Proiectarea Algoritmilor 2009-2010

Laborator 4

Backtracking și optimizări

1. Obiective laborator

a. Înțelegerea noțiunilor de bază legate de backtracking și optimizările aferente;

b. Conștientizarea necesității îmbunătățirii versiunii simple de backtracking și beneficiile fiecărei abordări în parte;

c. Familiarizarea atât cu problema satisfacerii constrângerilor, cât și cu metode prospective, euristici.

2. Aplicații practice

Răspunsul imediat – orice problemă care presupune o căutare în spațiul stărilor. De asemenea majoritatea problemelor din Inteligență Artificială pot fi reduse la problema satisfacerii constrângerilor, iar metodele prospective, respectiv euristicile pot fi aplicate într-o multitudine de probleme, fiind general valabile.

3. Descrierea problemei și a rezolvărilor

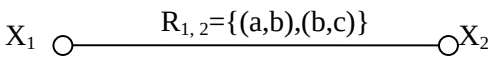
Pornind de la strategiile clasice de parcurgere a spațiului de stări, algoritmii de tip backtracking practic enumeră un set de candidați parțiali, care după completarea definitivă pot deveni soluții potențiale ale problemei inițiale. Exact ca strategiile de parcurgere în lățime/adâncime și backtracking-ul are la bază expandarea unui nod curent, iar determinarea soluției se face într-o manieră incrementală. Prin natura sa, bkt-ul este recursiv, iar in arborele expandat top-down se aplică operații de tipul prunning dacă soluția parțială nu este validă.

Notațiile utilizate sunt următoarele:

- $X_1, \ldots, X_N$ variabilele problemei, N fiind numărul de variabile ale problemei;
- $D_1, \ldots, D_N$ domeniile aferente fiecărei variabile;
- U - întreg care reprezintă indicele variabilei curent selectate pentru a i se atribui o valoare;
- F - vector indexat după indicii variabilelor, în care sunt memorate selecțiile de valori făcute de la prima variabila și până la variabila curentă

$$\text{Relatie}(U_1, F[U_1], U_2, F[U_2]) = \begin{cases} \text{true} & \text{daca exista restrictia } R_{U_1 U_2}(F[U_1], F[U_2]) \\ & R_{U_1 U_2} \subset D_{U_1} \times D_{U_2} \\ \text{false} & \text{altfel} \end{cases}$$

Reprezentarea grafică a unei relații pentru două variabile X_1 și X_2 cu domeniul {a, b, c} este următoarea:



O versiune generică a algoritmului de tip backtracking recursiv poate fi următoarea:

```
BKT (U, F)
foreach V of X_U
      F[U] ← V
      if Verifica (U,F) == true
      then
            if U < N
                  then BKT(U+1, F)
            else
                  Afișează valorile din vectorul F
                  break

Verifică (U,F)
      test = true
      I ← U - 1
      while I > 0
            test = Relație(I, F[I], U, F[U])
            I = I - 1
            if test == false
                  then break
      return test
```

Complexitatea algoritmului: complexitatea temporală este de **$O(B^d)$,** iar cea **spațială O(d)**, unde **B** este *factor de ramificare* (numărul mediu de stări posibil ulterioare în care nodul curent poate fi expandat) și **d** este *adâncimea soluției*.

Pornind de la versiunea inițială de BKT, putem aduce o **serie de îmbunătățiri** în următoarele direcții:

- **Algoritmi de îmbunătățire a consistenței reprezentării** care vizează consistența locală a arcelor sau a căilor în graful de restricții
- **Algoritmi hibrizi** care îmbunătățesc performanțele rezolvării prin reducerea numărului de teste; aici putem identifica următoarele subcategorii:
    - *Tehnici prospective:*
        - Căutare cu predicție completă
        - Căutare cu predicție parțială
        - Căutare cu verificare predictivă
    - *Tehnici retrospective:*
        - Backtracking cu salt
        - Backtracking cu marcare
- **Utilizarea euristicilor** în vederea optimizării numărului de teste prin luarea în considerare a următoarelor scenarii:
    - *Ordonarea variabilelor*
    - *Ordonarea valorilor*
    - *Ordonarea testelor*

