Stiuca Roxana-Elena
343C4

Tema 1 – Strategii de cautare

Cautari neinformate

- **Starea** problemei: un array bidimensional, reprezentand starea tablei ce a fost completata pana la acel nivel.
- **Starea initiala**: [] – o tabla goala.
- **Tranzitiile** se realizeaza prin completarea urmatorului rand, in toate felurile posibile avand in vedere specificatiile despre colorarea randului respectiv. In felul acesta, constrangerile de rand sunt adevarate din felul in care construim starile succesor. Constrangerile de coloana sunt verificate doar la final, dupa ce am completat intreaga tabla.
- **Observatie:** am putea verifica constrangerile de coloana pentru fiecare nod generat, a.i. sa taiem mult din nodurile care nu vor duce la solutie. Am incercat si aceasta varianta si este extrem de rapida (40s pentru testul 6, fata de variantele care nu faceau asta pentru care de la testul 3 incolo dureaza cateva minute). Am ales insa sa comentez aceasta schimbare din cod pentru ca aducea prea mult a “cautare informata”.

BFS

- Foloseste o coada. Initial coada contine starea initiala. La fiecare pas, se extrage un element, sunt generate nodurile succesor. Daca un nod succesor este solutie, aceasta este intoarsa. Altfel, nodul este adaugat in coada si algoritmul continua.
- **Avantaje**:
    - este o strategie completa, care garanteaza gasirea solutiei daca aceasta exista.
- **Dezavantaje/limitari**:
    - un beneficiu cunoscut pentru BFS este acela ca el exploreaza toate caile pas cu pas, a.i. daca exista o cale radacina-solutie mai scurta decat altele, BFS o va gasi pe aceea prima. Acest beneficiu insa este irelevant prin felul in sunt descrise tranzitiile, pentru ca, daca exista o solutie, ea se va afla mereu la o distanta fixa de radacina (numar de linii ale tablei).

DFS

- Idem BFS, doar ca foloseste o stiva.
- **Avantaje**:
    - depinde de restrictiile date ca input, dar se poate comporta foarte bine pentru atunci cand “nimereste” ca prima cale exploatata sa fie si cea care duce la solutie. Astfel poate genera mai putine noduri decat BFS.
- **Dezavantaje/limitari:**
    - poate fi o strategie incompleta, atunci cand lungimea cailor prin arbore este variata sau cand nu avem o conditie clara de cand am ajuns la o frunza. Aici nu e cazul prin felul in care am descris problema.

Iterative Deepening

- Foloseste implementarea pentru DFS, pe care o apeleaza cu un nivel maxim acceptat ce se incrementeaza dupa fiecare apel care nu a intors solutia.
- **Avantaje:**
    - in problemele unde nivelul la care este gasita solutia poate varia foarte mult, este mai potrivit decat un DFS simplu, pentru ca asigura gasirea solutiei si gaseste cea mai scurta cale la o solutie.
- **Dezavantaje:**
    - Pentru felul in care am decis sa descriem problema (stari si tranzitii), acest algoritm nu este adecvat. Solutia se va gasi intotdeauna la un anume nivel (egal cu inaltimea tablei), deci apelurile cu max_depth mai mici de atat vor esua intotdeauna.

