$3SG$ algoritmus a učenie sa akčných modelov v reálnom čase

Michal Čertický

Katedra Aplikovanej Informatiky, Univerzita Komenského
FMFI, Mlynská dolina, 842 48, Bratislava
Email: certicky@fmph.uniba.sk

Abstrakt

Akčný model, ako logická reprezentácia efektov a podmienok vykonateľnosti jednotlivých akcií, je nevyhnutnou podmienkou pre plánovanie. Manuálne definovanie takýchto modelov je však hlavne v prípade komplexnejších domén časovo náročné, a často vedie k chybám. Alternatívnym prístupom je nechať agentov, aby sa zo série pozorovaní sami naučili kedy sú ktoré akcie vykonateľné a čo spôsobujú. V tomto článku načrtneme návrh nového algoritmu na učenie sa akčných modelov, nazvaného 3SG (Simultaneous Specification, Simplification and Generalization) a priblížime niektoré jeho vlastnosti a výsledky prvých experimentov. Na rozdiel od existujúcich alternatív, 3SG súčasne produkuje pravdepodobnostné akčné modely s podmienenými efektami, a vyrovná sa so zlyhaním akcií, senzorickým šumom, a neúplnými pozorovaniami. Hlavným rozdielom oproti väčšine iných prístupov však je, že 3SG je online algoritmus, čo znamená že dostáva na vstupe vždy len jedno (najaktuálnejšie) pozorovanie. To na jednej strane znamená, že algoritmus je rýchly (polynomiálny vzhľadom na veľkosť vstupu) a teda použiteľný v reálnom čase aj v komplexných doménach. Na druhej strane ale nie je možné zaručiť presnosť indukovaných modelov. Ako doménu pre experimenty používame akčnú hru Unreal Tournament 2004, pričom vnímame rádovo stovky fluentov každú sekundu.

1 Úvod

Vedomosti o kauzálnych závislostiach, resp. efektoch a podmienkach jednotlivých akcií, sú prerekvizitou pre plánovanie a následné inteligentné správanie agentov. Takéto znalosti sa zvyčajne označujú pojmom *„akčný model“*. Akčné modely v umelých systémoch sú väčšinou ručne vytvorené znalcami danej domény, alebo programátormi. Ich podrobná špecifikácia je však v prípade komplexnejších domén náročná a zdĺhavá úloha. Okrem toho je občas potrebná ich aktualizácia, ktorá si vyžaduje komplikované revízie znalostí a často vedie k chybám. Automatické učenie sa akčných modelov sa preto v posledných rokoch stalo zaujímavou oblasťou výskumu a bolo navrhnutých niekoľko metód, využívajúcich širokú paletu postupov z umelej inteligencie.

Formálne budeme akčný model chápať ako dvojicu $\langle D, P\rangle$ kde D je symbolický[1] opis dynamiky domény v akomkoľvek jazyku (súčasné prístupy sa líšia vo výbere reprezentačnej štruktúry, resp. formalizmu) a P je pravdepodobnostná funkcia definovaná na prvkoch D. Učenie sa akcií sa potom dá definovať ako algoritmický proces, ktorý na základe pozorovaní spresňuje D, pričom môže využívať a modifikovať P.

Po tom, čo si v sekcii 2 stručne zhrnieme aktuálne metódy, predstavíme náš návrh – online algoritmus $3SG$. V sekcii 4 sa budeme venovať niektorým jeho vlastnostiam (tými dôležitými je korektnosť a zložitosť, ale formulujeme aj niekoľko menej podstatných poznatkov, ktoré však ilustrujú fungovanie algoritmu). Nakoniec v sekcii č. 5 popíšeme výsledky našich experimentov s využitím algoritmu $3SG$ na učenie sa v reálnom čase, pričom ako doménu použijeme akčnú FPS[2] hru Unreal Tournament 2004.

2 Súčasné riešenia

Spomedzi súčasných techník by sme ako prvý mali spomenúť algoritmus $SLAF$ [1], ktorý je ideovo najbližšie nášmu riešeniu. $SLAF$ postupne konštruuje výrokovologickú formulu a neskôr ju interpretuje pomocou SAT solvera. Využitím SAT solverov (konkrétne váhovaného MaxSAT solvera) je mu podobný systém $ARMS$ [10], ktorý sa však neučí na základe pozorovania stavov sveta. Akčné modely sa totiž učí spätne z úspešných plánov. Príkladom kompletne deklaratívnych metód sú dva vzájomne podobné prístupy založené na logickom programovaní ASP [2] a jeho reaktívnej verzii [5]. Ako zástupcov techník vzdialenejších od logiky spomenieme učenie sa cez perceptrónový algoritmus [9], a viacúrovňové greedy prehľadávanie [11].

