CAPITOLUL VI

PLANIFICARE

6.1 Problematica Planificării în Inteligenţa Artificială

Înainte de a da o definiţie planificării, aşa cum este ea înţeleasă în Inteligenţa Artificială, să considerăm următorul exemplu. Să ne imaginăm că o persoană X aflată la Bucureşti, doreşte să facă o călătorie la Londra cu avionul. Ca să îşi poată atinge acest scop, X trebuie să ducă la îndeplinire câteva acţiuni prealabile: să obţină o viză de la Ambasada Britanică, să-şi cumpere bilet de avion, să-şi facă bagajul, să se ducă la aeroport la data şi ora precizată pe bilet, să ia avionul şi, finalmente, să coboare la Londra. Aceste etape preliminare care trebuie satisfăcute pentru atingerea scopului final – călătoria de la Bucureşti la Londra – se numesc acţiuni, şi ele constituie un plan. Construirea explicită a unei succesiuni de astfel de acţiuni în vederea unui ţel final oarecare este o ***planificare***.

Până acum nimic nu pare să se lege de Inteligenţa Artificială. Să presupunem însă că X este un robot bucureştean, şi că ne-ar place ca acest robot, în momentul în care cineva îi comandă o călătorie la Londra (sau orice altceva), să fie capabil să-şi planifice singur (în sensul de mai sus) această călătorie, fără ca omul să trebuiască să-i precizeze explicit că are nevoie de o viză, de bilet, ş.a.m.d.

Planificarea în sensul Inteligenţei Artificiale înseamnă, deci, generarea unei secvenţe de acţiuni pentru un agent (de tipul unui robot) care poate schimba mediul (lumea) în care evoluează. Scopul planificării este atingerea unuia sau mai multor ţeluri (scopuri) exprimate explicit.

Ca urmare, o problemă de planificare poate fi definită prin:

- ***starea iniţială*** a lumii (mediului). În exemplul nostru, starea iniţială este: X se află în Bucureşti, fără viză britanică, fără bilet spre Londra şi fără vreun bagaj făcut.
- ***mulţimea de acţiuni*** (*operatori*) care pot fi folosite în schimbarea lumii (mediului). În exemplul nostru mulţimea de acţiuni reprezintă acţiunile pe care robotul X trebuie să le poată îndeplini. Putem presupune, de exemplu, că X ştie să obţină viză, să cumpere bilet de avion, să facă bagaje, să meargă dintr-un loc în altul, să urce în avion,să zboare cu avionul, să coboare din avion (de obicei, însă, operatorii sunt mult mai simpli, după cum vom vedea).
- ***mulţimea de scopuri***. În exemplul dat, scopul este unul singur – călătoria la Londra.

După ce ne-am format o idee generală despre planificare să atingem şi alte aspecte ale problemei.

După cum am mai specificat planificarea presupune stabilirea unei secvenţe de acţiuni care conduc la atingerea unui anumit scop. Cele mai multe probleme de planificare se referă la domenii complexe, pentru care reprezentarea şi prelucrarea descrierii complete a stărilor problemei de rezolvat poate deveni foarte costisitoare. Din

această cauză, de multe ori este nevoie să se adopte o strategie de reprezentare parţială a universului problemei cât şi o strategie de descompunere a problemei în subprobleme. Reprezentarea parţială a universului problemei se face modificând strategia de reprezentare integrală a stărilor pe parcursul rezolvării problemei în aceea de menţinere a unei stări curente şi înregistrarea schimbărilor aduse de o acţiune stării curente. Apare însă problema definirii a ceea ce înseamnă schimbări aduse de o acţiune. Referitor la acest aspect, raţionamentul despre acţiuni trebuie să rezolve trei probleme, clasice în Inteligenţa Artificială în special în cazul raţionamentului de bun simţ, cotidian:

- ***problema cadrului*** care presupune identificarea tuturor faptelor care nu s-au schimbat ca efect al executării unei acţiuni;
- ***problema calificării*** care constă în înregistrarea tuturor precondiţiilor necesare pentru executarea unei acţiuni;
- ***problema ramificării*** care se referă la necesitatea specificării tuturor consecinţelor unei acţiuni.

De cele mai multe ori este imposibilă specificarea integrală a tuturor acestor condiţii şi orice sistem de planificare trebuie să recurgă la ipoteze simplificatoare şi implicite.

Descompunerea problemei în subprobleme şi sinteza planurilor parţiale pentru fiecare subproblemă este o strategie utilă pentru stapânirea complexităţii problemelor de planificare dar care implică anumite dificultăţi. Metodele de generare a planurilor pot fi împărţite în două categorii: metode de planificare liniară si metode de planificare neliniară. În cazul planificării liniare, o secvenţă de scopuri este rezolvată prin satisfacerea fiecărui subscop, pe rând. Un plan generat de o astfel de metodă conţine o secvenţă de acţiuni care duce la satisfacerea primului subscop, apoi secvenţa de acţiuni care duce la satisfacerea celui de-al doilea subscop, şi aşa mai departe. Această metodă funcţionează în cazul în care subproblemele rezultate din descompunerea problemei iniţiale sunt independente sau pot fi astfel ordonate încât rezolvarea unei subprobleme să nu afecteze rezolvarea unei subprobleme anterioare.

Această metodă nu poate fi utilizată pentru rezolvarea problemelor de planificare în care subproblemele interacţionează. Problemele care pot fi descompuse în subprobleme independente (care nu interacţionează) le vom numi *decompozabile,* iar cele pentru care subproblemele intercaţionează le vom numi *aproape* decompozabile. În cazul problemelor aproape decompozabile, trebuie utilizată o metodă de planificare neliniară care generează planuri prin considerarea simultană a mai multor subprobleme obţinute din descompunerea problemei iniţiale. Planul este intâi incomplet specificat, şi rafinat ulterior considerând interacţiunile existente. O astfel de metodă se numeşte *planificare neliniară* deoarece planul nu este compus dintr-o secvenţă liniară de subplanuri complete.

Problemele de planificare presupun un proces de căutare. În general, această căutare este condusă de (orientată spre) scopuri: se porneşte de la starea finală şi se deduc secvenţele de operatori care fac trecerea dintre starea iniţială şi cea finală.

O altă abordare a planificării atinge o caracteristică importantă a domeniului problemei, şi anume dacă universul este sau nu previzibil. Într-o problemă în care universul nu este previzibil, nu putem fi siguri dinainte care va fi rezultatul unei operaţii, dacă lucrăm prin simulare. Revenind la exemplul nostru, operaţia de urcare în avion poate eşua dacă de exemplu aeroportul este închis, sau operaţia de a zbura cu avionul poate

eşua dacă avionul se prăbuşeşte la câteva minute după decolare sau avionul este deturnat de pe ruta spre Londra.

În cel mai bun caz putem considera mulţimea rezultatelor posibile, eventual într-o ordine în funcţie de probabilitatea apariţiei acestor rezultate. Putem produce planuri pentru fiecare rezultat posibil la fiecare pas, dar deoarece în general multe rezultate posibile sunt foarte improbabile, ar fi o pierdere de timp să formulăm planuri exhaustive.

Putem evita aceste situaţii în două moduri. Putem analiza lucrurile câte un pas la un moment dat, fără să încercăm să planificăm. Această abordare este considerată pentru ***sisteme reactive***. Ideea este să producem un singur plan care probabil va funcţiona bine. Dacă planul eşuează putem ignora şi relua procesul de planificare plecând de la situaţia curentă.

Pe de altă parte rezultatul neaşteptat nu invalidează tot restul planului, ci doar o parte din acesta. Probabil o modificare minoră a planului, cum ar fi adăugarea unui pas suplimentar, îl va face din nou aplicabil.

În lumea reală este foarte dificil să ajungem în situaţia în care fiecare aspect este previzibil. Întotdeauna trebuie să fim pregătiţi să avem planuri care eşuează. Dacă am produs planul prin descompunerea problemei în subprobleme, atunci impactul eşecului unui anumit pas asupra planului ar putea fi destul de localizat, ceea ce este un argument în favoarea descompunerii problemei.

Este recomandat ca fiecare operaţie executată să fie memorată, chiar mai mult este indicat să se reţină până şi motivul pentru care au fost executaţi paşii. Astfel, dacă un pas eşuează, vom putea determina cu uşurinţă care sunt părţile planului care depind de acest pas şi, prin urmare, modificarea acestora este posibilă fără probleme.

