UNIVERSITATEA TITU MAIORESCU

FACULTATEA: INFORMATICĂ
DEPARTAMENT: INFORMATICĂ
Programa de studii: INFORMATICĂ
DISCIPLINA: INTELIGENȚĂ ARTIFICIALĂ

IA – *Testul de evaluare nr.* 9

Principiile roboților colaborativi

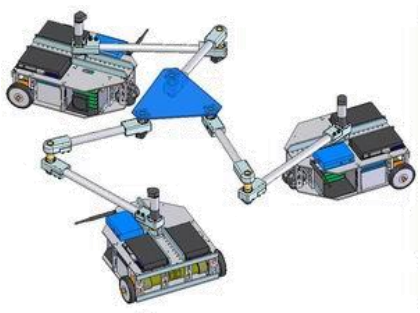
Grupa	Numele și prenumele	Semnătură student	Notă evaluare

Data: ____/____/_____

CS-I dr.ing.
Lucian Ștefăniță GRIGORE

Conf.dr.ing
Iustin PRIESCU

Ș.L.dr.ing.
Dan-Laurențiu GRECU



Cuprins

1. INTRODUCERE

Un sistem[1] robotizat constă în roboți autonomi cu capacități locale de detectare și comunicare, care nu dispun de control centralizat sau de acces la informații globale, situate într-un mediu necunoscut care efectuează o acțiune colectivă. Pe baza acestei definiții, se pot distinge cu ușurință sistemele robotice de roiuri de alte abordări multi-robot. Nu este neobișnuit ca sistemele multi-robot să nu aibă o anumită formă de descentralizare la nivelurile de calcul, comunicare și / sau de operare. Sistemele de roboți sunt adesea inspirate de sistemele naturale în care un număr mare de agenți simpli efectuează comportamente complexe colective - de exemplu, pește, păsări - prin interacțiuni repetate locale între ele și mediul lor. Paradigma de proiectare a roboților permite unui sistem multi-robot să depășească limitele menționate mai sus (detectarea locală, lipsa controlului centralizat etc.) și să opereze în mod autonom în mod coordonat.

O provocare principală în inteligența artificială este proiectarea sistemelor care, deși mențin controlul descentralizat, au agenți capabili să:

(i) dobândească informații locale prin simțire;
(ii) să comunice cu cel puțin un anumit subset de agenți;
(iii) să ia decizii pe baza datelor percepute în mod dinamic.

În timp ce descentralizarea leagă agenții de avantajele unui centru central de calcul și / sau de comunicare central, acesta oferă robustețea sistemului. Sistemul este astfel capabil să realizeze acțiuni colective globale sub o gamă largă de dimensiuni de grup (scalabilitate), în ciuda unei posibile pierderi bruște a mai multor agenți (robustețe) și în condiții necunoscute și dinamice (flexibilitate).

Au fost realizate o serie de eforturi notabile pentru proiectarea roiurilor robotice. Fiecare platformă robotizată explorează potențialul și fezabilitatea câtorva aspecte ale roiurilor și nu toate îndeplinesc toate cerințele unui sistem robotic de roiuri - în conformitate cu definiția de mai sus - sau sunt capabile să funcționeze în medii reale și necontrolate fără o infrastructură de sprijin. Progresele tehnologice implementate pe fiecare platformă fac dificilă punerea la dispoziția altora datorită particularităților fiecărui robot. Într-adevăr, aceste platforme au fost concepute cu considerații de bază în centrul designului lor. Ca o consecință, componentele centrale necesare pentru a permite rotirea sunt complet integrate în interiorul roboților. Mai mult, software-ul este adesea foarte dependent de specificațiile hardware datorită procesului inițial de co-design. Deși această integrare tehnică profundă, lipsită de modularitate, oferă platforme care se pot roti cu ușurință, este și un impediment serios pentru portabilitatea sa către alte platforme mobile.

Utilizatorii care au acces la mai multe unități ale unei platforme robotizate autonome existente - inclusiv în cele comerciale - și care încearcă să studieze roaming nu au în prezent altă opțiune decât să-și modeleze propriul cadru personalizat. O alternativă obișnuită folosită de mai multe grupuri de cercetare constă în simpla expunere a experimentelor care pot fi realizate cu platforma robotică de comerț existent (de exemplu, kilobot, e-puck etc.). Deși această alternativă are avantajul de a purta pe platforme bine testate, economisind astfel o cantitate semnificativă de timp, limitează totuși capacitatea de a proiecta experimente foarte originale. Soluția noastră tehnologică propusă vizează completarea acestui spațiu prin oferirea unui cadru platformă-agnostic - o combinație integrată a unui pachet hardware cu un strat software-ușor portabil la o mulțime de platforme hardware și facilitarea implementării diverselor algoritmi de rotire, reducând în același timp sarcina Asociate cu interfațare dependentă de platformă.

Un instrument hardware / software unificat de o platformă-agnostică, este capabil să:

(i) asambleze și să transforme o colecție de roboți mobili de bază în roiuri pline;
(ii) realizeze comportamente plutitoare versatile utilizând o bibliotecă software ușor de programat.

Natura modulară a suitei hardware și a software-ului permite evoluția, modernizarea și extinderea tehnologiei, independent de specificațiile platformei mobile. Prezentăm câteva rezultate

[1] http://journal.frontiersin.org/article/10.3389/frobt.2017.00012/full#B19

preliminare care validă tehnologia pe două sisteme de rotație foarte diferite. Această tehnologie de rotire este o combinație de hardware / software care permite o gamă largă de platforme multi-robot pentru a realiza rotirea versatilă și receptivă. Acest lucru se realizează prin oferirea fiecărui agent cu un hardware de interfață suplimentar, construit din componente low-cost și off-the-shelf, care sunt integrate cu o bibliotecă software special concepută. Credem că această tehnologie care decuplează considerațiile rotative de la robotică ar putea fi de interes atât pentru cercetători, cât și pentru educatori interesați în principal de studiul rocilor artificiale.

Software-ul scris în Python, dar testat și în C ++, este proiectat într-un mod modular, astfel încât tehnologia să fie portabilă între platforme cât mai aproape posibil, adică cu modificări hardware / software minime. Pentru a evalua eficacitatea tehnologiei propuse, am folosit-o pe două platforme diferite care funcționează în medii necontrolate: un robot de acționare diferențiat și o platformă de suprafață a apei (senzor de bord mobil). Mai mult, o serie de comportamente colective, cum ar fi consensul de poziție, apărarea perimetrală și marșul colectiv, sunt testate pe aceste platforme, confirmând astfel versatilitatea tehnologiei noastre bazate pe rotire.

Pentru a permite pornirea, este esențială realizarea unei comunicări distribuite și luarea deciziilor descentralizate. De exemplu, rolurile naturale realizează comportamente auto-organizatoare și decizii descentralizate prin intermediul schimburilor de informații distribuite prin semnalizarea locală asociate cu mecanisme de semnalizare uneori sofisticate și interacțiuni trofice. Semnalarea se referă la comunicarea care implică capacități senzoriale. Sistemele de roboți mimează roiurile naturale prin faptul că asigură comunicațiile dintre unități și se limitează la schimburile de informații locale prin interacțiuni cu rază scurtă de acțiune. Capacitatea de comunicare în spațiu și timp între platformele individuale conduce la comunicarea distribuită. Prin urmare, platformele ar trebui să fie capabile să stabilească o rețea dinamică și, eventual, o comutare a comunicațiilor și procesarea informații la nivel local, folosind exclusiv capacitățile de calcul de la bordul fiecărui agent individual.