Dintre metodele enumerate mai sus ne vom concentra asupra **CSP** (Constraint Satisfaction Problem) cu îmbunătățirea aferentă a consistenței reprezentării și asupra tehnicilor prospective, existând în cazul ambelor o îmbunătățire sesizabilă la nivelul apelurilor recursive / al expandărilor efectuate / al intrărilor în stivă.

3.1 Problema satisfacerii constrângerilor

Problema satisfacerii restricţiilor, în formularea cea mai generală, presupune existența unei mulțimi de variabile, unor domenii de valori potențiale pentru fiecare variabilă şi o mulțime de restricții care specifică combinațiile de valori acceptabile ale variabilelor (exact conceptul de relații definite anterior, cu tot cu restricțiile aferente). ***Scopul final*** îl reprezintă determinarea unei atribuiri de valori pentru fiecare variabila astfel încât toate restricțiile să fie satisfăcute.

Problema satisfacerii restricţiilor este, în cazul general, o problema grea, deci **NP-completă**, exponențială în raport cu numărul de variabile ale problemei. Din perspectiva strategiilor de căutare într-un spațiu de stări, traducerea problemei ar fi următoarea: pornind din starea inițială a procesului care conține restricțiile identificate în descrierea inițială a problemei, se dorește atingerea unei stări finale care a fost restricționată "suficient" pentru a rezolva problema.

Pornind de la premisa că CSP este o problema de căutare în clasa problemelor NP-complete, 2 aspecte fundamentale sunt de interes: reducerea cât mai puternică a timpului / spațiului de căutare, respectiv identificarea de sub-clase de probleme care pot fi rezolvate în timp polinomial.

Fiind o problemă de căutare, rezolvarea problemei satisfacerii restricţiilor poate fi făcută aplicând una din tehnicile de căutare a soluției în spațiul stărilor. Astfel, cea mai utilizată strategie de rezolvare a problemei CSP este backtracking-ul, variantă simplificată a căutării neinformate în adâncime. Aceasta strategie este preferată datorită economiei de spațiu atinse raportat la strategia de căutare în adâncime - O(B*d) sau pe nivel – O (B^d).

În fucție de particularizare şi anume în funcție de necesitatea determinării unei soluții sau a tuturor soluțiilor, satisfacerea tuturor constrângerilor sau relaxarea unora, putem avea următoarele categorii:

- CSP totală
- CSP parțială
- CSP binară – graf de restricții

Pentru noi, în cazul studiului de față, problemele de tipul CSP binare care pot fi reprezentate printr-un graf de restricții sunt de interes.

Un exemplu clasic de problemă care poate fi optimizată folosind CSP este așa numita problemă a zebrei [8] descrisă prin următoarele propoziţii:

1. Sunt cinci case, fiecare are o altă culoare, este locuită de persoane cu naţionalităţi diferite, cu animale de casă, băuturi si ţigări distincte.

2. Englezul locuieşte în casa roşie.

3. Spaniolul are un câine.

4. Cafeaua este băută în casa verde.

5. Ucraineanul bea ceai.

6. Casa verde este imediat la dreapta casei albe.

7. Fumătorul de Kent are melci.

8. Marlboro e fumat în casa galbenă.

9. În casa din mijloc se bea lapte.

