Математички Билтен
24 (L)
2000 (141–152)
Скопје, Македонија

ISSN 0351-336X

ПОСЛЕДИЧНО ПРОГРАМИРАЊЕ: АЛГОРИТАМ „ОД ПОТЦЕЛТА ВРАТИ СЕ"

Силвана Петрушева * и Стево Божиновски **

Апстракт:

Во трудов се презентирани експерименти со алгоритмот „од потцелта врати се" за решавање на проблемот на наоѓање најкраток пат од сите можни патишта, во околина во која има n состојби, од кои една е почетна, една е состојба на цел и некои состојби се неповолни и треба да се избегнат. Од секоја состојба можат да се превземат акции кои водат или до друга состојба, или до препрека.

За решавање на проблемот се користи агентен пристап. Архитектурата на агентот кој се користи овде е од невронски тип, поточно, се користи Невронската Кросбар Адаптивна Мрежа (NN - CAA). Опишан е математичкиот модел на овој агент. Програмот за алгоритмот е изработен во програмскиот пакет DELPHI 1.

Направена е споредба на резултатите со алгоритмот „од целта врати се" со што се покажува дека алгоритмот „од потцелта врати се" е поефикасен.

1. Вовед

Системот *агент - околина* е основен концепт во современото разбирање на интелигентните системи, [2]. Агентот е апстракција која може да репрезентира животно, робот, или друг систем со некое претпоставено ниво на интелигенција. Под *агентен пристап* подразбираме пристап при кој проблемот е зададен со некоја околина, а се конструира *архитектура* на агент која ќе биде користена да го реши дадениот проблем преку свое *однесување* во околината.

Пример на класа проблеми кои се решаваат со овој пристап се проблемите од Динамичко програмирање [1], каде околината се претставува со граф, чии темиња (состојби) имаат извесна *вредност*. Обично задачата е да се најде политика на однесување која од секоја состојба покажува како да се стигне до целната состој-

ба, оптимирајќи некој параметар кој е функција од вредностите на темињата. Кај динамичкото програмирање обично е *познат* графот кој треба да биде околина на агентот, и зададени се сите состојби и сите вредности на состојбите.

Посебна класа на динамичкото програмирање е *последичното програмирање*. Последичното програмирање е динамичко програмирање во кое графот не е познат, т.е. не е позната агентната околина. Тоа е потцел-цел програмирање, при кое агентот во својата архитектура има концепт дека доаѓање до потцел ја зголемува шансата да се достигне целта. Според тоа агентот развива стратегија за постигнување *на потцел*. Бидејќи нема модел на околината, т. е. графот не е зададен, агентот учи политика на однесување преку експериментирање, со метод на обид и погрешка. Агентот ќе научи оптимално однесување во околината учејќи ги последиците од своето однесување. Општо, концептот *последица* во последно време е препознаен [3] како еден од клучните во вештачката интелигенција.

2. CAA агенти

Во нашите истражувања, во кои се користи агентен пристап, ја користиме агентната архитектура наречена *кросбар адаптивна мрежа* (Crossbar Adaptive Array, CAA) [2]. Основните концептуални блокови на SAA се:

1) ОКОЛИНА НА ОДНЕСУВАЊЕ. Околината ги прима акциите на агентот и ја пресметува следната ситуација. Ситуациите не се директно видливи од агентот. Ситуацијата X е апроксимација на состојбата на околината. Ситуацијата може да биде презентирана со вектор со карактеристики за ситуацијата, каде компонентите на X се во општ случај реални броеви, или вектор што има една компонента со вредност 1, а другите нули. Овде се користи вториот случај и се работи со множество ортогонални вектори како влезни сигнали.

2) КРОСБАР АСОЦИЈАТИВНА МЕМОРИЈА. Тоа е матрица W во која компонентите $\{w_{aj}\}$ се користат на два начини: за проценка на внатрешната состојба на агентот поради сретнатата ситуација (колоните на W,) и за избор на акцијата на агентот во сретнатата ситуација (редовите на W). Матрицата W може да се посматра како табела од SAE компоненти (situation-action-emotion). Секоја компонента w_{aj} е кросбар компонента бидејќи се користи за пресметување на внатрешната состојба и на акцијата на агентот.