Unitatea de calcul a platformelor multi-robot este o resursă computațională limitată doar pentru a citi datele senzorilor de la robot și a le trimite la o unitate centrală la distanță. Procesorul la distanță ia deciziile adecvate pe baza algoritmului de control al supravegherii și trimite comenzi către robot. Deoarece avem nevoie de tehnologia care să fie ușor de portabil la diferite platforme robotizate, folosim o unitate dedicată de procesare independentă de procesorul de detectare și de acționare al robotului ca "creier" al fiecărui agent. În configurația clasică, calculele sunt efectuate de către Raspberry Pi/BeagleBone, computere cu o singură placă (SBC) echipate cu un număr de intrări și ieșiri multifuncționale. Aceste două SBC furnizează resurse ample de calcul pentru mărimea lor, având în vedere sarcinile la îndemână.

Această unitate computațională primește toate datele senzorilor din informațiile robotului și ale vecinilor prin rețeaua de comunicații pentru a lua o decizie adecvată bazată pe algoritmul de rotire, care este inclus în setul de instrumente software detaliat mai jos; Această ultimă parte este preîncărcată pe fiecare platformă înainte de orice operațiune colectivă.

Din punct de vedere teoretic, comunicarea distribuită poate fi mai bine analizată și înțeleasă folosind concepte teoretice de rețea - chiar și în absența unei rețele reale de comunicare fizică. Recent, studiile unor astfel de rețele de semnalizare în roiuri au dezvăluit nevoia de proprietăți structurale specifice ale rețelei - în ceea ce privește distribuția gradului, calea cea mai scurtă și coeficientul de grupare - pentru a obține o dinamică eficientă a consensului și controlabilitatea dinamică ridicată a roiului de către un subset dat de agenți de antrenare.

Din punct de vedere practic, platformele ar trebui să poată stabili o rețea dinamică, adică o rețea de comunicații. Recent, studiile unor astfel de rețele temporale au relevat necesitatea unor proprietăți structurale specifice ale rețelei - în ceea ce privește distribuția gradului, calea cea mai scurtă și coeficientul de grupare - pentru a obține o dinamică eficientă a consensului și un control dinamic ridicat al roiului de către un subset dat de agenți de antrenare. Cu o rețea de rețele, toți agenții sunt identici (din punctul de vedere al rețelei) și pot să facă schimb de informații cu un anumit set de vecinătăți (metrice, topologice, contururi, etc.) direct fără a implica un al treilea agent sau pentru a trece printr-un hub central sau un router. Aceasta conferă robustețea sistemului, deoarece pierderea

unui subset de agenți nu are un impact critic asupra funcționării restului, presupunând că nodurile se află în intervalul de comunicare limitat. Acest lucru este în contrast puternic cu configurația rețelei stea, unde pierderea agentului de rutare va opri funcționarea întregului sistem. Același scenariu se va întâmpla dacă agenții comunică într-o rețea de rețele, ci se bazează pe un hub computațional centralizat pentru procesarea informațiilor, așa cum este de fapt cazul multor sisteme multi-robot.

Componenta crucială necesară pentru a realiza un sistem complet descentralizat este dispozitivul de comunicare. Cele mai uzuale module folosite sunt XBee, pentru a stabili o rețea de rețele. Rețeaua de comunicații se bazează pe o interacțiune metrică în care informațiile sunt comunicate între agenți din intervalul de comunicare unul altuia (de obicei o linie de vizibilitate de 300 m). Dispozitivul este configurat în modul de difuzare în care informațiile trimise de fiecare agent sunt primite de toți vecinii din intervalul de comunicare. Domeniul de comunicare depinde de diferiți factori, cum ar fi puterea de ieșire a modulului și tipul de obstacole care blochează undele de frecvență radio. Schema bloc hardware a acestei unități de activare a roiului (SEU - Swarm-Enabling Unit).

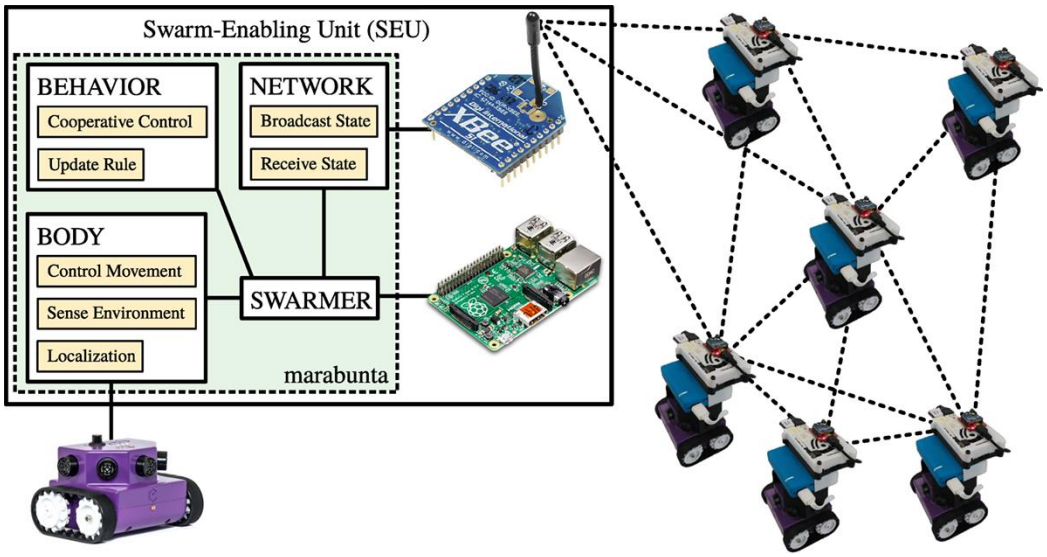


Fig. 1-1 Diagrama bloc a unității care permite rotirea (SEU).
http://journal.frontiersin.org/article/10.3389/frobt.2017.00012/full#B19

SEU servește ca o punte între un anumit robot și roi și este alcătuită dintr-un modul de comunicație și o unitate de procesare care rulează codul. La nivel de software, fiecare agent (sau swarmer) este compus din trei elemente:

1. "corpul" interacționează cu robotul pentru a-și controla mișcarea și a aduna informații de la starea sa și de la datele de mediu sesizate;
2. "rețeaua" interacționează cu modulul de comunicare pentru a transmite starea actuală a agentului la roi și pentru a aduna informații din starea altor agenți;
3. "comportamentul" conține strategia de control cooperativ.

În forma sa cea mai elementară, comportamentul este implementat de o regulă de actualizare care definește mișcarea robotului pentru o anumită fereastră de timp, având în vedere starea actuală a robotului și aceste date colectate din corp și din rețea.

Funcția de actualizare numește o metodă diferită a corpului (roșu), rețea (albastră) și comportament (verde) al agentului.

