Inteligență Artificială
Tema 2: **Planificare**

Tudor Berariu
tudor.berariu@gmail.com
Laboratorul AI-MAS
Facultatea de Automatică și Calculatoare

13 ianuarie 2015

1 Scopul temei

Scopul acestei teme îl reprezintă familiarizarea cu conceptul de planificare și implementarea unui algoritm pentru rezolvarea planificării într-un mediu dinamic.

În continuare se prezintă o descriere generală a problemei ce trebuie rezolvată (Secțiunea 2.1), a predicatelor folosite pentru reprezentarea cunoștințelor depspre universul problemei (Secțiunea 2.2), a operatorilor ce folosiți în planificare (Secțiunea 2.3) și a planificatorului robotului (Secțiunea 2.4).

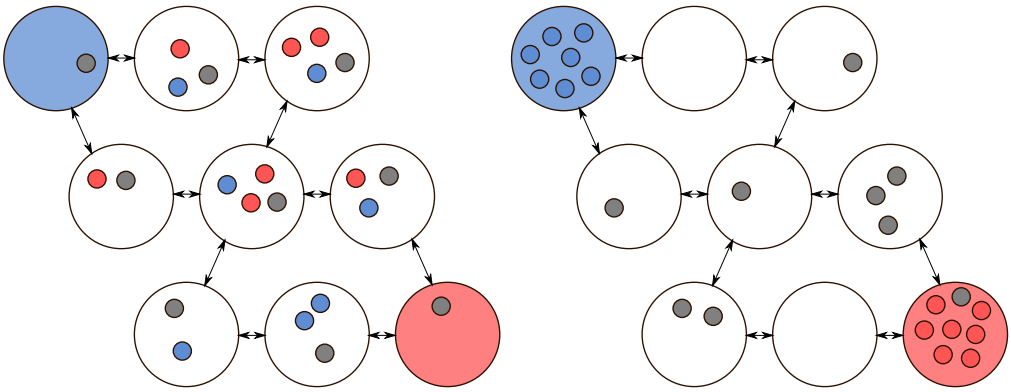
2 Problema celor doi roboți

2.1 Descrierea problemei

Camerele și depozitele Pe un etaj al unei clădiri se află un labirint de camere cu uși de acces între ele. Accesul nu este obligatoriu bidirecțional. Două dintre acestea au o destinație specială, fiind depozite: camera roșie și camera albastră. Restul camerelor sunt albe.

Sferele În aceste camere se află un număr de sfere de culoare roșie, albastră sau gri.

Roboții La început atât în camera roșie, cât și în camera albastră, se află câte un robot de aceeași culoare (roșu, respectiv albastru). Fiecare dintre aceștia are misiunea de a aduce în depozitul propriu toate sferele de aceeași culoare (vezi Figura 1).



(a) Configurația de start (b) O posibilă stare finală

Figura 1: Exemplu de scenariu (camerele și ușile sunt reprezentate prin noduri și arce în graf)

Fiecare dintre roboți se poate deplasa liber dintr-o cameră în oricare altă cameră vecină către care există acces. Fiecare dintre roboți are 2 spații interne de depozitare în care poate încărca câte o sferă indiferent de culoare. Deoarece aceste sfere sunt destul de grele, iar compartimentele se află în părțile laterale ale robotului, acesta nu se poate deplasa dintr-o cameră în alta cu un singur compariment încărcat. Așadar, robotul se poate muta, fie cu ambele spații interne de stocare goale, fie cu ambele încărcate cu câte o sferă. Un robot poate încărca și descărca orice sferă din și în orice cameră cu o singură excepție: nu se pot încărca sferele aflate deja în depozitul lor (sferele duse în depozitul lor nu mai pot fi mișcate).

Misiunea Pe etaj sunt N camere (depozitul roșu, depozitul albastru și $N-2$ camere albe), M sfere roșii, M sfere albastre și N sfere gri (inițial

câte o sferă gri în fiecare cameră, vezi Figura 1a). Un robot își încheie misiunea atunci când toate cele M sfere de aceeași culoare se află în depozitul corespunzător. Poziția finală a sferelor gri nu este importantă.