3) ОЦЕНУВАЧ НА СОСТОЈБА - ја оценува внатрешната состојба на агентот како функција од ситуацијата во која тој се наоѓа. Внатрешната состојба е сфатена како *емоција* на агентот, пожелност да се биде во таа ситуација [2],[4].

4) СЕЛЕКТОР НА АКЦИЈА: Се земаат во обзир две тенденции за

избор на акција: една е *политика на избор на акција* научена и меморирана во асоцијативната меморија, друга е акција *предложена* од системот од повисок ред (супервизор).

5) СИСТЕМ ОД ПОВИСОК РЕД (СУПЕРВИЗОР). Тоа е компонента од хиерархискиот контролен систем од CAA архитектурата. Тој може да посматра информација од сите споменати компоненти. Специфичната структура на овој систем е оставена недефинирана во CAA архитектурата. Во поедноставните случаи овој систем не бара додатна мемориска структура, додека во покомплексните задачи може да биде дефинирана и додатна мемориска структура. Овде се разгледува случај кога нема додатна меморија, единствена меморија на агентот е матрицата W.

3. Методи на агентно учење

Методите на агентно учење кои се од интерес во ова истражување се делат на три групи: учење со помош на учител, учење со поттик, и учење врз база на емоции. Кај *учењето со учител*, постои систем учител кој на агентот му кажува во која ситуација која акција е најдобра. Кај *учењето со поттик* [6], [7] постои учител кој не кажува која акција е најдобра, туку само кажува евалуација на акцијата што агентот веќе ја направил; агентот од таква евалуација треба самиот да извлече заклучок која е акцијата која понатаму треба да се превзема во слична ситуација. Кај *учењето врз база на емоција* не постои никаков надворешен учител ниту надворешен поттик. Агентот самиот, врз основа на внатрешната емоција која ја чувствува поради превземената акција, треба да заклучи која акција треба да ја превземе во иднина во дадената ситуација: акцијата што веќе ја пробал или некоја друга. CAA архитектурата го користи емоционалното учење; всушност оваа архитектура го има воведено ова учење во теоријата на учечките агенти [2,4].

4. Методот на учење кај CAA

Методот на учење кај CAA е дефиниран преку три функции кои се пресметуваат во тековната и следната состојба: функцијата на акција **Afunc** во тековната состојба, функцијата на проценка на внатрешната состојба **Vfunc** пресметана во последичната состојба, и функцијата за ажурирање на меморијата, **Wfunc** пресметана во тековната состојба. Последична состојба е онаа во која се доаѓа после превземање на акција во тековната состојба. Сите функции се дефинирани над SAE компонентите.

$$\text{состојба } j : y = \underset{a \in A(j)}{\mathbf{Afunc}}\{w_{aj}\} \quad \text{резултат : } y = i$$

состојба $k : v_k = \underset{b \in A(k)}{\mathbf{Vfunc}}\{w_{bk}\}$

состојба $j : \Delta w_{ij} = \mathbf{Ufunc}(v_k) = \mathbf{Ufunc}(\mathbf{Vfunc}\{w_{bk}\}) = \underset{b \in A(k)}{\mathbf{Wfunc}}\{w_{bk}\}$

состојба k: $y = \mathbf{Afunc}\{w_{bk}\}$.

Така, значи, кога е во состојба j, САА одбира акција според **Afunc** функцијата. Новата состојба k се пресметува и нејзините SAE компоненти се испитуваат, користејќи ја **Vfunc**, како резултат на што се пресметува вредноста v_k на состојбата k. Таа вредност се користи за ажурирање на SAE компонентата на акцијата што го иницирала процесот.

Вербално методот на учење кај САА можеме да го опишеме вака:

1) состојба j: изврши акција во зависност од SAE компонентите; пресметај k

2) состојба k: пресметај ја вредноста на состојбата, користејќи ги SAE компонентите

3) состојба j: зголеми ја SAE вредноста користејќи ја вредноста на состојбата k

4) $j = k$: оди во 1

5. Системот од повисок ред: модулирање на акции и емоции

Системот од повисок ред е центар од повисок ред кој ги одредува стратегиите на целиот систем. Тој ја одредува стратегијата за модулација на акција и емоција.

