Sterea Ștefan Octavian
334CB

Inteligență Artificială – Tema 1

Analiză comparativă a algoritmilor implementați

(PCSP vs HC)

1. Prezentarea generală a implementării

PCSP: Din punct de vedere matematic, am implementat o formă de PCSP + Constraint Propagation, dar am eficientizat algoritmul, specializându-l pe situația din problemă în mai multe feluri, ce vor fi prezentate în cele ce urmeaza.

Reprezentarea variabilelor și a domeniilor:

Pornind de la ideea că atunci când completăm un orar, nu completăm de fapt un singur orar, ci un orar comun, câte un orar pentru fiecare profesor și un orar al sălilor. Toate acestea reprezintă mulțimi de variabile (de tipuri/reprezentări diferite) ce au în relații între ele, date de constrângeri. Astfel, variabilele din algoritmul meu sunt:

- tupluri (zi, interval, sală), având valori (profesor, materie)
  [corespund orarului comun]
- tupluri (zi, interval, profesor) având valori (sală, materie)
  [corespund orarelor profesorilor]
- string profesor având valori numerice [numărul de ore pe care le ține]
- string curs având valori numerice [capacitatea totală ocupată în sălile în care a fost alocat cursul (inclusiv locuri libere)]
- string ‘U’ având valori tupluri (capacitatea totală ocupată de toate cursurile, capacitatea efectivă totală ocupată de toate cursurile) [capacitatea efectiva – fără locurile libere]

Aceste variabile au relații strânse între ele, atât de strânse încât multe dintre ele sunt complet determinate de valoarea altora, astfel că ar deveni ineficient să realizăm algoritmul constraint propagation verificând toate valorile consistente și eliminându-le din domenii. Astfel, am realizat următoarea separație: variabilele (zi, interval, sală) sunt cele pe care le luăm drept “independente” și algoritmul va face backtracking pentru a le atribui valori, celelalte sunt variabile dependente și vor fi atribuite atunci când o variabilă independentă este atribuită. În acest fel, multe dintre restricțiile ce le acoperă sunt aproape implicite – atunci când o variabilă dependentă este atribuită, se verifică și o condiție ce duce la încălcarea unei constrângeri.

Sterea Ștefan Octavian
334CB

Reprezentarea constrângerilor: o listă de variabile peste care se leaga, un predicat, și un cost. Costul este luat astfel: 1 pentru constrangerile optionale, infinit pentru constrangerile obligatorii. Astfel, algoritmul prioritizează și este obligat să satisfacă constrangerile obligatorii pentru a ajunge la costul acceptabil.

Constrângerea ca sălile alocate pentru un curs să aiba capacitatea necesară tuturor studenților înscriși la acel curs este una specială, deoarece este, într-un sens, "negativă" – în loc să fie satisfăcută până la o atribuire a variabilelor care o încalcă, ea este de la început încălcată și o atribuire potrivită a variabilelor o va satisface. Pentru c pe parcursul explorării algoritmului să putem calcula costul soluției încremental, evaluând doar constrângerile ce se leagă de variabila curentă și adunându-le la costul acumulat, am transformat și această restricție într-una "pozitivă", astfel: "Capacitatea totală nealocată a tuturor sălilor în întreg orarul este mai mare sau egală cu capacitatea totală necesară rămasă pentru toate cursurile"

În plus, pentru a reduce și mai mult căutarea am introdus o constrângere obligatorie adițională: capacitatea alocată unui curs nu trebuie să depășească cu capacitatea uneia din sălile alocate capacitatea necesară.

Alte eficientizări: am ordonat valorile din domenii după preferințele profesorilor, dând prioritate profesorilor care își doresc ziua și/sau slot-ul și apoi prioritate adițională profesorilor care vor mai puține sloturi.

HC: Presenting... stochastic first-choice random-restart (ex-simulated annealing-wannabe) shape-shifting hill climbing!