10. Norvegianul stă în prima casă din stânga.

11. Fumătorul de Chesterfield stă lângă cel care are o vulpe.

12. Marlboro este fumat în casa de lângă cea care are un cal.

13. Fumătorul de Lucky-Strike bea oranjadă.

14. Japonezul fumează West.

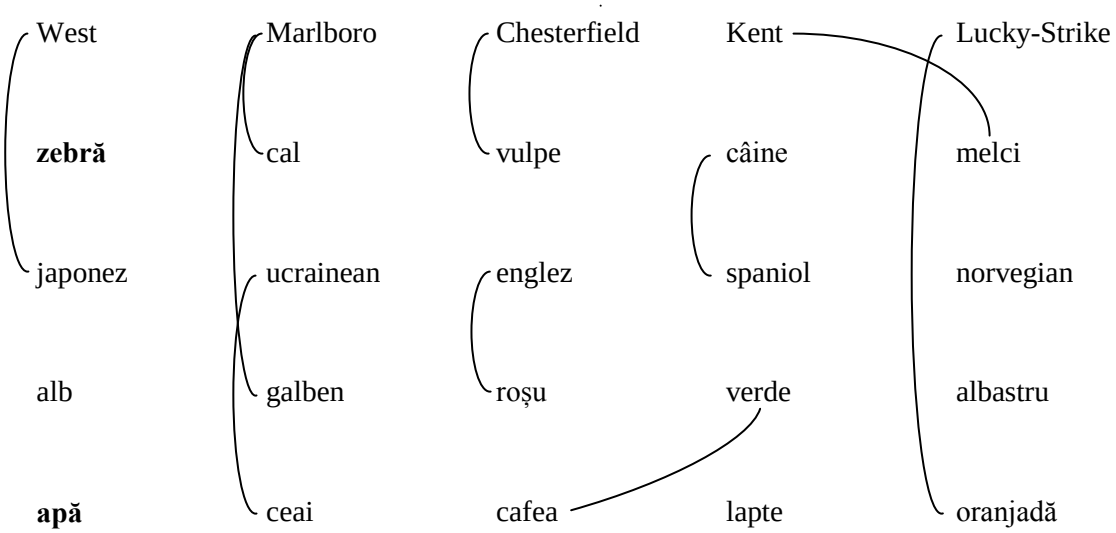
15. Norvegianul stă lângă casa albastră.

Întrebarea este: Cine bea apă si cine are o zebră?

Problema prezintă 5 case iar fiecare casă are 5 atribute specifice cu câte 5 valori fiecare:

- culoare: roșu, albastru, galben, verde, alb
- naționalitate: norvegian, ucrainean, englez, spaniol, japonez
- băutură: cafea, ceai, apă, lapte, oranjadă
- animal: zebra, câine, cal, vulpe, melci
- țigări: Kent, Marlboro, West, Chesterfield, Lucky-Strike

Prin urmare, există 5 case x 5 atribute/casă = 25 variabile; cum fiecare variabilă poate avea 5 valori rezultă că numărul maxim de tuple care pot fi generate si care trebuie verificate este $5^{25} \approx 3*10^{17}$. Folosind însă restricţiile problemei, o parte din ele fiind figurate prin arce în figura următoare, se poate obţine rapid o soluţie, folosind pe lângă CSP drept strategie de căutare backtrackingul.



Reţelele de restricţii constituie nu numai un punct de plecare pentru algoritmi eficienţi de rezolvare de probleme ci și o modalitate de exprimare declarativă a cunoștinţelor, care permite scoaterea în evidenta a relaţiilor locale între componente. Prelucrarea restricţiilor este un proces de utilizare a acestei clase de cunoștinţe pentru a deriva consecinţe utile si a produce o comportare inteligentă.

Definiții și algoritmi

Un **arc (X_i, X_j)** într-un graf de restricții orientat se numește ***arc-consistent*** dacă și numai dacă pentru orice valoare $x \in D_i$, domeniul variabilei X_i, există o valoare $y \in D_j$, domeniul variabilei X_j, astfel încât $R_{i,j}(x,y)$. **Graf de restrictii orientat** rezultat se numește ***arc-consistent***.