O preocupare naturală cu o rețea de comunicații atât de dinamică și distribuită este eficiența acesteia în menținerea unui flux susținut de informații între agenții swarming. Acest lucru a fost analizat și cuantificat în timpul experimentelor rotative cu sistemul BoB, cu geamanduri transmițând în mod continuu starea lor la 0,1 Hz. Gama de comunicare așteptată este de aproximativ 310 m, iar modulele sunt capabile să retransmită mesaje prin intermediul mai multor hamei în rețea, ceea ce înseamnă că, în principiu, orice modul poate să le difuzeze în mod global la toți agenții din colectivitate. Cu toate acestea, experimentele noastre arată că atunci când zeci de geamanduri

funcționează, comunicarea este departe de a fi perfectă, iar intervalul eficient de comunicare este semnificativ mai mic. Aceste rezultate oferă o măsură a domeniului de comunicare eficient într-o rețea mare și dinamică a unităților mobile XBee. Pe măsură ce crește numărul de geamanduri expediate, interferențele dintre ele vor determina scăderea numărului de mesaje. Acesta prezintă un exemplu clar de ce algoritmul de control distribuit trebuie proiectat pentru a oferi un comportament colectiv robust împotriva comunicării imperfecte.

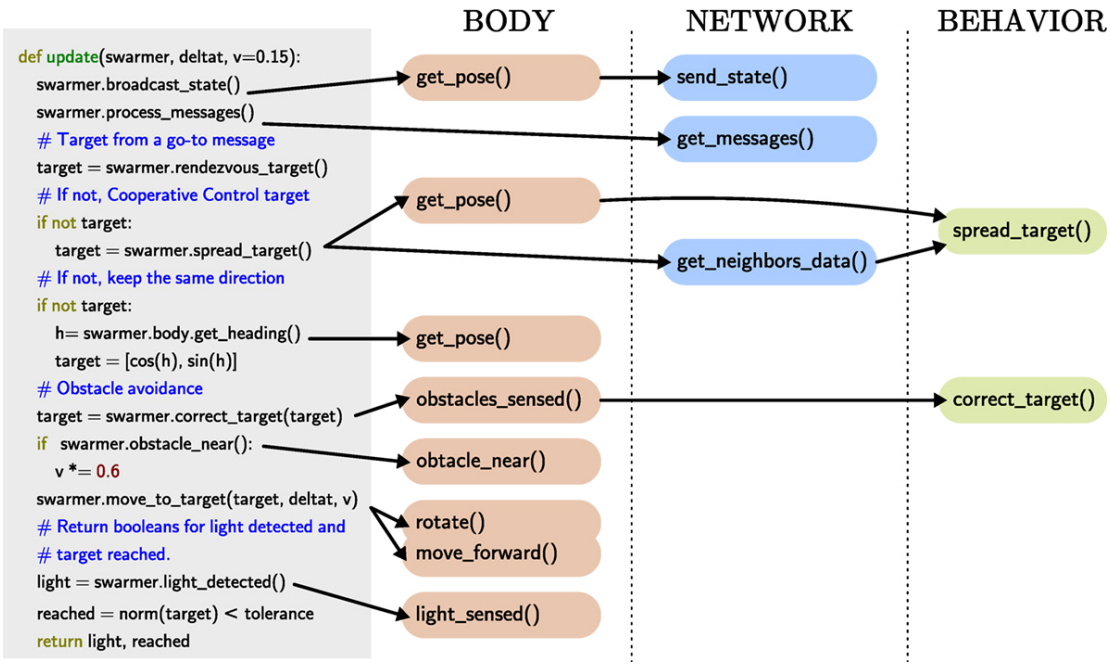


Fig. 1-2 Exemplu de comportament al agenților.
http://journal.frontiersin.org/article/10.3389/frobt.2017.00012/full#B19

Strategia de control cooperativ urmărește eficientizarea comportamentelor colective de către un sistem multi-robot. Pentru aceasta este nevoie de algoritmi de control complet descentralizați. Astfel de strategii de control cooperativa au primit o atenție deosebită din diferite comunități științifice cu scopuri diferite:

- in primul rând comunitatea grafică computerizată;
- apoi comunitatea fizică.

Comunitatea de control a stabilit ulterior un cadru formal, care a fost pus în practică și extins la sistemele multi-robot stabilind legătura dintre regulile de actualizare dinamică ale agenților care interacționează local și strategiile de control cooperativ pentru efectivele de roboți autonomi care zboară. Această încercare a fost alimentată de o activitate intensă de cercetare din partea biologilor și fizicienilor care au căutat să identifice regulile locale de actualizare la nivel de agent, ceea ce conduce la comportamentul observabil al animalelor colective.

Aceasta a dobândit abordări bazate pe cunoaștere, bazate pe abordări biologice, la elaborarea strategiilor de control cooperativ. O astfel de abordare a fost implementată și testată cu succes pe un mic lot de 10 quadrocoptere, folosind GPS pentru localizare și aceeași paradigmă de comunicare distribuită ca cea raportată aici. Folosind taxonomia specifică strategiei descentralizate de control cooperativ urmează o abordare bazată pe comportament bazată pe design.

Tehnologia prezentată a fost testată pe două platforme diferite:

A. eBot, careeste un robot de viteză diferențiat comercial dezvoltat de EdgeBotix, 2 o companie spin-off care dezvoltă roboți educaționali bazați pe roboți de cercetare dezvoltați de laboratorul SUTD MEC.3 Este echipat cu 6 aparate de măsurare cu ultrasunete sau cu infraroșu, unitate de măsurare inerțială (IMU), codificatoare cu două roți și doi senzori de lumină (Fig. 1-3). Viteza maximă a platformei este de 20 cm / s. Are dezvoltat un filtru extins

Kálmán (inclus în API-ul eBot) pentru a localiza robotul în timp real pe baza codificatoarelor IMU și a roților.

B. "BoB" (sistemul "Bunch of Buoys") este un multi-robot distribuit, bazat pe o ambarcațiune mică de suprafață dezvoltată (Fig. 1-4) inițial dezvoltată la MIT pentru a efectua o sesizare colectivă de mediu. (Videoclipurile de detectare colectivă cu mai mult de 50 U în timpul unui test pe teren sunt disponibile online.4,5,6) Este echipat cu un receptor global de satelit (GPS) de poziționare, compas MEMS și accelerometru cu 3 axe. Conceptele omnidirecționale de proiectare conduc la sistemul de propulsie vectorizat, permițând o agilitate maximă cu modificări de direcție aproape instantanee. Viteza maximă a platformei este de până la 1 m / s, deși considerațiile de proiectare sunt mai preocupate de poziționare decât de tranzit.

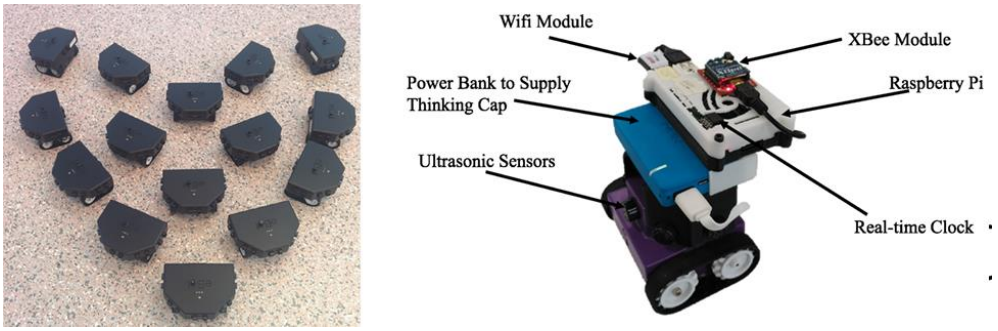


Fig. 1-3 eBot - robot.
http://journal.frontiersin.org/article/10.3389/frobt.2017.00012/full#B19

