Reprezentarea şi prelucrarea restricţiilor

(c) Stefan Trausan-Matu

1. Introducere

Restricţiile sunt un formalism de reprezentare şi rezolvare a problemelor intens folosit pentru modelarea şi rezolvarea problemelor complexe de căutare. Dat fiind faptul că multe probleme des întîlnite în realitate implică rezolvarea unor astfel de probleme de căutare, sunt depuse eforturi intense pentru găsirea unor tehnici şi algoritmi cit mai eficienţi pentru prelucrarea restricţiilor. De exemplu, pentru găsirea unei atribuiri de valori la o mulţime de variabile aflate în anumite relaţii (aşa numita "problemă a etichetării consistente") există deja un număr mare de lucrări publicate, din care amintim doar câteva [MFr85, Nad88, Dec90, DeP85, Fre88, FrQ85].

Să considerăm următoarea problemă [Bea96]:

Fie trei variabile A, B şi C, cu valori în domeniile precizate în figura 1 şi două restricţii, figurate prin segmente de dreaptă: (A=B şi A<C).

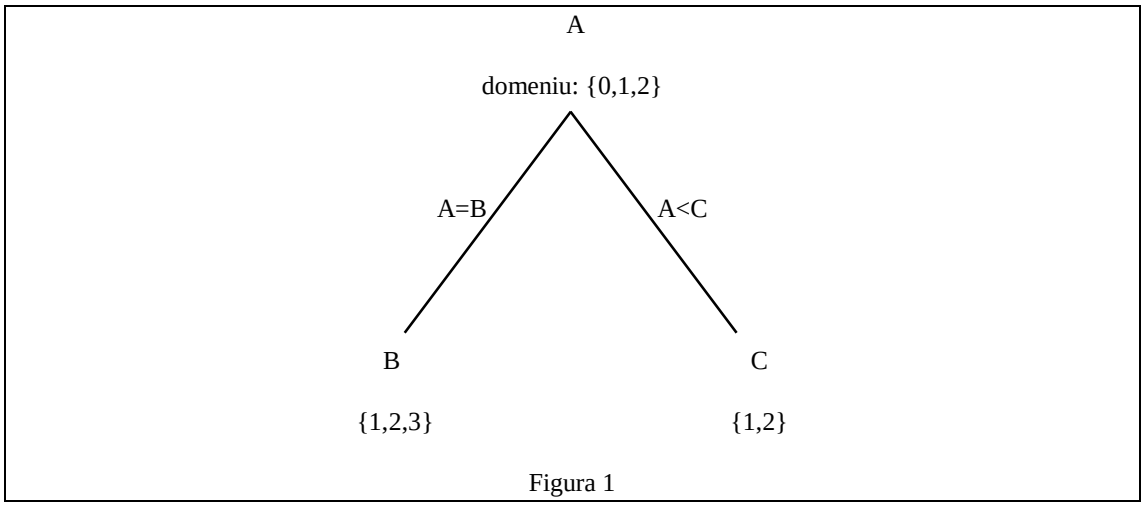

Figura 1

Problema cere să se găsească o atribuire de valori pentru cele trei variabile astfel încât toate restricţiile să fie satisfăcute.

Cel mai simplu algoritm de rezolvare a unei astfel de probleme este “generează şi testează” (GT), care generează toate tripletele de valori pentru variabile (în cazul de faţă 3x3x2=18 triplete de valori) şi verifică dacă se verifică restricţiile.

A,B,C

<0,1,1> nu este satisfăcută restricţia A=B
<0,1,2> nu este satisfăcută restricţia A=B
<0,2,1> nu este satisfăcută restricţia A=B
<0,2,2> nu este satisfăcută restricţia A=B
<0,3,1> nu este satisfăcută restricţia A=B

<0,3,2> nu este satisfăcută restricţia A=B
<1,1,1> nu este satisfăcută restricţia A<C
<1,1,2> SUCCES !
<1,2,1> nu este satisfăcută restricţia A=B
<1,2,2> nu este satisfăcută restricţia A=B
<1,3,1> nu este satisfăcută restricţia A=B
<1,3,2> nu este satisfăcută restricţia A=B
<2,1,1> nu este satisfăcută restricţia A=B
<2,1,2> nu este satisfăcută restricţia A=B
<2,2,1> nu este satisfăcută restricţia A<C
<2,2,2> nu este satisfăcută restricţia A<C
<2,3,1> nu este satisfăcută restricţia A=B
<2,3,2> nu este satisfăcută restricţia A=B

În cazul particular de faţă, cu trei variabile, extrem de simplu, problema poate fi rezolvată şi astfel dar, pentru probleme reale tipice, chiar dacă numărul de variabile şi cardinalul domeniilor asociate nu creşte foarte mult, numărul de combinaţii posibile este imens (de exemplu, în problema zebrei, prezentată ulterior, se ajunge la un număr de 298.023.223.876.953.125 combinaţii).

Pentru rezolvarea acestui tip de probleme, un algoritm des folosit este “backtracking” (BT). Acest algoritm nu mai construieşte toate tuplele şi apoi le verifică (ca la GT), ci le construieşte iterativ, menţinând o evidenţă a alegerilor făcute. În momentul în care se ajunge la o contradicţie (nu este satisfăcută o anumită restricţie), se revine la ultima decizie luată.

De exemplu, în cazul problemei anterioare, cu trei variabile, o rezolvare prin backtracking ar decurge după cum urmează (s-a folosit notaţia <x,y,z> pentru tripletul valorilor celor trei variabile, în ordinea A,B,C):

A=0
  B=1
    C=1 <0,1,1> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=1)
    C=2 <0,1,2> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=2)
    nu mai sunt variante pentru C ➔ revenire la ultima alegere (B=1)
  B=2
    C=1 <0,2,1> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=1)
    C=2 <0,2,2> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=2)
    nu mai sunt variante pentru C ➔ revenire la ultima alegere (B=2)
  B=3
    C=1 <0,3,1> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=1)
    C=2 <0,3,2> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=2)
    nu mai sunt variante pentru C ➔ revenire la ultima alegere (B=3)
  nu mai sunt variante pentru B ➔ revenire la ultima alegere (A=0)
A=1
  B=1
    C=1 <1,1,1> nu este satisfăcută restricţia A<C ➔ revenire la ultima alegere (C=1)
    C=2 <1,1,2> sunt satisfăcute ambele restricţii ➔ SUCCES !

În acest caz, s-a ajuns la soluţie după 7 încercări nereuşite, la fel ca la GT. Dacă, însă, se consideră o altă ordine a variabilelor în căutare, se poate ajunge la soluţie mai repede:

B=1
 A=0
    C=1 <0,1,1> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=1)
    C=2 <0,1,2> nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (C=2)
    nu mai sunt variante pentru C ➔ revenire la ultima alegere (A=0)
 A=1

C=1 <1,1,1> nu este satisfăcută restricţia A<C ➔ revenire la ultima alegere (C=1)
C=2 <1,1,2> sunt satisfăcute ambele restricţii ➔ SUCCES !

Există diverse posibilităţi de a îmbunătăţi BT, în afara alegerii unei ordini adecvate a considerării variabilelor. O primă posibilitate este să se testeze restricţiile de fiecare dată când s-a extins soluţia şi nu doar după ce s-a construit un întreg tuplu:

A=0
B=1
nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (B=1)
B=2
nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (B=2)
B=3
nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (B=3)
nu mai sunt variante pentru B ➔ revenire la ultima alegere (A=0)
A=1
B=1
C=1 <1,1,1> nu este satisfăcută restricţia A<C ➔ revenire la ultima alegere (C=1)
C=2 <1,1,2> sunt satisfăcute ambele restricţii ➔ SUCCES !

sau, pentru cea de-a doua ordine de alegere a variabilelor:

B=1
A=0
nu este satisfăcută restricţia A=B ➔ revenire la ultima alegere (A=0)
A=1
C=1 <1,1,1> nu este satisfăcută restricţia A<C ➔ revenire la ultima alegere (C=1)
C=2 <1,1,2> sunt satisfăcute ambele restricţii ➔ SUCCES !

Bineînţeles că numărul de calcule efectuate de BT pentru găsirea unei soluţii este, în general, mai mic decât la "generează şi testează".

Ce se observă din rezolvările de mai sus este faptul că se fac multe calcule inutile. De exemplu, atribuirea parţială de valori A=0 (sau, în altă notaţie, <0, , >), nu poate face parte din nici o soluţie dar este testată în mod repetat şi evident inutil. La o simplă analiză a problemei se poate observa că valoarea 0 din domeniul lui A nu trebuie încercată, pentru că în domeniul lui B nu există această valoare şi, prin urmare, datorită restricţiei A=B, nu poate face parte din nici o soluţie. Mai mult, şi valoarea 2 din domeniul lui A nu trebuie considerată, pentru că, datorită restricţiei A<C şi a faptului că nu există în domeniul lui C o valoare mai mare ca 2, nici ea nu poate face parte din soluţiile problemei. Prin urmare, domeniul lui A se poate reduce la o singură valoare: {1}.

Analiza problemei merge mai departe. Reducerea domeniului lui A la {1} are influenţe asupra domeniilor celorlalte variabile. Datorită restricţiei A=B, domeniul variabilei B poate fi redus şi el la {1}. Datorită restricţiei A<C, domeniul variabilei C este redus la {2}. Ce s-a obţinut, de fapt, este chiar rezolvarea problemei, doar prin considerarea restricţiilor şi eliminarea valorilor imposibile, printr-o "filtrare" (vezi secţiunea ….). O tratare sistematică a filtrării domeniilor de valori ale variabilelor, după ideea de mai sus, se poate face prin algoritmii de satisfacere a restricţiilor. Aceşti algoritmi consideră o reprezentare a problemei sub forma unei reţele de restricţii şi, în esenţă, folosesc informaţiile locale furnizate de topologia acestor reţele şi de restricţii pentru a elimina valori incompatibile din domeniile de valori ale variabilelor problemei. De exemplu, graful din figura 1 este o reţea cu două restricţii.