6.2 Planificare În Inteligenţa Artificială Tradiţională

Tehnicile de planificare pot fi aplicate într-o mare varietate de domenii de probleme. Pentru a putea compara metodele, le vom analiza pe un domeniu suficient de complex pentru a justifica utilitatea tuturor mecanismelor descrise şi suficient de simplu pentru a găsi exemple uşoare de urmărit. Un astfel de domeniu este **lumea blocurilor (cuburilor).**

Avem la dispoziţie o suprafaţă plată pe care se pot plasa blocuri. Există un număr de blocuri în formă de cub având aceeaşi dimensiune. Blocurile pot fi plasate unul pe altul. Un braţ de robot poate ridica un bloc pentru a-l muta într-o alta poziţie, fie pe suprafaţa plată, fie deasupra unui alt bloc. Braţul robotului poate ridica doar un singur bloc la un moment dat, deci nu poate ridica un bloc deasupra căruia există alt bloc. Fiecare bloc poate avea un singur bloc pe el la un moment dat.

Ca şi în cele mai multe probleme de Inteligenţă Artificială şi în cazul acestei probleme dându-se o stare iniţială şi o stare finală se cere să se determine şirul operatorilor care face trecerea între cele două stări.

Pentru a ajunge la starea finală care satisface scopul de atins, putem folosi o mulţime de operatori. Aplicarea unuia sau altuia dintre aceşti operatori atrage după sine o modificare de stare. Toate stările noi care pot fi generate din starea iniţială utilizând oricare din operatorii care pot fi aplicaţi la momentul respective unui anume bloc

formează o stare pe care în această lucrare o vom numi stare curentă, stare în care se află problema la un moment dat. Cu alte cuvinte, starea curentă reprezintă mulţimea tuturor stărilor care pot fi obţinute din starea iniţială aplicând orice secvenţă posibilă de operatori. Starea finală este şi ea o stare curentă în acest spaţiul de stări (dacă problema are soluţie). Soluţia problemei de planificare este secvenţa de operatori folosiţi pentru a trece de la o stare curentă la alta (pe drumul între starea iniţială şi cea finală)

Spaţiul stărilor poate fi reprezentat ca un graf în care nodurile sunt stări, iar muchiile operatori; o soluţie poate fi văzută ca un drum în acest graf, aşa cum e ilustrată în Figura 6.1.

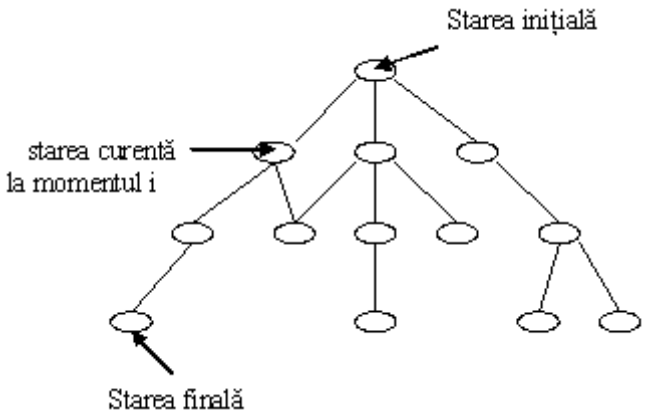

Figura 6.1 Spaţiul stărilor

Acţiunile care pot fi realizate de robot includ:

- **UNSTACK(A,B)** - Ia blocul A din poziţia sa curentă de pe blocul B. Braţul robotului trebuie să fie gol şi blocul A nu trebuie să aibă blocuri pe el.
- **STACK(A,B)** - Plasează blocul A peste blocul B. Braţul robotului trebuie să ţină blocul A, şi blocul B nu trebuie să aibă blocuri pe el.
- **PICKUP(A)** - Ia blocul A din poziţia sa curentă de pe masă. Braţul robotului trebuie să fie gol şi blocul A nu trebuie să aibă blocuri pe el.
- **PUTDOWN(A)** - Plasează blocul A pe masă. Braţul trebuie să ţină blocul A.

Pentru a specifica atât condiţiile în care au loc operaţiile cât şi rezultatele realizării operaţiilor, vom folosi predicatele:

- **ON(A,B)** - Blocul A este pe blocul B.
- **ONTABLE(A)** - Blocul A este pe masa.
- **CLEAR(A)** - Blocul A nu are blocuri pe el.
- **HOLDING(A)** - Bratul robotului ţine blocul A.
- **ARMEMPTY** - Bratul robotului este gol.

6.2.1 Componentele unui sistem de planificare

În cazul sistemelor de rezolvare a problemelor bazate pe tehnicile elementare este necesar să realizăm din următoarele funcţii:

- Alege cea mai bună regulă de aplicat pe baza celei mai bune informaţii euristice disponibile.
- Aplică regula aleasă pentru a determina noua stare a problemei.
- Detectează dacă s-a identificat o soluţie.
- Detectează blocajele, astfel încât ele să poată fi abandonate şi efortul sistemului să poată fi direcţionat în direcţii mai interesante.

În cazul sistemelor mai complexe pe care le vom discuta în continuare se cer tehnici pentru realizarea tuturor acestor funcţii. Deseori este importantă şi o a cincea operaţie:

- Detectează dacă s-a identificat o soluţie aproape corectă şi utilizează tehnici speciale pentru a o transforma într-o solutie absolut corectă.

Înainte să discutăm metode de planificare vom arunca o privire la modul în care aceste operaţii pot fi realizate.

Alegerea regulilor

În alegerea regulilor cea mai potrivită tehnică s-a dovedit a fi izolarea unei mulţimi de diferenţe dintre starea finală dorită şi starea curentă şi apoi identificarea regulilor relevante pentru reducerea acelor diferenţe. Dacă sunt identicate mai multe reguli, se pot exploata o varietate de informaţii euristice pentru a alege regula de aplicat. Această tehnică se bazează pe metoda de analiză ***means-ends* (*capete-medii*)**.

Aplicarea regulilor

În sistemele simple aplicarea regulilor este o operaţie simplă. Fiecare regulă specifică starea care rezultă din aplicarea ei. Acum va trebui să putem gestiona reguli care specifică o parte restrânsă a stării complete a problemei. Pentru aceasta există mai multe posibilităţi.

O primă posibilitate este să descriem pentru fiecare acţiune fiecare dintre schimbările pe care le aduce descrierii stării. În plus sunt necesare unele declaraţii care să precizeze că restul descrierii stării rămâne nemodificat.

Un exemplu al acestei abordări este descrierea unei stări exact aşa cum am precizat mai sus, sub forma unei mulţimi de predicate reprezintând faptele adevărate în aceea stare. Fiecare stare este reprezentată explicit ca parametru al predicatelor. De exemplu, starea S0 ar putea fi caracterizată de

$$ON(A,B,S0) \wedge ONTABLE(B,S0) \wedge CLEAR(A,S0)$$

şi regula care descrie operatorul **UNSTACK(x,y)** va fi

$$[CLEAR(x,s) \wedge ON(x,y,s)] \rightarrow [HOLDING(x,DO(UNSTACK(x,y),s)) \wedge CLEAR(y,DO(UNSTACK(x,y),s))]$$

Aici **DO** este o funcţie care specifică starea care rezultă din aplicarea unei anumite acţiuni asupra unei anumite stări. Dacă executăm **UNSTACK(A,B**) în starea S0 atunci folosind axioma despre **UNSTACK** şi ipoteza despre S0 putem demonstra că în starea S1 care rezultă din operaţia **UNSTACK** este valabil

$$HOLDING(A,S1) \wedge CLEAR(B,S1)$$

Despre S1 mai ştim că B este pe masă, dar cu ceea ce avem până acum nu putem deduce aceasta. Pentru a permite astfel de deducţii implicite avem nevoie de un set de reguli numite **axiomele cadrului** (***frame axioms***), care descriu acele componente ale stării care nu sunt afectate de operatori. De exemplu, avem nevoie să spunem că

$$ONTABLE(z,s) \rightarrow ONTABLE(z,DO(UNSTACK(x,y),s))$$
$$[ON(m,n,s) \wedge \neg EQUAL(m,x)] \rightarrow ON(m,n,DO(UNSTACK(x,y),s))$$

Avantajul acestei abordări este că un mecanism unic, rezoluţia, poate realiza toate operaţiile necesare pe descrierea stărilor. Totuşi, preţul plătit este numărul foarte mare de axiome necesar dacă descrierile stărilor problemei sunt complexe.