МОДУЛИРАЊЕ АКЦИИ: Ако системот за модулација од повисок ред не е присутен, САА изведува процес на селекција на акција на следен начин:

$$y = \underset{a}{\mathbf{Afunc}}\{w_{aj}\}$$

Ако е присутен, имаме: $y = \underset{a}{\mathbf{Afunc}'}\{w_{aj}, s_i\}$ каде с е реален број, кој е параметар на *стратегијата за барање* и **Afunc**' е модифицирана функција за селекција на акции со воведување на параметарскиот вектор s. Тој се генерира според некоја стратегија за превземање акции на системот од повисоко ниво. На пример, ако системот од повисоко ниво одлучи дека стратегијата на превземање акција ќе биде случајна, тогаш множеството $\{s\}$ се генерира според некоја униформна распределба. Случајната стратегија се користи кога САА системот извршува испитување во непознат простор од состојби.

состојба $k : v_k = \mathbf{Vfunc}\{w_{bk}\}$
$b \in A(k)$

состојба $j : \Delta w_{ij} = \mathbf{Ufunc}(v_k) = \mathbf{Ufunc}(\mathbf{Vfunc}\{w_{bk}\}) = \mathbf{Wfunc}\{w_{bk}\}$
$b \in A(k)$

состојба k: $y = \mathbf{Afunc}\{w_{bk}\}$.

Така, значи, кога е во состојба j, САА одбира акција според **Afunc** функцијата. Новата состојба k се пресметува и нејзините SAE компоненти се испитуваат, користејќи ја **Vfunc**, како резултат на што се пресметува вредноста v_k на состојбата k. Таа вредност се користи за ажурирање на SAE компонентата на акцијата што го иницирала процесот.

Вербално методот на учење кај САА можеме да го опишеме вака:

1) состојба j: изврши акција во зависност од SAE компонентите; пресметај k

2) состојба k: пресметај ја вредноста на состојбата, користејќи ги SAE компонентите

3) состојба j: зголеми ја SAE вредноста користејќи ја вредноста на состојбата k

4) $j = k$: оди во 1

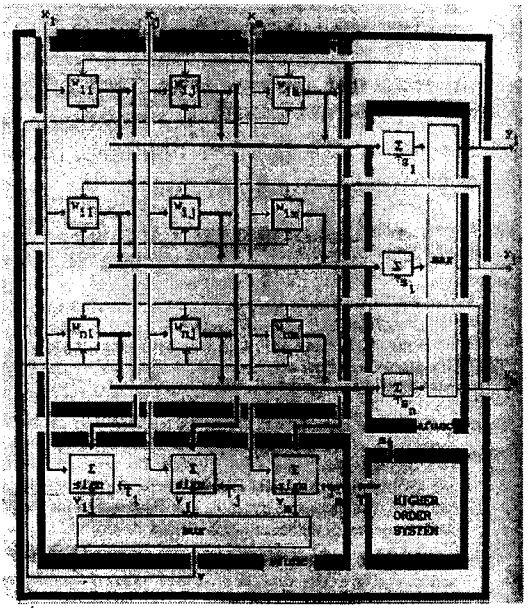
5. Системот од повисок ред: моделирање

[redacted]

МОДУЛИРАЊЕ ЕМОЦИИ: Пресметувањето на емоции се модулира како $v = \underset{a}{\mathbf{Vfunc}'}\{w_{aj}, T_j\}$ каде $\{T\}$ се параметри од некоја стратегија за избор на вредности на емоции. Со овој параметар однесувањето може да се модулира да биде повнимателно или поагресивно. На пример, ако има опасност, дали ќе се генерира предупредување или не. Ако предупредувањето не се генерира, САА нема да има концепт за ризик и ќе продолжи како порано. Ако предупредувањето е генерирано, тогаш можно е да има повеќе истражување пред да биде состојбата дизајнирана како пожелна.

6. NNCAA архитектура

Од генералната САА архитектура опишана погоре можат да се дефинираат повеќе типови конкретни САА архитектури со дефинирање на наведените функции.



Сл. 1

На Сликата 1 е даден пример на САА архитектура која се користи овде за решавање на проблемот и се означува како NNCAA архитектура, бидејќи вредноста на акцијата и состојбата се пресметуваат како елементи на формални неврони.

