Tema 1 - Inteligenta Artificiala

Andreea Nica (andreeanica16)

README

November 12, 2021

Problema de cautare in spatiul starilor

Pentru a modela problema Nonogramelor si a putea aplica algoritmii de cautare in spatiul starilor, a fost necesara definirea a ce inseamna o stare si cum se reprezinta ea, precum si care sunt posibilii vecini dintr-o stare. De aceasta reprezentare depind in intregime abordarile pentru fiecare algoritm in parte. Mai mult, reprezentari si considerente diferite au dus la comportamente diferite ale diferitililor algoritmi. Am ales aceasta reprezentare intrucat ea sintetiza cat mai complet si corect starea de la un moment dat a jocului. Mai mult, datorita eficientei sale, rularea tuturor algoritmilor este posibila pentru toate testele (toti algoritmii se termina intr-un timp rezonabil), astfel ca observatiile asupra diferentelor intre diversi algoritmi au fost complete.

- Starea este reprezentata ca o pereche ***(CONFIGURATION, ROW)*** reprezentant cum arata configuratia jocului, dupa ce au fost alese primele *ROW* randuri.
- Vecinii unui nod sunt reprezentati de toate configuratiile posibile viitoare dupa ce a fost ales randul urmator in toate modurile posibile. Practic, pentru generarea tuturor vecinilor unui nod alipim la configuratia curenta toate posibililele configuratii pentru randul urmator. Mai mult, daca o configuratie este explicit invalida, atunci ea nu va fi luata ca vecin al configuratiei curente (exact la fel ca atunci cand avem un obstacol, pozitia corespunzatoare obstacolului nu este considerata)
- Pentru a eficientiza procesul de generare a vecinilor, care include generarea pe rand a tututor posibilitatilor pentru o linie am folosit un algoritm de programare dinamica, pentru a genera linii valide. Astfel, algoritmul de generare a tuturor posibilitatilor pentru o linie ia fiecare componenta si o alipeste tuturor posibilitatilor pentru celelalte componente ramase. Algoritmul are o complexitate *O(K*N)* unde *K* este numarul maxim de componente, iar *N* este dimensiunea liniei. Liniile generate vor fi salvate in *known-configurations*, pentru a evita generarea de doua ori a tuturor posibilitatilor pentru o anumita linie.
- Pentru algoritmi nu este necesar retinerea nodurilor vizitate anterior, pentru ca, in general, nu se poate reveni la o stare parcursa deja, intrucat avansarea se face pe linii inainte.

1 Breadth First Search

Datorita modului de reprezentare a starilor si generare a vecinilor, algoritmul de BFS poate fi rulat pana la capat pentru toate testele, obtinand urmatoarele rezultate:

Rezultate Breadth First Search			
Test	Nodes Expanded	Nodes Generated	Time
Test 1	8	11	0.0012
Test 2	5	6	0.0010
Test 3	42	43	0.0471
Test 4	540	545	2.8964
Test 5	26163	26164	129.9147
Test 6	153	154	17.9270

1. ***Dezavantaje:*** Cu toate ca algoritmul gaseste solutia corecta, cautarea neinformata este o alegere extrem de ineficienta: pentru fieacare test se observa expandarea unui numar extrem de mare de noduri. Ca urmare, si timpul de rulare este unul extrem de ridicat. Intrucat solutia este mereu pe ultimul nivel, aproape toate nodurile generate sunt expandate, ceea ce duce la un numar extrem de mare de noduri parcurse pentru gasirea solutiei, implicand astfel un timp marit pentru determinarea solutiei.

   Este clar ca in problema Nonogramelor ar putea exista niste hinturi care sa ne ajute sa ne directionam catre gasirea problemei mai rapid, insa BFS nu tine cont de acestea. Mai mult, pentru gasirea solutiei este necesar parcurgerea aproximativa a intregului spatiu(deoarece solutia se afla pe ultimul nivel mereu, ceea ce este foarte ineficient, asa cum se vede si dim timpul ridicat de cautare).

2. ***Avantaje:*** Algoritmul BFS este complet, chiar si pentru o problema infinita, ceea ce inseamna ca un avantaj al acestuia este certitudinea ca vom gasi solutia problemei. Mai mult, un alt avantaj al algoritmului BFS este ca e foarte straight-forward de modelat pentru problema nonogramelor si de implementat.