Pentru gestionarea domeniilor complexe avem nevoie de un mecanism care nu cere un număr mare de axiome cadru explicite. Un exemplu de astfel de mecanism este cel folosit de sistemul de rezolvare a problemelor **STRIPS** şi descendenţii săi. Fiecare operator este descris de o listă de predicate noi pe care operatorul le face adevărate şi o listă de predicate vechi pe care operatorul le face false. Cele două liste se numesc **ADD** şi respectiv **DELETE**. Pentru fiecare operator este specificată şi o a treia listă, **PRECONDITION**, care conţine toate predicatele care trebuie să fie adevărate pentru ca operatorul să fie aplicabil. Axiomele cadrului sunt specificate implicit în STRIPS. Orice predicat neinclus in listele ADD sau DELETE ale unui operator nu este afectat de acel operator. Operatorii de genul STRIPS care corespund operaţiilor pe care le-am discutat sunt următorii:

STACK(x,y)
P: $CLEAR(y) \wedge HOLDING(x)$
D: $CLEAR(y) \wedge HOLDING(x)$
A: $ARMEMPTY \wedge ON(x,y)$

UNSTACK(x,y)
P: $ON(x,y) \wedge CLEAR(x) \wedge ARMEMPTY$
D: $ON(x,y) \wedge ARMEMPTY$
A: $HOLDING(x) \wedge CLEAR(y)$

PICKUP(x)

P: ONTABLE(x) $\wedge$ CLEAR(x) $\wedge$ ARMEMPTY
D: ONTABLE(x) $\wedge$ ARMEMPTY
A: HOLDING(x)

PUTDOWN(x)

P: HOLDING(x)
D: HOLDING(x)
A: ONTABLE(x) $\wedge$ ARMEMPTY

Prin faptul că axiomele cadru devin implicite cantitatea de informaţie necesară pentru fiecare operator a fost redusă semnificativ. Aceasta înseamnă că la introducerea unui atribut nou nu este necesar să introducem o axiomă nouă pentru fiecare din operatorii existenţi. In ceea ce priveşte utilizarea axiomelor cadrului în determinarea descrierilor de stări, să notăm că pentru descrieri complexe cea mai mare parte a descrierilor rămân nemodificate. Pentru a evita preluarea informaţiei de la o stare la alta, va trebui să nu mai reprezentăm starea ca parte explicită a predicatelor. Astfel, vom avea o singura bază de date de predicate care întotdeauna descriu starea curentă. De exemplu, descrierea stării S0 din discuţia anterioara va fi :

$$ON(A,B) \wedge ONTABLE(B) \wedge CLEAR(A)$$

şi descrierea stării obţinute după aplicarea operatorului UNSTACK(A,B) va fi :

$$ONTABLE(B) \wedge CLEAR(A) \wedge CLEAR(B) \wedge HOLDING(A)$$

Acest lucru se deduce prin utilizarea listelor ADD si DELETE specificate ca parte a operatorului UNSTACK.

Actualizarea descrierii unei stări unice este o modalitate bună de actualizare a efectelor unui şir de operatori dat. Dar în cadrul procesului de determinare a şirului de operatori corect, în situaţia explorării unui şir incorect trebuie să putem reveni la starea originală pentru a putea încerca un alt şir de operatori. Pentru aceasta tot ceea ce trebuie să facem este să memorăm la fiecare nod schimbările care s-au realizat în baza de date globală. Dar schimbările sunt specificate în listele ADD si DELETE ale fiecărui operator. Deci, tot ceea ce trebuie să indicăm este operatorul aplicat la fiecare nod, cu argumentele variabile legate de valori constante. De exemplu să considerăm aceeaşi stare iniţială pe care am folosit-o în exemplele anterioare. De această dată o vom denumi “nodul 1”. Descrierea ei în format STRIPS este :

$$ON(A,B) \wedge ONTABLE(B) \wedge CLEAR(A)$$

Asupra nodului 1 vom folosi operatorul UNSTACK(A,B) iar asupra nodului 2, produs de acesta vom folosi operatorul PUTDOWN(A), rezultând nodul 3. Descrierea stării corespunzatoare nodului 3 va fi următoarea:

$$\text{ONTABLE(B)} \wedge \text{CLEAR(A)} \wedge \text{CLEAR(B)} \wedge \text{ONTABLE(A)}$$

Dacă dorim acum să analizăm un alt drum, va trebui să facem backtracking peste nodul 3. Vom **adăuga** precatele din lista DELETE ale operatorului PUTDOWN, folosit pentru a trece de la nodul 2 la nodul 3, si vom **şterge** predicatele din lista ADD ale aceluiaşi operator PUTDOWN. După aceastea vom obţine descrierea stării nodului 2, identică cu cea pe care am obţinut-o în exemplul anterior, după aplicarea operatorului UNSTACK la situaţia iniţială:

$$\text{ONTABLE(B)} \wedge \text{CLEAR(A)} \wedge \text{CLEAR(B)} \wedge \text{HOLDING(A)}$$

Deoarece în domeniile complexe o specificare implicită a axiomelor cadrului este foarte importantă, toate tehnicile pe care le vom analiza exploatează descrierile de tip STRIPS ale operatorilor disponibili.

Detectarea unei soluţii

Cum ne dăm seama dacă starea curentă la care am ajuns este starea soluţiei? În sistemele simple de rezolvare a problemelor lucrurile sunt destul de simple, se verifică pur şi simplu descrierile stărilor. Dar dacă starile nu sunt reprezentate explicit ci folosind o mulţime de proprietăţi, atunci această problemă devine mai complexă şi modul de rezolvare depinde de modul în care sunt reprezentate descrierile stărilor. Pentru a putea descoperii dacă două reprezentări se potrivesc, trebuie să putem raţiona cu reprezentări.

O tehnică de reprezentare care a servit ca bază pentru multe sisteme de planificare este logica predicatelor, atractivă din cauza mecanismului inductiv pe care îl oferă. Să presupunem ca predicatul P(x) este parte a stării finale. Pentru a vedea dacă P(x) este satisfacut într-o stare oarecare încercăm să demonstrăm P(x) cunoscând declaraţiile care descriu acea stare şi axiomele care definesc domeniul problemei. Dacă putem construi o demonstraţie, atunci procesul de rezolvare a problemei se termină. Dacă nu, atunci trebuie să propunem un şir de operatori care ar putea rezolva problema. Acest şir poate fi testat în acelaşi mod ca şi starea iniţială, prin încercarea de a demonstra P(x) din axiomele şi descrierea stării obţinute prin aplicarea şirului respectiv de operatori.

Detectarea blocajelor

Pe măsură ce un sistem de planificare caută un şir de operatori care să rezolve o anumită problemă, trebuie să poată detecta când exploră un drum care nu poate conduce niciodată la soluţie, sau care pare improbabil să conducă la soluţie. Acelaşi mecanism de raţionare care poate fi folosit la detectarea unei soluţii poate fi deseori folosit şi la detectarea unui blocaj.

Dacă procesul de cautare raţionează înainte, poate elimina orice drum care conduce la o stare de unde starea finală nu poate fi atinsă precum şi orice drum care, chiar dacă nu exclude o soluţie, nu pare să conducă la o stare mai apropiată de soluţie decât starea de origine.

De asemenea, dacă procesul de căutare raţionează înapoi, poate opri un drum deoarece este sigur că starea iniţială nu poate fi atinsă sau deoarece se realizează un progrea prea mic. În raţionarea înapoi obiectivul problemei se descompune în sub-obiective care, fiecare, pot conduce la un set de sub-obiective adiţionale. Uneori este uşor să detectăm că nu există nici un mod în care să fie satisfăcute simultan toate sub-obiectivele dintr-o mulţime.