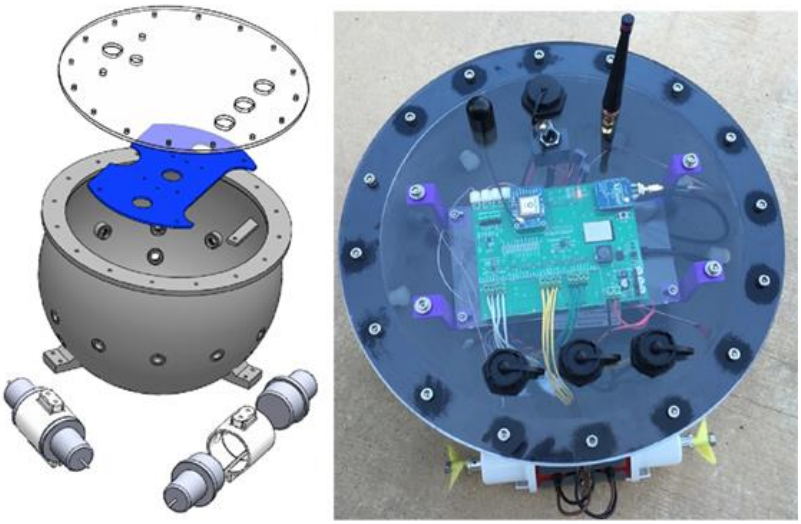


Fig. 1-4 BoB - robot.
http://journal.frontiersin.org/article/10.3389/frobt.2017.00012/full#B19

Robotul BoB are corp de aluminiu turnat ce atenuează interferența magnetică cu elemente senzoriale sensibile MEMS folosite pentru orientarea geamurilor. Multe porturi standard de extindere de-a lungul perimetrului permit adăugarea de dispozitive de măsurare de mediu fără probleme în teren. Un capac de acces mare de sus oferă acces ușor pentru întreținere, după cum este necesar. Toate componentele sunt adăpostite în siguranță în interiorul corpului de apă și pot fi văzute în partea de sus (colțul din dreapta sus).

În cele din urmă, geamul poate fi echipat cu mai mulți senzori pentru a-și monitoriza mediul. Flota de geamanduri (BoB) (Fig. 1-5) nu este limitată la omogenitate, iar unele unități pot fi echipate cu diverși senzori pentru a furniza niveluri diferite de cunoștințe despre apele înconjurătoare.

Fig. 1-5 Flotă de BoB – robot pe suprafața unui lac de acumulare.
http://journal.frontiersin.org/article/10.3389/frobt.2017.00012/full#B19

Software-ul, care reprezintă totodată și conexiunea dintre controlul robotului, managementul comunicațiilor distribuite și comportamentul colectiv este stabilită prin intermediul unui modul Python conceput modular pentru comportamentele diferite ale swarming și care subliniază portabilitatea și ușurința de implementare pe diferite platforme.

Acest modul, interacționează cu robotul la un nivel scăzut printr-o clasă dependentă de platformă ("corpul" agentului). În același mod, gestionarea comunicațiilor este făcută de o "rețea" de nivel scăzut și care depinde de platformă. Ambele clase sunt independente una de cealaltă și sunt, de asemenea, independente de comportamentul la nivel înalt al roiului, definit ca fiind clasa "comportamentală".

Fiecare platformă robotizată diferită, necesită o clasă proprie în urma presupunerii că procesorul care rulează codul are capacitatea de a interacționa cu robotul astfel încât să:

(i) poată trimite comenzi pentru a-l muta în spațiu;
(ii) solicite aceste date necesare localizării unității;
(iii) acceseze informațiile colectate de robot despre mediul său local, în mod tipic, cu ajutorul unui set de senzori.

Unitatea de activare a SEU nu necesită acces total la funcționarea interioară a robotului sau cunoașterea profundă a specificațiilor sale. De exemplu, dacă se utilizează roboți comerciali precum eBot sau e-puck, stabilirea unei conexiuni Bluetooth și utilizarea API-ului furnizat pentru a trimite comenzi către robot este suficient pentru a utiliza această tehnologie pentru a implementa o Roi de roboți. Clasele de corp pentru eBot și e-puck sunt incluse în Python (versiunea marabunta).

Comunicarea dintre agenți este gestionată de o clasă de rețea separată. Separarea acestui lucru de controlul corpului permite asocierea diferitelor platforme robot cu diferite protocoale de comunicare. Această implementare presupune că procesorul are capacitatea de a interacționa cu un modul de comunicare capabil să:

(i) difuzeze mesaje către agenții din apropiere;
(ii) citească mesajele difuzate de alți agenți.

Nu este nevoie ca modulul de comunicare să poată gestiona comunicarea directă sau recunoașterea nodurilor din apropiere. Modulul marabunta oferă o clasă de rețea pentru protocolul

Digimesh, astfel încât să se poată conecta un modul XBee în serie la computerul care execută software-ul pentru a obține comunicații distribuite între agenții furnizați cu tehnologia actuală.

Comportamentul colectiv al roiului este rezultatul comportamentului individual al agenților, implementat într-o clasă de comportament separată, care este hardware-agnostic. Deoarece comportamentul este independent de robotul folosit, această tehnologie permite o rotire eterogenă în care roboții diferiți efectuează comportamentele colective descrise în secțiunea anterioară. În plus, deoarece comportamentul este, de asemenea, independent de comunicarea dintre agenți, un roi poate avea comportamente eterogene în care diferiți agenți urmează diferite comportamente. O platformă fără frecare pentru a experimenta rotația eterogenă poate genera comportamente noi emergente interesante care rezultă din combinația comportamentelor (de exemplu, combinarea unui anumit raport dintre agenții care efectuează un consens și alții care efectuează apărarea perimetrală face ca roca să se împartă în subgrupuri).

Designul modular al tehnologiei permite o experimentare rapidă. Se poate folosi MockBody și MockNetwork incluse pentru a proiecta comportamente colective folosind iterativ simulări. De acolo, se poate schimba corpul la controlerul corect al robotului și se fac teste preliminare de la un singur terminal central de control (prin păstrarea unei "rețele machete" în terminal). În sfârșit, prin acordarea fiecărui robot cu un SEU cu comportamentul dorit și cu un modul de comunicare adecvat, sistemul este gata să efectueze experimente roaming într-un mod cu adevărat descentralizat.

Integrarea hardware și software este validată prin generarea unor date artificiale pentru pereți și prin poziția țintă pentru a localiza, se poate simula ușor comportamentul roiului și se repetă proiectarea algoritmului prin furnizarea comportamentului cu o clasă MockBody și rețea. Pentru experimentul de căutare și explorare, comportamentul unui agent rotativ este definit de regula de actualizare. Această cifră evidențiază dependențele cu metodele implementate în clasele corporale, de rețea și de comportament.

După iterarea designului comportamentului prin simulare, se poate echipa o flotă de 10 eBot-uri cu o unitate de tip "swarming-enabling" (SEU), care constă dintr-un Raspberry Pi cu Python și modulul marabunta încărcat pe acesta, un modul XBee, un modul Bluetooth și o sursă de energie. Pentru a opera acești roboți, trebuie stabilită o conexiune Bluetooth și să se trimită comenzi simple (cum ar fi activarea roților cu o anumită putere sau citirea valorilor fiecărui senzor) prin API-ul Python dedicat. Implementarea unei clase corporale pentru eBot este, în practică, o chestiune de extindere a API-ului furnizat pentru a obține o interfață cuprinzătoare pentru a interacționa cu robotul și pentru a returna datele senzorului într-un format convenabil, hardware-agnostic (de exemplu, oferind coordonatele estimate ale obstacolelor detectate, spre deosebire de datele brute ale senzorilor).