2 Depth First Search

La fel ca la BFS, algoritmul DFS poate fi rulat pana la capat pentru toate testele, datorita modului de reprezenatare a starilor si generare a vecinilor. Rezultatele sunt urmatoare:

Rezultate Depth First Search			
Test	Nodes Expanded	Nodes Generated	Time
Test 1	7	11	0.0010
Test 2	5	6	0.0007
Test 3	42	43	0.0450
Test 4	527	534	2.2055
Test 5	10329	10427	108.6512
Test 6	110	126	13.6241

1. ***Avantaje:*** Se observa ca algoritmul DFS este mult mai eficient decat altgoritmul BFS, pentru aceasta problema (pentru testul 5 numarul starilor se injumatateste). Motivul pentru care apare aceasta diferenta extrem de observabila pentru toate testele se datoreaza, in principal, faptului ca solutia problemei se afla mereu la aceeasi adancime fata de radacina (adancimea solutiei este ultimul rand) si nu exista stari care sa fie la o adancime mai mare de atat. Mai mult, nu exista stari finale care sa fie la o adancime mai mica. De aceea, in loc sa se parcurga pe rand toate starile de o anumita adancime, algoritmul DFS avanseaza mai rapid catre solutie, mergand in adancime, atingand mai repede ultimul rand, si astfel gaseste solutia mult mai rapid, fiind nevoit sa parcurga mai putine noduri. Acest lucru este oglindit atat de numarul de noduri mai mic expandate si generate, cat si timpului redus.

2. ***Dezavantaje:*** Desi algorimtul DFS are problema ca nu este complet, pentru aceasta problema si reprezentarea aleasa, nu ar trebui sa fie un motiv de ingrijorare, intrucat adancimea maxima este limitata la numarul de randuri. Mai mult, solutia se afla sigur la adancimea maxima si nu poate fi determinata mai rapid. Cu toate astea, un dezavantaj identificat este ca algoritmul DFS genereaza mai multe noduri pe care nu la va mai expanda ulterior, spre deosebire de BFS care expandeaza aproape toate nodurile generate. Acest lucru releva intocmai faptul ca nu este necesara parcurgerea in totalitate a spatiului pentru gasirea solutiei.

3 Iterative Deepening Search

Rezultatele rularii algoritmului Iterative Deepening Search sunt urmatoarele:

Rezultate Iterative Deepening Search			
Test	Nodes Expanded	Nodes Generated	Time
Test 1	16	28	0.0015
Test 2	14	18	0.0008
Test 3	305	326	0.0669
Test 4	5026	5169	14.1361
Test 5	33551	34502	168.0136
Test 6	1724	2109	101.4838

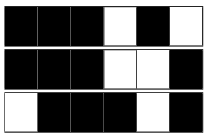
1. ***Dezavantaje:*** Deoarece, datorita modului de reprezentare a starilor si generarea vecinilor, solutia se afla intotdeauna la adancime maxima, iar adancimea maxima este egala cu numarul de linii, algoritmul Iterative Deepening Search nu aduce nici un beneficiu. Mai mult, el chiar este extrem de ineficient, pentru ca genereaza mult mai multe stari decat DFS si dureaza si mult mai mult timp. Solutie se afla sigur la adancimea = numarul de linii, asa ca nu are sens sa incercam gasirea ei la adancimi mai mici, pentru ca acest demers se va finaliza cu insucces. Astfel, toate rularile pentru *adancime <adancime maxima* vor fi total inutile.
2. ***Avantaje:*** Daca solutia nu se afla la adancimea maxima, atunci Iterative deepening putea sa reduca spatiul de cautare si sa nu mai exploreze nodurile care au trecut de o adancime limita, evitand riscul ca algoritmul sa se afunde prea tare pe ramura respectiva si sa nu gaseasca solutia. Cu toate acestea, pentru ca solutia se afla mereu la adancime maxima, algoritmul Iterative Deeepening Search nu aduce nici un avantaj concret.

4 A* Search

Pentru a reusi sa utilizam cunostintele euristice pentru cresterea eficientei si alegerea celui mai bun nod de expandat in cursul cautarii a fost nevoie de definirea functiei *f(nod), g(nod)* si *h(nod)*, mai exact de intelegere a notiunii de cost pentru problema Nonogramelor.