Inițial am implementat un simplu first-choice hill-climbing, plecând de la o stare inițială generate aleator, dar evaluând stările folosind o strategie pe bază de diferențe în cost față de starea anterioară (cost = cost + Δcost) (numesc această metodă "delta-ing"). Delta-ul poate fi calculate în O(1), ceea ce face această abordare pentru calcularea costului eficientă. Pentru a calcula costul, am folosit o sumă ponderată, dând ponderi mai mari constrângerilor hard. Vecinii unei stări sunt apoi toate orarele valide dpdv al repartizărilor cursurilor pe săli și profesori și al neduplicității unui professor într-un interval orar, care diferă de orarul curent printr-o singură atribuire materie-profesor într-un slot sau orarele în care două slot-uri sunt interschimbate între ele. Pentru a putea explora ambele tipuri de vecini, aceștia sunt generați intercalat.

Pentru a nu se bloca într-un minim local, am adaugat apoi random restart, rulând întreg algoritmul de un număr maxim de ori (parametrul max_restarts=1000) și salvând cea mai

bună solutie. Am obținut astfel rezultate bune, de cost 0, mai puțin pe testele mari, unde nu reușeam să obțin 0 constrângeri soft, iar timpul de rulare era foarte mare din cauza faptului că algoritmul nu se termina până nu facea toate cele 1000 de iteratii (aprox. 4 minute constrâns și 6-11 restricții soft încălcate).

Pentru a crește șansele algoritmului de a ajunge în minimul global în timp util, am mai adăugat un element nedeterminist: nu era fezabil să generez toți vecinii posibili ai unei stări pentru a face stochastic sau steepest-ascent, dar am amestecat listele cu toate sloturile după care generez vecinii, așadar apare o amestecare și în generarea lor (cam devine stochastic în loc de first-choice, dar îmi place să zic că e ambele simultan =))

Nici această schimbare nu s-a dovedit a da rezultate din păcate. Așa că am mai adăugat un element de nedeterminism: inspirat din simulated annealing, am încercat varianta în care, în loc să aleg întotdeauna primul vecin mai bun, am lăsat și o probabilitate de a explora un vecin mai prost. Aceasta a dus, într-adevăr la rezultate mai bune (se termina orarul mare cu cost 0, dar orarul constrâns tot nu găsea soluție, totuși, obțineam costuri ceva mai mici, 3-6 constrângeri soft încălcate).

Pentru câteva ore am încercat și o adaptare graduală de la first-choice la steepest-ascent (pe care aș fi numit-o steeper-ascent dacă ar fi mers xD): știind faptul că pentru orarele mai mici este găsită o soluție destul de repede, după puține stări explorate, m-am gândit să încerc să cresc numărul de vecini luați in considerare și ales minimul dintre ei pe parcursul algoritmului, crezând că aș putea face orarul constrâns să-și găsească o soluție în acest fel. Din nefericire, această abordare nu a dat roade, ducând la timpi chiar și mai mari.

Observație: calculez delta-ul unei soluții folosind o sumă ponderată unde constrângerile hard au o pondere mai mare decât cele soft. Rolul acestor ponderi este ca o stare să fie considerată "mai bună" și dacă aduce noi încălcări ale unor restricții. Practic, daca un vecin rezolvă o constrangere hard, dar încalcă una-două noi constrângeri soft, va putea fi considerat un vecin mai bun.

Într-o sclipire de geniu mi-a venit o idee nemaiîntâlnită: algorimul meu prioritizează în acest fel constrangerile hard, lăsând mult loc ca algoritmul să se blocheze într-un minim local dat de constrângeri soft. Așadar, pentru a da o șansă algoritmului să iasă din acea stare, am inversat temporat prioritățile, schimbând astfel funcția de cost cu totul, obținând noi minime locale, de această data date de contrângerile hard. Am lăsat la fiecare repetiție a algoritmului să realizeze un număr maxim de astfel de schimburi de strategie de câte ori se blochează. Eventual, algoritmul ajunge să gasească minimul global, trecând urmărind minime locale diferite pe un deal care își schimbă forma (de aici, shape-shifting :O)