Mutarea dintr-o cameră într-o altă cameră vecină, încărcarea unei sfere și descărcarea unei sfere se fac într-o unitate de timp. De asemenea, procesul care constă în actualizarea informațiilor despre pozițiile tuturor sferelor din toate camerele și conceperea unui plan (o secvență de acțiuni) necesită o unitate de timp. Cum robotul acționează într-un mediu dinamic, se poate întâmpla ca până la momentul aplicării unei operator, condițiile acestuia să nu mai fie adevărate. Dacă robotul încearcă să execute o acțiune ce nu se poate aplica (de exemplu: încărcarea unei sfere care nu se găsește în camera curentă), atunci acesta se blochează și mai are nevoie de o unitate de timp pentru a-și reveni. Aplicarea unui operator eronat (de exemplu: încărcarea unei a treia sfere, mutarea dintr-o cameră în alta care nu este vecină cu prima, etc.) se penalizează de asemenea prin blocarea robotului pentru o unitate de timp. După ce se deblochează, reintră în rutina de planificare.

Se caută ingineri care să programeze un planificator pentru acești doi roboți astfel încât aceștia să rezolve cât mai repede misiunea pe care au.

2.2 Reprezentarea cunoștințelor

Convenție Vom folosi nume cu litere mici pentru a ne referi la constante (e.g. `room1`, `blue`) și nume ce încep cu litere mari pentru a ne referi la variabile (`Room`, `Color`). De asemenea, vom folosi nume ce încep cu literă mică pentru a ne referi la predicate (e.g. `spheres(·, ·, ·)`) și nume cu litere mari pentru a ne referi la operatori (e.g. `Move(·)`).

Pentru a reprezenta cunoștințele despre mediu se vor folosi predicatele:

- `location(Room)` - cu semnificația că robotul se află în camera `Room`;
- `spheres(Color, Room, N)` - cu semnificația că în camera `Room` se găsesc `N` sfere de culoarea `Color`;
- `color(Room, Color)` - cu semnificația că încăperea `Room` are culoarea `Color`;

- `color(Color)` - cu semnificația că robotul are culoarea `Color`;
- `door(Room1, Room2)` - cu semnificația că se poate trece din camera `Room1` în camera `Room2`;
- `carries(Color, N)` - cu semnificația că robotul are încărcate în compartimentele interne `N` sfere de culoarea `Color`.
- `succ(N1, N2)` - predicat ce va fi adevărat pentru orice numere naturale `N1` și `N2` consecutive (`N2 = N1 + 1`);
- `greater(N1, N2)` - predicat ce va fi adevărat pentru orice numere naturale `N1` și `N2` pentru care (`N1` $>$ `N2`);
- `positive(N)` - predicat ce va fi adevărat pentru orice număr natural strict pozitiv `N`.

Intern robotul poate folosi oricâte alte predicate pentru a reprezenta complet starea sa sau pe cea a mediului, însă doar cele enumerate mai sus vor fi folosite pentru a transmite starea curentă planificatorului atât la începutul programului, cât și pe parcurs când este necesară replanificarea.

2.3 Operatori

Planurile conțin următoarii operatori:

- `Move(Room1, Room2)` care reprezintă acțiunea prin care agentul se mută din camera `Room1` în camera `Room2`. Acțiunea `Move` reușește doar dacă există o ușă între `Room1` și `Room2` și, fie ambele compartimente ale robotului sunt pline, fie ambele compartimente sunt goale (robotul cară zero sau două sfere).
- `Load(Color)` care reprezintă acțiunea prin care agentul culege o sferă de culoarea `Color` din camera în care se află și-o încarcă într-un spațiu intern de stocare liber. Acțiunea reușește întotdeauna dacă în camera în care se află agentul există cel puțin o sferă de culoare `Color`, camera nu are aceeași culoare cu sfera și robotul nu cară deja două sfere.
- `Unload(Color)` care reprezintă acțiunea prin care agentul descarcă o sferă de culoare `Color` în camera în care se află. Acțiunea reușește

numai dacă robotul are în compartimentele interne cel puțin o sferă de culoare `Color`.