Nu trebuie să sperăm că întotdeauna o problemă complexă poate fi rezolvată doar prin filtrări. Ce se poate obţine sigur însă este reducerea spaţiului de căutare, uneori chiar spectaculos, după cum se va vedea din acest capitol. De exemplu, să considerăm acum aşa numita problemă a zebrei [Dec90], 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 şi ţigări distincte.
2. Englezul locuieşte în casa roşie.
3. Spaniolul are un câine.
4. Cafeaua este băută în casa verde.
5. Ucrainianul 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 şi cine are o zebră?

Această problemă se referă la 5 case iar fiecare casă are 5 atribute specifice. Fiecare atribut are cinci valori:

1. roşu, albastru, galben, verde, alb
2. norvegian, ucrainian, englez, spaniol, japonez
3. cafea, ceai, apă, lapte, oranjadă
4. zebra, câine, cal, vulpe, melci
5. Kent, Marlboro, West, Chesterfield, Lucky-Strike

Prin urmare, avem 5 case x 5 atribute/casă=25 variabile. Cum fiecare variabilă poate avea 5 valori, numărul maxim de tuple care pot fi generate şi care trebuie verificate este 5^{25}=298.023.223.876.953.125. Dacă ar fi efectuate o sută de mii de calcule pe secundă, rezolvarea problemei ar necesita:

((((298.023.223.876.953.125 / 100.000) / 3.600) / 24) / 365) = 94.502,54 ani.

Folosind însă restricţiile problemei, o parte din ele fiind figurate prin arce în figura 2, se poate obţine rapid o soluţie (vezi [Dec90, GhT…..] şi secţiunea …..).

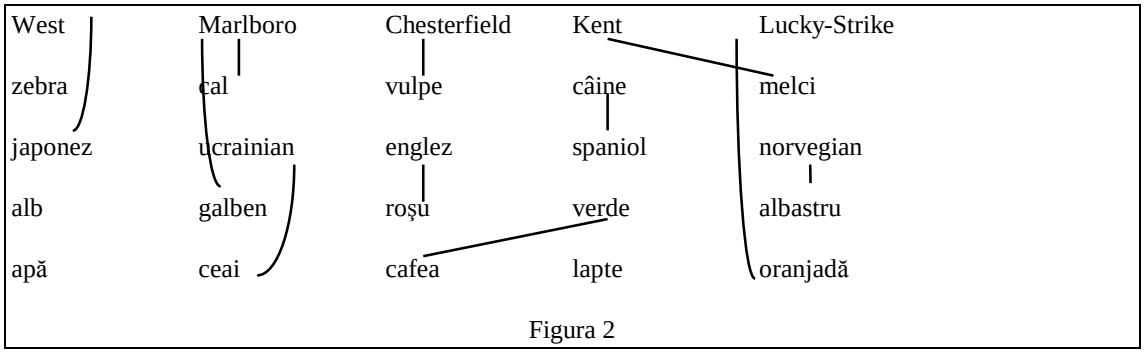

Figura 2

Reţelele de restricţii constituie nu numai un punct de plecare pentru algoritmi eficienţi de rezolvare de probleme ci şi o modalitate de exprimare declarativă a cunoştinţelor, care permite scoaterea în evidenta a relaţiilor locale între componente. Prelucrarea restricţiilor este un proces de utilizare a acestei clase de cunostinte pentru a deriva consecinţe utile şi a produce o comportare inteligenta. Existenţa unui modul specializat în acest scop într-un mediu de programare pentru dezvoltarea de sisteme bazate pe cunoştinţe

facilitează reprezentarea unor modele cauzale ale domeniilor abordate şi implementarea unor regimuri de rezolvare calitativă (de exemplu, fizică calitativă) a problemelor specifice.

Multe probleme din domeniul inteligentei artificiale pot fi definite sau reformulate în termenii prelucrării restricţiilor. Prelucrarea restricţiilor a fost folosita în mai multe domenii ale inteligentei artificiale cum ar fi recunoaşterea formelor [Dav87], analiza circuitelor electronice [deK84], raţionamente calitative în fizică [deK86b], simulări interactive, construirea de interfeţe grafice [Bor85, EpL88, SzM88] ordonanţarea producţiei [Fox83, FAS82, FSB89], proiectare [MiF90, MuS90, Ste81] şi diagnosticare. Una din ideile comune acestor abordări este modelarea unor tipuri de raţionamente utilizate de obicei de experţi în domeniile respective. Pentru rezolvarea unei probleme, aceştia, în general, nu apelează la instrumente matematice complexe cum ar fi rezolvarea de sisteme complexe de ecuaţii. Pentru a găsi rapid o soluţie ei apelează, în schimb, la raţionamente calitative, ce folosesc structura sistemului analizat, interacţiunile dintre componente, propagarea cauzala a modificărilor între acestea.

Pentru ilustrarea celor spuse mai sus dăm mai jos un exemplu simplu de sistem expert pentru diagnosticarea unui uscător de păr folosind raţionamente calitative pe modele cauzale. Acest sistem are cinci componente:

- un motor,
- o rezistenţă de încălzire,
- un cablu,
- o priză,
- o siguranţă.

Fiecare din aceste cinci obiecte poate fi descris ca o restricţie care instanţiază o restricţie generică, să-i spunem “element”. Acestă restricţie are următoarele terminale (variabile):

- funcţionează (F),
- bun (B),
- conectat (C),
- are_curent (A).

Fiecare din aceste variabile poate avea valoarea da, nu sau nedefinit. Reţeaua de restricţii va avea configuraţia din figura 3. Trebuie remarcat un fapt foarte important: Conexiunile între elemente reflectă relaţii cauzale între stări de fapt, cum ar fi faptul că, dacă un element este funcţionează, elementele conectate la el ar trebui să funcţioneze.

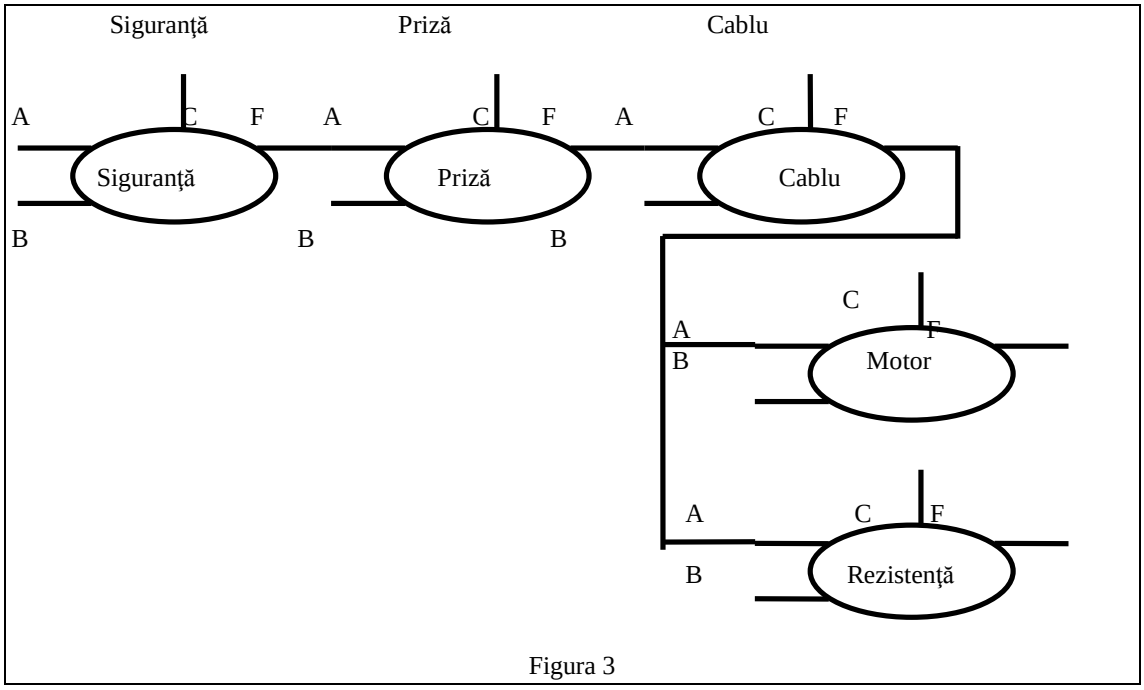

Figura 3

Restricţiile valabile pentru toate elementele sunt descrise de următoarele funcţii

Bun ← **dacă** conectat **şi** are_curent
atunci funcţionează
altfel dacă are_curent
atunci tipăreşte “Element neconectat!”
nedefinit

Conectat ← **dacă** merge
atunci da
altfel dacă bun **şi** are_curent
atunci nu
altfel nedefinit

Are_curent ← **dacă** funcţionează
atunci da
altfel dacă bun **şi** conectat
atunci nu
altfel dacă not conectat
atunci tipăreşte “Element neconectat!”
nedefinit

Funcţionează ← bun **şi** conectat **şi** are_curent

Se pot face diverse prelucrări, de exemplu:

1) Este curent la siguranta, e bună siguranţa, nu se ştie dacă e bun cablul, nu se ştie dacă e bună rezistenţa şi motorul, rezistenţa merge dar motorul nu, toate sunt conectate.

2) Toate sunt conectate, siguranţa, cablul şi ştecherul sunt bune, motorul merge dar rezistenţa nu.