АКЦИИ: УЧЕЊЕ И МОДУЛИРАЊЕ

Пресметувањето на акциите се изведува на стандарден „невронски" начин:

$$y_a = \begin{cases} i, & ако \quad \max\limits_a\{w_{aj} + s_a\} = w_{ij} + s_i \end{cases}$$

s_a е променлива за модулација на акциите од системот од повисок ред. Таа ја презентира стратегијата на барање. Најпростата стратегија на барање е случајната, имплементирана како:

$$s = \textbf{montecarlo}[-0.5, 0.5]$$

каде **montecarlo**[интервал] е случајна функција која дава вредности кои се рамномерно распределени во дефинираниот интервал. На овој начин, NN-CAA ќе извршува случајно движење сé додека SAE компонентите добијат вредности кои ќе завладеат со однесувањето. Во програмот s_a се пресметува како s_a = random − 0.5 (случајни броеви на $[-0.5, 0.5]$).

ФУНКЦИЈА ЗА ПРЕСМЕТУВАЊЕ СОСТОЈБА

Компонентите v_k, $k = 1, 2, ...m$ во NN-CAA се пресметуваат на „невронски" начин: $v_k(t) = sgn\Big(\sum\limits_{a=1}^{n} w_{ak}(t-1) + T_k\Big)$. Овде временската разлика е важна. Таа нагласува дека ситуацијата презентирана со x_k ќе биде оценета со векторот w_k, кој беше пресметан во претходниот чекор. Во NN-CAA случајот горната равенка може да се претстави како

$$v_k = \begin{cases} \text{sgn}\Big(\sum\limits_{a=1}^{n} w_{ak}(t-1) + T_k\Big) & \text{ако } x_k(t) = 1 \\ 0 & \text{во друг случај} \end{cases} \qquad \text{sgn}(.) = \begin{cases} -1 & (.) < 0 \\ 0 & (.) = 0 \\ 1 & (.) > 0 \end{cases}$$

Вредноста на невронскиот праг T_k се пресметува како $T_k(t) = \textbf{предупредување}\{w_{ak}(t-1)\}$ каде **предупредување** {.} е некоја функција која предупредува на јавувањето на негативни SAE компоненти во v_k. Тоа е всушност параметар пресметан од стратегијата како да се работи со негативните SAE компоненти при пресметувањето на состојбата. Мотивацијата за ваквата функција е концептот ризик, кој природно се јавува во проблеми какде треба да се донесува одлука за следна акција. Ризикот е всушност свесност за непожелен исход, ако се одбере некоја акција. Рискантна состојба е онаа во која веројатноста за за избор на акција која ќе продуцира непожелен исход е голема.

ФУНКЦИЈА НА УЧЕЊЕ И ВРАЌАЊЕ НАЗАД

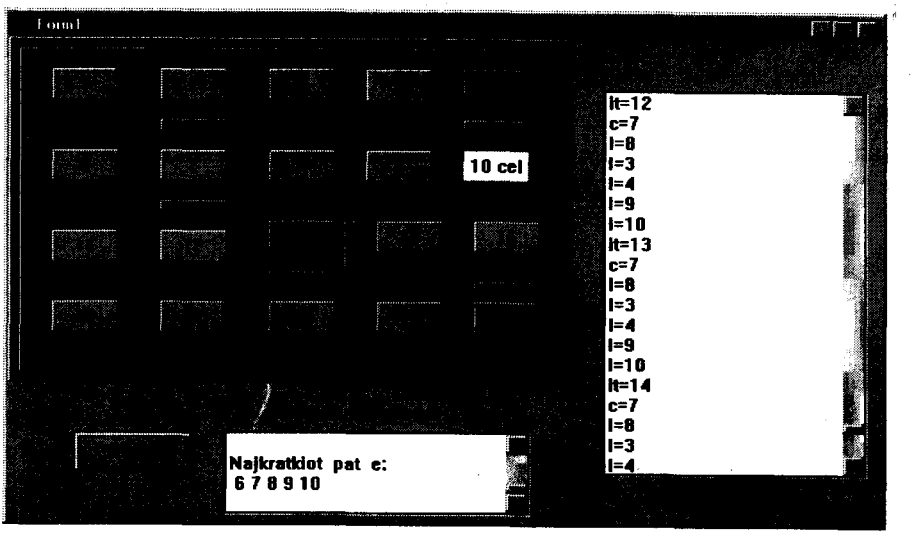
Правилото за учење во NN-CAA се дефинира како:

$$w_{aj}(t) = w_{aj}(t-1) + v_k(t)$$

Со ова правило едноставно адитивно се ажурираат SAE компонентите во *претходната* состојба, користејќи ја пожелноста на тековната состојба, која е последица од акцијата превземена во претходната состојба.