Fiecare dintre acești operatori se execută într-o unitate de timp.

Mai există un operator special, `Test(·)`, care îi permite robotului să verifice informații despre lumea înconjurătoare.

- `Test(Condition)` a cărui aplicare constă în verificarea conjuncției de predicate din `Condition`. În cazul în care condițiile sunt adevărate, planul este continuat prin aplicarea următoarei acțiuni, altfel, se abandonează planul curent pentru replanificare.
  `Condition` poate conține doar predicatele `spheres`, `succ`, `greater` și `positive` (restul informațiilor despre lume nu se schimbă pe parcurs). De exemplu, pentru a verifica faptul că în `room1` se găsesc cel puțin două bile roșii:

  $$\texttt{spheres(red, room1, N1)} \wedge \texttt{succ(N1, N)} \wedge \texttt{positive(N)}$$

  sau, pentru a verifica simultan că în camera `room1` se află cu cel puțin două sfere roșii mai multe decât în camera `room2`, iar în camera `room3` se află o singură sferă gri (de data aceasta în sintaxă `Racket`):

  ```
  ((spheres red room1 N1) (spheres red room2 N2) (succ N N1)
  (greater N N2) (spheres grey room3 1))
  ```

2.4 Planificatorul

Planificatorul robotului **nu** are memorie internă. El primește patru informații:

- obiectivul său, un predicat de forma `spheres(Color,Warehouse,M)` (e.g., pentru robotul roșu: `spheres(red, redWarehouse, m)`);
- starea lumii: culoarea lui, camera în care se află el, numărul de sfere de culoare gri, roșie și albastră din fiecare cameră, perechile de camere vecine și culorile camerelor;
- restul de acțiuni ce nu au fost executate, dacă a eșuat aplicarea unui plan sau verificarea condiției unui operator `Test`;

- o listă cu informații suplimentare, pe care robotul a reîntors-o odată cu ultimul plan (permite salvarea unui context de calcul și simulează memoria internă).

Rezultatul planificării conține 2 elemente:

- planul efectiv: o secvență de operatori dintre `Move`, `Load`, `Unload` și `Test`;
- o listă conținând orice informații; aceasta îi va fi retrimisă planificatorului dacă planul eșuează sau verificarea condiției unui operator `Test` eșuează.

3 Cerințe

3.1 [0.3p] Cerința 1: Descriere STRIPS

Folosind predicatele enumerate în secțiunea anterioară, dar și alte predicate suplimentare pe care le considerați necesare, descrieți următorii operatori folosind STRIPS: `Move`, `Load` și `Unload`.

Operatorul `Test` are un statut special și nu trebuie descris.

3.2 [0.7p] Cerința 2: Planificare simplă

Să se implementeze un planificator `memoryless-agent` care construiește planuri pentru aducerea unei sfere în depozit. Planificatorul va fi folosit astfel:

Algorithm 1 Funcționarea robotului

```
1: cât timp mai sunt bile de adus execută
2:     p ← memoryless-agent(goal, world-state)
3:     cât timp planul p mai conține operatori execută
4:         o_1 ← pop(p)
5:         dacă se poate aplica o_1 atunci
6:             execută o_1
7:         altfel
8:             întrerupe execuția planului
```

Cât timp mai există bile de adus în depozit, planificatorul este folosit pentru construirea unui plan care să aducă următoarea sferă în depozit. Atât timp cât se pot aplica operatorii planului, aceștia sunt executați. Dacă la un moment dat o acțiune nu poate fi aplicată, robotul se blochează pentru o unitate de timp și trebuie să replanifice.

Se poate alege orice strategie de planificare pentru rezolvarea acestei cerințe (căutare înainte, căutare înapoi, altceva).

Agentul `memoryless-agent` nu poate include operatorul `Test` în planul lui și nici nu-și poate salva informații pe care să le utilizeze în momentul replanificării.