Pentru astfel de experimente:

- nu este nevoie de majoritatea caracteristicilor modulului de comunicare;
- XBee poate fi setat în mod transparent și interfațat doar prin scrierea mesajelor care trebuie trimise;
- citirea celor recepționate printr-o conexiune serială.

Aceste lucruri fac implementarea clasei de rețea pentru un modul XBee destul de simplă. Sarcina principală a acestei clase este de a traduce mesajele primite și de a structura corect aceste date astfel încât celelalte elemente ale modulului să poată avea acces la acestea.

Prin comutarea MockBody pentru eBotBody și MockNetwork de către XbeeNetwork, codul utilizat pentru simulare poate rula pe SEU și face ca roiul să efectueze în mod autonom căutare și explorare.

2. AGENȚII INTELIGENȚI

Agenții inteligenți reprezintă entități computerizate care acționează în locul operatorilor umani, care adună și stochează informații, care trimit sau filtrează mesaje de e-mail. Utilizează noțiunea de cuvânt „cheie" care în viitorul foarte apropriat va deveni foarte utilă, ajutând la identificarea informațiilor, articolelor, imaginilor sau a unor pachete de date ce pot conține elemente de similitudine.

Agentul inteligent[2] reprezintă acel ceva care acționează într-un mediu. Un agent poate, de exemplu, să fie o persoană, un robot, un câine, un vierme, vântul, gravitația, o lampă sau un program de calculator ce cumpără și vinde. Uneori, agenții acționează numai într-un spațiu de informații și ei sunt numiți tot roboți.

Agenții interacționează cu mediul și aceștia de regulă au o structură fizică. Un robot este un agent care are de îndeplinit o misiune. Agenții primesc informații prin intermediul senzorilor lor. acțiunile agentului depind de informațiile pe care le primește de la senzorii săi. Acești senzori pot, sau nu pot, reflecta realitatea. Senzorii pot produce zgomote, pot fi deteriorați, au anumite limitări fizico-chimice-mecanice, astfel încât utilizarea lor la modul general nu este recomandată.

Un agent trebuie să acționeze pe informațiile pe care le are la dispoziție. Adesea, aceste informații sunt foarte slabe, respectiv mărimea fizică, care este de regulă un semnal electric, nu conține destulă informație pentru ca senzorul și elementele de traducere să descrie fenomenul fizic urmărit. Agenții acționează prin intermediul dispozitivelor de acționare – efectori sau actuatori.

Un agent de control este mesajul/comanda pe care agentul o trimite elementelor de acționare. Agenții de multe ori efectuează acțiuni pentru a aduna mai multe informații.

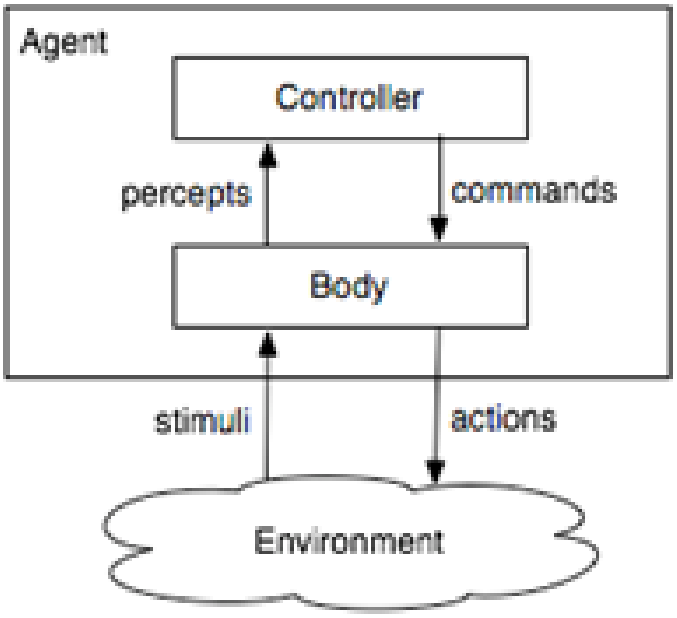


Fig. 2-1 Interacțiunea între un agent și mediul real

Agenții inteligenți sunt înzestrați cu „*rațiune*" pentru a putea fi utilizați să efectueze lucruri utile. Determinarea elementelor de rațional este dată de ceea ce considerăm a fi o „*acțiune corectă*". Implementarea *rațiunii* conferă agentului un anumit grad de decizie. Fiecare agent are o măsură privind criteriile de performanță. De aceea măsurarea performanței se va stabili conform unor proceduri ale unor autorități.

Standardele sunt definite în funcție de o multitudine de aspecte: mediu natural, mediu social, intervenții de urmărire, de decizie, economic etc. Toate fiind cuantificate în unități de timp, ținând cont de efectul urmărit și cauzele care pot conduce la diferite strategii.

O altă problemă pe care agenții inteligenți trebuie să o rezolve, o reprezintă distincția dintre „*raționalitate*" și „*omniscientă*[3]". Efectuarea unei acțiuni este rațională, dar lipsa unor elemente de decriptare a elementelor mediului de lucru, conduce la ideea că agentul respectiv nu poate „*să știe tot*".

[2] http://artint.info/html/ArtInt.html
[3] https://dexonline.ro/intrare/omniscien%C8%9B%C4%83/39032 - calitatea de a ști tot

Definirea noțiunii de „*corect*” pentru algoritmul de funcționare al unui agent este similară cu noțiunile de abstractizare ce se regăsesc în: „*jocul cu bile de sticlă*[4]”, „*GO*[5]” sau „*Labirint*[6]” și depinde de patru elemente (1):

i. gradul de succes – măsura performanței;
ii. percepția completă de la începutul unei acțiuni, până la momentul finalizării acțiunii;
iii. ce știe despre mediul înconjurător;
iv. ce acțiuni poate efectua.

Agentul rațional ideal trebuie să efectueze fiecare acțiune prin maximizarea performanțelor sale, pe baza datelor furnizate de sistemul de percepție. Aparent el respectă „*rațiunea*” cu care este înzestrat, dar el nu știe să se orienteze dual (stânga/dreapta, sus/jos). Raționale prima etapă este de evaluarea riscurilor - este de fapt un alt element de construcție al unui *agent inteligent*. Apoi trebuie să „*caute*” pe baza algoritmului de maximizare a performanței. Toate se subordonează „*analizei*” care are implementat un contor de timp.

În (9) se prezintă formatul ideal al unui agent inteligent:

1) biologic – asumarea elementelor din mediu;
2) comunicare;
3) arhitectură, planificarea task-urilor și control;
4) localizare, mapare[7] și explorare;
5) transport și manipulare obiecte;
6) coordonarea mișcării;
7) roboți reconfigurabili;
8) învățare – Deep Learning, Machine Learning;
9) probleme distribuite adiționale pentru sisteme de roboți mobili autonomi.