Reţeaua de restricţii de mai sus, pentru modelarea unui uscător de păr, deşi foarte simplă, ilustrează posibilităţile de utilizare a tehnicilor de propagare a restricţiilor pentru simulări calitative.

Ne-am propus ca în acest capitol să analizam problema prelucrării restricţiilor din cât mai multe perspective. In secţiunea următoare vom trece în revista elementele comune claselor de probleme de prelucrare a restricţiilor. Particularitatile fiecărei clase vor fi discutate în secţiunea a 3-a.

2. Restricţii, reţele de restricţii, probleme de prelucrarea restricţiilor

Definiţia 1. O **restricţie** c este o relaţie între mai multe **variabile** $v_1,\ldots,v_m$, (denumite nodurile sau celulele restricţiei). Fiecare variabila v_i poate lua valori într-o anumita mulţime D_i, denumită **domeniul** ei (ce poate fi finit sau nu, numeric sau nu).

Definiţia 2. Se spune că un tuplu (o atribuire) de valori (x_1, … x_m) din domeniile corespunzătoare celor m variabile **satisface** restricţia $c(v_1, \ldots, v_m)$, dacă (x_1, … x_m)$\in$ $c(v_1, \ldots, v_m)$.

Conform definiţiei de mai sus, o restricţie $c(v_1, \ldots, v_m)$ este o submulţime a produsului cartezian D_1 x D_2 x ... x D_m, constând din toate tuplele de valori considerate că satisfac restricţia c pentru submulţimea (v_1, . . ., v_m). Restricţiile pot fi exprimate prin:

- enumerarea tuplelor restricţiei,
- formule matematice, cum ar fi ecuaţiile sau inecuaţiile,
- precizarea unei mulţimi de reguli.

De exemplu, prima restricţie din problema din figura 1 ar putea fi exprimată prin:

{<1,1>,<2,2>}
A=B
{Dacă A=x atunci B=x; Dacă B=x atunci A=x}

Un exemplu de restricţie ale cârei variabile iau valori în mulţimi nenumerice, poate fi modelul unui tranzistor. Acesta are trei celule: colectorul, emitorul şi baza, valorile posibile pentru fiecare celula fiind sensul variaţiei semnalului la cele trei terminale (plus, minus sau zero). Acest exemplu este specific aplicaţiilor care modelează raţionamente calitative, în care nu se prelucrează valori numerice.

Un exemplu de restricţie cu n variabile sunt ecuaţiile cu n necunoscute (cu soluţii întregi, reale, booleene etc.). Restricţiile cu doua variabile, denumite **restricţii binare**, sunt domeniul predilect de studiu al diferiţilor algoritmi de prelucrare a restricţiilor [MFr85, Nad88, Dec90]. Restricţiile cu doar o singura variabila, denumite **restricţii unare**, sunt echivalente cu specificarea unui domeniu pentru variabila respectiva.

Definiţia 3. O **problemă de satisfacere a restricţiilor** (PSR) este un triplet <V,D,C>, format dintr-o mulţime V formată din n variabile $V=\{v_1, \ldots, v_n\}$, mulţimea D a domeniilor de valori corespunzătoare acestor variabile: $D=\{D_1, \ldots D_n\}$ şi o mulţime C de restricţii $C=\{c_1, \ldots, c_p\}$ între submulţimi ale V ($c_i(v_{i1}, \ldots, v_{ij}) \subseteq D_{i1}$ x D_{i2} x ... x D_{ij}). Conform definiţiei anterioare, o restricţie $c_i(v_{i1}, \ldots, v_{ij})$, este o submulţime a produsului cartezian D_{i1} x D_{i2} x ... x D_{ij}, constând din toate tuplele de valori considerate că satisfac restricţia pentru (v_{i1}, . . ., v_{ij}).

Definiţia 4. O **soluţie a unei PSR** <V,D,C> este un tuplu de valori <x_1, … , x_n> pentru toate variabilele V, din domeniile corespunzătoare D, astfel încât toate restricţiile din C să fie satisfăcute.

Altfel spus, problemele de satisfacerea restricţiilor au ca obiect un număr de restricţii care au celule (variabile) comune. Scopul urmărit este de a satisface toate restricţiile, adică de a găsi pentru toate

restricţiile tuple de valori care verifică relaţiile respective.

Există mai multe reprezentări posibile pentru o PSR. Primele două reprezentări discutate mai jos sunt bidimensionale, sub formă de graf, pe care le vom denumi **reţele de restricţii**. Cea de-a treia este o reprezentare simbolică, liniarizată.

a) Sub formă de graf neorientat, reprezentare folosită în figura 1, în care nodurile sunt variabilelei restricţiei iar arcele sunt restricţiile problemei. Această reprezentare este posibilă doar pentru PSR binare.

b) O altă reprezentare a unei PSR este printr-un graf care are ca noduri restricţiile şi ca arce legaturile între restricţii (respectiv celulele comune). De exemplu, PSR din figura 1 poate fi reprezentată şi prin graful din figura 2:

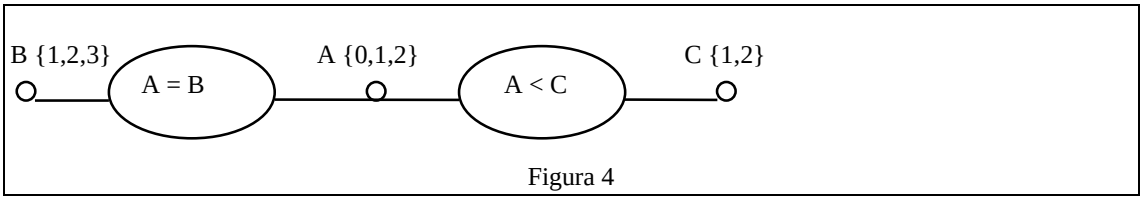

Figura 4

Avantajul unei astfel de reprezentări este faptul că ea permite şi utilizarea de restricţii n-are, în general, nu numai a acelor binare, ca în reprezentarea anterioară. De exemplu, ……..

c) Restricţiile sunt exprimate, de obicei, nu sub forma de tuple, ci prin notaţii matematice uzuale, cum ar fi ecuaţiile şi ineacuaţiile. PSR din figura 1 poate fi exprimată doar prin tripletul:

<{A,B,C},{{0,1,2},{1,2,3},{1,2}},{A=B,A<C}>

Ecuaţiile, inecuaţiile sau sistemele de ecuaţii şi inecuaţii sunt tot atâtea probleme de satisfacere a restricţiilor. În acest context, se pune întrebarea: Nu s-ar putea folosi tehnicile “clasice” de rezolvare a ecuaţiilor şi sistemelor de ecuaţii sau inecuaţii în rezolvarea oricărei PSR? Un răspuns poate fi dat plecând de la faptul că ecuaţiile şi sistemele de ecuaţii sunt doar un caz foarte particular de PSR. Există multe PSR caracterizate prin domenii de valori discrete (numerice sau nu) pentru variabile (şi nu continue, ca la ecuaţiile sau inecuaţiile algebrice), domenii nu neapărat aceleaşi pentru toate variabilele. În plus, restricţiile nu sunt limitate la ecuaţii şi inecuaţii, ele pot fi relaţii calitative şi nu cantitative (ca la ecuaţii, inecuaţii sau sisteme).

Reţelele de restricţii sunt o modalitate de reprezentare declarativa a cunostintelor referitoare la interconexiunile între componente, la domeniile de valori posibile pentru parametrii, la modul cum funcţionarea unui ansamblu este determinata de structura sa şi de comportarea componentelor sale. Reţelele de restricţii pot exprima legaturile cauzale între pârtile unui întreg. In urma modificării unei parţi aceste legaturi pot fi folosite pentru a propaga consecinţele modificării în scopul menţinerii coerentei întregului.

Mecanismul de inferenţa utilizat de un sistem de prelucrare a restricţiilor este **propagarea restricţiilor**. Aceasta consta într-un ciclu în care informaţiile referitoare la valorile nodurilor sunt propagate în reţea. Prin propagare se înţelege asignarea de valori variabilelor unei restricţii astfel încit aceasta să fie satisfăcuta şi, pe baza legaturilor între restricţii, asignarea de valori la restricţiile adiacente. Ciclul se încheie când nu mai au loc actualizări sau când este îndeplinita alta condiţie de terminare (de exemplu s-a ajuns la o incompatibilitate - exista restricţii ce nu pot fi satisfăcute).

Toate sistemele de prelucrarea restricţiilor oferă un număr de proprietati valoroase [Dav87]:

- Propagarea restricţiilor se bazează pe o structura de control simpla, aplicata unor module care fac actualizări simple. De aceea este uşor de codificat, de analizat şi de extins.

- Prelucrările sunt uşor de paralelizat (actualizarea poate fi făcuta simultan în mai multe locuri în reţea).

- Sunt adecvate unei abordării incrementale. Restricţii pot fi adăugate incremental în reţea, argumentele lor pot fi actualizate, efectele fiind propagate în reţea.

- Oprirea algoritmului de propagare într-un moment oarecare furnizează informaţii utile asupra stadiului prelucrărilor. Trasarea procesului de propagare a restricţiilor poate da informaţii utile asupra fluxului modificărilor în urma unei actualizări în reţea.

3. Clase de probleme de satisfacere a restricţiilor

Rezolvarea unei PSR poate cere găsirea unei singure soluţii ori a tuturor soluţiilor. În general, restricţiile pot avea oricâte variabile. O PSR este **binară** dacă fiecare restricţie este unară sau binară (implică cel mult două variabile). Studiile formale asupra algoritmilor de rezolvare a PSR şi a complexitatii acestora au fost făcute în general pentru probleme cu restricţii binare [MFr85, Nad88, Dec90].

