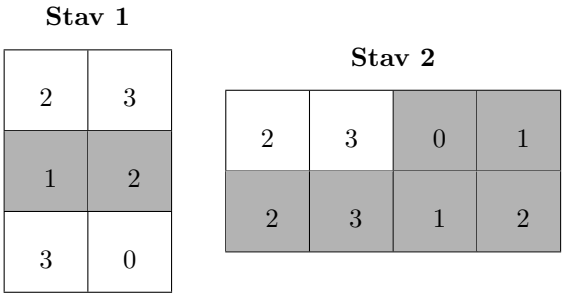
První čtyři akce hru neovlivní a nenutí žádného z hráčů změnit strategii. Až pátý prvek fronty (první akce Hráče 2 ve druhém stavu) umožní Hráči 2 hrát ryzí strategii $s_2^2 = [1, 0]$, čímž změní hodnotu hry z 1 na 0. Šestý a sedmý prvek strategie nemění.
Naplníme tedy znovu frontu:

$\mathbf{Q} = \{[112], [113], [121], [212], [223], [224]\}$

Hráči 1 se přidá třetí akce v prvním stavu, hodnota hry je pak 1.

$\mathbf{Q} = \{[112], [121], [212], [223], [224]\}$

Zde se přidá druhá akce Hráče 2 ze stavu dva. Algoritmus vyzkouší zbylé prvky fronty. Poté ji znovu naplní a zkontroluje, že již žádná akce startegie hráčů nemění a program se ukončí.



Obrázek 4.3: Výsledná rozšířená podhra z Obrázku 4.2

Vidíme, že hra se rozšířila přesně podle myšlenky Double Oracle algoritmu. Stav 1 odpovídá řešení příkladu z **Obrázku 3.11** a o Stavu 2 již z Příkladu 2 ze sekce 3.1.1 víme, že Hráč 2 bude pokaždé hrát ryzí strategii. Tedy pouze svou první akci a tím pádem se dá celá matice zkrátit pouze na první přechod.
Tímto jsme ale i zjistili, že by se volbou právě výše zmiňovaného přechodu na terminální stav 0 podhra zmenšila z $\Omega = 6$ na $\Omega = 5$. K lepšímu vybírání počáteční podhry by tedy byla vhodná heurestika, která by byla schopná efektivně odhadnout cestu k terminálnímu stavu, jenž má nejvyšší pravděpodobnost účasti ve finální rozšířené podhře.

Kapitola 5

Popis experimentů s výsledky

Testy proběhly na mém vlastním naprogramovaném prostředí v programovacím jazyce Java s využitím knihovny CPLEX, se kterou jsem počítal lineární program pro zjištění Nashova equlibria a hodnoty hry. Ke generování testovacích dat jsem využil svého generátoru náhodných her, na němž byly vytvořeny sady dat pro různé stupně stochasticity her.

5.1 Generování náhodných her

Pro testování na algoritmu rozšiřování podhry bylo potřeba vytvořit generátor her, abychom mohli provést měření na signifikantní a rozlišné množině her. Hry generujeme, protože neexistuje žádná knihovna her nebo tomu podobný generátor.

Definice 7 *Nechť h_i^p je počet akcí pro hráče p ve stavu i, pak Ω je velikost hry a definujeme ji následovně:*

$$\Omega = \sum_{i=1}^{l} h_i^1 \cdot h_i^2$$

Definice 8 *Zavedeme si parametr det, který nám bude určovat míru determinističnosti, resp. stochasticity přechodů mezi stavy herní matice. Parametr det nabývá hodnot $z < 0.0, 1.0 >$, kde $\boldsymbol{det} \cdot \sum_{i=1}^{l} h_i^1 + h_i^2$ je maximální počet stavů, do kterých se můžeme z daného přechodu dostat.*

Na **Obrázku 4.1** jsme viděli, že čas k vyřešení programu roste i s tím, jak je hra *stochastická*.