Výhodou $3SG$ oproti spomínaným technikám je nie len to, že súčasne produkuje *pravdepodobnostné* akčné modely s *podmienenými efektami* a vyrovná sa so *zlyhaním akcií*, senzorickým *šumom*, a *neúplnými pozorovaniami*, ale aj to, že ide o online algoritmus, vďaka čomu je dostatočne rýchly (to však ale znamená, že nie je zaručená presnosť jeho výsledkov).

[1] Poznatky potrebujeme mať reprezentované symbolicky, kvôli plánovaniu.

[2] FPS znamená *„First Person Shooter“*.

3 Algoritmus $3SG$

Podstatnou súčasťou nášho riešenia je návrh vhodnej **štruktúry na reprezentáciu** akčných modelov. Tá musí byť čo najkompaktnejšia, aby bolo možné učenie v reálnom čase pri veľkom množstve pozorovaní. Označme si najskôr množinu všetkých možných stavov sveta ako S a všetkých možných akcií ako A. Klasickým spôsobom reprezentácie, ktorý je použitý napríklad v [1], a podobá sa aj na prístup z [6] je tzv. *tranzičná relácia* $TR \subseteq S \times A \times S$. Tá vyjadruje vzťah medzi dvoma kompletnými popismi stavu sveta a akciou, a je najrobustnejšia vzhľadom na vyjadrovaciu silu, ale aj na priestorovú náročnosť. Trochu kompaktnejšou štruktúrou je jej modifikácia, ktorú voláme *efektová relácia* $ER \subseteq S \times A \times F$. Jej priestorová náročnosť je nižšia, pretože explicitne zachytáva iba vzťah medzi jedným kompletným popisom stavu sveta, akciou, a jedným fluentom (Fluentom nazývame ktorýkoľvek literál, ktorým vyjadrujeme takú vlastnosť našej domény, ktorá sa môže zmeniť pôsobením akcií alebo času. V tomto článku označujeme fluent opačný k fluentu f znakom $\overline{f}$ a množinu všetkých fluentov ako F.). Reprezentačná štruktúra s ktorou pracuje náš algoritmus $3SG$ ideovo vychádza z týchto dvoch, a volá sa **efektová formula**. Tento krát nejde o reláciu, ale o konečnú množinu[3] atómov dvoch druhov s rozdielnym významom:

1. a^f znamená že *"akcia* **a** *spôsobuje fluent* **f***"*
2. a^f_c znamená "aby **a** spôsobila **f**, musí platiť **c**" (resp. **c** je podmienkou pre a^f)

Efektovú formulu budeme označovať EF. Jej výhodou je kompaktnosť, ktorá ju robí vhodnou na použitie pre učenie sa v reálnom čase. Nevýhodou však je, že (na rozdiel napríklad od tranzičnej relácie) sa ňou nedajú reprezentovať disjunktívne podmienky efektov[4]. Akčné modely s ktorými budeme pracovať teda budú mať tvar $\langle EF, P\rangle$.

Vstupom pre $3SG$ algoritmus je trojica (o, a, o'), kde o' je najaktuálnejšie pozorovanie (neprázdna a konzistentná[5] množina pozorovaných fluentov), o je pozorovanie z predchádzajúceho časového kroku a a je vykonaná akcia[6]. Túto trojicu inak nazývame *príklad*. Každý takýto príklad môže (ale nemusí) byť považovaný za *pozitívny* alebo *negatívny* vzhľadom na niektorý z prvkov EF. Aktuálne počty pozitívnych a negatívnych príkladov budú pre nás dôležité, pretože na základe nich máme definovanú pravdepodobnosť P pre jednotlivé prvky EF:

$$\forall i \in EF :$$

$$P(i) = \begin{cases} 0 & \text{ak } pos(i) + neg(i) < minEx \\ \frac{pos(i)}{pos(i)+neg(i)} & \text{ak } pos(i) + neg(i) \geq minEx \end{cases}$$

kde $minEx$ je konštanta, ktorej význam objasníme o pár riadkov nižšie. Vyjadrenie počtu pozitívnych a negatívnych príkladov ($pos(i)$ a $neg(i)$) je založené na sémantike EF a teda bude odlišné pre obidva druhy prvkov. Ak si označíme množinu fluentov, ktoré v rámci príkladu zmenili hodnotu ako $\Delta(o, o')$ (teda $\Delta(o, o') = \{f | \overline{f} \in o \wedge f \in o'\})$, tak platí:

$$pos(a^f) = |(o, a, o') | f \in \Delta(o, o')|$$

$$neg(a^f) = \left|(o, a, o') | \overline{f} \in o'\right|$$

$$pos(a^f_c) = |(o, a, o') | c \in o \wedge f \in \Delta(o, o')|$$

$$neg(a^f_c) = |(o, a, o') | \overline{c} \in o \wedge f \in \Delta(o, o')|$$

Okrem toho hovoríme, že akčný model $\langle EF, P\rangle$ *pokrýva* príklad (o, a, o'), ak $\forall f \in \Delta(o, o') : a^f \in EF$. Podobne vravíme že akčný model je *v konflikte* s príkladom (o, a, o'), ak $\exists a^f \in EF : (\overline{f} \in o') \wedge (\forall a^f_c \in EF : c \in o)$. Ak je súčet pozitívnych a negatívnych príkladov pre nejaký prvok rovný k, tak vravíme že tento prvok je *podporený* k príkladmi. Hore spomínaná konštanta $minEx$ vyjadruje minimálny počet príkladov, ktorými potrebujeme mať podporený prvok, aby sme mu priradili nenulovú pravdepodobnosť ($minEx \in \mathbb{N}$).

Teraz, keď máme opísanú našu reprezentačnú štruktúru a vstupy, môžme prejsť k samotnému algoritmu (pseudokód je na obr. 1). Skratka $3SG$ znamená *"Simultaneous Specification, Simplification and Generalization"* a napovedá niečo aj o štruktúre samotného algoritmu[7]. Ten je zložených z troch zodpovedajúcich častí:

- Ako *generalizáciu* (zovšeobecnenie) chápeme pridávanie prvkov typu a^f do EF. Pridanie týchto prvkov totiž zväčšuje množinu príkladov, ktoré sú pokryté EF. O generalizáciu sa v našom algoritme stará prvý cyklus (riadky 0-10), ktorý po tom, čo odpozorujeme zmenu hodnoty f, do EF pridá hypotézu (prvok a^f) o tom, že ju spôsobila akcia a. Ak sa v EF už takýto prvok nachádza, tak vhodne upraví pravdepodobnosť nie len jeho, ale aj všetkých jeho podmienok súvisiacich s aktuálnym pozorovaním.
- *Špecifikáciou* rozumieme zas pridávanie prvkov typu a^f_c do EF, o čo sa stará druhý cyklus (riadky 12-18). Podmienky a^f_c totiž zmenšujú počet príkladov pokrytých EF, tým že kladú nové požiadavky na aplikovateľnosť už naučených efektov a^f. Tento cyklus iteruje cez všetky prvky typu a^f, pre ktoré je vstupný príklad negatívny, znižuje ich pravdepodobnosť a pridáva do EF niekoľko hypotéz o tom, že efekt a^f

[3] Efektová formula môže byť chápaná ekvivalentne ako množina atómov, alebo ako ich konjunkcia (teda formula). Názov *efektová formula* bol inšpirovaný štruktúrou použitou v [1].

[4] Učenie sa efektov s disjunktnými podmienkami je však podstatne ťažší problém a náš algoritmus, ani väčšina súčasných metód ho nerieši. Schopnosť reprezentovať takéto podmienky by preto bola pre nás zbytočná.

[5] Vravíme, že množina fluentov je konzistentná, ak neobsahuje dva vzájomne opačné fluenty.

[6] Treba podotknúť, že pozorovania o a o' môžu byť neúplné. Nemá však zmysel uvažovať prázdne pozorovania.

[7] Mimochodom, názov $3SG$ algoritmu je inšpirovaný algoritmom $SLAF$ popísaným v [1].

```
VSTUPY: Najnovší príklad (o,a,o'), kde a je vykonaná akcia a o,o' sú pozorovania (pred a po nej).
        Momentálny akčný model ⟨EF,P⟩, ktorý budeme modifikovať (môžeme začínať aj s prázdnym).
VÝSTUP: Modifikovaný akčný model ⟨EF,P⟩.

Foreach f ∈ Δ(o,o') do {
    // generalization of EF
    If a^f ∉ EF then Pridaj a^f do EF.
    Else {
        Uprav P(a^f) tým, že zvýšiš pos(a^f).
        Foreach c ∈ o do {
            if a^f_c ∈ EF then Uprav P(a^f_c) tým, že zvýšiš pos(a^f_c).
            if a^f_c̄ ∈ EF then Uprav P(a^f_c̄) tým, že zvýšiš neg(a^f_c).
        }
    }
}

Foreach f také, že f̄ ∈ o' do {
    If a^f ∈ EF {
        Uprav P(a^f) tým, že zvýšiš neg(a^f).
        // specification of EF
        Foreach c také že c̄ ∈ o do: if a^f_c ∉ EF then Pridaj a^f_c do EF.
    }
}

// simplification of EF
Foreach a^f_c ∈ EF staršie ako memoryLength do {
    If P(a^f_c) < minP then Vymaž a^f_c z EF.
}
Foreach a^f ∈ EF staršie ako memoryLength do {
    If ( P(a^f) < minP and nemáme v EF žiadne a^f_c ) or ( pos(a^f) + neg(a^f) < minEx ) {
        Vymaž a^f z EF.
    }
}
```

Obr. 1: Pseudokód algoritmu $3SG$.

nenastal, preto že a^f je podmienené nejakým fluentom c, ktorý práve neplatí.