Pentru aceasta, am facut urmatoarea observatie: atunci cand incepem prelucrarea unei linii, indiferent de ce configuratie alegem, anumite celule vor fi sigur colorate Negru sau Alb.

De exemplu, pentru o linie de lungime *6* cu configuratiiele $\{3, 1\}$, variantele posibile pentru linia respectiva sunt:



Observam ca celulele 1 si 2 sunt Negre in toate configuratiile, restul pot fi negre sau albe. Aceste celule care au aceeasi culoare in toate configuratiile corecte vor fi definite ca celule ***SIGURE***. Restul de celule (0, 3, 4, 5) trebuie ***GHICITE*** (pentru fiecare va trebui sa alegem o culoare, negru sau alb).

Pe baza acestei observatii vom defini functia *g(nod)* astfel:

g(nod) = Numarul de celule ghicite pana acum pentru a ajunge in configuratia curenta

Evident, cu cat numarul de celule pe care trebuie sa le ghicim este mai mic, cu atat configuratia este mai buna, astfel ca numarul de celulele ***GHICITE*** poate reprezenta o metrica buna pentru cost in problema Nonogramelor.

Avand definita functia *g(nod)* si notiunea de cost pentru problema Nonogramelor, pentru a aplica A* trebuie definita functia euristica *h(nod)*.

Euristica 1

Pentru euristica 1 am folosit urmatoarea functie:

h1 = Numarul de celule care mai trebuies ghicite daca presupun ca doar coloanele corespunzatoare unei celule GHICITE din linia curenta mai poate avea celule care trebuiesc GHICITE, restul de coloane fiind sigure pentru toate campurile

Functia este **admisibila**, deoarece un element SIGUR pe o linie nu imi poate garanta ca nu mai exista elemente de GHICIT pe coloana respectiva. Astfel, toate elementele care ar mai fi de ghicit pe o coloana care contine un element SIGUR de pe linia curenta sunt ignorate, fiind luate in considerare doar elementele de GHICIT de pe coloanele cu elemente de GHICIT din linia curenta. Cum numarul de celule de GHICIT produs de functia *h1(nod)* este mai mic decat numarul real de celule de GHICIT si cum numarul de celule de ghicit pentru configuratia finala este 0 (in configuratia finala toate celulele au fost completate, deci nu mai trebuie ghicit nimic) demonstreaza ca *h1(nod)* este **admisibila**

1. ***Avantaje:*** Cautarea informata are avantajul ca solutia este, in general, gasita mult mai rapid, intrucat alegerea starii urmatoare de expandat se bazeaza pe o euristica, pe anumite considerente. Se observa, in special pentru testul 4, ca nodurile generate si expandate se reduc si de zeci de ori.
2. ***Dezvantaje:*** Eficienta algoritmul A* este dependent de euristica folosita si de cat de bine poate sa surprinda aceasta si sa aproximeze starea nodului curent. Cu toate ca pentru anumite configuratii euristica nu este cea mai informata, observam ca pentru teste precum 4 si 6 se obtin beneficii majore prin folosirea algoritmului A*.

Rezultatele rularii sunt spectaculoase, asa cum se poate vedea si din:

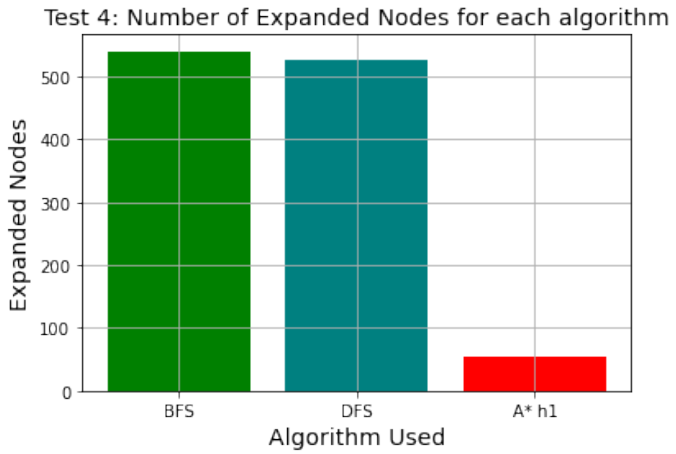


Figure 1: Test 4: Comparare numari noduri expandate pentru BFS, DFS si A* folosinf euristica h1.