O altă clasificare a PSR poate fi în funcţie de **caracterul static sau dinamic al definiţiei PSR**. In primul caz, configuraţia problemei (tripletul <V,D,C>) este fixă pe parcursul întregului proces de rezolvare de probleme. În cel de-al doilea caz reţeaua este modificată dinamic, pe parcursul rezolvării problemei. Modificarea reţelei poate consta în schimbarea restricţiilor (de exemplu în probleme de relaxare a restricţiilor [Fox83] - una sau mai multe restricţii sunt modificate, relaxate, în scopul ieşirii dintr-o situaţie de incompatibilitate) sau chiar a topologiei reţelei, prin adăugarea sau eliminarea de restricţii [Ste81]. O variantă

În sfârşit, o ultimă clasificare a PSR poate fi făcută în funcţie de **prelucrările efectuate în cadrul unui pas al ciclului de propagare**. O prima clasa este cea în care fiecare nod al reţelei are asignata la început o mulţime (finita sau nu) de valori posibile, la un pas al propagării filtrându-se mulţimile valorilor nodurilor conectate la o restricţie astfel încit aceasta să fie satisfăcuta [Dav87]. În cea de a doua clasă de probleme, fiecare nod are ataşat în orice moment cel mult o valoare, propagarea făcându-se prin calculul (pe baza unei restricţii legate la un nod) şi asignarea unei valori pentru nodul respectiv în funcţie de valorile celorlalte noduri ale restricţiei. În acest caz, dacă restricţiile sunt relaţii algebrice şi la un moment dat nu sunt cunoscute valorile pentru suficiente noduri, o posibilitate de a extinde prelucrările este de a face propagări de expresii simbolice, ce conţin variabile introduse temporar, ale căror valori pot fi deduse ulterior [SuS80].

3.1 Probleme de filtrare

Caracteristic problemelor de filtrare este faptul că variabilele (celulele reţelei) au domenii finite de valori. Prin propagare se filtrează aceste mulţimi (prin filtrarea unei mulţimi înţelegem eliminarea de elemente din ea conform unui criteriu dat) astfel încit, în final, toate restricţiile să fie satisfăcute. În aceste probleme, propagarea poate lua în considerare repetat aceleaşi celule ale aceloraşi restricţii, procesul oprindu-se în momentul când nu se mai efectuează modificări ale mulţimilor asociate celulelor. Oprirea procesului poate fi făcuta şi conform altor criterii. De exemplu, faptul că una din mulţimi devine vida indică ajungerea la o incompatibilitate, procesul de propagare putând fi oprit.

Algoritmii folosiţi în probleme de filtrare sunt denumiţi algoritmi de consistenta reţelelor. Un exemplu uzual de astfel de algoritm este cel al lui Waltz [Dav87]. Un algoritm de K-consistenta înlătura toate inconsistentele submulţimilor de mărime K ale unei reţele cu n variabile. Pentru K = 1,2,3 algoritmii sunt denumiţi de **consistenţa nodurilor**, **arcelor** şi respectiv **căilor**.

Să considerăm o PSR binară cu n variabile, cu n restricţii unare, cu r restricţii binare şi reprezentarea ei prin reţeaua de restricţii G, cu variabile drept noduri şi restricţii drept arce. Să notăm cu D_i domeniul variabilei I,

cu Q_i un predicat care verifică restricţia unară pe variabila I şi cu P_{ij} predicatul care reprezintă restricţia binara pe variabilele i şi j. Acesta corespunde unei muchii între nodurile i şi j din graful de restricţii G. O muchie între i şi j se înlocuieşte cu arcele orientate de la i la j şi de la j la i. Consideram că D_i are mărimea maximă a, şi nu exista nici o structura interna a lui D_i sau P_{ij} care să poată fi folosita pentru o eficientizare a algoritmului.

Un algoritm pentru consistenta nodurilor, care se termina în O(an), este NC-1 [MFr85]:

```
procedura NC(i)este:
  pentru fiecare x ∈ Di
   repetă
    dacă not Qi (x)
      atunci şterge x din Di
 sfârşit;
Sfârşit.

Algoritm NC-1 este:
  pentru i ← 1 până la n executa NC(i)
  sfârşit;
Sfârşit.
```

Acest algoritm elimină din domeniul de valori al fiecărui nod valorile care nu satisfac restricţiile care au ca argument variabila din nodul respectiv.

Pentru consistenţa arcelor au fost propuşi mai mulţi algoritmi. Algoritmii prezentaţi în această carte folosesc amândoi funcţia de filtrare REVISE:

```
funcţia REVISE ((i,j)) este:
 STERS ← fals
 pentru fiecare x ∈ Di execută
   dacă nu exista y ∈ Dj a.i. Pij(x,y)
    atunci
      şterge x din Di;
      STERS ← adevărat
 sfârşit;
 întoarce STERS
Sfârşit.
```

Funcţia REVISE este apelată pentru un arc al grafului de restricţii (binare) şi şterge acele valori din domeniul de definiţie al unei variabile pentru care nu este satisfăcută restricţia pentru nici o o valoare corespunzătoare celelaltei variabile a restricţiei.

De exemplu, dacă apelăm REVISE((A,B)) în problema din figura 1, execuţia va decurge ca mai jos:

Înainte de intra, domeniile variabilelor sunt: pentru A {0,1,2} şi pentru B {1,2,3}

STERS ← fals
x=0
  y=1 A=B nu este satisfăcut
  y=2 A=B nu este satisfăcut
  y=3 A=B nu este satisfăcut
  deoarece A=B nu a fost satisfăcut de nici o valoare → şterge x=0 din A ; STERS ← adevărat
x=1
  y=1 A=B este satisfăcut

x=2
  y=1 A=B nu este satisfăcut
  y=2 A=B este satisfăcut
întoarce STERS=adevărat

Efectul lateral al funcţiei este ştergerea lui 0 din A deoarece nu există o valoare corespunzătoare în B pentru care restricţia să fie satisfăcută.

REVISE face cel mult a^2 evaluări ale lui P_{ij}, deci are o complexitate $O(a^2)$.

Un prim algoritm de consistenţa arcelor este cel denumit în literatura de specialitate prin acronimul AC-1 ("Arc Consistency" -1):

Algoritm AC-1 **este**:
 NC-1;
 Q <- {(i,j) | (i,j)∈ arce(G), i ≠ j}
 repeta
  executa
   SCHIMBAT <- fals
   pentru fiecare (i,j)∈ Q
    executa SCHIMBAT <- (REVISE ((i,j)) sau SCHIMBAT)
   sfârşit
 până non SCHIMBAT
Sfârşit.

Acest algoritm apelează mai întâi algoritmul de consistenţa nodurilor, după care apelează REVISE pentru fiecare din arcele grafului. Se repetă acest ciclu de apelări ale lui REVISE pentru fiecare din arcele grafului până când nu mai există nici o eliminare de valori din domeniile variabilelor.

Pentru problema din figura 1, execuţia algoritmului decurge după cum urmează:

	A	B	C
Consistenţa nodurilor nu elimină nimic deoarece în problemă nu există restricţii unare. Q←{(A,B),(B,A),(A,C),(C,A)} SCHIMBAT ← fals	{0,1,2}	{1,2,3}	{1,2}
SCHIMBAT ← REVISE((A,B)) sau SCHIMBAT=adevărat efect lateral → 0 este şters din A	{1,2}	{1,2,3}	{1,2}
SCHIMBAT ← REVISE((B,A)) sau SCHIMBAT=adevărat efect lateral → 3 este şters din B	{1,2}	{1,2}	{1,2}
SCHIMBAT ← REVISE((A,C)) sau SCHIMBAT=adevărat efect lateral → 2 este şters din A	{1}	{1,2}	{1,2}
SCHIMBAT ← REVISE((C,A)) sau SCHIMBAT=adevărat efect lateral → 1 este şters din C	{1}	{1,2}	{2}
Deoarece SCHIMBAT este adevărat, se reia ciclul SCHIMBAT ← fals	{1}	{1,2}	{2}
SCHIMBAT ← REVISE((A,B)) sau SCHIMBAT=fals	{1}	{1,2}	{2}
SCHIMBAT ← REVISE((B,A)) sau SCHIMBAT=adevărat efect lateral → 2 este şters din B	{1}	{1}	{2}
SCHIMBAT ← REVISE((A,C)) sau SCHIMBAT=adevărat	{1}	{1}	{2}
SCHIMBAT ← REVISE((C,A)) sau SCHIMBAT=adevărat	{1}	{1}	{2}
Deoarece SCHIMBAT este adevărat, se reia ciclul SCHIMBAT ← fals	{1}	{1}	{2}

SCHIMBAT ← REVISE((A,B)) sau SCHIMBAT=fals	{1}	{1}	{2}
SCHIMBAT ← REVISE((B,A)) sau SCHIMBAT=fals	{1}	{1}	{2}
SCHIMBAT ← REVISE((A,C)) sau SCHIMBAT=fals	{1}	{1}	{2}
SCHIMBAT ← REVISE((C,A)) sau SCHIMBAT=fals	{1}	{1}	{2}
Deoarece SCHIMBAT=fals, se iese din ciclu şi din algoritm	{1}	{1}	{2}

Să vedem acum care este complexitatea algoritmului AC-1.

Pentru ciclul mare sunt cel mult na iteraţii deoarece cazul cel mai defavorabil este cel în care se elimină la o iteraţie o singură valoare şi nu există mai mult de na valori pentru variabile.