- *Simplifikáciou* (zjednodušením) EF je jednoducho povedané zabúdanie tých prvkov, ktoré sa za nejaký časový interval (určený konštantou $memoryLength$) nestihli dostatočne potvrdiť (teda majú príliš nízku pravdepodobnosť).

4 Vlastnosti

Prvou dôležitou vlastnosťou $3SG$ je, že ide o *online* algoritmus. V prípade učenia sa akcií to znamená, že namiesto celej histórie odpozorovaných príkladov dostáva na vstupe vždy len jeden (najnovší) príklad, ktorý použije na nezvratnú modifikáciu akčného modelu. Nevýhodou online algoritmov oproti ich offline alternatívam je, že ich výsledok závisí od poradia vstupov ktoré dostávajú, a teda ich výsledky môžu byť menej presné. Na druhej strane ale online algoritmy pracujú s podstatne menšími vstupmi, čo ich robí výrazne rýchlejšími a vhodnými pre učenie sa v reálnom čase.

Závislosť na poradí vstupov nás vedie k otázke korektnosti algoritmu. Vo všeobecnosti označujeme algoritmy za korektné, ak pre akýkoľvek vstup skončia v konečnom čase a vrátia správny výsledok. Definícia "správneho výsledku" však v našom prípade bude musieť byť pomerne uvoľnená z dvoch dôvodov:

1. Online algoritmy ako ten náš nemajú prístup ku konkrétnym predchádzajúcim ani budúcim vstupom.
2. Aj keby k nim prístup mali, tak vďaka senzorickému šumu a možnosti zlyhania akcií, či dokonca zmien samotných pravidiel domény nemôžeme predpokladať konzistenciu nového vstupu so staršími.

Za správny výsledok teda považujeme taký akčný model, ktorý *pokrýva nový príklad* a zároveň platí, že *ak v ňom vznikol konflikt* s nejakým starším príkladom, musí to byť *spôsobené odstránením konfliktu* s novým príkladom, alebo jeho *pokrytím*.

Veta 1 (korektnosť)**.** *Algoritmus 3SG bez zabúdania ($memoryLength = \infty$) je korektný podľa hore uvedenej definície.*

Dôkaz. Aby sme mohli konštatovať korektnosť, musíme dokázať platnosť nasledovných troch tvrdení:

i. *Algoritmus $3SG$ skončí v konečnom čase pre akýkoľvek vstup*. Táto podmienka nám vyplýva zo štruktúry algoritmu – prvé 2 cykly (riadky 0-18 na obr. 1) totiž iterujú vždy len cez konečnú množinu pozorovaní (či už o, alebo o'), pričom túto množinu nijako nemodifikujú. Zvyšné 2 cykly (riadky 21-28) potom iterujú cez množinu EF, pričom z nej len vymazávajú prvky. Všimnime si, že toto platí aj pre algoritmus so zabúdaním.

ii. *Výsledný akčný model vždy pokrýva nový príklad*. Podľa definície pokrývania vyššie, musí platiť že pre všetky $f \in \Delta(o, o')$ máme v EF prvok a^f. O to sa nám priamo stará prvá časť algoritmu, ktorá a^f do EF pridá – viď riadok 2. Ak nemáme povolené zabúdanie, žiadna iná časť algoritmu potom a^f nemôže odstrániť.

iii. *Ak vo výslednom akčnom modeli vznikol konflikt so starším príkladom, musí to byť spôsobené buď odstránením konfliktu s novým príkladom, alebo jeho pokrytím*. Toto tvrdenie triviálne vyplýva z faktu, že (pokiaľ nie je povolené zabúdanie) akákoľvek úprava EF je spôsobená buď snahou o pokrytie nového príkladu (pridávanie a^f na riadku 2), alebo pridávaním nových podmienok a^f_c (riadok 16) pre tie a^f, vzhľadom na ktoré je nový príklad negatívny. Podľa definície konfliktu (vyššie) totiž pridanie takých podmienok a^f_c, že $\overline{c} \in o$ odstraňuje konflikt s daným príkladom (pretože ak sa v o vyskytuje $\overline{c}$, nemôže tam byť zároveň aj c).