Planificatorul agentului `memoryless-agent` va fi apelat de fiecare dată astfel:

```
(memoryless-agent goal world-state null null).
```

unde `goal` va fi `(spheres Color Warehouse (+ 1 M))`, `Color` este culoarea robotului, `Warehouse` reprezintă depozitul, iar `M` este numărul de bile duse deja acolo.

Agentul nu va face diferența între o planificare de la zero și o replanificare după eșecul aplicării unui operator.

Testarea se va face folosind planificatorul `memoryless-agent` atât pentru agentul roșu și `dummy-agent` sau `memoryles-agent` pentru cel albastru.

```
(run scenario4 memoryless-agent dummy-agent)

(run scenario2 memoryless-agent memoryless-agent)
```

Recomandare pentru o implementare frumoasă Încercați să construiți un planificator care să funcționeze independent de problema dată (separați algoritmul care construiește planul de detalii ce țin de problema descrisă).

3.3 [BONUS 0.2p] Cerința 3: Tehnici de planificare avansată

Să se implementeze un agent `advanced-agent` care este mai eficient decât `memoryless-agent`. Se poate folosi orice strategie pentru căutarea / construirea planului. Mai mult, agentul poate include în acțiunile din plan operatorul `Test(Condition)`, unde `Condition` este o listă predicate dintre `spheres`, `succ`, `greater` și `positive`. Dacă acestea nu sunt adevărate în starea curentă, se apelează funcția:

```
(advanced-agent goal world-state rest-of-actions saved-info).
```

Pentru a da mai multe posibilități de implementare, `advanced-agent` va primi întotdeauna un singur obiectiv: starea în care toate sferele se află în depozit.

Sarcina acestei cerințe este de a construi un planificator cât mai eficient. Se recomandă exploatarea următoarelor direcții:

- optimizarea căutării planului prin folosirea unei euristici care să o ghideze;
- repararea unui plan eșuat fără a reface replanificarea de la zero (vezi Figura 2);
- gestionarea simultană a mai multor planuri.

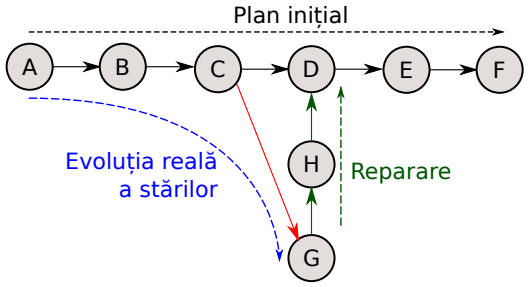


Figura 2: Replanificare: repararea unui plan

În Figura 2 este reprezentat un plan inițial care într-un mediu static ar fi dus la secvența de stări: $A \to B \to C \to D \to E \to F$. Cum însă mediul

este unul dinamic, din starea C s-a ajuns în starea G (tranziția marcată cu roșu). Repararea unui plan presupune găsirea unei secvențe de acțiuni care să ducă mediul înapoi în starea D de unde să se reia restul de acțiuni din planul inițial. Practic replanificarea va produce un plan $G \to H \to D \to E \to F$ realizat prin concatenarea planului de reparare cu restul planului precedent.

Testarea `advanced-agent` se va face prin plasarea lui în același scenariu cu `memoryless-agent` (primul va fi planificatorul robotului roșu, iar celălalt planificatorul robotului albastru). Se va adăuga un al patrulea parametru funcției `run`, `#t`.

```
(run scenario4 advanced-agent memoryless-agent #t)
```

4 Trimiterea temei

Cerința 1 se trimite într-un fișier pdf:

`Nume_Prenume_Grupa_IA_T2.pdf`

Cerințele 2 și 3 se completează în fișierul Racket atașat (`planning.rkt`) unde trebuie implementate funcțiile `memoryless-agent` și [opțional] `advanced-agent`. Așa cum s-a discutat mai sus, fiecare dintre acestea primește 4 parametri:

1. `goals` - un predicat reprezentând obiectivul robotului;
2. `world-state` - listă de predicate care descrie starea lumii (culoarea robotului, camera în care se află, culorile tuturor camerelor, ușile dintre camere, numărul de sfere din fiecare culoare din fiecare cameră);
3. `left-actions` - listă de acțiuni ce au rămas neexecutate din planul precedent [doar pentru `advanced-agent`];
4. `info` - valoarea pe care a reîntors-o aceeași funcție odată cu planul precedent (memoria) [doar pentru `advanced-agent`];

și reîntorc o pereche `(plan . info)` unde `plan` este o listă de acțiuni, iar `info` poate lua orice valoare.