O cale de lungime *m* prin nodurile $i_0,\ldots,i_m$ ale unui graf de restricții orientat se numește ***m-cale-consistentă*** dacă și numai dacă pentru orice valoare $x \in D_{i0}$, domeniul variabilei i_0 și o valoare $y \in D_{jm}$, domeniul variabilei i_m, pentru care $R_{i0,\,im}(x,y)$, există o secvență de valori $z_1 \in D_{i1} \ldots z_{m-1} \in D_{im-1}$ astfel încât $R_{i0,\,i1}(x,z_1)$, …, $R_{im-1,\,im}(z_{m-1},y)$. **Graf de restrictii orientat** rezultat se numește ***m-arc-consistent***.

Realizarea arc-consistenței pentru un graf de restricții se poate realiza folosind următorii algoritmi:

AC-1:

```
Crează Q ← { (Xi, Xj) | (Xi, Xj) ∈ Mulțime arce, i≠j}
repeat
      modificat = false
      foreach (Xi, Xj) ∈ Q
            modificat = modificat or Verifică(Xk, Xm)
until modificat==false
```

AC-3:

```
Crează Q ← { (Xi, Xj) | (Xi, Xj) ∈ Multime arce, i≠j}
while Q nu este vida
      Elimină din Q un arc (Xk, Xm)
      if Verifică(Xk, Xm) then
            Q ← Q ∪ { (Xi, Xk) | (Xi, Xk)∈Multime arce, i≠k,m}
```

Verifică (X_k, X_m)

```
delete = false
foreach x ∈ Dk
      if nu există nici o valoare y ∈ Dm astfel încât Rk, m(x,y) then
            elimină x din Dk
            delete = true
return delete
```

Pornind de la următoarele notații:

- **N** - numărul de variabile;
- **a** - cardinalitatea maximă a domeniilor de valori ale variabilelor;
- **e** - numărul de restricții.

complexitățile algoritmilor precedenți sunt următoarele:

- Algoritmului de realizare a arc-consistentei - *AC-1* are în cazul cel mai defavorabil complexitatea **O(a^2*N*e)**
- Algoritmului de realizare a arc-consistentei - *AC-3*: complexitate timp este **O(e*a^3)**; complexitate spatiu: **O(e+N*a)**
- Algoritmului de realizare a arc-consistentei - *AC-4* care presupune o îmbunătățire a complexității în timp: **O(e*a^2)**
- Algoritmul de realizare a 2-cale-consistentei - *PC-4*: complexitate timp **O(N^3*a^3)**

În strânsă legătură cu definiția formală a problemei satisfacerii restricțiilor se află noțiunile de grad al problemei şi aritate a problemei satisfacerii restricțiilor. Astfel, *gradul unei variabile* este cardinalitatea domeniului ei de valori, *aritatea unei restricții* este numărul de variabile specificate de restricție; *gradul problemei* este gradul maxim al variabilelor problemei, iar *aritatea problemei* este aritatea maximă a restricțiilor ei.

Mai departe, în cadrul unui graf de restricții se poate defini o ordonare a nodurilor asociate variabilelor problemei, ordonare ce reprezintă ordinea de selectare a variabilelor în algoritmul de backtracking.

Astfel, într-un graf de restricții ordonat putem defini noțiunile de:

- *lățimea unui nod* egală cu numărul de arce care leagă acel nod de nodurile anterioare lui în ordonare;
- *lățimea unei ordonări a nodurilor* egală cu lățimea maximă a nodurilor;
- *lățimea unui graf de restricții* este egală cu valoarea minimă a lățimilor de ordonare a nodurilor, pentru toate posibilitățile de ordonare a nodurilor.

Teoremă. *Dacă un graf de restricții arc-consistent are lățimea egală cu unu (arbore), atunci problema asociată grafului admite o soluție fără backtracking*.

Teorema. *Dacă un graf de restricții 2-cale-consistent are lățimea egală cu doi, atunci problema asociată grafului admite o soluție fără backtracking*.