□

Veta 2 (zložitosť)**.** *Označme si veľkosť vstupnej EF ako n a veľkosť väčšieho zo vstupných pozorovaní o, o' ako m. Časová zložitosť $3SG$ algoritmu je polynomiálna vzhľadom na veľkosť vstupu. V prípade, že EF implementujeme ako B-strom, je táto zložitosť konkrétne $\mathcal{O}\left((m^2 + n^2) \log n\right)$.*

Dôkaz. Najskôr si vyjadríme zložitosť všetkých štyroch *Foreach* cyklov nášho algoritmu, pričom sa budeme odvolávať na pseudokód z obr. 1. Operácie ktoré v tomto dôkaze nespomíname sa vykonávajú v konštantnom čase a môžeme ich zanedbať.

i. V prvom cykle iterujeme cez množinu $\Delta(o, o')$ (riadok 0) ktorá má v najhoršom prípade m prvkov. V každej iterácii potrebujeme zistiť, či $a^f \in EF$ (riadky 2 a 3), pričom vyhľadanie prvku v B-strome má zložitosť $\mathcal{O}(\log n)$ [3]. Tu máme 2 možnosti, pričom do výpočtovo náročnejšej vetvy sa dostávame, ak $a^f \in EF$. V nej sa nachádza vnorený *Foreach* cyklus (riadok 5), ktorý opäť iteruje cez najviac m prvkov a pre každý z nich potrebuje zisťovať, či sú 2 rôzne prvky obsiahnuté v EF (riadky 6 a 7). Toto zisťovanie nám teda dáva zložitosť $2 \cdot \log n$. Zložitosť prvého *Foreach* cyklu v najhoršom prípade je teda: $m \cdot m \cdot (2 \cdot \log n) \leq \mathcal{O}(m^2 \cdot \log n)$

ii. V druhom cykle iterujeme opäť cez množinu maximálne m prvkov (riadok 12) a v každej iterácii zisťujeme prítomnosť prvku v EF (riadok 13) so zložitosťou $\mathcal{O}(\log n)$. Vnorený cyklus na riadku 16 nám potom maximálne m krát najskôr zisťuje prítomnosť nejakého prvku v EF a ak tam nie je, tak ho pridá. Keďže zložitosť vkladania prvku do B-stromu je tak isto $\mathcal{O}(\log n)$ [3], dáva nám celý tento cyklus pre najhorší prípad zložitosť:
$m \cdot ((\log n) + m \cdot 2(\log n)) \leq \mathcal{O}(m^2 \cdot \log n)$

iii. V treťom, predposlednom cykle algoritmus vymazáva (zabúda) niektoré staré prvky z EF. Aby sme vedeli ktoré prvky vymazať, musíme otestovať všetkých n prvkov a v najhoršom prípade ich vymazať všetky (riadok 22). Keďže vymazávanie z B-stromu beží tak isto v čase $\mathcal{O}(\log n)$ [3], dostávame pre tento cyklus zložitosť: $\mathcal{O}(n \cdot \log n)$

iv. Posledný cyklus nám opäť prechádza cez všetkých n prvkov EF (riadok 24), ale pre potenciálne každý z nich tu ešte potrebujeme overiť neexistenciu relevantných prvkov a^f_c (riadok 25), čo pre nás znamená ďalšiu iteráciu cez n-prvkovú množinu EF. Dokopy s vymazávaním a^f teda dostávame: $n \cdot n \cdot \log n \leq \mathcal{O}(n^2 \cdot \log n)$

Ak teraz skombinujeme zložitosti všetkých štyroch cyklov (bežiacich sériovo), dostaneme výslednú časovú zložitosť $3SG$ algoritmu:

$$\begin{aligned} & \mathcal{O}(m^2 \cdot \log n) + \mathcal{O}(m^2 \cdot \log n) + \mathcal{O}(n \cdot \log n) + \mathcal{O}(n^2 \cdot \log n) \\ = \; & \mathcal{O}\left((m^2 + m^2 + n + n^2) \log n\right) \\ = \; & \mathcal{O}\left((m^2 + n^2) \log n\right) \end{aligned}$$

□

Poznámka 1. *Implementácia EF pomocou B-stromu nie je pre náš problém úplne optimálna. Všimnite si, že v poslednom cykle potrebujeme iterovať cez celú množinu EF aby sme zistili, čo z nej potrebujeme vymazať. Ak by sme mali EF vhodnejšie štrukturovanú, s ohľadom na vzťahy medzi jej prvkami, vedeli by sme tento proces zrýchliť.*

Po vyhlásení korektnosti a vyjadrení zložitosti algoritmu sa ešte pozrime na niekoľko jeho vlastností, ktoré môžu ponúknuť hlbší vhľad do jeho fungovania. Konkrétne budú nasledujúce vety hovoriť o tom, za akých podmienok sa naučíme efekty akcií. Začnime teda definíciou tohoto poj-mu.