Zadefinujme si *gen(n, h, l, seed, det)* jako funkci, která vrací hru jako pole stavů, pak platí, že:

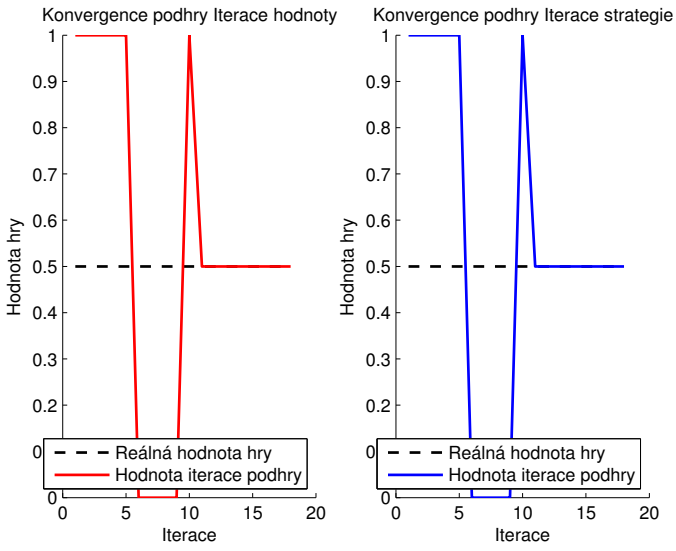
- n je počet požadovaných nagenerovaných náhodných her
- h je maximální velikost hrany matice stavu, tj. maximální počet akcí pro hráče u jednoho stavu hry
- l je maximální počet stavů hry
- *seed* je unikátní klíč ke každé nagenerované hře
- *det* je parametr stochasticity přechodů mezi stavy hry

Generátor tedy přijme na vstupu parametry hry a generuje z nich hry takové, že vstup nám určuje maximální hodnoty daných specifikací hry jako je třeba největší možná šířka stavu apod.
Nejdříve se vygenerují velikosti stavů, poté se postupně vytváří přechody pro různé kombinace akcí a podle parametru *det*, jak již bylo popsáno v jeho definici, se pro každý takový přechod nastaví pravděpodobnostní rozložení. Pravděpodobnost, že dvěma akcím bude přidělen přechod na jiný stav, je dána *rovnoměrným rozdělením*. To nám maximalizuje šanci, že bude existovat do každého stavu alespoň jeden přechod a že do větších matic bude takových přechodů více.

5.2 Měření na konkrétních příkladech

Zajímá nás, jak vypadá průběh algoritmu, co se týče hodnoty hry. Jak se mění v čase a zároveň, jak ořezává hry na nějakém větším vzorku dat.

5.2.1 Příklad na deterministické hře



Obrázek 5.1: Konvergence algoritmu rozšiřování podher na příkladě ze sekce 4.2.1

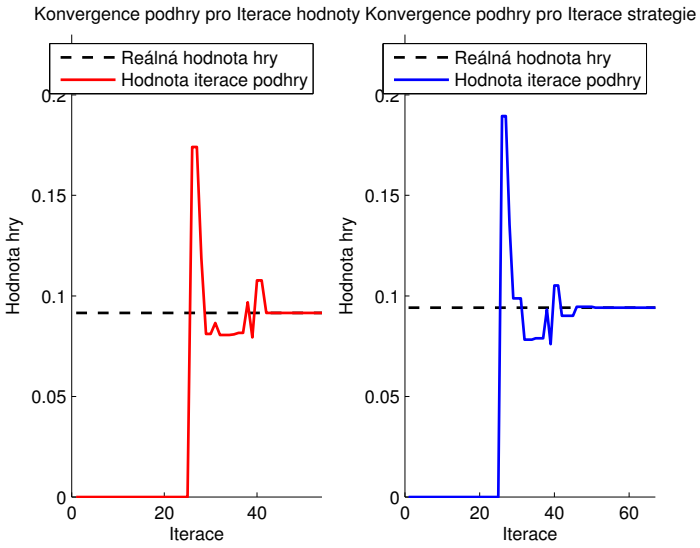
Na **Obrázku 5.1** můžeme vidět, jak se mění hodnota hry po jednotlivých iteracích. Skoky znamenají přidání stavu, který tuto hodnotu ovlivňuje. V tomto konkrétním případě to bude právě podmatice ze Stavu 1, která neprve přilepší Hráči 2, poté se vrátí ve prospěch Hráče 1, až se doplní na čtvercovou "jednotkovou"podmatici s hodnotou hry *0.5*. V místech grafů, kde je hodnota konstantní, probíhalo přidávání akcí, které hodnotu Nashova equlibria nemění. I na délce jednotlivých rovných úseků můžeme pozorovat počet nepřidaných dostupných akcí. Pro oba iterační algoritmy probíhalo přidávání akcí stejně.

Algoritmis tak dokázal hru zjednodušit zhruba o 43% její původní velikosti.

5.2.2 Příklad na stochastické hře

Uvádíme další příklad, tektokrát na stochastické nekonečné hře. Hra byla vytvořena generátorem jako *gen(1, 4, l0, 218, 0.25)*. Pro složitost hry uvádíme pouze její rozměry. Díky tomu, že je hra stochastická, může obsahovat cykly, které bychom rádi zjednodušili (v tomto případě tomu tak skutečně je).

