Tema 1 IA - Generator Orare

Ceaușu Matei Călin, grupa 331CA

April 30, 2024

1 Descrierea Problemei

1.1 Cerința problemei

Scopul temei este să se genereze o posibilă soluție pentru problema orarului. Problema are următoarele date de intrare:

- O listă de intervale în care este posibil să se țină cursuri
- O listă de materii și numărul de studenți care trebuie să participe la cursul respectiv
- O listă cu profesori, cu materiile la care profesorul poate să țină curs și cu constrângerile pentru programul fiecărui profesor. Constrângerile conțin zilele în care preferă să fie cursurile, zilele în care nu și intervalele preferate și cele care trebuie evitate
- O listă de săli, capacitatea lor și materiile care se pot ține în sala respectivă
- O listă de zile în care se pot ține cursurile

Cerințele profesorilor reprezintă constrângerile soft ale acestei probleme. Este de preferat ca aceste constrângeri să fie încălcate și în final să se obțină un orar valid. Situația ideală este ca orarul final să nu încalce nicio constrângere, însă acest lucru nu este întotdeauna posibil.

1.2 Reprezentarea stărilor și a restricților

Un element important în rezolvarea problemei este modul de reprezentare al unei stări. Pentru ca problema să fie mai ușor de rezolvat, trebuie aleasă o reprezentare care permite facil generarea vecinilor, verificarea respectării condițiilor și verificarea stării, în sensul dacă aceasta este finală sau nu.

1.2.1 Reprezentarea stării

Pentru a reprezenta stările, am gândit în felul următor: un tuplu format din zi, interval și sală determină unic o pereche de profesor și materia pe care o ține. Așadar am reprezentat orarul ca o matrice tridimensională în care cele 3 dimensiuni sunt ziele, intervale și sălile, iar valorile din matrice sunt perechi formate din profesori și materia pe care o predă profesorul respectiv în intervalul respectiv.

1.2.2 Detalii privind partea tehnică

În ceea ce privește detaliile tehnice, orarul este reprezentat ca un dicționar care are drept cheie ziua și ca valoare are un alt dicționar. Acest dicționar are drept cheie o pereche de valori care reprezintă intervalul orar și ca valoare un alt dicționar. Acest ultim dicționar are drept cheie un string care reprezintă sala în care se ține cursul, iar valoarea este o pereche formată din numele profesorului și numele materiei pe care o predă. Toate aceste informații sunt stocate în clasa State. Pentru ambii algoritmi, am implementat metode care întorc o listă de vecini și o metodă care verifică dacă starea curentă este una finală. O stare este finală dacă toate materiile au fost acoperite. Pentru a avea o implementare mai eficientă, vecinii nu sunt noi stări, ci o reprezentare minimală a unei noi stări. Adică, vecinul este reprezentat ca un tuplu format din zi, interval, sală, profesor și materie care urmează a fi adăugat la orarul curent pentru a obține vecinul respectiv. Astfel, nu se vor face copii la stări care nu vor fi expandate. De asemenea, vecinii unei stări sunt generați astfel încât în orice moment de timp nu este încălcată **nicio** constrângere hard, cu excepția celei de a acoperi toate materiile. Această constrângere determină dacă starea este finală sau nu. După ce se selectează vecinul care urmează să fie expandat, se va apela metoda care aplică acțiunea dată și întoarce starea vecină. Nu în ultimul rând, există o metodă care întoarce numărul de constrângeri soft încălcate. Acestea sunt utile pentru a ghida algoritmii spre soluția optimă.

1.3 Cerința Bonus

Cerința bonus a acestei probleme este de a ține cont și de pauza dintre cursurile profesorilor. Astfel, în lista de preferințe a fiecărui profesor se găsește o constrângere care menționează pauza maximă acceptată de fiecare profesor.

Pentru a ține cont de preferințele fiecărui profesor, am reținut într-o structură de date internă fiecărei stări a orarului și intervalele din fiecare zi în care profesorul are cursuri. Când se adaugă un nou curs, se va itera prin această listă de intervale. Se caută poziția în care ar trebui să fie inserat intervalul pentru a păstra lista crescătoare și se compară capetele intervalelor vecine. Dacă diferența dintre ele depășește pragul admis de profesor, se crește numărul de conflicte ale stării. Față de problema fără această constrângere, efortul computațional este mai mare, întrucât la fiecare nouă stare există noi condiții de verificat.