Definícia 1. Efektom budeme nazývať dvojicu $E = \langle a^f, C = a^f_{c0}, a^f_{c1}, \ldots, a^f_{ci} \rangle$ kde C je množinou podmienok

pre a^f. Hovoríme, že sme sa *naučili efekt* E práve vtedy, keď vzhľadom na náš akčný model $\langle EF, P\rangle$ platí nasledovné:

i. $a^f \in EF$

ii. $\forall a_c^f \in C : (a_c^f \in EF) \wedge \big(P(a_c^f) \geq minP\big)$

iii. $(C = \emptyset) \Rightarrow \big(P(a^f) \geq minP\big)$

Inak povedané, a^f a všetky jeho podmienky musia byť obsiahnuté v EF a musia mať priradenú dostatočne vysokú[8] pravdepodobnosť P. Ak je E nepodmienený efekt (resp. $C = \emptyset$), tak vyžadujeme aby samotné a^f bolo dostatočne pravdepodobné.

Pre nasledujúce vety zavedieme značenie, kde $pos(a^f)_c$ bude znamenať *„počet pozitívnych príkladov pre a^f kde bol odpozorovaný fluent c“* (podobne pre negatívne príklady).

Veta 3. Uvažujme 3SG algoritmus bez zabúdania (memoryLength = ∞). Ak odpozorujeme dostatok pozitívnych a žiadne negatívne príklady vzhľadom na a^f (presnejšie $pos(a^f) \geq minEx \wedge neg(a^f) = 0$), tak môžeme s istotou tvrdiť, že sa *naučíme nepodmienený efekt* $E = \langle a^f, \emptyset\rangle$. Taktiež môžeme povedať, že sa *nenaučíme* žiaden iný *podmienený efekt* $E' = \langle a^f, C \neq \emptyset\rangle$.

Veta 4. Ak odpozorujeme aspoň jeden negatívny príklad vzhľadom na a^f, tak na naučenie sa *nepodmieneného* efektu $E = \langle a^f, C = \emptyset\rangle$ potrebujeme zabúdanie (teda $memoryLength$ musí byť konečné číslo).

Veta 5. Uvažujme $3SG$ algoritmus bez zabúdania (memoryLength = ∞). Ak odpozorujeme dostatok pozitívnych príkladov vzhľadom na a^f (teda $pos(a^f) \geq minEx$), pričom počet tých kde sme videli fluent c je o dosť väčší ako počet tých, kde sme videli $\overline{c}$ pre každé naše a_c^f (presnejšie $\forall a_c^f \in C : pos(a^f)_c \geq minP \cdot \big(pos(a^f)_c + pos(a^f)_{\overline{c}}\big)$) a zároveň každé $a_c^f \in C$ je podporené aspoň $minEx$ príkladmi a existuje aspoň jeden negatívny príklad vzhľadom na a^f, pri ktorom platilo $\overline{c}$ tak môžeme s istotou tvrdiť, že sa naučíme *podmienený* efekt $E = \langle a^f, C \neq \emptyset\rangle$.

Dôkaz. Definícia 1 stanovuje niekoľko podmienok, ktoré musia byť splnené, aby sme mohli povedať že sme sa *naučili* efekt E. Postupne si ukážeme ako z predpokladov vety vyplýva každá z týchto podmienok:

i. $a^f \in EF$ pretože $3SG$ algoritmus nám pridá a^f do EF pri prvom pozitívnom príklade (riadok 2 na Obr. 1) a zároveň predpokladáme, že $pos(a^f) \geq minEx$, pričom $minEx > 0$.

ii. Na to, aby $a_c^f \in EF$ potrebujeme (podľa riadku 16) aby $neg(a^f)_{\overline{c}} > 0$, čo je priamo našim predpokladom. Teraz si uvedomme, že podľa sémantiky EF sú pozitívne príklady pre a_c^f práve tie, ktoré sú pozitívne aj pre a^f a zároveň v nich platí c. Podobne, negatívne príklady pre a_c^f sú tie, ktoré sú pozitívne pre a^f, ale platí v nich $\overline{c}$. Náš predpoklad $pos(a^f)_c \geq minP \cdot \big(pos(a^f)_c + pos(a^f)_{\overline{c}}\big)$ si teda môžeme prepísať do tvaru $pos(a_c^f) \geq minP \cdot \big(pos(a_c^f) + neg(a_c^f)\big)$. Úpravou dostávame: $pos(a_c^f) / \big(pos(a_c^f) + neg(a_c^f)\big) \geq minP$, čo je ekvivalentné s podmienkou $P(a_c^f) \geq minP$ z definície, keďže tiež predpokladáme, že a_c^f je dostatočne podporené ($pos(a_c^f) + neg(a_c^f) \geq minEx$).

iii. To že platí $(C = \emptyset) \Rightarrow \big(P(a^f) \geq minP\big)$, dokážeme sporom. Predpokladajme negáciu tejto vety: $(C = \emptyset) \wedge \big(P(a^f) < minP\big)$ To, že C je prázdna množina znamená (za predpokladu neprázdnych pozorovaní), že nemáme ani jeden negatívny príklad vzhľadom na a^f (algoritmus totiž pri negatívnych príkladoch vzhľadom na a^f pridáva na riadku 16 do EF podmienky a_c^f). Keďže predpokladáme, že $pos(a^f) \geq minEx$, a nemáme žiadne negatívne príklady, pravdepodobnosť $P(a^f)$ sa musí rovnať 1. Nemôže byť teda menšia ako $minP$.

