**Домашна работа – Вештачка Интелигенција**

Трајче Проданов – 221164

1. ***Проблем со пребарување на простор на состојби***

Овој проблем опфаќа **k** број на Пакмани, кој се движат низ простор **МхN** и зад себе оставаат ѕид. Целта е секој од **k**-те Пакмани да стигнат до знаменцето кое е наменето за нив, притоа не смеат да стапнат на исто место каде има стапнато друг Пакман. Пакманите може во секој чекор да ги применуваат следните акции: **Горе, Долу, Лево, Десно** и **Стоп.** За минималната состојба имаме **k** пакмани, и табла со **MxN** број на полиња. Тоа може да се претстави како **(МxN)k .** Односно:

* k двојки од вредности ((x1, y1), (x2, y2) ... (xk, yk))

Сега да го видиме второто ограничување – „Пакманите оставаат ѕид на полето кое ќе стапнат“. Ова може да го претставиме како **MxN** логички променливи со вредност 0 или 1, во зависност од тоа дали има ѕид на тоа поле или не. Со тоа добиваме:

* **MxN** логички променливи за секое поле од лавиринтот претставени како матрица кои претставуваат дали на полето има настапнато пакман или не.

За максималниот број на состојби кои може да произлезат од минималната состојба може да се пресмета со следната формула:

* **2(MxN) x (МxN)k**

Следно имаме фактор на разгранување – кој е всушност бројот на можни состојби кој може да произлезат од дадена состојба. За да го пресметаме факторот на разгранување, треба да земеме во предвид сите акции кој пакманот може да ги преземе од дадена состојба, но тука треба да бидат вклучени и акциите кои не се возможни од истата таа состојба – пример ако има десно од него ѕид, тогаш акцијата десно не може да ја преземе.

Еве како би го пресметале максималниот фактор на разгранување:

* Почнуваме од тоа дека секој Пакман има 5 можни акции (горе, долу, лево, десно, стоп)
* За секој Пакман, треба се земе во предвид дали било која од овие акции е возможна во дадената состојба, односно дали при преземање на акцијата Пакманот ќе влезе во невозможна состојба.

На пример, ако нема ѕидови или други пакмани кои го блокираат патот на дадениот Пакмен, секој Пакмен може да ги преземе сите 5 акции, но ако има препреки, тогаш тој број се намалува.

Следно ќе ги претставиме почетната и крајната состојба на проблемот за сликите дадени во проблемот.

Иницијалната состојба може да ја претставиме на следниот начин

* (((0, 4), (0, 1)), ((0,0,0,0,0,0,0) ..., (1, 0, 0, 0,0,0,0) ..., (1, 0, 0, 0, 0, 0, 0), ...)) – каде првиот tuple е позициите на пакманот, а вториот е матрица MxN, односно tuple со 5 други tuples кои во себе содржат 7 елементи, односно 1 или 0 за тоа каде има ѕид (почнувајќи од доле лево). Тука не ги вклучуваме ѕидовите што се поставени на почетокот на играта, и истите се непотребни бидејќи би имало проверка која спречува илегални акции. Но, исто така би можело и при дефинирање на иницијалната состојба, местото каде има ѕидови да се пополни со 1.
* Бидејќи знаменцата и другите ѕидови се статични и не се движат, нивните позиции се чуваат надвор од иницијалната состојба.

За крајната состојба имаме на сликата имаме:

* (((4,4), (4,6)), ((1,1,0,1,0,0,0), (0,1,0,1,1,1,1), (0,1,1,1,1,1,1), (1,1,1,1,1,1,1), (1,1,1,1,1,1,1)) – каде (4,4) и (4,6) се крајните позиции на двата Пакмани соодветно

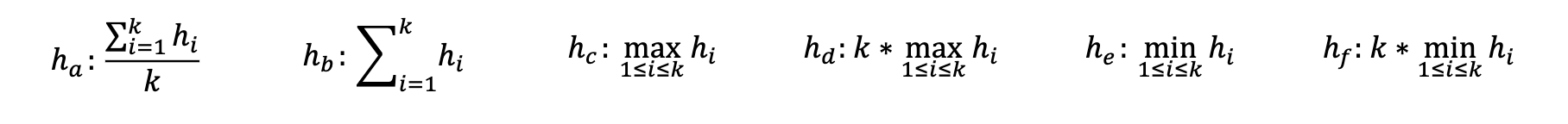
Ако проблемот се наоѓа во можната состојба по првиот чекор, претставена со втората слика:

A screenshot of a game

Description automatically generated

* Тогаш можните акции за првиот Пакман се „десно, долу“, а за вториот, тие се само „десно“

Една можна хевристика за дадената слика може да биде менхетан растојанието за секој пакман до крајната позиција. Менхетан растојанието се пресметува како сума од абсолутните разлики во хоризонтална и вертикална позиција меѓу две точки. Но бидејќи имаме k Пакмани, тогаш за хевристика може да ја земаме сумата од сите менхетан растојанија за k-те Пакмани. Таа хевристика би била допустлива бидејќи по правило, хевристиката треба да е помала од вистинската цена да се стигне до целта.



Кои од следните хевристики се допустливи?

За да го пресметаме тоа, треба да се осигуриме дека секоја од овие хевристики е помала од вистинската вредност. Па така имаме:

ha : оваа хевристика е допустлива бидејќи го зема просекот на сите вредности на hi, а просекот никогаш не е поголем од вистинската максималната вредност.

hb: оваа хевристика е исто допустлива бидејќи ги сумира вредностите, што никогаш не го преценува трошокот

hc: оваа хевристика исто така е допустлива бидејќи никој еден h не е поголем од вистинската вредност

hd: оваа евристика не е допустлива, бидејќи може да прецени трошок ако k пати максималното hi​ е поголемо од вистинскиот трошок.

he: исто така може да биде допустлива, затоа што не преценува трошок, ако ниту едно h не е поголемо од вистинската вредност

hf: оваа хевристика не е допустлива, бидејќи може да прецени трошок ако k пати минималното hi​ е поголемо од вистинскиот трошок.

За да го избереме најдобриот алгоритам за проблемот со Пакман играчите, треба да ги анализираме нивните карактеристики и како се вклопуваат со карактеристиките на проблемот.

DFS (Depth-First Search) - гарантира најбрзо доаѓање до целта само ако целта се наоѓа на горното ниво на дрвото на пребарување. Во спротивно, DFS не е оптимален бидејќи може да заврши во бесконечен циклус, ако пребарувачката патека влегува во циклус. За проблемот со Пакман играчите, DFS може да не биде најдобар избор бидејќи не гарантира оптимално решение и може да се најде во циклуси или да оди во погрешна насока.

BFS (Breadth-First Search) - BFS гарантира најбрзо доаѓање до целта ако сите ребра имаат иста тежина. Тоа значи дека ако сите стапки за движење на Пакманите имаат иста тежина, тогаш BFS е оптимален. За проблемот со Пакман, каде што стапките имаат иста тежина, BFS би бил најдобар избор бидејќи гарантира најкратка патека до целта.

UCS (Uniform Cost Search) - UCS е оптимален за проблеми каде што стапките имаат различни тежини. Тоа значи дека ако има препреки во мрежата на лавиринтот и стапките имаат различни тежини, UCS гарантира оптимално решение. За проблемот со Пакман играчите, каде што треба да се земе предвид оставањето на ѕидови како препреки, UCS може да биде добар избор, но ќе има некои трошоци за пресметување на различните стапки.

A\* (A-star) - е оптимален алгоритам за проблеми каде што има информација за трошоците на стапките и информација за раздалеченоста до целта. Тоа го прави најдобар избор за проблемот со Пакман играчите, бидејќи може да се користи хевристиката за да се предвиди најбрзото доаѓање до целта, а чувањето на информацијата за трошоците на стапките гарантира оптимално решение.

Според карактеристиките на проблемот, алгоритамот A\* би бил најдобриот избор. Тоа е бидејќи имаме информација за трошоците на стапките (изградбата на ѕидови) и можеме да користиме хевристика за предвидување на најкратката патека до целта. Ова го прави A\* оптималниот алгоритам за проблемот со Пакман играчите.