Corectarea unei soluţii aproape corecte

Tehnicile pe care le vom discuta sunt deseori utile în rezolvarea problemelor aproape decompozabile. O modalitate de a rezolva astfel de probleme este să presupunem că sunt complet decompozabile, să rezolvăm subproblemele separate şi să verificăm dacă după combinarea subsoluţiilor, acestea conduc efectiv la o soluţie a problemei originale. Desigur că dacă acest lucru este adevărat, nu mai este nimic altceva de făcut. Dar dacă nu este adevărat, avem la dispoziţie o varietate de posibilităţi de continuare. Cea mai simplă este de a renunţa la soluţia propusă, să căutăm alta şi să sperăm că va fi mai bună. Deşi este o soluţie simplă, poate să conducă la pierderea unei cantităţi mari de efort.

O altă abordare este să analizăm situaţia propusă prin aplicarea şirului de operaţii care corespund soluţiei propuse şi să comparăm acestă situaţie cu obiectivul urmărit. În cele mai multe cazuri diferenţa dintre cele două stări va fi mai mică decât diferenţa dintre starea iniţială şi cea finală. În acest moment sistemul de rezolvare a problemelor poate fi apelat din nou pentru a elimina această diferenţă nouă. Apoi prima soluţie va fi combinată cu a doua pentru a forma o soluţie a problemei originale.

O modalitate mai bună de a corecta o soluţie aproape corectă este să utilizăm cunoştinţe specifice despre ceea ce a funcţionat greşit şi să aplicăm o *corecţie*. De exemplu, să presupunem că motivul pentru care soluţia propusă este inadecvată este că unul dintre operatori nu poate fi aplicat deoarece în momentul în care ar fi trebuit aplicat precondiţiile sale nu erau satisfăcute. Aceasta se poate întâmpla dacă operatorul ar avea două precondiţii iar şirul de operatori care face adevărată a doua precondiţie o anulează pe prima. Dar, probabil, această problemă ar apărea dacă am încerca să satisfacem precondiţiile în ordine inversă.

O modalitate şi mai bună de a corecta o soluţie incompletă este să le lăsăm incomplet specificate până în ultimul moment posibil. Atunci, când este disponibilă cât de multă informaţie posibilă, vom completa specificaţia într-un astfel de mod încât să nu apară conflicte. Această abordare este numită **strategia celui mai mic angajament (*least-commitment strategy*)** şi poate fi aplicată în mai multe moduri. Unul este de a amâna decizia asupra ordinii de realizare a operaţiilor. În exemplul anterior, în loc să decidem în ce ordine vom satisface o mulţime de precondiţii ar trebui să lăsăm ordinea nespecificată până la sfârşit. Apoi vom analiza efectele fiecărei substituţii pentru a determina dependenţa dintre ele, după care vom putea alege şi ordinea.

6.2.2 Tehnici de planificare

În acest capitol vom prezenta pe scurt cele mai importante tehnici de planificare folosite în Inteligenţa Artificială.

6.2.2.1 Planificarea folosind stive de obiective (STRIPS)

Această tehnică de planificare este dintre primele tehnici dezvoltate pentru rezolvarea problemelor compuse care pot interacţiona. Această abordare este folosită de STRIPS şi se bazează pe utilizarea unei stive unice care conţine atât scopuri cât şi operatori care au fost propuşi pentru a rezolva acele scopuri. De asemenea se reţine într-o bază de date mulţimea predicatelor care descrie starea curentă şi într-o altă bază de date operatorii descrişi sub forma listelor PRECONDITION, ADD şi DELETE.

În scopul ghidării procesului de căutare se pot folosi diverse informaţii euristice, (detectarea drumurilor neprofitabile precum şi considerarea unor interacţiuni între scopuri care pot conduce la determinarea unei soluţii globale bune). Aceste euristice vor fi identificate în funcţie de problema concretă în care se aplică mecanismul de planificare.

În cele ce urmează vom descrie paşii algoritmului STRIPS. Vom face următoarele notaţii:

- ***SI*** – starea iniţială
- ***SF***– starea finală
- ***StivaScop*** – stiva ce conţine sopurile
- ***StareCurenta*** – starea în care se află problema la un moment dat (starea curentă)
- ***StareFinala*** – starea finală
- ***StivaOperatori*** – stiva ce conţine operatorii pe parcursul aplicării algoritmului.

1. Se adaugă în ***StivaScop*** diferenţele dintre ***SI*** şi ***SF***.
StareCurentă ← ***SI***

2. Dacă ***StivaScop*** = ∅ şi ***StareCurenta*** = ***StareFinala*** atunci
STOP
StivaOperatori este soluţia

3. Fie ***e*** ← vârful stivei ***StivaScop***
dacă ***e*** este operator **atunci**
@se adaugă ***e*** în ***StivaOperatori***
@se scoate ***e*** din vârful ***StivaScop***
altfel
dacă există în ***StareCurentă*** **atunci**
@se scoate ***e*** din ***StivaScop*** (din vârf)
altfel
@se caută operatorii O_1,..,O_k care au în lista ADD pe ***e***
pentru i =1,k execută
@se scoate ***e*** din vârful ***StivaScop***
@**dacă** nu se descoperă vreun blocaj **atunci**
@se adaugă O_i în ***StivaOperatori***
@se adaugă ***StivaOperatori*** predicatele din lista PRECONDITION a operatorului O_i
sf-dacă
sf-dacă

sf-dacă

4. Salt la pasul 2.

6.2.2.2 Planificarea neliniară folosind declararea limitărilor

Metoda de planificare folosind stive de scopuri abordează problemele care implică scopuri multiple prin rezolvarea scopurilor în ordine, câte unul la un moment dat. Acestă metodă constă în generarea unui şir de operatori pentru atingerea primului scop, urmat de un şir complet de operatori pentru al doilea scop, ş.a.m.d. Aşa cum am văzut problemele dificile crează interacţiuni între scopuri. Operatorii folosiţi pentru rezolvarea unei subprobleme pot interfera cu soluţia unei subprobleme anterioare. Majoritatea problemelor cer un plan combinat în care mai multe subprobleme sunt abordate în acelaşi timp. Un astfel de plan se numeşte ***plan neliniar*** deoarece nu este compus dintr-un şir liniar de subplanuri complete.

În generarea planurilor neliniare se foloseşte ***adăugarea paşilor***, euristică care constă, după cum îi spune şi numele, prin adăugarea unor paşi pentru realizarea scopurilor sau precondiţiilor. Iată euristicile folosite în general:

- **Adăugarea paşilor** (***step addition***) – Crearea unor paşi noi pentru un plan.
- **Promovarea** (***promotion***) – Introducerea limitării (constrângerii) ca un pas să apară înaintea altuia într-un plan final.
- **Revalidarea** (***declobbering***) – Plasarea unui pas (posibil nou) s2 între doi paşi mai vechi s1 şi s3,astfel încât s2 revalidează unele precondiţii ale lui s3 care au fost negate de s1.
- **Stabilirea simplă** (***simple establishment***) – Atribuirea unei valori pentru o variabilă pentru a asigura precondiţiile unui anumit pas.
- **Separarea** (*separation*) – Prevenirea atribuirii anumitor valori pentru o variabilă.

În continuare vom da o descriere succintă a paşilor algoritmului de planificare neliniară **TWEAK**.

Algoritm: Planificare neliniară (TWEAK)

1. Iniţializează S la mulţimea propoziţiilor din starea scop.
2. Şterge din S o propoziţie nerezolvată P.
3. Rezolvă P folosind una din cele cinci operaţii de modificare a planului.
4. Revizuie toţi paşii din plan incluzând orice pas introdus prin adăugarea paşilor, pentru a vedea dacă există vreo precondiţie nesatisfăcută. Adaugă la S noua mulţime de precondiţii nesatisfăcute.
5. Dacă S este vidă, termină planul prin convertirea ordinii parţiale de paşi într-o ordine totală şi instanţiază orice variabile după cum este necesar.
6. Altfel mergi la pasul 2.

Nu orice şir de operaţii de modificare a planului conduce la o soluţie. Nedeterminismul paşilor 2 şi 3 trebuie implementat ca un fel de procedură de căutare,

care poate fi ghidată de euristici. De exemplu, dacă atât promovarea cât şi adăugarea unui pas rezolvă o anumită dificultate, atunci probabil este bine ca mai întâi să se încerce promovarea. TWEAK foloseşte backtracking de tip Breadth-First direcţionat de dependenţe precum şi euristici de ordine.