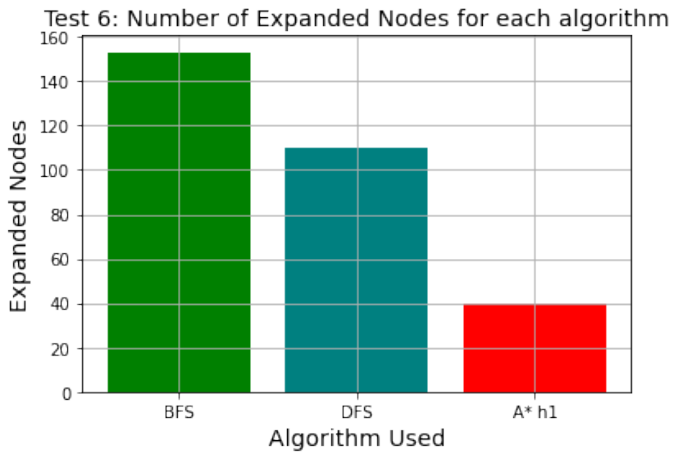


Figure 2: Test 6: Comparare numari noduri expandate pentru BFS, DFS si A* folosinf euristica h1.

Rezultate A* Search cu h1			
Test	Nodes Expanded	Nodes Generated	Time
Test 1	8	11	0.0031
Test 2	5	6	0.0009
Test 3	13	27	0.0321
Test 4	53	92	1.1020
Test 5	17000	17040	125.4402
Test 6	40	75	5.3931

Euristica 2

Pentru euristica 2 am folosit urmatoarea functie:

h2 = Numarul de celule care mai trebuiesc ghicite daca presupunem ca toate celulele negre de pe o coloana au fost ghicite corect

Functia este **admisibila**, deoarece celulele negre de pe o coloana nu sunt garantat ghicite, deci voi produce un numar mai mic decat numarul real de celule care au ramas de ghicit. Cum numarul de celule de GHICIT produs de functia *h2(nod)* este mai mic decat numarul real de celule de GHICIT si cum numarul de celule de ghicit pentru configuratia finala este 0 (in configuratia finala toate celulele au fost completate, deci nu mai trebuie ghicit nimic) demonstreaza ca *h2(nod)* este **admisibila**

1. ***Avantaje:*** Observam ca, si in cazul acestei euristici, ca si pentru h1, se obtin gasiri mai rapide ale solutiei, reducand foarte mult nodurile generate si expandate (mai ales pentru Test 4 si Test 6)
2. ***Dezvantaje:*** Euristica h2 este mai putin informata decat euristica h1 pentru ca produce mai multe stari si are si un timp de executie mai mare. De aceea, observam cat de dependenta este performanta algoritmului A* de euristica aleasa.

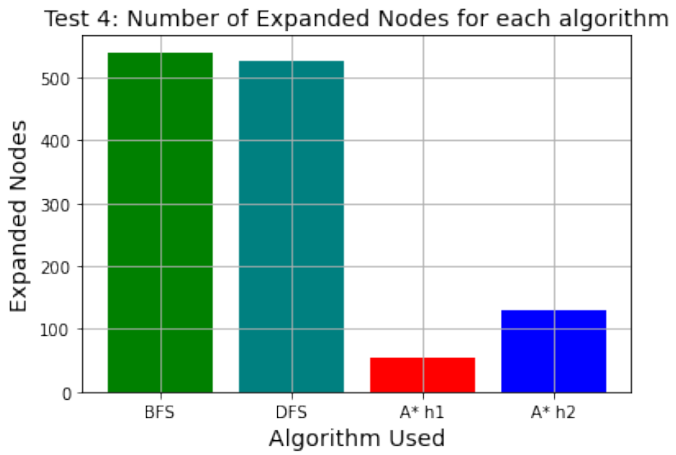


Figure 3: Test 4: Comparare numari noduri expandate pentru BFS, DFS si A* folosinf euristica h1 si h2.