1. ***Проблеми кои задоволуваат услови***

Проблемот на крстозборот може да се дефинира како проблем на исполнување услови (CSP) на следниот начин:

**Променливи:**

За секоја празна почетна позиција во крстозборот, имаме по една променлива која претставува местото каде треба да се вметне збор. На пример, ако крстозборот е со димензии 5x5 и имаме 8 почетни позиции, ќе имаме 8 променливи, по една за секоја од 8-те празни почетни позиции.

**Домени:**

Домените на променливите се составени од зборови од дадениот лексикон. За секоја променлива, доменот се состои од сите зборови кои се можно да се вметнат на соодветната позиција. Во дадениот проблем зборовите се следните:

„set, cyber, pod, myman, faked, cross, elite, ebike, met, rod, naked, new, amino, net, pakman, sudoku“

**Услови (ограничувања):**

За секоја две пресекнати променливи (зборови кои се пресекуваат), ги имаме следниве услови:

* Буквите кои се пресекуваат во двата збора треба да бидат исти.
* Почетните позиции за зборовите кои треба да се вметнат во крстозборот се означени со броеви:
  + Хоризонтално: 1, 4, 6, 7 и 8.
  + Вертикално: 1, 2, 3, 4 и 5.

**Цел:**

Целта на проблемот е да се најдат валидни зборови од лексиконот што може да се вметнат во крстозборот, при што се задоволуваат сите ограничувања.

Проблемот на крстозборот се решава со постепено пристапување на решението преку измена на домените на променливите и примена на условите за да се задоволат сите ограничувања. Започнувајќи со унарните услови, домените на променливите се анализираат и се избираат вредности кои не ги нарушуваат унарните ограничувања. Откако ќе се исполнат унарните услови, се продолжува со анализирање на бинарните ограничувања помеѓу пресекнатите променливи, со цел да се најдат валидни комбинации на зборови што се вметнуваат во крстозборот.

Графот на ограничувања ги прикажува ограничувањата меѓу променливите (полињата од таблата) кои мора да бидат задоволени за да се најде валидно решение. Во овој случај, ограничувањата се организирани околу преклопувањата на зборовите во таблата.

За пример на граф на ограничувања може да се земе следното:

* Променливите, односно квадратната табла со почетните полиња означени со соодветни броеви
* Ребра, секој збор има прво поле и насока (хоризонтално или вертикално). Ребро се додава помеѓу две променливи (полиња) ако овие полиња треба да бидат исти.

A diagram of a network

Description automatically generated

Започнуваме со унарните услови, на пример:

* поле со број 5 вертикално, не може да собере збор поголем од 3 букви. Што значи ги отстрануваме сите зборови поголеми од 3 букви
* поле со број 1 хоризонтално и вертикално, треба да земеме во предвид еден од зборовите кои содржат 3 букви за хоризонтално и 5 букви кои почнуваат со тие почетни букви од зборовите со 3 букви. Што значи преостанати почетни букви од зборовите со 5 букви се „S, P, M, R, N“. Истото може да се направи и за 4
* За 2, 3, 6, 7 – остануваат сите
* Исто така, за 8, преостануваат само тие со 3 букви
* Зборовите со должина поголема од 5 ги отстрануваме од домените на сите променливи

Со црвено се означени вредностите отстранети по примена на унарните услови.

При **arc consistency enforcing** меѓу секој од ребрата може да го забележиме следното:

* **(1 – 2)** – бидејќи ги знаеме вредностите кои може да се постават хоризонтално, тогаш втората буква од доделената вредност на 1 хоризонтално, ќе биде почетната буква на 2. Можни почетни букви за 2 се „e, o“, па кај 2 ги отстрануваме сите што не почнуваат со „е“ и „о“.
* **(1 – 3)** – истото може да го примениме и овде. Каде ќе ја гледаме третата буква од зборот. Може да ги отстраниме зборовите кои не почнуваат на следните букви: „t, d, w“. Од тука може да заклучиме дека проблемот нема решение, но да продолжиме и за останатите.
* **(1 – 4)** – слично на првите две точки, втората буква на 4 мора да е една од следните букви: „y, a, u, e, o“.
* **(1 – 5)** – нема ребро, така да не се прави ништо овде.
* **(1 – 6)** – како и претходните, втората буква мора да биде една од третата буква на зборовите со 5 букви од доменот на 1. Тие се: „t, d, w, m, k“
* **(1 – 7)** – „а, е“ како можна втора буква од 7, но 1 може да биде со должина 3, па не отстрануваме вредности
* **(1 – 8)** – последната буква од 1 мора да е иста со почетната од 8, но како и претходниот пример, 1 може да биде со должина 3, па и овде не отстрануваме
* **(2 – 1)** – не правиме ништо бидејќи нема преостанатиот домен е различен
* **(2** **– 3)** – 2 и 3 немаат раб, но немаат исти вредности преостанато во доменот.
* **(2 – 4,5,6,8)** – не се случува ништо
* **(2** **– 7)** – 3тата буква од 7 мора да биде „i“
* **(3** **– сите)** – 3 нема преостанати вредности во доменот
* **(4 – 1)** – втората буква на 1 мора да е еднаква со втората буква на 4, па така преостануваат зборовите што имаат барем една иста втора буква
* **(5 – 4)** – исто како и 1 – 8, 4 може да има вредност со должина 3, па не се отстранува ништо
* **(6 – 7)** – немаат ребро
* **(7 – 2)** – вредностите кои не интересираат се “elite” и “ebike”, што значи третата буква на 7 треба да биде „t“ или „k“. Бидејќи „amino“ ја нема таа 3та буква, се отстранува со arc consistency enforcing. Овде исто ќе видиме дека 7 ќе нема преостанати вредности по исполнување на 2. Што значи еве уште еден доказ дека проблемот ќе нема решение.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Set |  | X | X |  |  | X | X |  |
| Cyber | X | X | X | X | X | X | X | X |
| Pod |  | X | X |  |  | X | X |  |
| Myman |  | X | X |  | X | X | X | X |
| Faked | X | X | X | X | X | X | X | X |
| Elite | X |  | X | X | X | X |  | X |
| Ebike | X |  | X | X | X | X |  | X |
| Met |  | X | X |  |  | X | X |  |
| Rod |  | X | X |  |  | X | X |  |
| Naked |  | X | X |  | X | X | X | X |
| New |  | X | X |  |  | X | X |  |
| Amino | X | X | X | X | X |  | X | X |
| Net |  | X | X |  |  | X | X |  |
| Pakman | X | X | X | X | X | X | X | X |
| sudoku | X | X | X | X | X | X | X | X |
|  |  |  |  |  |  |  |  |  |

Иако проблемот нема решение поради отстранувањето на сите вредности од променливата **3** по примена на конзистентност на ребрата, нека илустрираме како би се пристапило кон наоѓање на решение користејќи ги дадените хевристики и алгоритамот за враќање наназад со проверка напред.

1. **Minimum Remaining Values (MRV):** Приоритетно ќе ја избереме променливата со најмал број на преостанати вредности во нејзиниот домен. Во овој случај, откако ќе се примени конзистентност на ребрата, променливата 3 нема преостанати вредности, затоа ќе треба да избереме друга променлива.
2. **Least Constraining Value (LCV):** По изборот на променлива за доделување на вредност, приоритетно ќе избереме вредност која ќе ги исклучи најмалку вредности во домените на другите променливи. Ова помага да се намали шансата за конфликти подоцна во пребарувањето.
3. **Backtracking with forward checking:** Итеративно доделуваме вредности на променливите, применуваме проверка напред за да ги скратиме домените на другите променливи врз основа на доделата направена. Ако доменот на променливата стане празен, се враќаме назад на најскорешната додела и се обидуваме со друга вредност.

Според сценариото каде променливата **3** нема преостанати вредности, алгоритамот за враќање назад би се вратил на најскорешната додела и обидел да го проба другата вредност за променливата пред променливата **3**. Доколку не се пронајде решение, алгоритамот би продолжил да се враќа назад сè додека не се исцрпат сите можности.

Иако овој конкретен проблем нема решение, пристапот опишан горе го демонстрира како би се користеле MRV, LCV и алгоритамот за враќање назад со проверка напред за пребарување на решение во проблем на задоволување на ограничувања.