La fiecare iteraţie se apelează de 2r ori REVISE, care, am văzut mai sus, are complexitatea $O(a^2)$.

În consecinţă, complexitatea lui AC-1 este $O(a^3nr)$

Pentru un graf de restricţii complet (cum ar fi, de exemplu, pentru problema reginelor pe tabla de şah cu n linii şi coloane), $r=n(n-1)/2$ ceea ce duce la o complexitate de $O(a^3n^3)$ pentru AC-1. Deoarece multe grafuri de restricţii sunt rare (numărul de restricţii este liniar în funcţie de numărul de noduri - de exemplu, grafurile planare) r este O(n) şi astfel, AC-1 se termină într-un timp de $O(a^3n^2)$.

Algoritmul lui Waltz, folosit pentru recunoaşterea configuraţiile spaţiale din imagini planare, prezentat în [Dav87], este un al doilea exemplu de algoritm de consistenta arcelor, notat de obicei prin AC-2. Algoritmul AC-3, o generalizare a algoritmului lui Waltz, se termina într-un timp de $O(a^3r)$ [MFr85]:

algoritm AC-3 **este**:
NC-1;
Q <- {(i,j) | (i,j) ∈ arce(G), i ≠ j}
cât timp Q nevid **execută**
Selectează şi şterge un arc (k,m) din Q;
dacă REVISE ((k,m))
atunci Q <- Q ∩ {(i,k) | (i,k)∈ arce(G), i ≠ k, i ≠ m, }
sfârşit
Sfârşit.

Structura de date Q pentru acest algoritm este o coadă. Spre deosebire de AC-1, un arc pentru care s-a apelat REVISE este şters din coadă. În schimb, dacă REVISE întoarce adevărat (adică au fost şterse valori), se adaugă în coada Q la sfârşit toate arcele care punctează către nodul de plecare al arcului considerat, cu excepţia situaţiei când ele deja există în Q sau când sunt inversele arcului deja considerat. Terminarea algoritmului se face în momentul în care nu mai există arce în coada Q.

	A	B	C	Q
Consistenţa nodurilor nu elimină nimic deoarece în problemă nu există restricţii unare. Se construieşte Q	{0,1,2}	{1,2,3}	{1,2}	{(A,B),(B,A),(A,C),(C,A)}
selectează şi şterge (A,B) REVISE((A,B))→ adevărat efect lateral → 0 este şters din A	{1,2}	{1,2,3}	{1,2}	{(B,A),(A,C),(C,A)}
selectează şi şterge (B,A) REVISE((B,A)) → adevărat efect lateral → 3 este şters din B	{1,2}	{1,2}	{1,2}	{(A,C),(C,A)}
selectează şi şterge (A,C) REVISE((A,C)) → adevărat	{1}	{1,2}	{1,2}	{(C,A),(B,A)}

efect lateral → 2 este şters din A se adaugă (B,A) în Q				
selectează şi şterge (C,A) REVISE((C,A)) → adevărat efect lateral → 1 este şters din C	{1}	{1,2}	{2}	{(B,A)}
selectează şi şterge (B,A) REVISE((B,A)) →adevărat efect lateral → 2 este şters din A se adaugă (C,A) în Q	{1}	{1}	{2}	{(C,A)}
selectează şi şterge (C,A) REVISE((C,A)) →fals Sfârşit	{1}	{1}	{2}	

Se vede uşor că AC-3 este mai eficient ca AC-1 deoarece face mult mai puţine apeluri ale lui REVISE. AC-3 apelează REVISE o dată pentru fiecare arc din graful G şi practic propagă local modificările făcute de REVISE prin adăugarea unor arce potenţial afectate în coada arcelor considerate.

O altă variantă de algoritm de consistenţa arcelor, o îmbunăţăţire a lui AC-1, este denumit AC-4. Acest algoritm are o complexitate şi mai mică, de $O(ra^2)$ [Bea96]. El are trei paşi. În primul pas este apelat algoritmul de consistenţă a noduri. În al doilea pas se consideră toate arcele grafului de restricţii şi, pentru fiecare arc toate combinaţiile de valori posibile, eliminându-se valorile a căror prezenţă nu este sprijinită de valori compatibile cu ele în restricţii (similar cu funcţia REVISE). În plus fată de REVISE, în acest al doilea pas se construieşte o evidenţă a numărului de valori care sprijină fiecare valoare a unei variabile pentru fiecare restricţie, precum şi o listă a valorilor srijinite de fiecare valoare. Pentru aceasta sunt folosite matricile SRIJINIRI, SPRIJINIT-DE şi E\SEC.

SPRIJINIRI [variabilă, valoare, restricţie]

ia valorile 0,1,2, …, semnificând numărul de valori ale altor variabile a căror valori sunt sprijinite de restricţie şi de existenţa valorii variabilei respective.

De exemplu, în problema cu trei variabile din figura 1, SPRIJINIRI [C,2,A<C] = 2 deoarece valoarea 2 a variabilei C este sprijinită de două valori (A=0 şi A=1) în restricţia A<C.

Pentru fiecare valoare a unei variabile este menţinută o listă:

SPRIJINIT-DE [variabilă, valoare]

ale cărui elemente conţin triplete <variabilă, valoare, restricţie> sprijinită de valoarea variabilei precizată.

De exemplu, SPRIJIN[A,1]={(C,2,A<C),(B,1,A=B)}

E\SEC păstreză lista perechilor (variabilă, valoare) care nu sunt sprijinite.

În cel de-al treilea pas al algoritmului AC-4, pe baza informaţiilor din cele trei matrici, se elimină valorile care nu sunt sprijinite.

algoritm AC-4 **este**:

;; PAS I

NC-1;
Iniţializează SPRIJINIRI [variabilă, valoare, restricţie]=0
Iniţializează SPRIJINIT-DE [variabilă, valoare]=nil
Iniţializează E\SEC=nil

;; PAS II

pentru fiecare arc(i,j) **execută**
 pentru fiecare $x \in D_i$ **execută**
  pentru fiecare $y \in D_j$ **execută**
   dacă $P_{ij}(x,y)$
    atunci
     incrementează SPRIJINIRI [i, x, (i,j)]
     adaugă (i, x, (i,j)) în SPRIJINIT-DE [j, y]
  sfârşit
  dacă SPRIJINIRI [i, x, (i, j)]=0
   atunci
    adaugă (i, x) în E\SEC
    elimină x din D_i
 sfârşit
sfârşit

;; PAS III

cât timp E\SEC $\neq$ { } **repetă**
  Extrage (i, x) din E\SEC
  S$\leftarrow$ SPRIJINIT-DE [i, x]
  pentru fiecare (j, y, (j, i)) din S **repetă**
   Decrementează SPRIJINIRI [j, y, (j, i)]
   dacă SPRIJINIRI [j, y, (j, i)] = 0
    şi (j, y) $\notin$ E\SEC
    şi $y \in D_j$
    atunci
     adaugă (j, y) în E\SEC
     elimină j din D_j
 sfârşit
sfârşit
Sfârşit.

Să exemplificăm algoritmul AC-4 pe problema din figura 1.

După pasul I, matricea SPRIJINIRI are, următorile valori:

A,0, (A,B)	A,0, (A,C)	A,1, (A,B)	A,1, (A,C)	A,2, (A,B)	A,2, (A,C)	B,1, (B,A)	B,2, (B,A)	B,3, (B,A)	C,1, (C,A)	C,2, (C,A)
0	0	0	0	0	0	0	0	0	0	0

matricea SPRIJINIT-DE are valorile:

A,0	nil

A,1 nil
A,2 nil
B,1 nil
B,2 nil
B,3 nil
C,1 nil
C,2 nil

şi E\SEC=nil

Să considerăm că arcele grafului sunt în ordinea (A,C), (C,A), (A,B), (B,A).

Înainte de decizia din prima iteraţie din ciclul mare al pasului II, au fost executate ciclurile imbricate pentru arcul (A,C) şi matricele au valorile:

SPRIJINIRI

A,0, (A,B)	A,0, (A,C)	A,1, (A,B)	A,1, (A,C)	A,2, (A,B)	A,2, (A,C)	B,1, (B,A)	B,2, (B,A)	B,3, (B,A)	C,1, (C,A)	C,2, (C,A)
0	2	0	1	0	0	0	0	0	0	0

SPRIJINIT-DE
A,0 nil
A,1 nil
A,2 nil
B,1 nil
B,2 nil
B,3 nil
C,1 (A, 0, (A,C))
C,2 (A, 0, (A,C)), (A, 1, (A,C))

Efectul deciziei din prima iteraţie este că, datorită faptului că SPRIJINIRI [A,2,(A,C)] = 0, valoarea 2 este eliminată din domeniul variabilei A şi în E\SEC este adăugată perechea (A,2).

După iteraţia corespunzătoare arcului (C,A), matricele vor avea valorile:

SPRIJINIRI

A,0, (A,B)	A,0, (A,C)	A,1, (A,B)	A,1, (A,C)	A,2, (A,B)	A,2, (A,C)	B,1, (B,A)	B,2, (B,A)	B,3, (B,A)	C,1, (C,A)	C,2, (C,A)
0	2	0	1	0	0	0	0	0	1	2

SPRIJINIT-DE
A,0 (C, 1, (C,A)), (C, 2, (C,A))
A,1 (C, 2, (C,A))
A,2 nil
B,1 nil
B,2 nil
B,3 nil
C,1 (A, 0, (A,C))
C,2 (A,0, (A,C)), (A,1, (A,C))

Nu mai este eliminată nici o valoare şi E\SEC rămâne {(A,2)}.