2 Soluția folosind Hill Climbing

2.1 Descrierea algoritmului și detalii tehnice

Una din soluțiile impuse de cerință este să rezolvăm problema folosind algoritmul Hill Climbing. Pentru a rezolva acest task, am ales să folosesc "Random Restart Hill Climbing", împreună cu "Stochastic Hill Climbing". În cadrul algoritmul "Stochastic Hill Climbing", se pornește de la o soluție inițială, în cazul nostru un orar gol. Dintre vecinii stări, se alege aleator unul care respectă condiția impusă. În cazul nostru, condiția e ca numărul de conflicte să nu crească față de starea precedentă. Dacă nu există vecini care respectă condiția impusă, se alege starea urmatoare aleator din stările care au un număr minim de conflicte. Acest algoritm este unul de tip Greedy, de aceea timpul de execuție este foarte mic comparativ cu alți algoritmi de căutare. Acest fapt este dat și de numărul maximum de iterații pentru fiecare căutare și de numărul maximum de reporniri. Algoritmul "Random Restart Hill Climbing", spre deosebire de varianta din laborator, se oprește atunci când se gasește o soluție cu 0 constrângeri soft încălcate sau atunci când s-a atins limita de reporniri impusă.

2.2 Descrierera comportamentului algoritmului pe testele date

Pentru testele mai relaxate, precum: dummy, *orar mediu relaxat* și *orar mare relaxat*, soluția este găsită într-o singură repornite, iar toate soluțiile sunt exacte. În ceea ce privește testul mai constrâns *orar mic exact*, algoritmul poate necesita mai multe reporniri pentru a găsi soluția exactă (de obicei maximum 4 reporniri). Însă, pentru testul cel mai constrâns, *orar constrans incalcat*, algoritmul nu găsește o soluție exactă și deci atinge limita maximă de reporniri. Cea mai bună soluție găsită pentru testul dat este cea cu 2 constrângeri încălcate. Pentru a obține o soluție bună am modificat numărul maximum de reporniri.

3 Soluția folosind A*

3.1 Descrierea algoritmului și detalii tehnice

Algoritmul A* se aseamănă foarte mult cu algoritmul lui Dijkstra. Astfel, se pornește de la o stare și pe baza costului și a euristicii stabilite, se caută o cale către o stare considerată finală. Astfel, scoate o stare din frontieră și se adaugă noile stări vecine, neexplorate sau vor fi readăugate dacă se găsește o cale mai scurtă pentru ele. În cazul nostru, frontiera este de fapt o coadă cu priorități. Aceasta asigură alegerea unui nod cu cost minimum din cele care nu au fost explorate. Costul minimum înseamnă că există o șansă mai mare ca ele să ducă la o soluție bună sau chiar exactă. Alegerea euristicii depinde foarte mult de constrângerile problemei. Euristica influențează foarte mult alegerea viitoarelor stări și deci timpul de execuție al programului.

3.2 Alegerea euristicii

Euristica pe care eu am ales-o am cosiderat că trebuie să depindă în principal de numărul de conflicte ale stării curente și de numărul de studenți neasignați. Având în vedere că numărul de studenți este mult mai mare decât numărul posibil de conflicte, am decis să adaug un factor de scalare α. Așadar, euristica are forma:

$$h(x) = număr\ conflicte * \alpha + număr\ studenți\ neasignați$$

Această euristică este admisibilă pentru soluțiile exacte, adică cele cu nicio constrângere încălcată. Însă, așa cum am menționat, nu orice set de date de intrare are o soluție exactă. În cazul în care nu există o soluție exactă sau nu vrem să găsim soluția exactă, această euristică nu va fi nulă în starea finală. Pentru problema dată, mi se pare mai important numărul de conflicte încălcate decât numărul de studenți. Așa că se va inițializa α cu numărul total de studenți. Pe scurt, orice stare cu 0 conflicte va avea un cost mai mic decât o stare cu mai multe conflicte.

Această euristică nu are un comportament adecvat în cazurile în care nu se vrea soluția exactă sau în cazul în care nu există. În aceste cazuri, se va modifica valoarea coeficientului într-o valoare mai mică.