6.2.2.3 Planificare ierarhică

De cele mai multe ori pentru rezolvarea unor probleme dificile este nevoie de generarea unor planuri lungi. Pentru a face acesta în mod eficient este important să eliminăm unele părţi ale problemei până când găsim o soluţie care abordează aspectele principale. Apoi se poate încerca şi o completare a detaliilor corespunzătoare. Primele astfel de încercări au implicat utilizarea macro-operatorilor, în care operatorii mari erau construiţi din operatori mici. O abordare mai bună a fost dezvoltată în sistemul ABSTRIPS, care planifica într-o ierarhie de spaţii abstracte. În fiecare dintre acestea erau ignorate precondiţiile aflate la un nivel de abstractizare mai scăzut.

ABSTRIPS foloseşte următoarea modalitate de rezolvare: fiecărei precondiţii i se atribuie o valoare critică care reflectă dificultatea presupusă de satisfacere a precondiţiei. Se rezolvă problema complet considerând doar precondiţiile ale căror valori critice sunt cele mai mari posibile. Pentru acesta se operează la fel ca STRIPS luând în considerare doar precondiţiile cu valoare critică cea mai mare. Acest procedeu se repetă cu deosebirea că se consideră precondiţiile de la nivelul critic imediat următor. În selectarea operatorilor sunt ignorate toate precondiţiile mai puţin critice decât nivelul considerat. Acest procedeu se continuă prin luarea în considerare a precondiţiilor din ce în ce mai puţin critice până ce toate precondiţiile regulilor originale au fost satisfăcute.

Planificarea dezvoltată în sistemul ABSTRIPS mai este numit şi ***length first***. Acest nume i-a fost dat deoarece explorează în întregime planuri la un anumit nivel de abstractizare înainte de a analiza detaliile de nivel scăzut. Este foarte important în metoda planificării ierarhice ca valorile critice asociate predicatelor din precondiţii să fie cele potrivite. Acele precondiţii pe care nici un oprator nu le poate satisface sunt considerate cele mai critice.

6.2.2.4 Sisteme reactive

Tehnicile de planificare descrise până în acest moment presupuneau construirea unui plan pentru realizarea întregului obiectiv înainte de a acţiona într-un fel sau altul. ***Sistemele reactive*** abordează problema de a decide ce anume trebuie să facem într-un mod cu totul diferit. Ideea de bază constă în evitarea planificării şi folosirea situaţiei observabile ca o indicaţie la care să se reacţioneze.

Un sistem reactiv trebuie să aibă acces la o bază de cunoştinţe care să descrie acţiunile care trebuie luate şi circumstanţele în care aceste acţiuni trebuie luate. Sistemele reactive sunt foarte diferite de celelalte tipuri de planificare pe care le-am discutat până

acum deoarece aleg câte o acţiune la un moment dat şi nu anticipează şi selectează un şir întreg de acţiuni înainte de a face primul pas.

Să luăm acum un exemplu simplu de sistem reactiv: ***termostatul***. Obiectivul unui termostat este de a păstra constantă temperatura unui încăperi. Regulile simple pe care le foloseşte un termostat sunt de forma ***situaţie-acţiune***:

- Dacă temperatura în încăpere este de cu ***x*** grade peste temperatura dorită, atunci porneşte instalaţia de aer condiţionat.
- Dacă temperatura în încăpere este de cu ***x*** grade sub temperatura dorită, atunci opreşte instalaţia de aer condiţionat.

Sistemele reactive sunt capabile să surprindă comportări surprinzător de complexe, în special în aplicaţii reale cum ar fi deplasarea unui robot. Sistemele reactive operează robust în domenii care sunt dificil de modelat absolut complet şi absolut corect, fapt ce reprezintă principalul avantaj asupra planificatorilor tradiţionali. În domenii complexe şi imprevizibile abilitatea de a planifica un şir exact de paşi înainte ca aceştia să fie aplicaţi are o valoare îndoielnică. Sistemele reactive au o abilitate de răspuns foarte mare, deoarece evită explozia combinatorială implicată în planificarea deliberată. Acest lucru le face atractive pentru aplicaţii în timp real precum conducerea unei maşini şi mersul pe jos.

Sistemele reactive nu se bazează pe un model al lumii (mediului) sau pe structuri explicite ale obiectivului urmărit, de aceea performanţa lor în astfel de aplicaţii este limitată.

Eforturile de a înţelege comportarea sistemelor rective au servit la ilustrarea multor limitări ale planificatorilor tradiţionali. De exemplu, este vital să alternăm planificarea şi execuţia planului. Un sistem inteligent cu resurse limitate trebuie să decidă când să înceapă să gândească, când să nu mai gândească şi când să acţioneze. De asemenea, când sistemul interacţionează cu mediul obiectivele de urmărit apar în mod natural. Este nevoie de un mecanism de suspendare a planificării pentru ca sistemul să se poată concentra asupra obiectivelor cu prioritate mare. Unele situaţii cer atenţie imediată şi acţiune rapidă. Pentru acest motiv unii planificatori deliberativi includ subsisteme reactive (adică mulţimi de reguli de genul situaţie-acţiune) bazate pe experienţa de rezolvare a problemelor. Astfel de sisteme în timp învaţă să fie reactive.

6.2.2.5 Alte tehnici de planificare

Alte tehnici de planificare pe care nu le-am discutat includ următoarele:

- **Tabele Triunghiulare** - Tabelele triunghiulare oferă un mod de a înregistra obiectivele pe care fiecare operator trebuie să le satisfacă şi obiectivele care trebuie să fie adevărate pentru ca operatorul să poată fi executat corect. Dacă se întâmplă ceva neprevăzut în timpul execuţiei unui plan, tabelul oferă informaţiile necesare pentru a corecta planul.
- **Metaplanificare –** O tehnică de raţionare nu doar despre problema de rezolvat, ci şi despre însuşi procesul de planificare.
- **Macro-operatori –** Permit unui planificator să construiască noi operatori care reprezintă şiruri de operatori utilizate frecvent.

- **Planificare bazată pe cazuri –** Reutilizează planurile vechi pentru a construi planuri noi.

6.3 Rezolvarea distribuită a problemelor şi planificarea în Inteligenţa Artificială Distribuită

În legătură cu problema planificării trebuie să răspundem la două întrebări importante: "Ce este un plan ?" şi "De ce avem nevoie să planificăm?".

După cum am văzut în secţiunea anterioară, un plan în IA este văzut ca o secvenţă de acţiuni prin care agentul este capabil să-şi modifice mediul, în scopul realizării sarcinii (sarcinilor). Procesul de generare a unui plan se numeşte ***planificare***. Trebuie să cunoaştem: *starea iniţială, acţiunile posibile* ale agenţilor prin aceştia care pot schimba mediul, *mulţimea de scopuri* care trebuie îndeplinită.

Planificarea în SMA presupune în plus rezolvarea următoarelor probleme: controlul căutării, reprezentarea acţiunilor, protecţia scopului, modelarea timpului, ordonarea scopurilor.

În continuare vom aborda problematica rezolvării distribuite a problemelor şi a planificării în sistemele cu mai mulţi agenţi.

Planificarea distribuită are foarte multe în comun cu rezolvarea distribuită a problemelor. Pe de o parte în timp ce se rezolvă problema distribuită, fiecare agent îşi crează un plan al acţiunilor şi apoi îşi coordonează acţiunile pentru a evita interacţiunile negative, astfel realizând un plan multiagent. Pe de altă parte, planificarea în SMA poate fi privită ca o problemă distribuită care trebuie rezolvată.

Rezolvarea distribuită a problemelor se ocupă cu împărţirea sarcinilor între agenţi pentru realizarea scopului în timp ce sistemele multiagent se ocupă cu comportamentul agenţilor care acţionează în scopul rezolvării problemei date.

Atunci când se construieşte un sistem de rezolvare distribuită a problemelor se fac următoarele presupuneri asupra principalelor elemente ale sale. Se presupune că:

- Sarcinile, resursele şi/sau capacităţile sunt distribuite inegal între agenţi, motivând agenţi să lucreze împreună mult mai eficient, pentru realizarea scopului propus.
- Agenţii sunt creaţi să lucreze unul cu celălalt.
- Există probleme care trebuiesc rezolvate.
- Sarcinile trebuie să îndeplinească anumite cerinţe.