7. Експерименти со алгоритмот „од потцелта врати се" и споредба со алгоритмот „од целта врати се"

За екпсерименталниот дел на ова истражување е припремена програмска опрема во вид на симулационе програм изработен во програмскиот пакет DELPHI 1, [6]. Дизајнот на екранот е даден на слика 2.



Сл. 2

Претставени се $n = 20$ состојби од кои почетни можат да бидат состојбите: 1,6,11 и 16 а состојбата 10 е цел во која треба да се стигне, избегнувајќи ги непожелните состојби: 5,13 и 20. Од секоја состојба се можни 3 акции, кои водат или до друга состојба, или до препрека (ѕид).

Почетно знаење е матрицата W на SAE компоненти. За секоја состојба j ($j = 1,2,\ldots 20$), вредностите w_{ij} ($i = 1,2,3$) се зададени со почетни вредности. За состојбата 10 тие имаат вредност 1: $w_{1,10} = w_{2,10} = w_{3,10} = 1$. За непожелните состојби 5,13 и 20 тие имаат вредност -1 ($w_{1,5} = w_{2,5} = w_{3,5} = -1$), а за сите останати состојби имаат почетни вредности за w_{ij} едакви на 0.

Пресметувањето на функциите **Afunc, Vfunc** и **Ufunc** е опишано во точка 5. Тие функции се исти кај двата алгоритми „во потцелта врати се" и „во целта врати се".

Агентот влегува во непозната околина од n состојби, од зададена почетна сосотојба, барајќи ја целта.

Експериментот содржи неколку итерации.

Во секоја итерација, агентот се движи случајно додека не наиде на веќе претходно научена потцел. Оттаму до целта агентот се движи детерминистички, користејќи ја научената политика како да се однесува во секоја ситуација за да дојде до целта.

Потцел се дефинира кога ќе се достигне целта. Целта се смета како последица од претходната ситуација од која е дојдено до целта. Всушност, вака потцелта станува нова цел. Кај алгоритмот од „целта врати се" [2], во секоја итерација агантот се движи случајно додека не наиде потцел, а детерминистички оттаму до целта, од каде почнува нова итерација. За разлика од тој алгоритам, кај алгоритмот од „потцелта врати се" [4], не се оди до крај, агентот почнува нова итерација веднаш штом ке наиде на потцел. Процесот на учење кај двата алгоритми запира кога почетната состојба станува потцел, што значи натаму секоја итерација содржи само детерминистичко однесување на агентот.

Агентот кој научил некое однесување во околината, од почетната состојба до состојбата на цел оди директно, значи, тој научил *план* како да ја најде сосотојбата на цел почнувајќи од почетната состојба, избегнувајќи ги непожелните состојби.

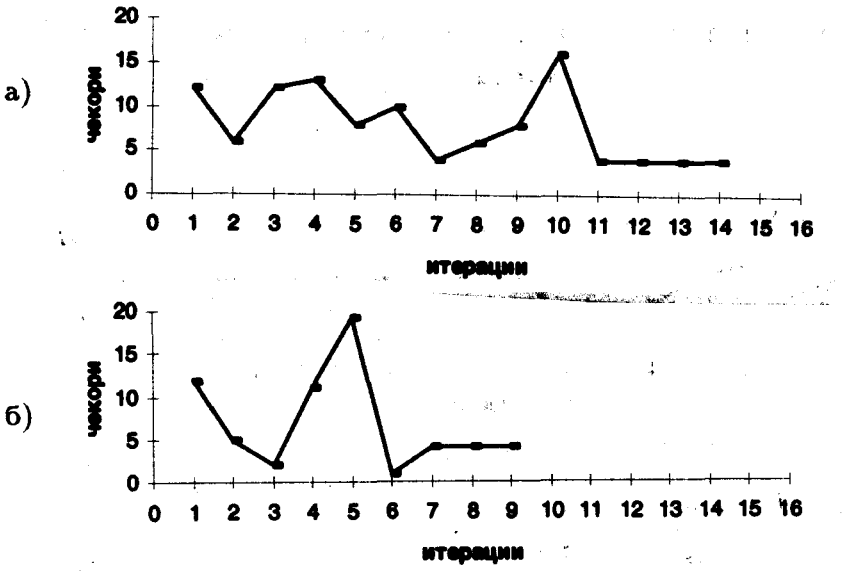
Алгоритмите гарантираат решение во смисол на наоѓање пат до целта. За да се решава проблемот на најкраток пат, потребно е, покрај матрицата W, да се воведе уште една мемориска варијабла во која ќе се меморира должината на најкраткиот пат постигнат во еден репродуктивен период. Под *репродуктивен период* се подразбира период во кој агентот го решава проблемот, наоѓа еден пат, и во општ случај во еден репродуктивен период тој нема да го најде најкраткиот пат. По наоѓањето на еден пат, започнува нов репродуктивен период кога агентот учи нов пат, независно од претходното решение. Варијаблата најкраток пат се пренесува по генетски пат до следната генерација агенти. Така генетската околина е оптимизациона околина која овозможува памтење само на најкраткиот пат во низа репродуктивни периоди. Со веројатност 1, овој опримизационен период ќе заврши со наоѓање на најкраткиот пат.