După considerarea celui de-al treilea arc, (A,B), se obţin valorile:

SPRIJINIRI

A,0, (A,B)	A,0, (A,C)	A,1, (A,B)	A,1, (A,C)	A,2, (A,B)	A,2, (A,C)	B,1, (B,A)	B,2, (B,A)	B,3, (B,A)	C,1, (C,A)	C,2, (C,A)
0	2	1	1	0	0	0	0	0	1	2

SPRIJINIT-DE

A,0	(C, 1, (C,A)), (C, 2, (C,A))
A,1	(C, 2, (C,A))
A,2	nil
B,1	(A, 1, (A,B))
B,2	nil
B,3	nil
C,1	(A, 0, (A,C))
C,2	(A,0, (A,C)), (A,1, (A,C))

şi este eliminată valoarea 0 din domeniul lui A, E\SEC conţinând acum {(A,0),(A,2)}

În sfârşit, după ultimul arc considerat, (B,A), matricele vor conţine:

SPRIJINIRI

A,0, (A,B)	A,0, (A,C)	A,1, (A,B)	A,1, (A,C)	A,2, (A,B)	A,2, (A,C)	B,1, (B,A)	B,2, (B,A)	B,3, (B,A)	C,1, (C,A)	C,2, (C,A)
0	2	1	1	0	0	1	0	0	1	2

SPRIJINIT-DE

A,0	(C, 1, (C,A)), (C, 2, (C,A))
A,1	(C, 2, (C,A)), (B, 1, (B,A))
A,2	nil
B,1	(A, 1, (A,B))
B,2	nil
B,3	nil
C,1	(A, 0, (A,C))
C,2	(A,0, (A,C)), (A,1, (A,C))

Valorile 2 şi 3 pentru B sunt elimnate iar E\SEC va conţine {(B,3),(B,2)(A,0),(A,2)}

În acest moment, domeniile celor trei variabile conţin:

A {1}
B {1}
C {1,2}

Se trece la pasul al treilea al algoritmului, eliminarea valorilor nesprijinite. Să considerăm prima valoare din E\SEC, (B,3). Deoarece această pereche nu este sprijinită de nimeni se trece la următoarea valoare din E\SEC: (B,2). Nici această valoare nu este sprijinită. Următoarea valoare din E\SEC este (A,0). Prin urmare, S va lua valoarea {(C, 1, (C,A)), (C, 2, (C,A))}. Pentru fiecare din aceste două valori se va face decrementarea valorii corespondente din SPRIJINIRI, matricea devenind:

SPRIJINIRI

A,0, (A,B)	A,0, (A,C)	A,1, (A,B)	A,1, (A,C)	A,2, (A,B)	A,2, (A,C)	B,1, (B,A)	B,2, (B,A)	B,3, (B,A)	C,1, (C,A)	C,2, (C,A)
0	2	1	1	0	0	1	0	0	0	1

Deoarece SPRIJINIRI [C, 1, (C,A)] = 0 şi sunt satisfăcute celelalte condiţii ale deciziei din algoritm, se

adaugă (C,1) şi se elimină 1 din domeniul lui C. Eşec va conţine acum lista: {(C,1),(A,2)} şi, prin urmare, se vor mai executa încă două iteraţii. În prima, pentru (C,1), S va lua valoarea {(A, 0, (A,C))} şi, în consecinţă, SPRIJINIRI [A, 0, (A,C)] va deveni 1.

Pentru (A,2), ultima valoare din E\SEC, nu se va mai face nimic, deoarece SRIJINIT-DE [A,2]=nil.

În concluzie, algoritmul s-a terminat şi domeniile variabilelor au valorile următoare:

A {1}
B {1}
C {2}

Să analizăm acum complexitatea algoritmului AC-4. În pasul al doilea, (numărul de arce înmulţit cu mărimea domeniilor la pătrat) complexitatea este $O(ra^2)$. Pasul al treilea este executat, în cel mai rău caz, câte o dată pentru fiecare valoare, adică de na ori. În ciclul interior pot fi examinate cel mult a elemente, deci complexitatea pasului va fi $O(a^2n)$. Complexitatea algoritmului AC-4 va fi deci $O(ra^2+a^2n)=O(ra^2)$ în cazul în care numărul de restricţii este mai mare decât numărul de variabile, r>n.

Spre deosebire de ceilalţi algoritmi, în AC-4 trebuie evaluată şi memoria suplimentară. Cel mai mult spaţiu îl ocupă matricea SPRIJINIT-DE. Aceasta poate avea câte un element pentru fiecare valoare posibilă, deci an elemente. Fiecare element poate conţine sprijin de la toate valorile opsibile, deci iar an. Volumul de spaţiu ocupat este deci $O(a^2n^2)$.

Algoritmii de consistenţa căilor extind ideile algoritmilor de consistenţă a nodurilor şi a arcelor la considerarea de triplete de noduri din graful de restricţii. Acesţi algoritmi au complexităţi mai mari, de $O(n^3a^3)$, atât ca timp cât şi ca spaţiu [Bea96]. Din fericire, prin aplicarea dinamică a algoritmilor de consistenţa arcelor se pot obţine aceleaşi rezultate.

3.2 Probleme de asignare de valori

Din perspectiva PSR, folosită până acum, caracteristic acestui tip particular de probleme este faptul că domeniile de valori pentru variabile sunt intervale de numere reale, deci conţin o infinitate de valori. Vrem să remarcăm deosebirea faţă de problemele de filtrare, la care exista un număr finit de valori în domeniile variabilelor. Datorită numărului infinit de valori posibile pentru variabile, algoritmii de filtrare nu mai sunt utilizabili (ei ciclau după toate valorile posibile). De aceea, pentru aceste probleme este necesară o altă abordare. Vom privi astfel problemele pe care le-am grupat în aceasta clasă ca având ca punct de plecare o reţea ale cărei restricţii nu au iniţial valori asignate pentru toate celulele componente. Propagarea se va face plecând de la cel puţin o valoare dată pentru o variabilă.

Prelucrările efectuate au ca scop calcularea valorilor lipsă în funcţie de valorile definite. Între restricţii au loc propagări de valori de-a lungul conexiunilor din reţea. De exemplu, în reţeaua din figura de mai jos, formată din două sumatoare A1 şi A2 (care reprezintă restricţia t1+t2=s), doar celulele t1 din A1 şi s din A2 au iniţial asignate valorile 3 respectiv 17. Valoarea 3 este propagata la t1 din A2. Datorita restricţiei A2, celulei sale t2 ii va fi atribuita valoarea 14. Aceasta valoare este propagata la s din A1, în final calculându-se valoarea celulei t2 din A1 (11).

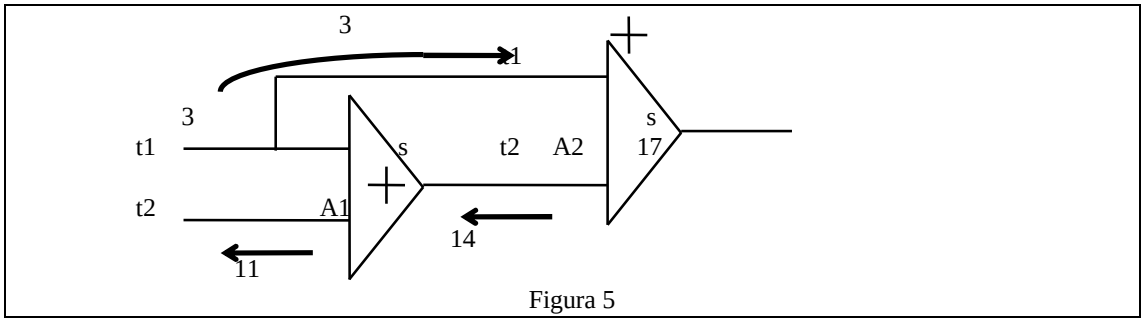

Figura 5

Această clasă de probleme este foarte utilă în aplicaţii de modelare şi simulare. Ea este adecvată, de exemplu, modelării şi simulării circuitelor electronice [MEP88]. În acest scop, se definesc restricţii pentru modelarea elementelor de circuit şi a legilor lui Kirchhof. Un circuit particular este reprezentant de o reţea de restricţii. Pe baza valorilor tensiunii şi curentului în unele puncte ale circuitului şi a valorilor parametrilor elementelor de circuit se pot obţine valorile necunoscute.

Mai multe detalii referitor la astfel de probleme sunt date în [SuS80], tot aici fiind analizate şi citeva aspecte care apar în cursul prelucrărilor: eliminarea ciclurilor şi tratarea inconsistentelor. In cele ce urmează vom prezenta şi noi problematica acestor aspecte.

3.2.1 Eliminarea ciclurilor

In figura de mai jos este reprezentata aceeaşi reţea formata din doua sumatoare A1 şi A2. Celulele t2 din A1 şi s din A2 au asignate valorile 11 respectiv 17.