Percepție axa OX	Acțiune axa OZ
1.0	1.000000000000000
1.1	1.048808848170152
1.2	1.095445115010332
1.3	1.140175425099138
1.4	1.183215956619923
1.5	1.224744871391589
1.6	1.264911064067352
1.7	1.303840481040530
1.8	1.341640786499874
1.9	1 .378404875209022
⋮	⋮

```
function: SQRT(X)
    z ← 1.0                    /*initial guess*/
    repeat until |z²-x|<10⁻¹⁵
            z ← z − (z²−x)/(2z)
    end
    return z
```

Fig. 2-2 Maparea ideală și rezolvarea ei prin aplicarea metodei celor mai mici pătrate.

Inițiatorul mediului de testare al mediului Inteligenței Artificiale, Alan Turing[8], a considerat că agenții reali și artificiali sunt identici, chiar dacă în lucrare consideră că agenții artificiali trebuie să dețină un algoritm (H&S) care să îi permită imitarea comportamentului uman.

Datele interne din structura unui agent inteligent se actualizează pe măsură ce informațiile care provin din sistemul de percepție ajung să fie interpretate de elementele cognitive ale agentului. Agenții primesc informația în mod secvențial, respectiv, informație după informație, astfel încât la intrare există adusă doar o singură percepție. Care este secvența pe care o va primi agentul ține de proiectant.

[4] https://merlinstore.ro/jucarii/257-joc-cu-bile-de-sticla-standard-solitaire.html
[5] http://www.netinform.ro/blog/2016/03/10/inteligenta-artificiala-il-invinge-pe-campionul-jocului-go/
[6] http://www.scritub.com/sociologie/psihologie/Strategii-de-rezolvare-a-probl2332213815.php
[7] https://dexonline.ro/intrare/mapare/216699 - cartografiere
[8] http://www.turing.org.uk/scrapbook/test.html

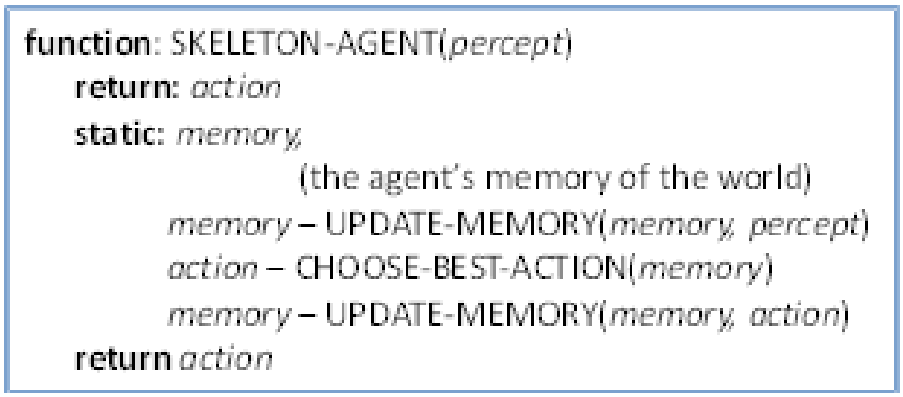


Fig. 2-3 Structura unui agent inteligent.

În continuare, măsurarea performanței din Fig. 2-3 se face aplicând din exterior un mod de comparare al unui comportament. Se scriu astfel datele într-un tabel de căutare (Fig. 2-4) , tabel care funcționează prin menținerea, în memorie, a întregii secvențe de percepție cu ajutorul unui indice, care conține acțiunea corespunzătoare pentru toate secvențele posibile provenite din sistemul de percepție. Tabelul descrie de fapt valorile de risc:

1) numărul de intrări necesare pentru jocul de șah - $10^{43} \div 10^{47}$ – care pot fi asociate unei competiții normale, iar dacă se ia în calcul complexitatea jocului (game-tree complexity), din punct de vedere combinatoric, numărul estimat este în jur de 10^{120} (Shannon number[9]);
2) timpul pentru construirea tablei de joc;
3) faptul că agentul nu are autonomie deloc, deoarece calculul celor mai bune acțiuni este în întregime încorporat, în cazul în care mediul va suferi schimbări, într-un mod neașteptat, agentul va fi pus în dificultate și nu va putea învinge un competitor uman;
4) agentul dispune de un mecanism tip machine-learning, astfel încât, să aibă o anumită autonomie, însă timpul de învățare al tuturor mișcărilor posibile ar fi prohibit.

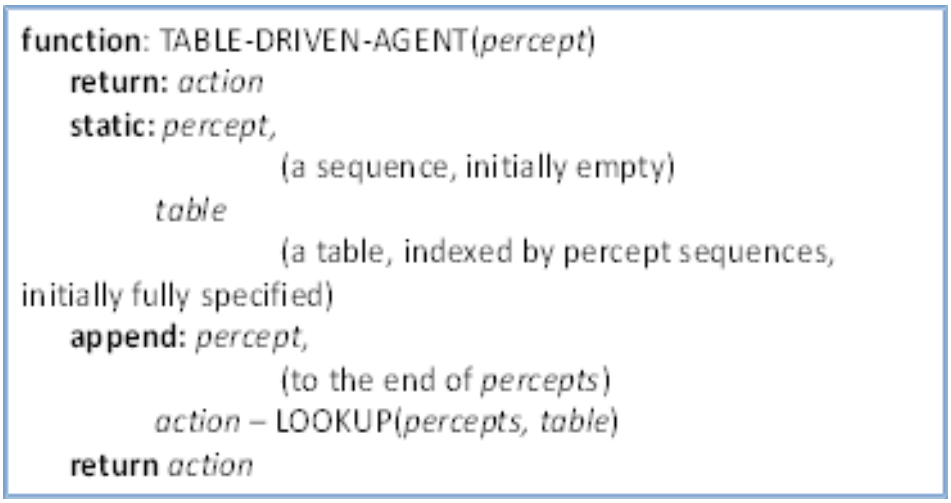


Fig. 2-4 Tabel de căutare predefinit pentru un anumit agent.

Pentru a putea rezolva problema de mai înainte s-a apelat la noțiunea de „*Structură de Control Ierarhică – Hierarchical Control*[10]”. Agentul este împărțit în senzori și un sistem de percepție complex, care trimite informațiile către un controler din care vor pleca informațiile pentru sistemul de acționare. Această structură este deficitară, întrucât este foarte lentă. De exemplu în cazul unui robot care trebuie să parcurgă un labirint ar trebui să dureze un timp foarte mare.

[9] https://en.wikipedia.org/wiki/Claude_Shannon
[10] http://artint.info/html/ArtInt_37.html

De asemenea capacitatea de a descrie mediul înconjurător este limitată.