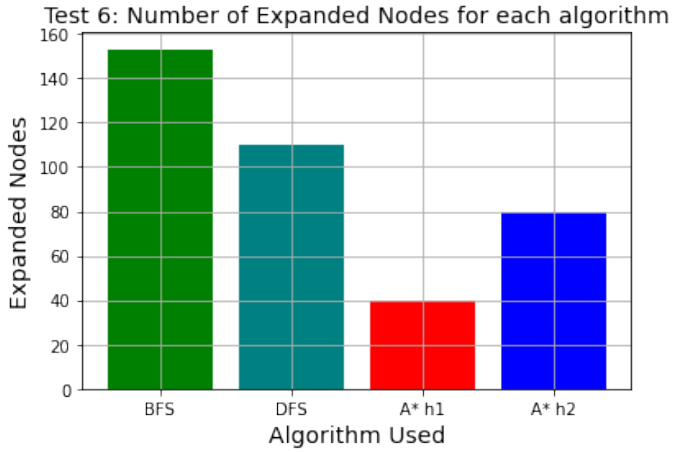


Figure 4: Test 6: Comparare numari noduri expandate pentru BFS, DFS si A* folosinf euristica h1 si h2.

Rezultate A* Search cu h2			
Test	Nodes Expanded	Nodes Generated	Time
Test 1	8	11	0.0028
Test 2	5	6	0.0009
Test 3	10	23	0.0143
Test 4	129	196	0.8141
Test 5	19172	19431	91.0836
Test 6	80	114	6.8600

Problema satisfacerii restrictiilor

5 Maintaining Arc Consistency

Pentru problema satisfacerii restrictiilor, fiecare linie si fiecare coloana reprezinta cate o variabila. Domeniul fiecarei variabile reprezinta configuratiile valide care respecta restrictiile pe linia sau coloana respectiva.

Exista arce doar intre o linie si o coloana sau intre o coloana si o linie. Restrictia dintre o linia si o coloana verifica ca la intersectia liniei cu coloana sa se afle acelasi tip de celula. Astfel, problema este modelata complet prin folosirea de arce intre o linie si o coloana sau intre o coloana si linie.

Rezultatele obtinute de algoritm sunt:

Rezultate Maintaining Arc Consistency			
Test	Nodes Expanded	Nodes Generated	Time
Test 1	11	11	0.0061
Test 2	11	11	0.0062
Test 3	21	21	0.0555
Test 4	31	31	0.4621
Test 5	36	36	0.7792
Test 6	51	51	2.8167

1. ***Avantaje:*** Algorimtul MAC este de departe cel mai eficient pentru problema Nonogramelor. Acest lucru este extrem de vizibil prin timpul extrem de mic obtinut si numarul foarte mic de noduri generate si expandate. Graful de restrictii este arc-consistent, iar simpla rulare a realizarii arc-consistentei aproape ca rezolva problema (mai raman foarte putine variabile neinstantiate dupa rularea MAC initiala). Astfel, se obtine un timp extrem de bun pentru toate testele, inclusiv pentru Testul 5. Mai mult, problema Nonogramelor este extrem de usor de modelat pentru CSP, fiind o abordare foarte naturala, fireasca si usor de implementat.
2. ***Dezvantaje:*** Desi complexitatea temporala a BackTracking ar putea fi una ridicata (exponentiala), prin folosirea MAC se reduce extrem de mult domeniul fiecarei variabile. Pe problema Nonogramelor si pe testele propuse, domeniul se reduce aproape pentru toate variabile la solutie, ramanand un graf cu un brach factor extrem de mic, astfel facand problema Nonogramelor extrem de eficienta.

6 Bonus

Pentru bonus am ales sa implementez utilizarea euristicilor in rezolvarea problemei satisfacerii restrictiilor: ordonarea variabilelor in vederea atribuirii.

Am implementat Minimum remaining value, la fiecare pas alegand variabila cu cele mai putine valori legale.

Deoarece MAC reduce inca de la inceput pentru mai toate variabilele domeniul la o singula valoare, Minimum Remaining Value nu eficientizeaza prea tare rularea (pentru ca programul este deja extrem de eficient). Daca ar exista mai multa variabile care sa aiba diverse valori diferite in domeniu, atunci aceasta strategie ar imbunatati major performantele algoritmului. Cum insa mai toate domeniile sunt formate dintr-o singura valoare, rezultatele nu sunt atat de uluitoare pe cat ne-am astepta.

Rezultate Minimum Remaining Value			
Test	Nodes Expanded	Nodes Generated	Time
Test 1	11	11	0.0061
Test 2	11	11	0.0015
Test 3	21	21	0.2047
Test 4	31	31	0.4621
Test 5	36	36	0.4630
Test 6	51	51	2.26871