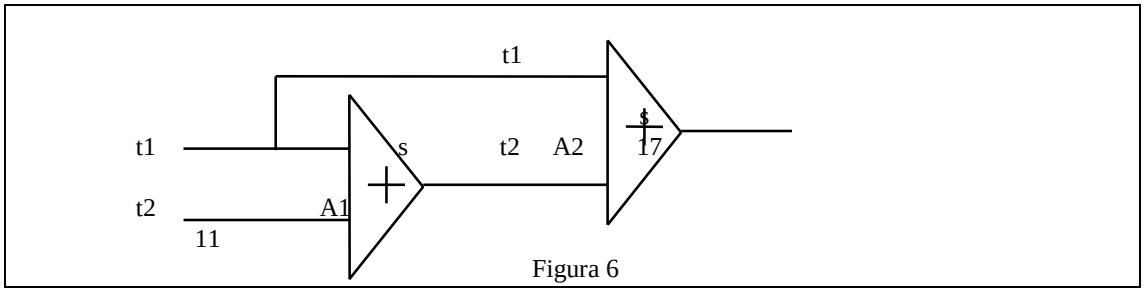

Figura 6

Se observa că prin propagare nu se pot determina valorile celorlalte celule. De exemplu, pentru determinarea valorii lui s din A1 ar trebui să fie cunoscuta valoarea lui t1 a aceleiaşi restricţii. Datorita conexiunilor (t1 din A1 egal cu t1 din A2 şi s din t1 acelaşi cu t2 din s) rezulta că, folosind valoarea 17 a lui s din A2 ar putea fi determinat t1 din A1 dacă se ştie s din A1. Aceasta situaţie (echivalenta cu un sistem de doua ecuaţii - cele doua restricţii - cu doua necunoscute - cele doua celule fara valori) este, din perspectiva problemei de propagare de restricţii, un ciclu în procesul de propagare. Găsirea unor asignări de valori se poate face introducând o variabilă temporară - de exemplu x, ca valoare a celulei t1 din A1 (figura 3) - atribuind expresia simbolică x + 11 lui s din A1, propagând x şi x + 11 la A2 (t1 din A2 este x şi t2 din A2 este x + 11) si, conform restricţiei A2 obţinând x + (x + 11) = 17 adică x = 3 (figura 4).

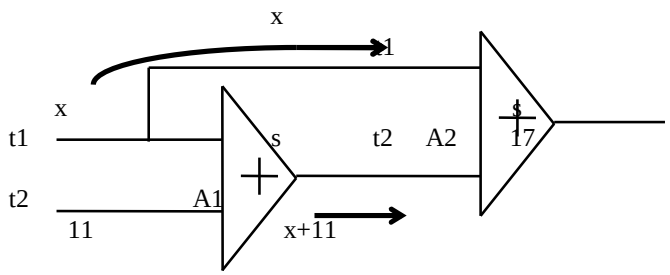

Figura 7

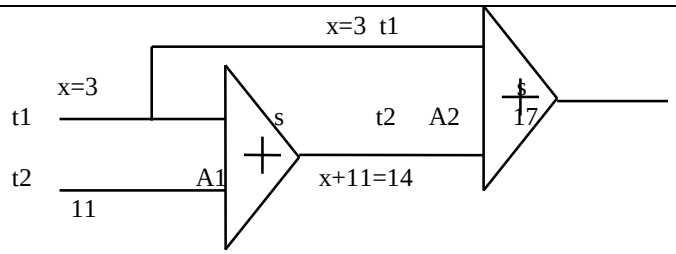

Figura 8

În concluzie, pentru a găsi asignări de valori în astfel de cazuri care implică cicluri, o soluţie este de a introduce variabile temporare pentru celule din ciclu, de a propaga expresii simbolice rezultate din aplicarea restricţiilor şi de a rezolva ecuaţiile determinate de egalarea expresiilor propagate cu valorile (eventual simbolice) celulele altor restricţii [deK86b]. Bineînţeles, aceasta soluţie presupune existenta unor proceduri de calcul simbolic pentru restricţiile din reţea [SuS80].

O a doua soluţie pentru eliminarea ciclurilor (propusa tot în [SuS80]) este definirea unor descrieri redundante. In termenii reţelelor de restricţii aceasta înseamnă suprapunerea peste cicluri a unor alte restricţii, astfel încit valorile tuturor celulelor să poată fi determinate doar prin propagare simpla.

3.2.2 Tratarea inconsistentelor

In decursul unui proces de propagare pot apare situaţii când unei celule care are asignată o valoare i se propagă o altă valoare. In acest caz în reţea a apărut o inconsistenţă. Rezolvarea acestei situaţii poate fi făcută prin modificarea valorii iniţiale a unei celule, modificarea valorilor tuturor celulelor ale căror valori au fost calculate în funcţie de valoarea modificată, după care se continuă propagarea. Pentru a putea efectua eficient aceste actualizări şi a nu relua de la început toată propagarea este foarte utila memorarea justificărilor oricăror valori calculate în reţea [Dav87]. Aceste justificări permit, în cazul unei inconsistenţe, căutarea unor variabile a căror valoare iniţială necorespunzatoare a determinat eşecul. De asemenea, ele permit modificarea numai a valorilor din reţea care depind de o anumită valoare. Tipărirea justificărilor poate fi utilă şi ca o modalitate de explicare a modului cum o anumită valoare a fost calculată în procesul de propagare.

O clasă de mecanisme care pot fi folosite pentru implementarea mecanismului descris mai sus sunt sistemele de gestiune a dependenţelor inferenţelor (TMS [Day79] sau ATMS [deK86a]). Aceste sisteme realizează toate prelucrările legate de memorarea justificărilor inferenţelor făcute, ele putând fi cuplate cu mecanismele de inferenţa folosite în procesul de rezolvare de probleme.

Mecanismele de detectare şi rezolvare a inconsistenţelor pot fi utilizate şi pentru verificarea corectitudinii valorilor variabilelor unei reţele. In caz de inconsistenţă se poate încearca modificarea valorilor pentru restabilirea consistenţei.

3.3 Probleme de etichetare consistentă

Problemele de etichetare consistentă (denumite uneori şi probleme de satisfacere a restricţiilor) sunt o extindere a problemelor de filtrare în sensul că, se cere ca, pentru fiecare variabilă, să se determine în final o singura valoare (din mulţimea de valori posibile pentru ea) [MFr85, Nad88]. Rezolvarea acestui tip de probleme necesită algoritmi care sunt combinaţii ale algoritmilor de filtrare (care determina mulţimile de valori posibile pentru variabile) cu algoritmi de căutare în arbori (pentru alegerea unei singure valori din mulţimea valorilor posibile) [Nad88].

Pentru căutarea în arbori au fost concepuţi mai mulţi algoritmi. Cei mai utilizaţi algoritmi sunt cei de tip backtrack (BT). De multe ori aceşti algoritmi nu sunt acceptabili deoarece ei pot duce la un timp de căutare exponenţial. De exemplu, în notaţiile de la secţiunea ….., se obţine pentru cazul cel mai defavorabil un timp $O(ra^n)$ [MFr85]. De fapt, se ştie că problema colorării grafurilor, un exemplu tipic de problema de etichetare consistenta, este NP-completa [DeP85]. De aceea, au fost făcute cercetări în vederea găsirii unor algoritmi care să îmbunătăţească performanţele BT sau să limiteze sau chiar să elimine BT [Dec90, deK86a, DeP85, Fre88, FrQ85, MFr85, Nad88]. Soluţiile propuse pot fi grupate, după aspectul avut în vedere pentru optimizare, astfel [RoB87]:

- **Exploatarea proprietăţilor specifice** problemei prin evitarea, pe cât posibil, a redundanţelor în spaţiul de căutare.

- **Filtrare** (în [RoB87] este folosit termenul de propagare a restricţiilor), combinată cu sisteme de genul **ATMS** [deK86a] precum şi forme de **învăţare** [Dec90].

- **Îmbunătăţirea avansului** în căutarea în arbore. Decizia asupra modului de extindere a unei soluţii parţiale, adică alegerea variabilei căreia i se atribuie următoarea valoare, are un impact deosebit asupra mărimii spaţiului de căutare. Găsirea unei ordini iniţiale între variabile poate duce la câştiguri semnificative de timp [RoB87]. Astfel, considerând graful cu restricţii drept arce şi variabile drept noduri, prin **eliminarea ciclurilor** [Dec90], prin tehnici de **anticipare** (lookahead) [Nad88] sau prin găsirea unei **ordini adecvate de parcurgere** se poate obţine o soluţie cu un BT redus sau chiar fara BT [DeP85].

Pentru exemplificare amintim îmbunătăţirea obţinută în secţiunea 1 prin alegerea unei alte ordini de considerare a variabilelor.

- **Îmbunătăţirea revenirilor**. Construirea unor algoritmi de BT inteligent, necronologic, care revin după un eşec nu la ultima alegere făcuta ci la realegerea variabilelor care au generat eşecul, pot imbunatati considerabil performantele [RoB87, Nad88, Dec90]. În acest scop este necesară păstrarea justificărilor fiecărei inferenţe făcute, de exemplu prin mecanisme de gen ATMS [Dec86a].

De exemplu, pentru problema discutată în prima secţiune, un BT necronologic, dirijat de dependeţe ar fi:

A=0
  B=1
    C=1 <0,1,1>
      nu este satisfăcută restricţia A=B ➔ revenire la alegerea care a cauzat eşecul (A=0)
A=1
  B=1
    C=1 <1,1,1>
      nu este satisfăcută restricţia A<C ➔ revenire la alegerea care a cauzat eşecul (C=1)
    C=2 <1,1,2> sunt satisfăcute ambele restricţii ➔ SUCCES !

Referitor la efectele imbunătăţirilor propuse mai sus au fost făcute aâit analize formale cât şi studii empirice [RoB87, Dec90]. Concluziile sunt diferite în funcţie de fiecare problema în parte. Algoritmii trebuie adaptaţi pentru particularităţile problemei şi, dacă este posibil, trebuie combinate în acelaşi algoritm [RoB87] concepte ontogonale (de exemplu, BT inteligent şi anticipare).

Problemele de asignare de valori pot fi considerate probleme de etichetare consistenta. Diferenţierea este nu de natura conceptuala ci metodologică. In primul caz, de obicei, asignarea de valori nu este precedată de o etapa de filtrare. Determinarea unei valori pentru o celula nu implică o alegere dintr-o mulţime de valori posibile (la problemele de filtrare) ci efectuarea unui calcul pentru determinarea ei (la problemele de asignare de valori).