Fiind dată o ordonare *d* a variabilelor unui graf de restricții R, graful R este *d-arc-consistent* dacă toate arcele având direcția *d* sunt arc-consistente. Fie un graf de restricții R, având ordonarea variabilelor *d* cu lățimea egală cu unu. Daca R este d-arc-consistent atunci căutarea după direcția *d* este fără backtracking.

Realizarea d-arc-consistenței unui graf de restricții cu ordonarea variabilelor (X_1,…,X_N) poate fi realizată în următoarea manieră:

```
for i = N..1
      foreach (Xj, Xi) with j < i
            Verifica(Xj, Xi)
```

Complexitatea în timp a algoritmului este **O(e*a^2)**

3.2 Tehnici prospective

Principiul este simplu: fiecare pas spre soluție nu trebuie să ducă la blocare. Astfel, la fiecare atribuire a variabilei curente cu o valoare corespunzătoare, toate variabilele sunt verificate pentru a depista eventuale condiții de blocare. Anumite valori ale variabilelor neinstanțiate încă pot fi eliminate deoarece nu vor putea să facă parte din soluție niciodată.

Următorii algoritmi analizați implementează strategia de căutare neinformată cu realizarea unor grade diferite de k-consistență.

Backtracking cu predicție completă

```
Predicție(U, F, D)
foreach L of D[U]
      F[U] ← L
      if U < N then
            DNEW ← Verifică_Inainte (U, D[U], D)
            if DNEW != null
                  then DNEW  ← Verifica _Viitoare (U, DNEW)
            if DNEW != null
                  then Predictie (U+1, D, DNEW)

Verifica_Înainte (U, L, D)
inițializează DNEW
for U2 = U+1..N
      foreach L2 of D[U2]
            if Relatie(U, L, U2, L2) == true
                  then introduce L2 in DNEW[U2]
      daca DNEW[U2] vidă
             atunci return null
return DNEW

Verifica_Viitoare (U, DNEW)
if U+1 < N then
      for U1 = U+1..N
            foreach L1 of DNEW[U1]
                  for U2 = U+1..N
                        foreach L2 of DNEW[U2]
                              if Relatie (U1, L1, U2, L2) == true
                              then break L2
```

```
                        if nu s-a gasit o valoare consistenta pentru U2
                        then
                              elimina L1 din DNEW[U1]
                              break U2
            if DNEW[U1] vidă then return null
return DNEW
```

Backtracking-ul cu predictie parțială presupune modificarea doar a funcției Verifică_Viitoare din prisma domeniului de vizibilitate a variabilei U_2 care acum variază exclusiv de la U_1+1, nu direct de la U+1. Rezultatul imediat este înjumătățirea numărului de operații efectuate la nivelul funcției.

```
Verifica_Viitoare (U, DNEW)
…
      for U1 = U+1..N
            foreach L1 of DNEW[U1]
                  for U2 = U1+1..N
                        foreach L2 of DNEW[U2]
                              …
```

Backtracking-ul cu verificare predictivă elimină apelul Verifica_Viitoare(U, DNEW) complet din funcția de Predicție

```
Predicție(U, F, D)
      …
            DNEW ← Verifică_Inainte (U, D[U], D)
            if DNEW != null
                  then DNEW ← Verifica_Viitoare (U, DNEW)
            if DNEW != null
                  …
```

Discuția care se ridică imediat este *care dintre cele 3 metode este mai eficientă*? Părerile sunt împărțite în sensul că uneori costul rafinărilor ulterioare poate fi mai mare decât costul expandării efective a nodului curent, dar totodată se poate obține o reducere semnificativă a numărului de apeluri recursive prin eliminarea unor soluții neviabile. Certitudinea este că oricare dintre aceste metode reduce corespunzător numărul de intrări în stivă, dar trebuie luat în considerare în funcție de specificul problemei și costul operației de Verifica_Viitoare.