Acțiunile vor fi liste în care pe prima poziție se regăsește numele operatorului, urmat de argumentele acestuia (obiecte din lumea problemei).

Pentru cerințele 2 și 3 se vor adăuga câteva explicații în fișierul pdf:

- algoritmul ales pentru planificare;
- comparații între agenții implementați.

5 Scenarii de test

În fișierul `planning.rkt` sunt definite șase scenarii: `scenario0`, . . ., `scenario5`.

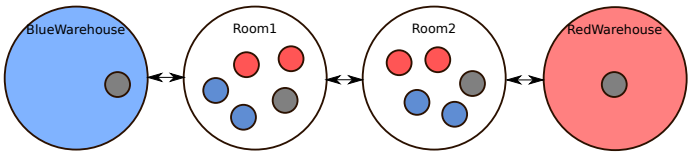


Figura 3: Starea inițială pentru scenariul 0

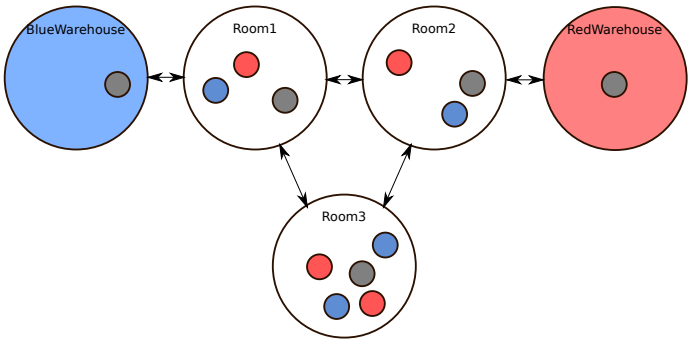


Figura 4: Starea inițială pentru scenariul 1

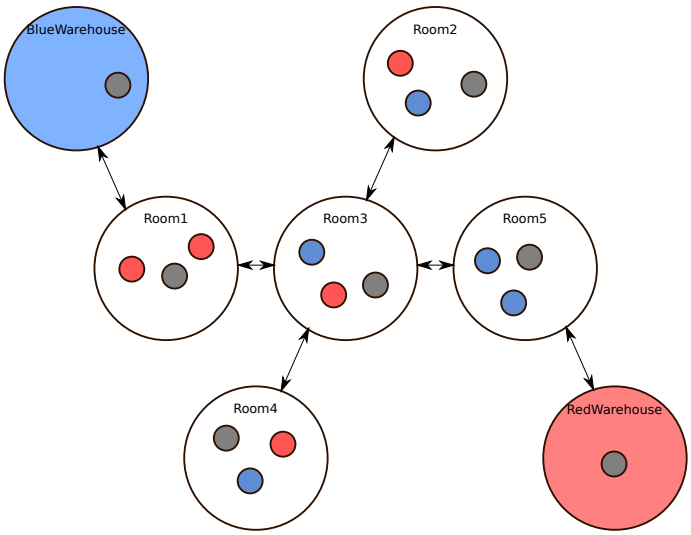


Figura 5: Starea inițială pentru scenariul 2

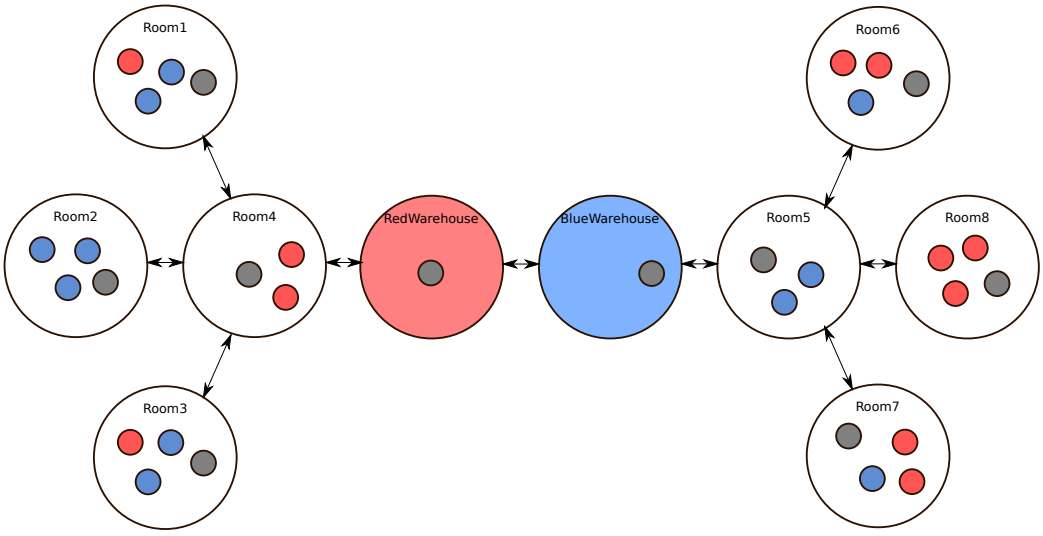


Figura 6: Starea inițială pentru scenariul 3

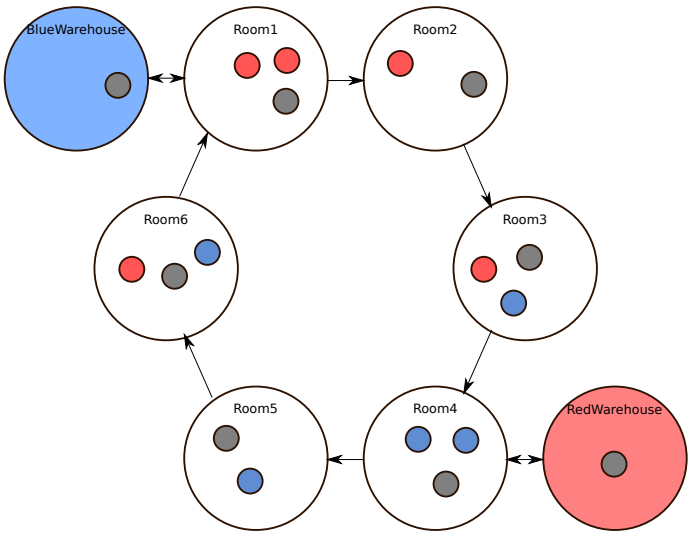


Figura 7: Starea inițială pentru scenariul 4

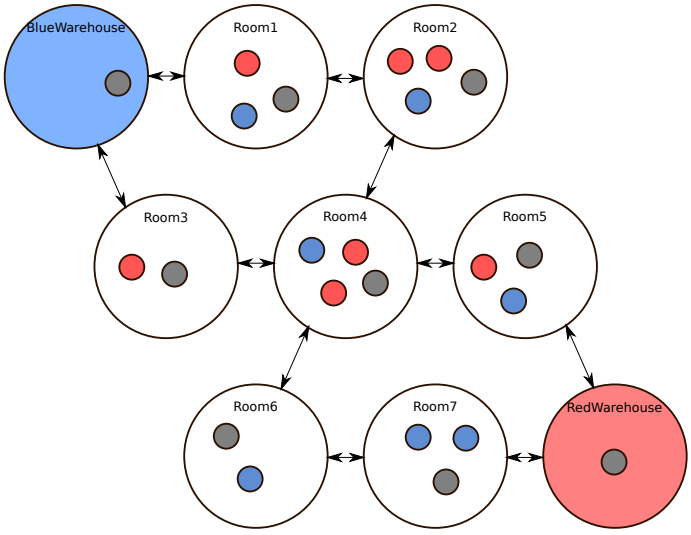


Figura 8: Starea inițială pentru scenariul 5