Vygenerovaná hra	**Zjednodušená hra**
Stav 1 = 4x3	Stav 1 = 4x1
Stav 2 = 2x2	Stav 2 = 0x0
Stav 3 = 2x3	Stav 3 = 2x2
Stav 4 = 2x2	Stav 4 = 2x2
Stav 5 = 2x3	Stav 5 = 2x2
Stav 6 = 3x3	Stav 6 = 0x0
Stav 7 = 2x2	Stav 7 = 1x1
Stav 8 = 3x4	Stav 8 = 3x3
Stav 9 = 2x2	Stav 9 = 2x2
Stav 10 = 2x2	Stav 10 = 1x1

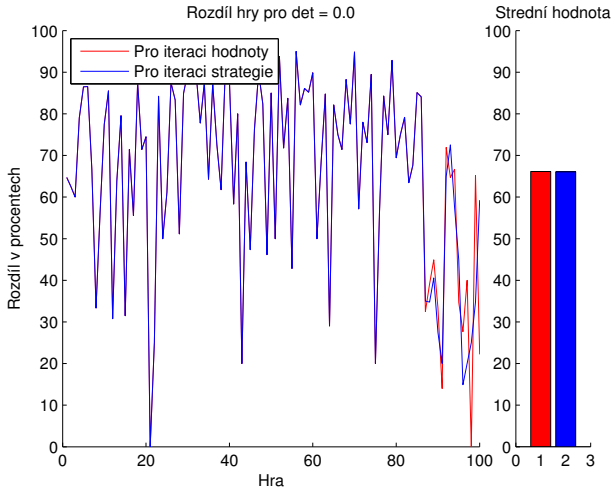


Obrázek 5.2: Konvergence algoritmu rozšiřování podher na vygenerovanén příkladě nekonečné stochastické hry

Vidíme, že ve hře jsme objevili dokonce stavy, které jsou nedosažitelné *(0x0)*. Pak jsou zde stavy, jež dosažitelné jsou, ale vytvářejí pouze přechodnou cestu *(1x1)*. V neposlední řadě jsou pak zjednodušeny stavy, které neovlivňují rozhodování hráčů.
Tímto jsme hru zmenšili zhruba o polovinu (rozdíl průběhu zjednodušení pro oba iterační algoritmy vysvětlíme později) a jak vidíme na rozměrech, výrazně jsme usnadnili výpočet strategie hráčů.

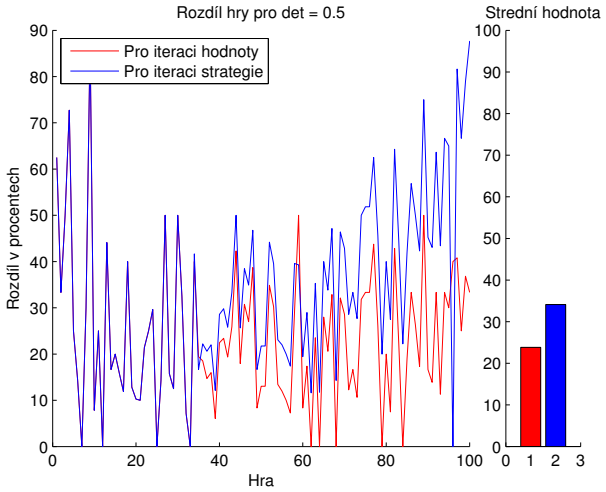
5.3 Měření na větší sadě dat

Níže uvedené výsledky testování proběhly na hrách do velikosti 5×5, do počtu stavů 5, za ukončovací podmínky $\varepsilon = 0.001$ a hodnoty klíče 160. Jelikož k měřením používáme náhodný generátor, můžeme předpokládat, že rozměrnější matice s větším počtem stavů budou mít podobné náhodné rozložení přechodů a tedy bude hra podobnou měrou obsahovat cykly.
U některých z níže uvedených grafů nabývá občas hodnota procentuálního zmenšení hry velikosti nula. To znamená, že tyto hry nebyly vůbec nijak zjednodušeny. Jsou buď dílem náhody nagenerovány tak malé, že nemohou být zmenšeny (zpravidla jsou to hry o jednom stavu), anebo hry tak dobře vytvořené, že po permutaci řádků a sloupců u všech stavů zjistíme, že na diagonále matic jsou přechody do terminálních stavů.
Níže uvedené grafy nenesou žádnou závislot os na sobě. Osa x udává rozdílná na sobě nezávislá měření a grafy jsou zde pro porovnání. Udávají, jak se mění zmenšení her v procentech v závislosti na paramentru *det* při statistickém vzorku 100 her. Ty byly na ose x seřazeny podle absolutního rozdílů zmenšení velikosti stavových matic obou iteračních algoritmů. Na základě toho můžeme demonstrovat, jak moc se výsledky ořezání liší i mezi hodnotovým a strategickým algoritmem.