Експериментите покажуваат дека програмот ги наоѓа сите можни патишта од зададена почетна состојба (1,6,11 и 16) и најкраткиот меѓу нив.

Овде се прикажани по 2 експерименти за сите 4 почетни состојби: 6, 1, 11 и 16, еден експеримент со алгоритамот „од целта врати се" и еден со со алгоритамот „во потцелта врати се" и

споредени се резултатите. Таа споредба најдобро се гледа од *кривата на учење* која ја дава зависноста на бројот на чекори во една итерација од бројот на итерацијата.

На сл. 3, сл. 4, сл. 5 и сл. 6 дадени се кривите на учење при почетна состојба 6,1,11 и 16 соодветно со а) алгоритмот „од целта врати се" и б) алгоритмот „од потцелта врати се".



Сл. 3

При почетна состојба 6 (сл. 3) со алгоритамот „во целта врати се" потребни се вкупно 95 чекори додека се најде едно решение (пат) во 11 итерација, а со алгоритамот „во потцелта врати се" потребни се 50 чекори додека се најде едно решение во 7 итерација.

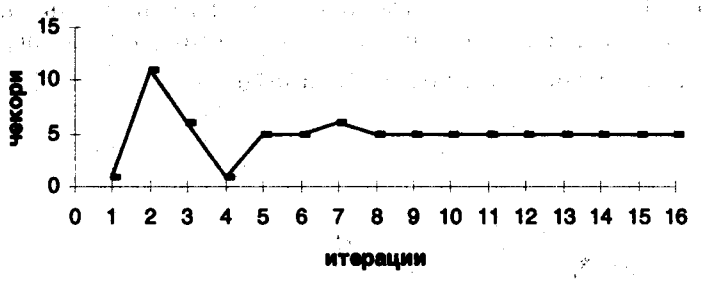
При почетна состојба 1 (сл. 4) со алгоритамот „во целта врати се" потребни се вкупно 35 чекори додека се најде едно решение (пат) во итерација 8, а со алгоритамот „во потцелта врати се" потребни се 29 чекори додека се најде едно решение во итерација 7.

При почетна состојба 11 (сл. 5) со алгоритамот „во целта врати се" потребни се вкупно 67 чекори додека се најде едно решение (пат) во 8 итерација, а со алгоритамот „во потцелта врати се" потребни се 56 чекори додека се најде едно решение во итерација 9.

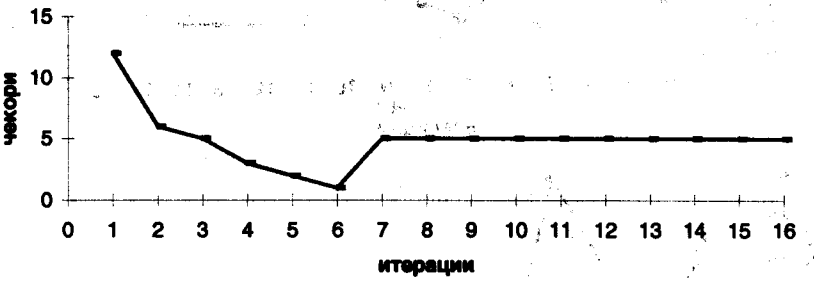
При почетна состојба 16 (сл. 6) со алгоритамот „во целта врати се" потребни се вкупно 44 чекори додека се најде едно решение

(пат) во итерација 8, а со алгоритамот „во потцелта врати се" потребни се 29 чекори додека се најде едно решение во итерација 7.