A*

- La fel ca toate reprezentarile problemei de pana acum, starile sunt completari partiale ale tablei, pana la un anume rand, iar tranzitiile se fac prin completarea unui nou rand. Mai mult, dintre starile viitor, le eliminam pe acelea care contrazic constrangerile pe coloana.
- Dintre functiile algoritmului A*, nu am folosit g pentru ca functia euristica nu inceasta sa gaseasca cost minim (costul, in sensul de lungimea caii radacina-solutie este oricum constant), ci incearca sa ajunga la o solutie si asociaza o valoare h cat mai mare acelor table partiale cu probabilitate mica in a conduce la o solutie.
- Nu exista posibilitatea de a genera stari pe care sa le avem deja in coada sau pe care sa le fi explorat deja, de aceea acele verificari nu au fost implementate.
- **Avantaje:**
    - pentru functii euristice admisibile, foarte bune, poate rezolva problema foarte repede.
    - se foloseste si de intuitia umana in rezolvarea Nonogramelor, prin euristica.
- **Dezavantaje:**
    - nu am reusit sa gasesc nicio functie euristica care sa functioneze bine pentru toate cele 6 teste.
    - functia euristica aleasa poate determina ca algoritmul sa se comporte foarte bine pentru subsetul de input-uri targetat, dar sa se comporte foarte prost pentru ceva mai aleator si deci comportamentul algoritmului pentru un input oarecare este greu de decis.

Euristica 1

- se bazeaza pe observatia ca Nonogramele modeleaza imagini alb-negru. In aceste imagini, este comun sa existe forme delimitate de linii negre, adica continuitate.
- cu cat o solutie are o discontinuitate mai mare (adica un camp negru are doar alb pe langa), cu atat este mai improbabil sa conduca la solutie, deci i se asociaza o valoare h mai mare.
- rezultatele obtine prin aceasta euristica sunt foarte dependente de imaginea reprezentata de Nonograma.

Euristica 2

- se bazeaza tot pe observatia ca in spatele solutiei se ascunde o imagine alb-negru. Iar aceste imagini sunt deobicei balansate (adica au ori mijlocul umplut, ori au marginile umplute pentru a crea perimetrul unei forme).
- se asociaza o valore mai mare pentru linii nou adauge mai ne-balansate.
- functioneaza foarte bine pe figuri simetrice sau aproape simetrice.

Problema satisfacerii restrictiilor

- **Variabilele** sunt randurile tabelei [Row0, Row1, ..., Row_height-1] si coloanele [Col0, Col1, ..., Col_width-1].
- **Domeniile** variabilelor sunt posibilele colorari ale randurilor sau coloanelor date de constrangerile de linii si coloane primite ca input.
- **Arcele** sunt intre perechi (Row_i, Col_j) pentru fiecare i si j. **Restrictia** (i,j) este ca Row_i[j]==Col_j[i] (adica punctul in care randul si coloana se intersecteaza are aceeasi valoare).
- Aplicam algoritmul **MAC**, care mentine consistenta arcelor prin aplicarea **AC3**:
    - Incepem cu o tabla goala, pe care o vom popula rand cu rand.
    - Alegem randul cu domeniul cel mai mic. II asignam pe rand cate o valoare din domeniul sau si aplicam AC3 pentru a reduce domeniul coloanelor si randurilor. In AC3, incepem cu coada care contine doar arcele din randul curent, scoatem pe rand cate un arc, verificam daca domeniile trebuie reduse si daca da, adaugam arcele care incep din randul sau coloana al carui domeniu s-a modificat.
    - Daca ajungem la un domeniu vid sau daca prin incercarea colorarii celoralalte randuri nu am ajuns la o solutie, facem backtracking pentru a incerca toate celelalte asignari posibile.
- **Imbunatatire bonus aplicata:** la fiecare apel bkt_mac, se alege asignarea randului cu cel mai mic domeniu (in realitate, multe variabile ajung sa aiba doar cate o valoare posibila si in felul asta scadem branch factorul la acel nivel).
- **Avantaje:**
    - rapid.
    - exploateaza putine noduri pentru ca mentine consistenta arcelor la fiecare pas.
    - starea (tabla) construita pana in orice moment este corecta.
- **Dezavantaje/limitari:**
    - spatiu suplimentar folosit pentru mentinerea variabilelor si a domeniului.
    - cost temporal initial pentru crearea domeniilor.
    - pentru fiecare asignare se aplica AC3, care adauga un cost temporal.