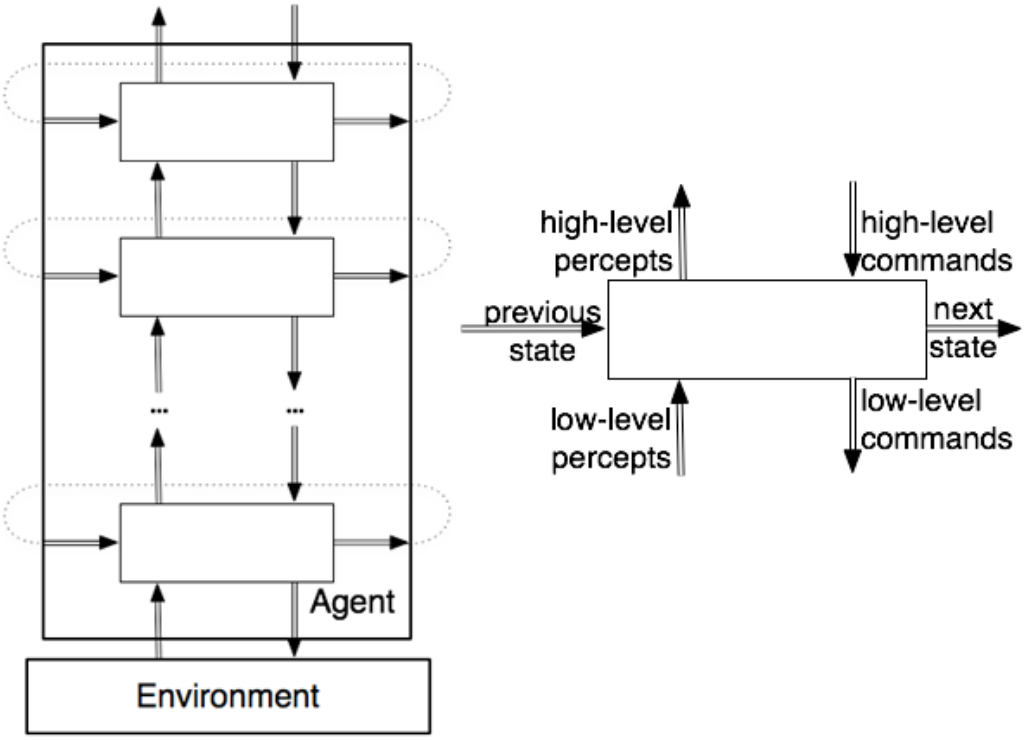


Fig. 2-5 Structura idealizată a unui agent cu control ierarhic.

Arhitectura alternativă este o structură ierarhică de controlere (Fig. 2-5), fiecare layer (strat) vede Layerele de mai jos, ca pe un corp virtual care provine din sistemul de percepție și care apoi trimite informațiile către sistemul de acționare. Straturile de nivel inferior sunt capabile să ruleze mult mai rapid și oferă o vedere mai simplă către straturile superioare, concomitent cu ascunderea informațiilor neesențiale.

Există trei tipuri de intrări pentru fiecare strat, la fiecare dată:

- caracteristicile stării de certitudine, denumite valori sau caracteristici anterioare;
- caracteristicile reprezentând percepțiile din stratul ierarhic de mai jos;
- caracteristicile reprezentând comenzile din stratul ierarhic de mai sus.

Există trei tipuri de ieșiri din fiecare strat, la fiecare dată:

- percepțiile de nivel superior pentru stratul de mai sus;
- comenzile de nivel inferior pentru stratul de dedesubt;
- următoarele valori pentru caracteristicile stării de certitudine.

Pentru a pune în aplicare un controler, fiecare intrare de la un strat trebuie să obțină valoarea sa de undeva. Fiecare comandă de intrare sau de precepte trebuie să fie conectată la o ieșire a unui alt strat. Analiza informațiilor în Layerele superioare utilizează metode discrete și robuste, pe când în Layerele inferioare analiza este de continuu și robust. Controlerul care are implementate ambele sisteme de analiză discret/robust și continuu/robust se numesc controlerele hibride.

Raționamentele cantitative folosește un aparat matematic ce utilizează analiza numerică prin metode integro-diferențiale. Raționamentele calitative folosesc în plus *logica*. Raționamentul calitativ determină legile cantitative ce trebuie aplicate. Valorile discrete pot lua forme de reprezentare diverse: obiective, comenzi cu termeni fuzzy, motivații calitative

În cazul unui robot care se deplasează obiectivele, comenzile și motivațiile sunt date de spațiul, viteza, accelerația și obstacolele ce pot apare în traseul robotului. Bracarea roților poate fi gestionată ca fiind o acțiune instantanee, dar care trebuie ajustată permanent în funcție de timp, destinație, hartă și traiectorie. Senzorii de poziție oferă coordonatele (prin suprapunere GPS și cel triaxial), dacă este echipat cu ambele sisteme de poziționare. Dacă nu, la întâlnirea unui obstacol va efectua un viral la stânga și apoi la dreapta (spațiu/timp). Altfel spus harta este o secvență de locații citite într-o anumită ordine prestabilită (Fig. 2-6).

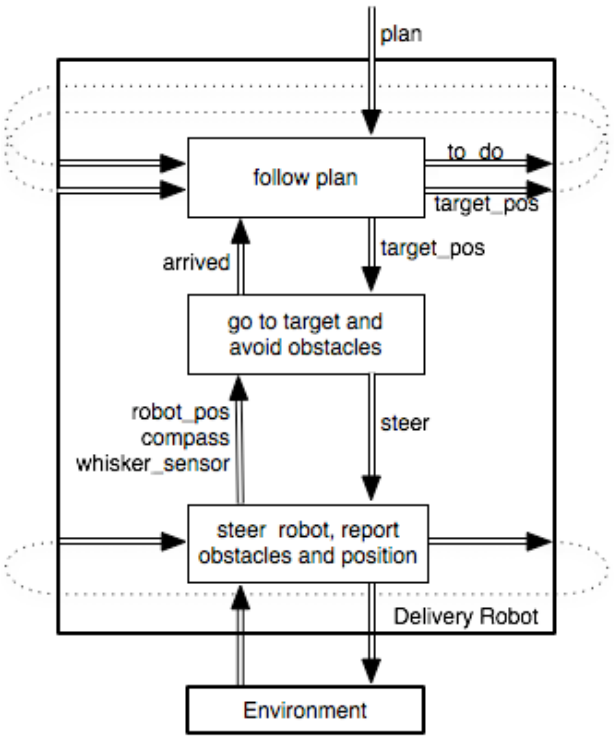


Fig. 2-6 Descompunerea structura idealizate a unui robot în agenți supuși unui control ierarhic.

În interiorul unui layer avem caracteristici ce sunt funcții ale altor caracteristici (cinematică, dinamică, curent consumat, absorbit, sisteme termodinamice, hidraulice etc.) și ale intrărilor provenite de la alte layere. Dacă informațiile ajung într-un centru de prelucrare (Fig. 2-7) pentru a se lua o decizie privind ocolirea unor obstacole, atunci elementele efectoare (motoarele electrice) vor primi comenzile de modificare a turației astfel încât dinamica robotului să fie modificată.

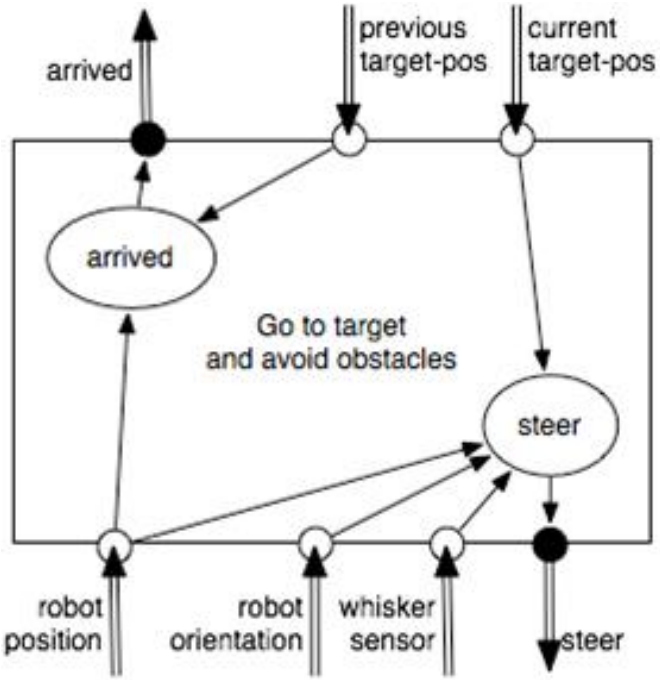


Fig. 2-7 Layerul de mijloc în cadrul structurii ierarhice.