6.3.1. Exemple de probleme distribuite

Turnurile din Hanoi

Este o problemă foarte cunoscută şi des utilizată pentru ilustrarea tehnicilor de planificare în SMA. Domeniu problemei constă în existenţa a 3 tije şi a unui număr dat de discuri de mărimi diferite. Acţiunea posibilă este aceea de a muta un disc (numai unul la un moment dat) de pe o tijă pe alta fără a plasa un disc mai mare pe unul de dimensiuni mai mici. Scopul este de a muta toate discurile de pe o tijă iniţială pe o alta dată.

Problema Turnurilor din Hanoi este un exemplu de problemă care permite utilizarea planificării în paralel pentru îmbunătăţirea timpului de găsire a soluţiei.

Încărcarea unui domeniu de pe Internet

În acest domeniu agenţii primesc un set de documente pe care trebuie să le încarce de pe Internet. Scopul fiecare agent fiind minimizarea costul de încărcare. Acest lucru se poate realiza prin comunicarea documentelor necesare.

6.3.2. Partajarea sarcinii şi partajarea rezultatului

Există două mecanisme de comunicare între agenţi. Pentru a profita de autonomia şi independenţa agenţilor, colaborarea se bazează pe partajarea sarcinii şi combinarea soluţiei.

Partajarea sarcinii

Partajarea sarcinii foloseşte metoda descompunerii problemei în subprobleme şi abilitatea agenţilor de a lucra în paralel. Când un agent realizează că este suprasolicitat de sarcini încearcă să descompună sarcina (sarcinile) în subsarcini, pe care apoi să le plaseze şi altor agenţi, care au capacităţile necesare rezolvării acestor subprobleme. Apoi fiecare agent realizează subsarcina dată (dacă este nevoie utilizează aceeaşi metodă, de partajare a sarcinii în cazul în care consideră că subproblema este încă prea mare), după care fiecare agent trimite rezultatul unui agent (de obicei soluţia se trimite agentului de la care s-a primit sarcina).

Partajarea sarcinii este uşoară atunci când agenţii au capacităţi identice – astfel de sisteme se numesc ***sisteme multiagent omogene***. Din păcate în cele mai multe cazuri capaciţăţile agenţilor nu sunt nici pe de aproape identice – astfel de sisteme se numesc ***sisteme multiagent eterogene***. Partajarea sarcinii în sisteme eterogene în care problema de rezolvat nu poate fi descompusă în subprobleme egale necesită utilizarea unor mecanisme speciale. Într-un caz simplu, un agent trebuie să păstreze infomaţii despre capacităţile celorlalţi agenţi şi apoi să selecteze un agent capabil (după informaţiile deţinute de el) să realizeze o subsarcină dată. Din păcate poate apărea cazul în care un agent este deja ocupat cu realizarea altei sarcini; caz în care este indicat ca agentul să efectueze complet acea sarcină. Un agent poate verifica dacă un alt agent este sau nu ocupat cu realizarea unei cerinţe comunicând cu acesta. Dacă toţi agenţii care sunt capabili să realizeze o sarcină sunt ocupaţi la un moment dat se pot face următoarele lucruri: să se aştepte până agentul în cauză devine liber, să se încerce descompunerea problemei într-un al mod, să se reverifice capacităţile necesare realizării sarcinii în scopul găsirii altor agenţi capabili să realizeze sarcina, etc.

Partajarea rezultatului

Rezultatele obţinute de un agent pot fi intermediare sau finale. Deoarece agenţii pot avea capacităţi diferite, şi cunoştinţe şi viziuni asupra domeniului diferite, ei pot genera rezultate diferite deşi realizează aceeaşi sarcină.

Pentru ca partajarea rezultului să se desfăşoare în cele mai bune condiţii agenţii trebuie să fie atât competenţi cât şi ”prudenţi”. Ei trebuie să fie competenţi în interpretarea rezultatelor primite şi de asemenea ei trebuie să găsească echilibrul perfect între trimiterea rezultatelor şi schimbarea de informaţii pentru ai ajuta pe ceilalţi în lucrul

lor. O problemă importantă a agenţilor este de a decide care sunt informaţiile pe care le pot împărţi cu ceilalţi şi cât de des să facă acest lucru. O altă problemă este cum să privească agentul rezultatele obţinute de ceilalţi, să acorde acestora credibilitate şi prioritate mai mare decât propriilor sale rezultate sau nu?

Există mai multe strategii pentru rezolvarea acestor dileme:

- Folosirea unei **structuri partajate** (container) în locul propagării aleatoare a rezultatului agenţilor din sistem. Acest mecanism este foarte util atunci când agenţii nu ştiu care agent ar fi interesat de rezultatele obţinute de ei.
- **Distributed Constraint Heuristic Search** (DCHS) constă în crearea unui agent "resursă" care planifică folosirea resurselor.
- Crearea unei **structuri organizatorice** în scopul reducerii comunicării. Ideea acestei strategii constă în faptul că "agenţii au un rol general de jucat în efortul colectiv, şi folosind cunoştinţele rolului, agentul poate lua decizii de interacţionare mai bune"**.** Structura organizatorică va conţine roluri, priorităţi şi responsabilităţi. Rolul descrie tipul sarcinii pe care agentul trebuie să o satisfacă şi câteva reguli prioritare pentru a-l ajuta să lucreze eficient în situaţii de sarcini multiple. Responsabilităţile sunt cele care într-adevăr ajută la reducerea comunicării, deoarece fiecare agent este informat doar cu rezultatele care îl pot ajuta la realizarea sarcinii sale. Astfel agentul ştie pe cine să anunţe cînd are o informaţie nouă. De asemenea agentului îi este impus cu care agent poate să comunice, şi de la care agent poate prelua informaţii.
- Pentru minimizarea informaţiilor trimise excesiv pot fi utilizate strategii diferite de comunicare. O alternativă ar fi ca să se aştepte ca agenţii să solicite rezultatele înainte ca ele să fie trimise.
- Agenţii pot rezolva foarte uşor problema comunicării prin introducerea relaţiilor între sarcini. Relaţiile între sarcini precizează efectele pe care le poate produce un agent asupra altor agenţi. În general sunt definite patru tipuri de relaţii: *relaţii împuternicite* – atunci când rezultatul acţiunii unui agent este solicitat de un alt agent pentru îndeplinirea sarcinii; *relaţii facilitate* – rezultatele obţinute de un agent ajută alţi agenţi la obţinerea unor soluţii mai bune, sau mai rapide; *relaţii interzise* – opusul celor permise ; *relaţii interzise* – opusul celor facilitate.

Toate strategiile prezentate până acum se pot aplica unui sistem cu o organizare relativ statică; cu relaţii între sarcini simple. Pentru ca agenţii să se descurce şi în domenii mai complicate, unde relaţiile dintre acţiuni sunt mai complexe, ei trebuie să fie capabili să analizeze situaţia curentă şi să genereze un plan de comunicare şi interacţiune cu ceilalţi agenţi. În continuare ne vom axa pe aceste probleme.

6.3.3. Planificarea distribuită

În general, planificarea poate fi văzută ca o problemă de creare a unui plan. Atunci când procesul de planificare este împărţit între o mulţime de agenţi vorbim despre construirea planurilor distribuite. Dacă rezultatul procesului de planificare (care este de fapt un plan) este distribuit spre execuţie între mai mulţi agenţi, se numeşte plan distribuit. Termenul de planificare distribuită este folosit pentru a descrie faptul că agenţii

participă în formarea unui plan distribuit sau acţionează în scopul realizării planului distribuit, sau ambele.

În continuare vom discuta tehnici de planificare distribuită.

Planificare centralizată pentru planuri distribuite

Planificarea centralizată este folosită atunci când crearea planului este sarcina unui singur agent (doar el este capabil să creeze planul, el este singurul care deţine informaţiile necesare). Planul trebuie să fie un plan ordonat paţial pentru ca să poate fi distribuit apoi unei mulţimi de agenţi. Agenţii execută subplanurile primite spre îndeplinire, având informaţii corecte despre mediului în care evoluează şi a propriilor lor acţiuni. Execuţia subplanurilor se face în paralel.

Desigur pentru ca tehnica planificării centralizate să funcţioneze eficient este foarte important să se găsească cea mai eficientă descompunere a problemei în subprobleme şi distribuirea sarcinii să fie cea mai potrivită. Acest lucru este destul de greu de îndeplinit pentru că agenţii capabili să îndeplinească subsarcina dată poate să fie ocupat la momentul respective. Descompunerea problemei şi distribuirea sarcinii s-ar putea face de mai multe ori până când agenţii să fie capabili să înceapă realizarea planului.