□

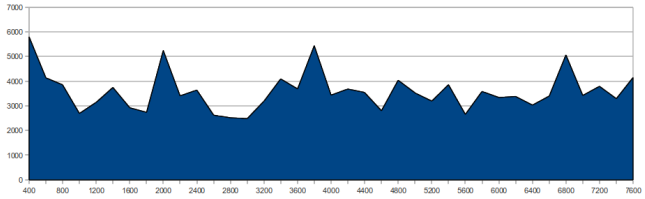
5 Experimenty

Na vyjadrení zložitosti (veta 2) vidíme, že pre nás bude hrať dôležitú úlohu *veľkosť* našej *reprezentácie akčného modelu*. Okrem nej nás zaujíma *rýchlosť behu* $3SG$ algoritmu v praxi a samotné *výsledky*, ktoré produkuje. Ako testovaciu doménu sme použili akčnú FPS hru Unreal Tournament 2004, ku ktorej sme pristupovali pomocou frameworku Pogamut 3 [7]. Agent volal $3SG$ algoritmus po každej odpozorovanej akcii, čo bolo priemerne 7.432 krát za sekundu, pričom priemerná veľkosť jeho pozorovaní bola 70 fluentov (teda priemerná veľkosť o aj o' bola 70). Nasledujúce výsledky zodpovedajú jednej hre, trvajúcej 48.73 minút (na extenzívnejšie praktické experimenty tu bohužiaľ nie je dosť miesta).

5.1 Veľkosť akčného modelu

Počas jednej hry sme odpozorovali a spracovali 21733 príkladov, ktoré zodpovedali 7632 časovým krokom (v jednom kroku totiž mohlo byť pozorovaných súčasne viac akcií). Za veľkosť akčného modelu nám stačí považovať veľkosť EF, nakoľko funkcia P je reprezentovaná implicitne vzorcom, pričom jej definičný obor je práve EF. Pri tomto experimente sme mali konštanty nastavené takto: $minP = 0.9, minEx = 3, memoryLength = 50$. To znamená, že sme z EF zabúdaním vymazávali tie prvky, ktoré sa za 50 krokov (asi 19.154 sekundy) nestihli potvrdiť aspoň 3 príkladmi aspoň na 90%. Graf na obr. 2 znázorňuje vývoj veľkosti EF počas tejto 48.73-minútovej hry.

[8] Minimálna hodnota priradenej pravdepodobnosti je daná konštantou $minP$ z intervalu $(0, 1)$, napríklad $0,9$.



Obr. 2: Vývoj veľkosti EF (os y) v čase (jednotkou na osi x je poradové číslo časového kroku).

Vidíme, že vďaka zabúdaniu sa veľkosť nášho akčného modelu udržuje na viac-menej konštantnej úrovni. Bez zabúdania nám táto veľkosť samozrejme nekontrolovane narástla.

5.2 Rýchlosť

Pre meranie rýchlosti sme najskôr vytvorili záznam za sebou idúcich príkladov odpozorovaných agentom počas hry, a až potom sme opakovane volali $3SG$ sériovo pre všetky zaznamenané príklady. Algoritmus bol naprogramovaný v jazyku Python a pristupoval k akčnému modelu uloženému v MySQL databáze. Experiment bežal pod 64-bitovým Linuxovým OS, na systéme s procesorom Intel(R) Core(TM) i5 3.33GHz. Spracovanie 21733 príkladov zo 48.73-minútovej hry nám trvalo len 12 minút a 10.138 sekundy. To znamená, že za takýchto podmienok môžene používať 3SG algoritmus efektívne v reálnom čase.

5.3 Výsledky

Pre zlepšenie čitateľnosti výsledkov sme EF preložili do plánovacieho jazyka PDDL [8]. Pri tomto preklade sme ignorovali všetky tie prvky EF, ktoré mali pravdepodobnosť menšiu ako $minP$. Taktiež sme kvôli zostručneniu zápisu zlúčili tie skupiny efektov kde to bolo možné bez straty informácie pomocou zavedenia premenných a v prípade, že všetky efekty nejakej akcie mali rovnakú podmienku, označili sme ju za podmienku vykonateľnosti danej akcie.