а)



б)



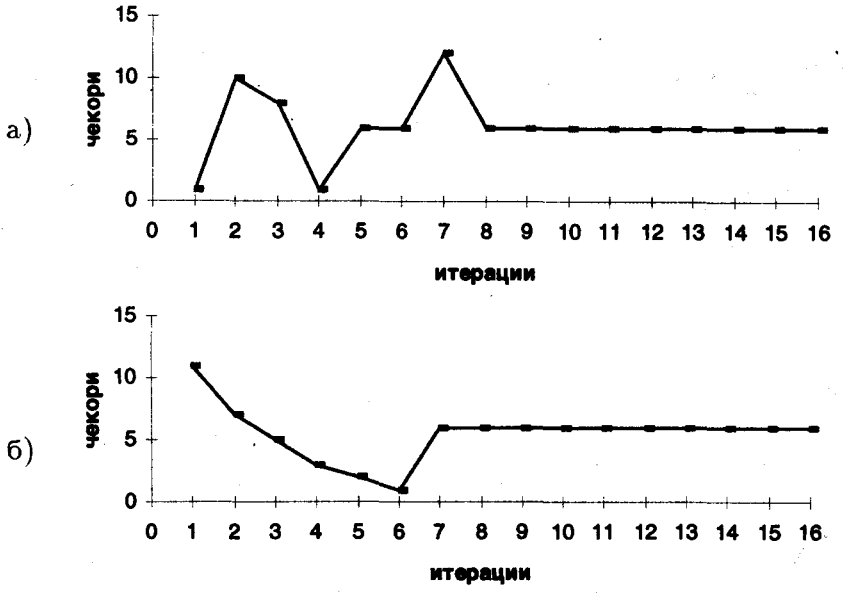
Сл. 4

а)

б)

Сл. 5

Со алгоритмот „од целта врати се" потребни се 800 репродуктивни периоди за да се добијат сите можни патишта, а со алгоритмот „од потцелта врати се" потребни се само 100 репродуктивни периоди за генерирање на сите патишта.



Сл. 6

Очигледно е дека „во потцелта врати се" е поопшта и поефикасна процедура од оригиналната „во целта врати се" САА процедура, и е многу едноставно решение како за детерминираните, така и за случајните околини.

Литература:

[1] Bellman R.: *Dynamic Programming*, Princeton University Press, 1957.

[2] Božinovski S.: A self-learning system using secondary reinforcement. In R. Trappl (ed.) *Cybernetics and Systems Research* p. 397-402, North Holland, 1982.

[3] Божиновски С.: *Вештачката интелигенција*, Гоцмар, Скопје, 1994.

[4] Bozinovski S.: *Consequence Driven Systems: Teaching, Learning, and Self-learning Agents*, Gocmar Press, Amherst, 1995.

[5] Kraus L.: *Programsko okruzenje DELPHI sa resenim zadacima*, Beograd 1996.

[6] Singh S., Norvig P., Cohn D.: *How to make software agents do the right thing: An Introduction to reinforcement learning*, Harlequin Inc., 1996.

[7] Sutton R., Barto A.: *Reinforcement Learning: An Introduction*, MIT Press, Cambrige, MA, 1998.

CONSEQUENCE PROGRAMMING: ALGORITHM "AT SUBGOAL-GO-BACK"

Silvana Petruševa * and Stevo Božinovski **

S u m m a r y

In this paper some experiments with the algorithm "at subgoal-go-back" are presented, for solving the problem for finding shortest path from all paths in an environment with n states. One is a starting state, one is a goal state and some states are unpleasant and they should be avoided. From each state actions can be taken which can lead to another state or to some obstacle. Agent based approach is used The Neural Network Crossbar Adaptive Array (NN-CAA) is chosen as agent architecture. The mathematical model of the agent and the algorithm of the program, described here, are implemented in DELPHI 1. The results of the "at goal-go-back" - algorithm and of "at subgoal -go-back" - algorithm are compared and it is presented that "at subgoal-go-back" - algorithm is more efficient.

* Katedra po matematika
Gradežen fakultét email: matematika@stobi.ga.ukim.edu.mk
R. MAKEDONIJA

** Institut za kompjuterska tehnika i informatika
Elektro-tehnički fakultet
1000 Skopje
R. MAKEDONIJA