Planificare distribuită pentru planului centralizate

Planificarea distribuită poate avea loc chiar dacă planul rezultat va fi executat doar de un singur agent. Aici, construirea distribuită a unui plan centralizat poate fi privită ca o tehnică specială de rezolvare distribuită a problemelor. Prima dată, problema (scopul) trebuie descompusă şi distribuită printre agenţi prin folosirea unor metode de partajare a sarcinilor. Apoi coordonarea poate fi realizată fie prin schimbarea unui singur plan parţial, fie prin partajarea rezultatului.

Planificarea distribuită pentru planuri centralizate este folosită eficient în domenii complexe ca: industria, logistica, etc.

Planificarea distribuită pentru planuri distribuite

Cea mai complexă configuraţie pentru planificarea distribuită este aceea în care atât procesul de planificare cât şi execuţia este distribuită. În astfel de cazuri de planificare în SMA, se poate ca agenţii să aibă informaţii unii despre ceilalţi. Astfel poate fi imposibil şi chiar inutil să încercăm să generăm un plan multiagent complet, conţinând toate acţiunile agenţilor din sistem. Pe de altă parte, este de preferat să se minimizeze interacţiunile dintre planurile agenţilor, şi dacă este posibil şi raţional chiar să se elimine cooperarea dintre agenţi în scopul realizării obiectivelor.

Aşadar agenţii trebuie să cunoască planul celorlalţi agenţi, ei trebuie să fie capabili să rezolve conflicte şi să gasească relaţiile dintre sarcini. Cele mai populare tehnici sunt:

- Interclasarea planului.
- Formarea iterativă a planului.

- Negocierea.

Interclasarea planului

Interclasarea planului nu se ocupă cu distribuirea scopurilor. Această tehnică este folosită atunci când fiecare agent crează un plan pentru activităţile sale fără a le "discuta" cu ceilalţi agenţi. Apoi agenţii comunică între ei pentru a-şi coordona acţiunile în scopul evitării conflictelor.

În continuare vom schiţa algoritmul de interclasare a planului pentru un singur agent – interclasarea planului centralizat. Precizăm că toţi agenţii şi-au generat planul prin folosirea unor metode de planificare. Apoi toţi agenţii trimit planul unui singur agent (agentul coordonator) care va încerca să le îmbine, şi va adăuga constrângeri dacă este nevoie pentru a asigura că acţiunile în conflict nu vor fi executate în acelaş timp. Deci sarcina de bază a agentului coordonator este să găsească secvenţele de acţiuni din planurile agenţilor care ar putea cauza conflicte. În general, astfel de probleme necesită o analiză profundă: având starea iniţială şi o secvenţă de acţiuni care pot fi executate asincron, se caută stările la care se poate ajunge în urma acţiunilor, se enumeră cele care include conflicte după care se plasează constrângeri în secvenţele de acţiuni care au dus la conflict.

Pentru a restrânge aria de căutare a spaţiului vom prezenta un algoritm propus de Georgeff.

Pentru aplicarea acestei metode se presupune că:

- agenţii cunosc stările iniţiale posibile;
- fiecare agent a construit un plan corect şi total odonat (cuprinzând acţiunile $a_1 \ldots a_n$; astfel că a_1 se poate aplica stării iniţiale, a_i stării rezultate prin aplicarea lui a_{i-1}, iar a_n să ducă la starea scop);
- acţiunile sunt reprezentate în sistem STRIPS.

Folosirea operatorilor STRIPS restrânge spaţiul de căutare a interecţiunilor dintre planuri.

Algoritmul de interclasare analizează planurile agenţilor examinând perechi de acţiuni pe care doi agenţi le pot efectua în acelaş timp. De exemplu acţiunile a_i şi b_j vor fi analizate după o acţiune pe care agenţii A şi B o pot efectua în acelaş timp. Există o mulţime de configuraţii posibile de dependenţă între a_i şi b_j.

- *Independenţa* – două acţiuni pot fi executate în acelaşi moment dacă precondiţiile, condiţiile în curs, şi postcondiţiile sunt satisfăcute în aceeaşi stare. Astfel de acţiuni se numesc **comutabile**;
- *Precedenţa* – dacă starea anterioară aplicării unei acţiuni este modificată de efectul uneia dintre cele două acţiuni (să zicem a_i) şi satisface precondiţiile acţiunii (b_j);
- *Conflictul* -- dacă nici o acţiune nu o poate preceda pe cealaltă.

Având precizare legăturile dintre acţiuni, putem trece la analizarea situaţiilor care pot apărea.

Definim *situaţiile nesigure* astfel:

- situaţia de a executa a_i şi b_j împreună este nesigură dacă ele nu comută;

- situaţia de a începe a_i înainte de b_j este nesigură dacă a_i nu precede b_j ;
- situaţia de a începe a_i şi b_j este nesigură dacă situaţiile succesorilor a fost nesigură;
- situaţia de a începe a_i şi de a termina b_j este nesigură dacă situaţia de termina a_i şi de a termina b_j sunt nesigure;
- situaţia de a termina a_i şi b_j este nesigură dacă situaţiile succesoare ambelor acţiuni sunt nesigure.

Analizarea siguranţei situaţiilor reduce semnificativ spaţiul de căutare a interacţiunilor datorită dependenţei acţiunilor. Evident, dacă o acţiune comută cu toate celelalte acţiuni ea poate fi scoasă din spaţiul de căutare pentru că nu cauzează conflicte. Acest lucru este util în sistemel multiagent slab cuplate (*loosely-coupled*), unde fiecare agent lucrează individual şi majoritatea acţiunilor sunt independente. În acest caz planul de interclasare se reduce la o secvenţă scurtă care poate fi uşor analizat. Algoritmul se continuă prin definirea spaţiului de situaţii nesigure şi plasarea unor restricţii în planul agenţilor. De obicei restricţiile pot suspenda activitatea unor agenţi a căror acţiuni viitoare vor intra în conflict cu acţiunile curente ale celorlalţi agenţi.

Algoritmul de interclasare a planului este mult mai complex când avem de a face cu planuri care conţin scopuri temporale. De exemplu, în domeniul DD agenţii nu au doar simpla sarcină de a transporta un obiect, ci ei trebuie să o facă într-un anumit interval de timp (sau până la o anumită dată). În astfel de aplicaţii, interclasarea planului se bazează pe programarea activităţilor între intervale de timp fixate. Agenţii îşi costruiesc planul de acţiuni după un orar propriu prioritar (anumite acţiuni trebuie terminate înainte ca altele să fie începute). Astfel procesul de interclasare a planului devine o problemă distribuită de satisfacere a constrângerilor (DCSP), problema de a găsi o programare a sarcinilor agenţilor având în vedere constrângerile ordinii precedente, a datelor limită şi a resurselor consumate.

Interclasarea planului poate fi realizat şi în mod descentralizat, astfel: agenţii identifică scopurile, generează planuri, îşi schimbă planurile locale între ei; planurile sunt interpolate folosind timpul şi mesajele.

Formarea iterativă a planului

Aşa cum ne-am putut da seama, interclasarea planului poate fi folosit pentru a face posibilă generarea şi executatea planurilor paralele, tehnică prin care fiecare agent îşi generează propriul plan luând în considerare doar scopurile proprii. Dar, în practică deciziile şi planurile unui agent sunt foarte rar independente de acţiunile celorlalţi. Există o mulţime de abordări care se ocupă cu situaţii în care agenţii trebuie să ia în considerare constrângerile globale şi să fie conştienţi de acţiunile posibile ale celorlalţi agenţi în generarea planului propriu. Toate aceste abordări utilizează tehnici de coordonare în timpul procesului de formare a planului în loc să facă acest lucru după (aşa cum face tehnica de interclasare a planului).