3.3 Descrierera comportamentului algoritmului pe testele date

- În cazul orarului *dummy* și în cazul *orarului mic exact*, timpul de execuție este relativ mic și se expandează un număr mic de stări.
- În cazul orarelor *orar mediu relaxat* și *orar mare relaxat*, timpul de execuție crește mult, ceea ce era de așteptat, însă în continuare euristica dată funcționează și se obțin soluții exacte.
- În cazul *orarului constrans incalcat*, euristica trebuie modificată, întrucât timpul de execuție este prea mare și consumă prea multă memorie RAM. Astfel, pentru a obține o soluție finală, trebuie relaxată euristica și modificat coeficientul. Se va obține un orar cu constrângeri opționale încălcate.

4 Comparația algoritmilor

Comparația dintre cei doi algoritmi se va realiza pe baza timpului de execuție și al numărul de reporniri în cazul algoritmului Hill Climbing și al numărul de reporniri în cazul algoritmului A*. Cu toate acestea, rezultatele pot varia în funcție de sistemul de calcul pe care se rulează programul.

4.1 Algoritmul Hill Climbing

Așa cum am menționat, acest algoritm este unul Greedy și tinde să fie mai rapid decât alți algoritmi de căutare.

Test	Timp execuție (s)	Număr Reporniri	Conflicte soft
dummy	0.01796	0	0
orar mediu	2.41485	0	0
orar mare	4.90682	0	0
orar mic exact	0.27689	0	0
orar constrans	285	300 (max limit)	1
orar bonus	213	32	0

Se poate observa că în cazul în care nu este nevoie de restart, adică se găsește o soluție exactă, timpul de rulare este relativ mic (sub 5 secunde). Pentru *orarul constrans*, unde nu există nicio soluție exactă, timpul de execuție depinde de limita de reporniri. Aici se observă ideea de trade-off, și anume, dacă vrem o soluție care durează mai puțin, dar cu o probabilitate scăzută de a obține un număr mic de constrângeri soft încălcate se scade limita de reporniri. Dacă dorim să obținem efectul invers, se crește limita de reporniri.

4.2 Algoritmul A*

În cazul algoritmului A*, pe lângă timpul de execuție, este foarte relevant și numărul de stări explorate. Algoritmul explorează un spațiu mult mai mare față de algoritmul Hill Climbing și prin urmare consumă și o cantitate mult mai mare de memorie.

Test	Timp execuție (s)	Număr st. expandate	Conflicte soft
dummy	0.15685	434	0
orar mediu	202	155264	0
orar mare	1085	326002	0
orar mic exact	24	24317	0
orar constrans	72	56704	10
orar bonus	490	227364	2

Rezultatele pentru testele mari pot varia în funcție de starea sistemului. În cadrul testelor mele, pentru testul orar mare relaxat, întreaga memorie RAM a fost ocupată și a început procesul de "memory swap", crescând astfel drastic timpul de execuție.

În cazul testului *orar constrans incalcat*, s-a folosit o altă euristică, cu un coeficient de scalare mult mai mic ($\alpha = 1$). Astfel, rezultatele pentru acest test pot varia de la o rulare la alta.

În cazul testului *orar constrans incalcat*, s-a folosit o altă euristică, cu un coeficient de scalare mult mai mic ($\alpha = 0$). Astfel, rezultatele pentru acest test pot varia de la o rulare la alta.

4.3 Comparație cot la cot

În ceea ce privește timpul de execuție, memoria ocupată și complexitatea, din analiza făcută reiese că algoritmul Hill Climbing este mai potrivit pentru problema dată. Cu toate acestea, în situația în care datele de intrare erau mai mari, exista posibilitatea ca alegerea algoritmului A* să fi fost cea corectă, întrucât explorează un spațiu mult mare mare ca algoritmul Hill Climbing.

5 Concluzii

Algoritmul A* necesită o euristică mult mai complicată pentru a ghida algoritmul către o soluție exactă. Acesta explorează mult mai multe stări și, în consecință, timpul de execuție este mult mai mare. În schimb, algoritmul Hill Climbing ajunge destul de rapid la o soluție acceptabilă, daca nu chiar exactă. Acest lucru este dat de faptul că algoritmul Hill Climbing este unul de tip Greedy. Algoritmul pornește de la o soluție inițială și explorează vecinătatea în speranța că se va găsi o soluție mai bună. Având în vedere că algoritmul se bazează pe șansă, îmbinarea algorimului Stochastic Hill Climbing cu Random Restart, asigură creșterea probabilității de a găsi o soluție bună.