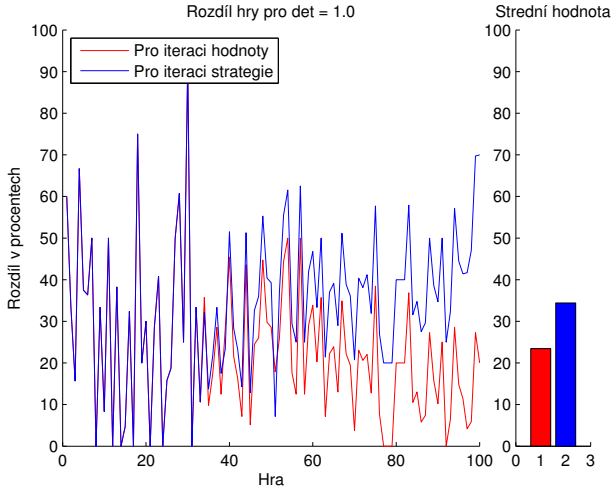
Obrázek 5.3: Graf porovnávající procentuální zmenšení herních matic na vzorku náhodných her při det = 0.0

Na grafu pro čistě deterministickou hru vidíme, že půměrného zlepšení (tedy zmenšení velikosti původní hry) můžeme dosáhnout u obou iteračních algoritmů zhruba stejně. Zhruba o $66.1 \pm 2.2\%$ a pro oba algoritmy stejně (liší se jen o desetinu procenta a odchylka o setinu).

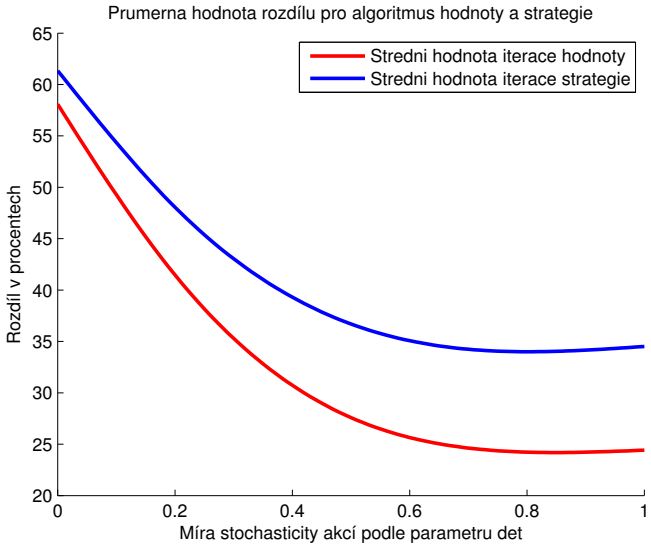


Obrázek 5.4: Graf porovnávající procentuální zmenšení herních matic na vzorku náhodných her při det = 0.5

U her při *det = 0.5* bylo na stejném vzorku průměrné zlepšení u iterace hodnoty o $23.8 \pm 1.6\%$ a u strategie $34.1 \pm 2.0\%$. Při *det = 1.0* bylo průměrné zlepšení u iterace hodnoty o $23.4 \pm 1.8\%$ a u strategie $34.4 \pm 1.8\%$.



Obrázek 5.5: Graf porovnávající procentuální zmenšení herních matic na vzorku náhodných her při det = 1.0



Obrázek 5.6: Graf závislosti průměrného zmenšení hry na parametru det

Můžeme si všimnout, že u iterace strategie klesá procentní zmenšení her v závislosti na parametru det pomaleji, než je to u iterace hodnoty. Po důkladném prozkoumání vkládání akcí a běhu algoritmu jsem zjitil, že je to dáno ztrátou informace díky nedostatečné přesnosti datového typu *double*. Stane se pak to, že algoritmus rozšíří stav o další akce, i když jde o chybu v nepřesnosti typu double. Toto se děje především ve hrách se stavem(vy), obsahujícím takovou kombinaci přechodů na terminální stav nějakého hráče, že se hodnota hry tohoto stavu blíží buďto 0 nebo 1. Jelikož je tento stav zacyklený sám do sebe, algoritmu pak stačí přidat odkaz na jiný stav s malou hodnotou utility a vzniká tím chyba zaokrouhlení. Toto může i stavy, ve výjimečných případech - kdy hra splňuje výše uvedené předpoklady, rozšířit na jejich plnou velikost.
Zvažoval jsem použití přesnějších typů jako např. *bigdecimal*, ale jelikož jde o objekty a bylo by potřeba v daném typu uchovávat hodnotu hry a strategie, tedy nejpoužívanější proměnné v algoritmech, nakonec jsem toto řešení zamítl.