Exemple tipice de probleme de etichetare consistenta sunt [DeP85] înţelegerea desenelor în trei dimensiuni, colorarea grafurilor, analiza circuitelor electronice. Un alt exemplu tipic de problema de etichetare consistenta îl constituie aplicaţiile de alocare de resurse. De exemplu, pentru o baza aeriana exista un număr de zboruri cărora trebuie să li se aloce avioane şi piloţi [MCKG88]. Totodată, exista o mulţime de restricţii cum ar fi: un avion sau un pilot nu pot efectua doua zboruri în acelaşi timp; fiecare pilot ştie să piloteze un anumit tip de avion; în funcţie de caracteristicile zborului este posibila utilizarea unor anumite tipuri de avioane etc.. Pentru a găsi o alocare de avioane şi piloţi la zboruri, se construieşte o reţea de restricţii, se alocă fiecărui zbor avioanele şi piloţii posibili, după care se lansează algoritmul de filtrare pentru a elimina incompatibilitatile din reţea. O soluţie poate fi găsita după alternarea unor alegeri (se alocă un anume avion şi pilot la un zbor), filtrări şi eventual reveniri.

3.4 Relaxarea restricţiilor

Relaxarea restricţiilor este utilizată în probleme caracterizate printr-o reţea puternic restricţionată, în care obţinerea unei soluţii prin propagare nu este posibilă. Astfel, după o (eventuală) etapă de propagare în urma căreia s-a ajuns la o incompatibilitate, reţeaua este analizată în scopul determinării restricţiilor care pot fi relaxate. Relaxarea unei restricţii înseamnă slăbirea condiţiei sale astfel încât aceasta să poată avea soluţii (de exemplu, în probleme de asignare de valori) sau să includă noi elemente în mulţimile de valori posibile pentru variabile. Cel de al doilea caz poate să apară în probleme de filtrare, noile elemente introduse datorită relaxării restricţiilor permiţând eventual obţinerea unei soluţii globale.

Problema găsirii restricţiilor care pot fi relaxate implică o analiza globala a reţelei şi a cauzelor eşecului. În acest scop pot fi utilizate euristici sau chiar un sistem expert. In [Fox83] este descrisa o aplicaţie pentru ordonanţarea producţiei care foloseşte relaxarea unor restricţii de timp în scopul programării unui lot pentru fabricaţie.

3.5 Adaugarea/eliminarea de restricţii

În clasele de probleme considerate până acum se presupunea că topologia reţelelor de restricţii rămâne aceeaşi pe parcursul procesului de prelucrare. Un caz interesant de studiat este cel în care restricţii sunt adăugate sau eliminate dinamic în reţea. Aceste operaţii constituie de fapt o reformulare a problemei iniţiale de prelucrare a restricţiilor, în urma lor fiind necesare propagării pentru a rezolva noua problemă. Eliminarea unei restricţii poate fi considerată ca un caz limită de relaxare, efectul acestei operaţii putând fi găsirea unei soluţii într-o problemă care dusese la o incompatibilitate. Introducerea unei noi restricţii are în general efectul de a restrânge numărul soluţiilor. Alegerea restricţiilor de eliminat poate implică un modul ATMS si/sau eventual o mulţime de euristici.

Unele probleme de planificare pot fi abordate pe baza acestei modalitati de prelucrare a restricţiilor. Astfel, în procesul rezolvării unei probleme pot fi definite un număr de restricţii ce sunt luate în considerare

impunând propagarea de valori având eventual efecte asupra secvenţei de prelucrări ce sunt efectuate în continuare. Un exemplu de astfel de abordare este [Ste81].

Bibliografie

[Bar90] Barbuceanu, M., "The MODELS environment for knowledge acquisition: automating the task specific architecture approach", Proceedings of AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems, Banff, nov. 1990.

[BaT88] Barbuceanu, M., Trausan-Matu, St., "The XRL2 Manual", ITCI Bucuresti, 1988.

[Bar87] Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, A., Woolf, M., "Constraint hierarchies", OOPSLA '87 Proceedings, pp. 48-60.

[Bea96] Beale, St., HUNTER-GATHERER, Applying Constraint Satisfaction, Branch-and-Bound and Solution Sybthesis to Computational Semantics,Report MCCS-96-289, New Mexico State University, 1996.

[Col87] Colmerauer, A., "Opening the Prolog III Universe", Byte, Aug. 1987, pp. 177-182.

[Dav87] Davis, E., "Constraint propagation with interval labels", Artificial Intelligence, 32(1987), pp. 281-331.

[Dec90] Dechter, R., "Enhancements schemes for constraint processing: Backjumping, learning, and cutset decomposition", Artificial Intelligence, 41 (1989/90), pp. 273-312.

[deK84] de Kleer, J., "How circuits work", Artificial Intelligence, 24(1984), pp. 205-280.

[deK86a] de Kleer, J., "An assumption-based TMS", Artificial Intelligence, 28(1986), pp. 127-162.

[deK86b] de Kleer, J., Brown, J. S., "Theories of causal ordering", Artificial Intelligence, 29(1986), pp. 33-61.

[deK86c] de Kleer, J., "Problem solving with the ATMS", Artificial Intelligence, 28(1986), pp. 197-224.[DeP85] Dechter, R., Pearl, J., "The anatomy of easy problems: A constraint-satisfaction formulation", Proceedings of IJCAI '85, 1985, pp. 1066-1072.

[Do79] Doyle, J., "A truth maintenance system", Artificial Intelligence, 12 (1979).

[EpL88] Epstein, D., LaLonde, W. R., "A Smalltalk window system based an constraints", Proceedings OOPSLA88, San Diego, 1988, pp. 83-94.

[FAS82] Fox, M., Allen, B., Strohm, G., "Job-shop Scheduling: An investigation in constraint-directed reasoning", Proceedings of the Second Conference of the AAAI, Pittsburgh, 1982.

[FLD85] Fink, P., Lusth, J., Duran, J., "A general expert system for diagnostic problem solving", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 7, no.5, sep. 85. pp 553-560.

[Fox83] Fox, M., "Constraint directed search: a case study of job-shop scheduling", Raport de cercetare, CMU-RI-TR-83-22, Univ. Carnegie-Mellon, 1983.

[Fre88] Freuder, E.C., "Backtrack-free and backtrack-bounded search", in Kanal L., Kumar, V. (eds), "Search in artificial intelligence", Springer Verlag, 1988, pp. 343-369.

[FrQ85] Freuder, E.C., Quinn, M.J., "Taking advantage of stable sets of variables in constraint satisfaction problems", Proceedings of IJCAI '85, 1985, pp. 1076-1078.

[FSB89] Fox, M., Sadeh, N., Baykan, C., "Constrained heuristic search", Proceedings of IJCAI89, Detroit, 1989.

[GhT93] Ghiculete, Gh., Trăușan-Matu, \St., A cOnstraint-Space Based CSP Solving Method, Studies in Informatics and Control, vol.2, no.2, June 1993, pag. 109-116.

[Giu89] Giuse, D., "KR: constraint-based knowledge representation", Raport de cercetare CMU-CS-89-142, Carnegie Mellon University, 1989.

[GJV87] Guesgen, H. W., Junker, U., Voss, A., "Constraints in a hybrid knowledge representation system", Proceedings of IJCAI-87, pp. 30-33.

[Las87] Lassez, C., "Constraint logic programming", Byte, Aug. 1987, pp. 171-176.

[MCKG88] Mott, D. H., Cunningham, J., Kelleher, G., Gadsden. J. A., "Constraint-based reasoning for generating naval flying programmes", Expert Systems, August 1988, Vol. 5, No. 3, pp. 226-246.

[MEP88] Motta, E., Eisenstadt, M., Pitman, K., West, M., "Support for Knowledge acquisition in the Knowledge Engineer's Assistant (KEATS)", Expert Systems, febr. 1988, vol. 5, nr. 1, pp. 6-27.

[MiF90] Mittal, S., Falkenhainer, B., "Dynamic constraint satisfaction problems", Proceedings of AAAI90, 1990, pp. 25-32.

[MFr85] Macworth, A. K., Freuder, E. C., "The complexity of some polynomial network consistency algorithms for constraint satisfaction problems", Artificial Intelligence, 25(1985), pp. 65-74.

[MuS90] Murtagh, N., Shimuro, M., "Parametric engineering design using constraint-based reasoning,

Proceeding of AAAI90, 1990, pp. 505-510.
[Nad88] Nadel, B.,A., "Tree search and arc consistency in constraint satisfaction algorithms", in Kanal, L., Kumar, V. (eds.), "Search in artificial intelligence", Springer-Verlag 1988, pp. 287-342.
[RoB87] Rosiers, W., Bmynooghe, M., "Empirical study of some constraint stisfaction algoritms", in Jorrand, Ph., Sgurev, V. (eds.), "Artificial Intelligence II: Methodology, Systems, Applications", North Holland, 1987, pp. 173-180.
[Ste81] Stefik, M., "Planning with constraints (Molgen: Part1)", Artificial Intelligence, 16(1981), pp. 111-140.
[SuS80] Sussman, G. J., Steele, G. L., "CONSTRAINTS - A language for expressing almost-hierarchical descriptions", Artificial Intelligence, 14 (1980), pp. 1-39.
[SzM88] Szekely, P., Myers, B. A., "A user interface toolkit based on graphical objects and constraints", Proceedings OOPSLA88, San Diego, 1988, pp. 36-45.
[Tra89] Trausan-Matu, St., "Micro-XRL: An object-oriented programming language for microcomputers", Raport de cercetare, Technical Cybernetics Institute, Bratislava, 1989.