O astfel de metodă este *căutarea planului combinat* propus de Ephrati şi Rosenschein. Fiecare agent generează un set de planuri posibile care pot duce la realizarea scopului, apoi fiecare agent este angajat în căutatea planurilor care pot fi

combinate spre evitarea conflictelor. Stările din domeniu şi rezultatul acţiunilor agentului sunt reprezentate sub forma unor seturi de axiome. Fiind dată o declaraţie curentă care stabileşte starea, fiecare agent prezintă schimbările care vor rezulta în urma unei acţiuni din planul său. Toate schimbările sunt combinate în scopul producerii unei noi stări posibile; care sunt ordonate folosind algoritmul A*. Starea cea mai bună este aleasă şi procesul se repetă până când nici un agent nu mai deţine axiome. În această metodă coordonarea este atinsă în timpul procesului de corectare a planului când un agent are acţiuni limitate deoarece starea următoare nu include rezultatele acţiunii sale (probabil deoarece au fost în conflict cu scopul celorlalţi agenţi).

Corkill a sugerat folosirea planificării ierarhice – aşa numita ***planificare ierarhică distribuită.*** El a ilustrat acest concept în Lumea Blocurilor, adăugând un "plan decompozabil" al cărui ţel este de a determina scopurile pentru a fi distribuite. Astfel fiecare agent primeşte o copie a planului. De la acest punct fiecare agent păstrează un model a planurilor altor agenţi şi înnoieşte modelul de căte ori are informaţia ca acesta s-a modificat. La un nivel mai înalt de detaliere a procesului de planificare agenţii decid când să comunice cu ceilalţi agenţi despre schimbările pe care doresc să le facă asupra stării curente. Astfel, fiecare agent poate detecta dacă acţiuniile sale vor intra în conflict cu efectele acţiunilor viitoare ale celorlalţi agenţi. Agentul care descoperă un conflict poate să-i ceară unui agent să aştepte (prin comanda WAIT) până când execuţia acţiunii se poate face în siguranţă. Algoritmul se termină când se ajunge la un plan detaliat şi sincronizat.

La fel ca planificarea ierarhică în sisteme cu un singur agent şi planificarea ierarhică distribuită s-a dovedit a fi foarte eficientă; coordonarea şi interacţiunile pot fi rezolvate timpuriu în procesul de planificare.

Negocierea în Planificarea Distribuită

Până acum am discutat două abordări ale planificării distribuite care aveau ca scop utilizarea coordonării în scopul recunoaşterii şi rezolvării conflictelor. Coordonarea poate fi de asemenea folosită pentru a încuraja munca cooperativă. Coordonarea în SMA poate fi realizată în mai multe feluri, una dintre ele ar fi: *negocierea dintre agenţi*.

Dar pentru început să definim mai exact noţiunea de negociere.

"Negocierea este un proces prin care o decizie comună este atinsă de doi sau mai mulţi agenţi, fiecare încercând să atingă un scop sau obiectiv individual. Agenţii prima dată îşi precizează poziţia, care poate fi conflictuală, şi apoi încearcă să se deplaseze spre ţintă făcând concesii şi căutând alternative".

Cele mai importante caracteristici ale negocierii sunt:

- limbajul folosit de agenţii participanţi;
- protocolul urmat de agenţi în negocieri;
- procesul de decizie pe care îl foloseşte fiecare agent.

Cercetătorii de mecanisme de negociere pentru coordonare au fost separate în două categorii diferite, în funcţie de felul în care încurajează negocierile productive şi satisfăcătoare. Prima categorie favorizează negocierile utilizând mediul ca un suport de informaţii de negociere asemenea unui mecanism de reglementare. Se încearcă impunerea unor reguli asupra mediului în care negocierea a început:

- eficienţa – folosirea minimă a resurselor pentru obţinerea unei înţelegeri;

- etabilitate – agenţilor nu le este permis să se abată de la deciziile deja luate;
- simplitate – mecanismele de negociere cer un număr scăzut de agenţi;
- distribuire – nu ar trebui să existe un sistem central de luare a deciziilor;
- simetria – mecanismul de negociere trebuie să fie absolut corect faţă de toţi agenţii.

Ca rezultat s-au identificat trei tipuri de medii : domenii orientate pe valoare, domenii orientate pe stări şi domenii orentate pe sarcini. Ne vom ocupa doar domeniile orientate pe sarcini (task-oriented domains - TOD). O definiţie a TOD ar fi:

"*Un domeniu orientat pe sarcini este unul în care agenţii au un set de sarcini de îndeplinit, toate resursele necesare îndeplinirii sarcinilor sunt disponibile, şi agenţii îşi pot îndeplini sarcinile fără ajutor sau interferenţe cu alţi agenţi*".

Un exemplu de TOD ar fi ***Încărcarea unui domeniu de pe Internet***, domeniu deja descris la începutul acestui capitol. Datorită necesităţii de a negocia, în mediul din acest domeniu se pot fixa următoarele reguli şi constrângeri:

- pentru început agenţii trebuie să precizeze documentele de care au nevoie;
- documentele care sunt comune pentru doi sau mai mulţi agenţi sunt atribuite aleator unui agent;
- încărcarea unui document are un preţ;
- fiecare agent are acces la documentele încărcate de el, de asemenea are acces şi la documentele din setul original de documente încărcate de alţi agenţi.

Acest set de reguli garantează simplitatea, simetria, distribuirea şi eficienţa negocierii dintre agenţi. Stabilitatea va depinde de strategia aleasă de agent ca prim pas al protocolului. Cel mai bine e ca agenţii să declare lista reală a documentelor necesare (adică să fie "oneşti").

A doua categorie se axează pe negocierea centralizată. Aici ideea este de a găsi cea mai bună strategie pentru un agent aflat într-un mediu anume. Până acum s-au dezvoltat două abordări – una dintre ele se bazează pe acţiunea de a vorbi, şi cealaltă pe conceptul de raţionament economic. Acest ultim concept a fost implementat de Rosenschein and Zlotkin într-un protocol de negociere. Este bazat pe presupunerea că numărul de agenţi participanţi este limitat, agenţii cunosc acelaşi limbaj şi ei trebuie să ajungă la o soluţie comună.

6.3.4. Combinarea planificării şi a execuţiei

În IAD planificarea este de cele mai multe ori o problemă de sine stătătoare, de obicei agenţii trebuie să efectueze acţiunile planificate în scopul atingerii obiectivelor. În unele cazuri planul multiagent poate fi anulat de evenimente neaşteptate din timpul execuţiei. Pentru rezolvarea acestei probleme s-au dezvoltat mai multe metode. Cel mai simplu mod de a repara un plan multiagent este de a repeta planificarea şi coordonarea planurilor individuale. Această abordare poate fi complet ineficace în unele domenii şi poate duce la întârzieri semnificative în timpul de execuţie. Pentru a evita recoordonarea, agenţii pot fi proiectaţi astfel încât aceştia să aibă acces la o bibliotecă de planuri reutilizabile dacă planul propriu eşuează. Folosirea planurilor abstracte permite ca agentul să replanifice în caz de eşec fără a afecta planul multiagent

O altă abordare foloseşte algoritmul de interclasare a planului dar pentru planuri mai mari. Acele planuri include ramuri din planul original care se pot refolosi ca o alternativă de atingere a aceluiaşi scop sub alte circumstanţe. Deci, dacă o acţiune eşuează în timpul execuţiei sau este inaplicabilă din anumite cauze, o altă ramură a planului poate fi folosită fără a se începe generarea unui nou process. Această metodă se numeşte ***planificare contingentă*** deoarece agentul prevede alternative de contingenţă a timpului de execuţie.

Planurile care trebuie îmbinate devin mult mai mari şi spaţiul de căutare a conflictelor devine mai greu de construit deoarece toate combinaţiile de execuţie a planului trebuie examinate. Problema finală poate fi simplificată prin asocierea unor ramuri care pot fi într-adevăr urmate (conform unor verificări). Astfel combinaţiile de ramuri cu condiţii contradictorii sunt eliminate.

Alte abordări se axează pe validarea coordonării pre-planificării. Coordonarea pre-planificării se bazează pe presupunerea că restricţiile pot fi analizate înainte de a începe planificarea propriu zisă. Unii cercetători încearcă să fixeze ***legi sociale*** pentru agenţi. O lege socială poate fi definită ca “o interzicere a unor alegeri de acţiuni în contexte particulare”. Un exemplu de legi sociale pot fi considerate legile rutiere, ele de exemplu interzic intrarea în intersecţii la lumina roşie. În SMA legile sociale pot fi codificate de către programatori sau se pot deduce.