Kapitola 6

Závěr

Předmětem této práce bylo prozkoumat, zda jde hru zjednodušit, abychom uspíšili konvergenci iteračních algoritmů. Zjistil jsem, že postupné rozšiřování podhry skutečně dokáže akce her podstatně ořezat.
Jenže s rostoucím počtem akcí se může stát, že hodnota hry už dávno konvergovala k optimu, ale algoritmus ještě stále zkouší přidávat akce. Stejně tak se může stát, že při opakovaném přidávání akcí do fronty, ze které postupně vylepšujeme podhru, se nám stále vrací akce, jež hodnotu hry nikdy nezmění.
V práci jsem se tímto problémem nezabýval, ale způsob stavění fronty může být další faktor k urychlení algoritmu k rozšiřování podhry. Například opakovaně přidávané akce, které po několik iterací stále nemění hodnotu hry, je možné oklasifikovat nižší prioritou než akce neprozkoumané. Tedy najít vhodnou heuristiku, jak vybírat, kterými akcemi hru rozšiřovat.
K úspoře času nám reálně ale algoritmus nepomohl. Je tu ovšem potenciál pro zrychlení algoritmu rozšiřování podhry a to nejen v optimalizaci výše uvedeného pseudoalgoritmu, ale především v lepší volbě rychlejšího programovacího jazyka. To by pomohlo i s dalším problémem, který se vyskytoval v jazyce Java. Nedostatečná přesnost datových typů a objektový model jazyka byl, co se týče rychlosti běhu algoritmu, spíše na škodu.
Stochastické nekonečné hry mají v současné době široké využití a je proto žádoucí při hledání optimálních strategií tento proces uspíšit. Vhodné osekávání akcí hráčů se zdá být jako jedno z možných vylepšení.

Literatura

[1] Kristoffer Arnsfelt Hansen, Rasmus IbsenJensen, and Peter Bro Miltersen: *The complexity of solving reachability games using value and strategy iteration*. Computer Science-Theory and Applications, Springer Berlin Heidelberg, 2011. 7790.

[2] Branislav Bošanský, Viliam Lisý, Jiří Čermák, Roman Vítek, a Michal Pěchouček: *Using Doubleoracle Method and Serialized AlphaBeta Search for Pruning in Simultaneous Moves Games*. In Proceedings of the 23rd International Joint Conference on Artificial Inteligence (IJCAI), 2013.

[3] Yoav Shoham, and Kevin LeytonBrown: *Multiagent systems: Algorithmic, gametheoretic, and logical foundations*. Cambridge University Press, 2009. `http://www.masfoundations.org/mas.pdf`

[4] Antonín Kučera: *Stochastické hry dvou hráčů*. Druhé setkání Pražského informatického semináře, 2014. `http://www.praguecomputerscience.cz/index.php?l=cz&p=2`

[5] Everett, H: *Recursive games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games Vol. III. Annals of Mathematical Studies, vol. 39*, Princeton University Press, Princeton, 1957.

[6] Shapley, L.S.: *Stochastic games*. Proceedings of the National Academy of Sciences, U.S.A. 39, 1953.

[7] Howard, R.: *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[8] Condon, A.: *On algorithms for simple stochastic games. In: Advances in Computational Complexity Theory DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13*. 1993.

[9] Nash, J.: *Equilibrium points in n-person games*, Proceedings of the National Academy of Sciences USA, 36, 1950. Reprinted in H. Kuhn (Ed.), Classics in Game Theory, Princeton, NJ: Princeton University Press, 1997 .

[10] H. B. McMahan, G. J. Gordon, and A. Blum: *Planning in the Presence of Cost Functions Controlled by an Adversary. In ICML, pages 536–543*, 2003.

[11] S. Ganzfried and T. Sandholm: *Computing an approximate jam/fold equilibrium for 3-agent no-limit Texas hold'em tournaments*, AAMAS, 2008.