Analiza comparativa

STRATEGIE	TIMP TEST1 5X5	TIMP TEST2 5X5	TIMP TEST3 10X10	TIMP TEST4 15X15	TIMP TEST5 15X20	TIMP TEST6 25X25
BFS	0.0005	0.0029	>10 min			
DFS	0.0015	0.0019	>10 min			
ITERATIVE DEEPENING	0.0017	0.0038	>10 min			
A* (H1)	0.0002	0.0003	0.013	2.66	tle	35.96
A* (H2)	0.0013	0.0003	0.014	2.51	tle	43.01
CSP	0.0011	0.0012	0.0067	1.5	0.93	16.75

Cum rezolvarea unei nonograme este o problema NP-completa, ne asteptam la complexitate exponentiala si deci la cresteri mari in timpul de rulare odata cu cresterea dimensiunii problemei.

Se observa clar ca strategiile informate sunt mult mai optime. CSP are timpi comparabili cu DFS/BFS pentru teste mici, datorita costului initial de creare a domeniilor pentru fiecare variabila. Dupa aceea, strategiile neinformate nu reusesc sa se termina intr-un timp adecvat, in timp ce CSP (mai exact MAC backtracking, cu reducere a domeniilor prin AC3) are un timp OK pentru table mai mari. A* este mai imprevizibil, depinzand mult de input, dar tot are timpi mai buni si rezonabili pentru 5/6 teste.

Tot prin timpii de rulare ai CSP si A*, observam ca pentru o strategie informata, timpul de rulare nu depinde doar de dimensiune (heightXwidth), ci si de proprietati ale tablei. Testul 4 dureaza mai mult decat testul 5 desi are dimeniunea unei linii mai mic, dar de fapt domeniile pentru variabile din testul 5 sunt mai restranse.

Cele doua euristici pentru A* dau rezultate asemanatoare pentru aceste teste, dar euristica 2 pare a fi mai putin informata si adecvata. Niciuna dintre ele nu functioneaza pe testul 5, care nu are un pattern asa usor de identificat.

STRATEGIE	NODURI GENERATE /EXPANDATE TEST1	NODURI GENERATE /EXPANDATE TEST2	NODURI GENERATE /EXPANDATE TEST3	NODURI GENERATE /EXPANDATE TEST4	NODURI GENERATE /EXPANDATE TEST5	NODURI GENERATE /EXPANDATE TEST6
BFS	18/7	53/16				
DFS	21/17	44/38				
ITERATIVE DEEPENING	36/36	75/73				
A* (H1)	10/5	5/5	42/42	785/785	tle	187/187
A* (H2)	10/10	5/5	42/42	785/785	tle	187/187
CSP	6/5	5/5	11/10	27/15	27/15	29/25

Pentru cautarile neinformate, avem putine teste care se termina intr-un timp rezonabil, deci mai putine date pe care putem sa le analizam.

Se observa insa ca DFS expandeaza mai multe noduri decat BFS, chiar daca numarul de noduri generate este apropiat. Aceste diferente se datoreaza si proprietatilor problemei.

Iterative Deepening expandeaza si genereaza cele mai mult noduri. Observam si ca majoritatea nodurilor generate sunt si expandate, acest lucru fiind datorat rularilor in care se atinge adancimea maxima si sunt expandate toate nodurile pana acolo.

A* genereaza foarte multe noduri si, aparent, exploateaza la fel de multe. Acest lucru se datoreaza faptului ca tranzitiile se fac catre stari care verifica si constrangerile de coloane, deci care pot fi expandate foarte mult.

CSP genereaza putine noduri pentru ca genereaza rar noduri care nu pot conduce la solutie, verificand mereu atat conditiile de rand, cat si cele de coloana. De asemenea, observam ca numarul de noduri expandate este egal cu inaltimea tablei. Inseamna ca prin mentinerea consistentei arcelor, el reuseste sa construiasca din prima solutia (adica backtracking-ul va merge pe o singura cale radacina-frunza, pe care se si afla solutia). CSP este cea mai de incredere strategie de rezolvare.