Prvým príkladom zobrazeným na obr. 3 (nižšie) je akcia $changeWeapon(A, B, C)$, ktorá je naučená presne ako by sme očakávali – agent A môže zmeniť zbraň ktorú práve používa z B na C iba ak B a C sú rôzne. Ak to spraví, tak bude držať C a nebude naďalej držať B. Druhá akcia, $move(A, B)$, má viac naučených efektov (pre stručnosť uvádzame 4 z nich). Prvé dva sú očakávateľné – agent A sa bude nachádzať na pozícii B a nebude naďalej na žiadnej z okolitých pozícií Z. Ďalšie dva efekty už sú pre nás menej intuitívne, avšak naučené sú správne. Pokrývajú totiž odpozorované príklady. Konkrétne efekt 3 hovorí, že ak agent príde na pozíciu $playerstart25$, tak zomrie. To je spôsobené jednoducho tým, že ho na tejto pozícii vždy niekto zabíjal. Podobne, ak agent prišiel na $playerstart20$, tak sa jeho zdravie dostalo na maximálnu hodnotu. To je spôsobené zrejme tým, že sa na danej pozícii nachádzala lekárnička. Ešte je zaujímavé, že zo šiestich naučených efektov, sa tie pre nás intuitívne (prvé 4) agent naučil veľmi rýchlo (mal ich v EF už pred časovým krokom 150, čo predstavuje menej ako 58 sekúnd). Neintuitívne efekty (posledné 2) mu trvali dlhšie (šiesty niečo pod 2,5 a piaty skoro 16 minút).

6 Záver

Ukázali sme, že zložitosť algoritmu $3SG$ je polynomiálna vzhľadom na veľkosť vstupu a tiež že je za určitých podmienok korektný. Prvé praktické experimenty naznačujú, že má potenciál pre využitie v reálnom čase - a to aj v pomerne komplexných doménach. Je však potrebné dôkladnejšie preskúmať jeho efektívnosť, či už ďalšími praktickými experimentami vo viacerých prostrediach, alebo z teoretického hľadiska kompetitívnou analýzou [4]. Následne sa môžeme začať venovať skúmaniu vplyvu samotného správania učiaceho sa agenta na rýchlosť a presnosť učenia.

Poďakovanie

Tento príspevok vznikol za podpory Vedeckej grantovej agentúry Ministerstva školstva, vedy, výskumu a športu SR a Slovenskej akadémie vied v rámci projektu VEGA č. 1/1333/12.

Literatúra

[1] E. Amir, A. Chang: Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, Volume 33 Issue 1: 349-402 (2008)

[2] M. Balduccini: Learning Action Descriptions with A- Prolog: Action Language C. In: *Proceedings of Logical Formalizations of Commonsense Reasoning*, AAAI Spring Symposium (2007)

[3] R. Bayer, E. McCreight: Organization and Maintenance of Large Ordered Indices. In: *Mathematical and Information Sciences Report No. 20*, Boeing Scientific Research Laboratories (1970)

[4] A. Borodin, R. El-Yaniv: Online Computation and Competitive Analysis, *Cambridge University Press* (1998)

[5] M. Čertický. Action Learning with Reactive Answer Set Programming: Preliminary Report. In: *Proceedings of The Eighth International Conference on Autonomic and Autonomous Systems*, ICAS'12 (2012).

[6] T. Eiter, E. Erdem, M. Fink, and J. Senko. Updating action domain descriptions. In: *Proceedings of 19th Int. Joint Conf. on Artificial Intelligence* (IJCAI'05): 418–423 (2005)

```
(:action changeWeapon
        :parameters (?a ?b ?c)
        :precondition (and (differentWeapon ?b ?c) (differentWeapon ?c ?b))
        :effect (and
                (equippedWeapon ?a ?c)
                (not (equippedWeapon ?a ?b))
        )
)
(:action move
        :parameters (?a ?b)
        :effect (and
                (onPosition ?a ?b)
                (when (and (edge ?b ?z) (edge ?z ?b)) (not (onPosition ?a ?z)))
                (when (= ?b playerstart25) (dead ?a))
                (when (= ?b playerstart20) (health ?a maximum))
                … atď.
        )
)
```

Obr. 3: Príklad naučených sa akcií (zapísaných v plánovacom jazyku PDDL).

[7] J. Gemrot, R. Kadlec, M. Bida, O. Burkert, R. Pibil, J. Havlicek, L. Zemcak, J. Simlovic, R. Vansa, M. Stolba, T. Plch, C. Brom: Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: *Agents for Games and Simulations*, LNCS 5920: 1-15, Springer (2009)

[8] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins: PDDL - The Planning Domain Definition Language. *Draft*. Available at http://www.cs.yale.edu/dvm (1998).

[9] K. Mourao, R. P. A. Petrick, M. Steedman. Learning action effects in partially observable domains. In: *Proceedings of 19th European Conference on Artificial Intelligence*, ECAI'10 (2010)

[10] Q. Yang, K. Wu, Y. Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence, Volume 171*: 107-143 (2007)

[11] L. S. Zettlemoyer, H. M. Pasula, L. P. Kaelbling. Learning probabilistic relational planning rules. *MIT TR* (2003)