Un aspect important este că toate cele trei variante de tehnici prospective pot fi îmbunătățite prin introducerea de euristicii, lucru echivalent cu o reordonare dinamică a variabilelor la fiecare avans în căutare. Experimental, s-a dovedit că introducerea acestor euristici (ex. selecția următoarei variabile urmărind ca aceasta să aibă cele mai puține valori rămase în domeniul propriu) furnizează rezultate foarte bune.

3.3 Euristici

Ordonarea variabilelor urmărește ca variabilele legate prin restricții explicite (specificate de mulțimea de restricții definită în problemă) să fie consecutive:

- în cazul în care se doresc toate soluțiile sunt preferate mai întâi variabilele care apar într-un număr mic de restricții și au domenii de valori cu cardinalitate mică
- în cazul în care se dorește numai o soluție, criteriul este cel opus.

Ordonarea valorilor pleacă de la premisa că nu toate valorile din domeniul variabilelor apar în toate restricțiile:

- în cazul în care se doresc toate soluțiile sunt preferate mai întâi variabilele cele mai restricționate, cu cele mai puține atribuiri posibile;
- în cazul în care se dorește numai o soluție, criteriul este cel contrar

Ordonarea testelor presupune începerea cu variabila precedentă cea mai restricționată.

4. Concluzii şi observaţii

Metodele descrise pot fi aplicate pe o plajă largă de probleme, iar optimizările prezentate pot duce la scăderi drastice ai timpilor de execuție. Combinarea anumitor metode, precum tehnici prospective cu euristici duce la rezultate și mai bune, demonstrate în practică. Astfel, majoritatea problemelor care presupun parcurgeri în spațiul stărilor pot fi abordate pornind de la unul dintre algoritmii descriși.

Responsabil laborator: Mihai Dascălu (mihai.dascalu@cti.pub.ro)

5. Referinţe

[1] Curs BLIA, Prof. Ing. Adina Magda Florea

[2] Introducere in Algoritmi, Thomas H. Cormen; Charles E. Leiserson, Ronald R. Rivest, Cliff Stein (1990)

[3] The Art of Computer Programming, Donald E. Knuth (1968)

[4] CSP Tutorial http://4c.ucc.ie/web/outreach/tutorial.html

[5] The Complexity of Some Polynomial Network Consitency Algorithms for Constraint Satisfaction Problems disponibil la http://cse.unl.edu/~choueiry/Documents/AC-MackworthFreuder.pdf

[6] http://en.wikipedia.org/wiki/Backtracking

[7] http://en.wikipedia.org/wiki/Constraint_satisfaction_problem

[8] Dechter, R., "Enhancements schemes for constraint processing: Backjumping, learning, and cutset decomposition", Artificial Intelligence, 41 (1989/90), pp. 273-312

[9] Reprezentarea si prelucrarea restricţiilor, manuscris de publicat Ștefan Trăușan-Matu

6. Aplicații

Problema analizată este **problema clasică a reginelor**: pe o tablă NxN trebuie așezate N regine astfel încât să nu existe conflicte între oricare 2 piese (o regină poate elimina altă regină dacă segmentul determinat de cele două piese este paralel cu oricare din axele sau diagonalele tablei de joc).

Algoritmul se oprește când se găsește **prima soluție validă.**

Se cere să se modeleze această problemă folosind următoarele abordări:

1. Backtracking simplu
2. BKT + CSP (AC1)
3. BKT + predicție prospectivă parțială + o euristică

Pentru bonus:

1. Implementarea algoritmului AC3 în loc de AC1
2. Se vor rula toate cele 3 metode pe dimensiuni N variabile (4,8,16,32) și se vor înregistra atât ***timpii totali de execuție*** (care pot fi influențați și de alte procese din sistemul de operare, respectiv alocări și procese interne ale JVM, etc.) cât și ***numărul de intrări în recursivitate***. *Se vor figura grafic rezultatele* și se va atribui o *interpretare corespunzătoare*.