Sterea Ștefan Octavian
334CB

2. Rezultate obținute

Pentru CSP, înainte de a realiza ordonarea domeniilor, toate testele mai puțin orarul constrâns se terminau destul de repede, dar cel constrâns nu reușea să găsească vreo soluție de cost mai mic decât 11 în timp util. Acest lucru se datora faptului că pierdea enorm de mult timp atribuind valori greșite slot-urilor de la începutul orarului și ajungea până la finalul lui fără să încalce restricții obligatorii dacă era lăsat de costul acceptat. Pierdea foarte mult timp atribuind intervale nepreferate unor profesori cu foarte puține sloturi dorite. După sortare, rezultatele obținute sunt urmatoarele:

dummy

```
cost=0, iterations=48

real    0m0.181s
user    0m0.015s
sys     0m0.000s
```

orar_mic_exact

```
cost=0, iterations=370

real    0m0.174s
user    0m0.015s
sys     0m0.000s
```

orar_mediu_relaxat

```
cost=0, iterations=1687

real    0m0.213s
user    0m0.000s
sys     0m0.031s
```

orar_mare_relaxat

```
cost=0, iterations=4963

real    0m0.344s
user    0m0.000s
sys     0m0.031s
```

orar_constrans_incalcat

```
cost=0, iterations=284

real    0m0.183s
user    0m0.000s
sys     0m0.015s
```

```
cost=0, iterations=47311

real    0m1.046s
user    0m0.015s
sys     0m0.015s
```

Pentru HC:

Evaluated_states reprezintă numărul de stări scoase din generator și evaluate pentru a determina dacă sunt mai bune, iar visited_states reprezintă numărul de stări explorate

dummy

```
cost=0, evaluated_states=522, visited_states=13

real    0m0.175s
user    0m0.015s
sys     0m0.015s
```

orar_mic_exact

```
cost=0, evaluated_states=17066, visited_states=71

real    0m0.268s
user    0m0.015s
sys     0m0.031s
```

Sterea Ștefan Octavian
334CB

orar_mediu_relaxat

```
cost=0, evaluated_states=5830, visited_states=70

real    0m0.216s
user    0m0.000s
sys     0m0.031s
```

orar_mare_relaxat

```
cost=0, evaluated_states=95010, visited_states=183

real    0m0.703s
user    0m0.016s
sys     0m0.015s
```

orar_constrans_incalcat

```
cost=0, evaluated_states=285400, visited_states=806

real    0m1.734s
user    0m0.015s
sys     0m0.031s
```

```
cost=0, evaluated_states=5830033, visited_states=16491

real    0m32.803s
user    0m0.000s
sys     0m0.031s
```

3. Explicarea rezultatelor

Așa cum se observă în imaginile de la CSP orar_constrans_incalcat, rezultatele variază. Componenta nedeterministă este jucată de funcția sort atunci când două valori au aceeași prioritate, ceea ce poate duce la ordonări mai bune sau mai proaste. Pentru a aduce determinism, ar trebui aranjată formula pe care o folosesc să dau prioritate unor valori față de altele să ia în calcul mai multe aspecte legate de preferințele profesorilor și distribuția acestora.

Și în cazul HC-ului, la orar_constrans_incalcat observăm o variație mare, atât în numărul de stări explorate, cât și în timpul de rulare, datorată gradului mare de nedeterminism și abordării statistice.

Numărul de 1000 de iterații folosit este, în momentul actual, exagerat, căci algoritmul ar funcționa bine și cu un număr mai mic, dar este luat pentru a asigura o probabilitate cât mai mare ca rezultatul să fie minim.

Sterea Ștefan Octavian
334CB

4. Comparație

HC este ceva mai rapid decât CSP, în mod special pe orarele mari, dar fără trucul folosit cu schimbarea funcției de cost, nu reușea să găsească niciodată soluția de cost minim pe cel constrans. CSP este mai precis din acest punct de vedere, HC având o abordare euristică și statistică.

Dacă ne uităm la evaluated_states de la hc, pare că stă prost, dar acelea sunt doar niște comparații și calculul costului este facut în timp constant, deci metrica mai bună aici (comparabilă cu iterations de la CSP) este visited_states. Din acest punct de vedere, observăm că HC reușeste să "sară" mai rapid la soluție. Abordarea diferă: HC se blochează repede și apoi trebuie restartat într-un fel sau altul, iar gasirea minimului global este garantată de caracterul nedeterminist și de numărul mare de repetări. CSP încearcă să construiască soluția, eliminând variante până obține soluții din ce în ce mai bune, dar este vulnerabil la a face greșeli devreme care sa coste scump târziu (nu reușeste să-și descopere greșelile timpuriu)