Acest layer nu menține starea de certitudine internă, astfel încât, funcția de tranziție să fie o mulțime ne vidă. Funcția de comandă va specifica direcția robotului în funcție de intrările sale. Faptul că robotul atinge punctul stabilit, atribuie o valoare (care este dată de funcția de mișcare) ținând cont de valoarea anterioară și de o valoare prag predefinită:

```
arrived ←distance(previous_target_pos,robot_pos)<threshold
if whisker_sensor=on
    then steer←left
else if  straight_ahead(robot_pos,robot_dir,current_target_pos)
    then  steer←straight
else if  left_of(robot_position,robot_dir,current_target_pos)
    then steer←left
else steer←right
end if
if arrived and not empty(to_do)
    then
        target_pos' ←coordinates(head(to_do))
        to_do'←tail(to_do)
end if
```

dacă:

- *arrived* ← înseamnă atribuire, distanța este distanța euclidiană, iar pragul este o distanță în unitățile corespunzătoare;
- *straight_ahead (robot_pos, robot_dir, current_target_pos)* este adevărat atunci când robotul este la *robot_pos*;
- *robot_dir* reprezintă direcția când poziția este țintă actuală;
- *current_target_pos* este stabilită ca direcție în sensul longitudinal al robotului (valoare prag);
- *left_of* este funcția de testare în cazul în care obiectivul este spre stânga robotului.

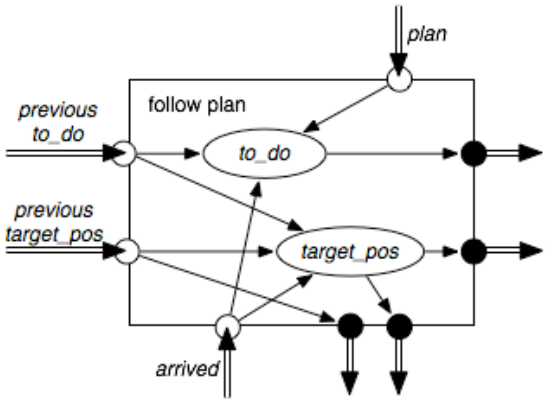


Fig. 2-8 Layerul de deasupra în cadrul structurii ierarhice.

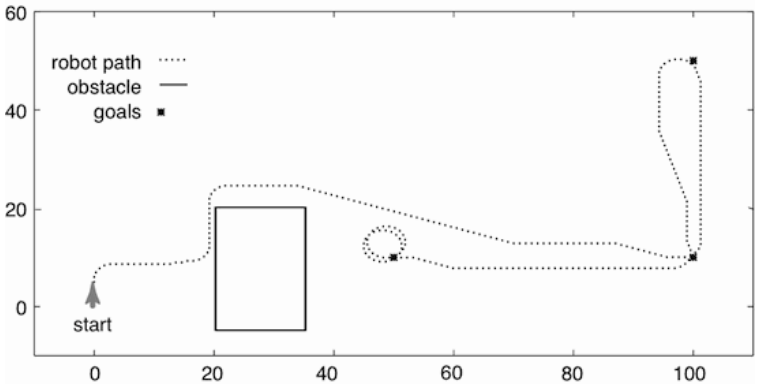


Fig. 2-9 Simulare a deplasării robotului.

3. REZOLVAREA PROBLEMELOR.

Inteligența artificială include scopuri, obiective, acțiuni etc. Un alt tip de agent este cel care trebuie să rezolve probleme (Problem Solving). Este un agent bazat pe obiective. Ei caută secvențe ale unor acțiuni corespunzătoare unor spații dorite. Agentul formulează o vizualizare a problemei, ce decurge din procesul de formulare ce are la dispoziție cunoașterea actuală și rezultatele acțiunilor. Procedural, dacă avem definițiile exacte ale problemelor, se poate construi un proces de căutare pentru găsirea soluțiilor:

- formularea problemei:
  - tipuri de cunoștințe;
  - probleme și soluții bine definite;
  - măsurarea performanței de rezolvare a problemelor;
  - alegerea spațiilor și a acțiunilor;
- probleme:
  - virtuale;
  - reale;
- căutarea soluțiilor:
  - generarea unor secvențe de acțiune;
  - structurarea datelor sub forma arborilor de decizie;
- strategii de căutare:
  - lățimea primei benzi de căutare;
  - respectarea unui cost al căutării;
  - adâncimea primei benzi de căutare;
  - limita adâncimii primei benzi de căutare;
  - căutare iterativă;
  - căutare bidirecțională;
- evitarea repetării spațiilor;
- constrângeri.

3.1 Agenții de rezolvare a problemelor

Agenții inteligenți acționează astfel încât activitatea de rezolvare a problemelor să presupună ca procesele de identificare sau de construire a unui obiect cu anumite caracteristici, să reprezinte soluția problemei. Această acțiune parcurgerea unor succesiuni de stări, pentru a fi maximizată performanța. Cerințele unei *activități de rezolvare a problemelor* (ARP) sunt:

- *structurarea simbolică*: a descrierii problemei inițiale și a fiecărui obiect implicat – baza de date;
- *descrierea instrumentelor computaționale*: necesare investigării sistematice a obiectelor – operatori de transformare;
- *metoda de planificare*: care indică ordinea de aplicare a transformărilor, *astfel încât* (a.î.) soluția problemei să fie găsită, iar costurile să fie reduse – strategia de control;

```
function: SIMPLE-PROBLEM-SOLVING-DRIVEN-AGENT(percept)
    return: action
    inputs: p, a      (percept)
    static: s,        (a sequence, initially empty)
            state,    (some description of the current world state)
            g         (a goal, initially null)
            problem, a problem formulation
    state ← UPDATE-STATE(state, p)
    if s is empty then
            g ← FORMULATE-GOAL(state)
            problem ← FORMULATE-PROBLEM(state, g)
            s ← SEARCH(problem)
    action ← RECOMMENDATION(s, state)
    s ← REMAINDER(s,state)
    return action
```

Fig. 3-1 Agent de rezolvare a problemelor.

Un agent cu mai multe opțiuni imediate de valori necunoscute poate decide ce să facă dacă analizează diferite secvențe posibile care duc la o stare cu valoare cunoscută.

Tipuri de probleme ce pot fi definite, în funcție de domeniul de științe aplicat:

- **deterministă**:
  - *observabilă* $\Longrightarrow single-state\ problem$ - agentul știe starea iar soluția e o secvență;
  - *neobservabilă* $\Longrightarrow conformant\ problem$ (nu există senzori) - agentul știe/~~știe~~ unde se găsește soluția, care dacă $(\exists)$ este la fel o secvență;
- **nedeterministă**:
  - *observabilă parțial/~~parțial~~* $\Longrightarrow contingency\ problem$ problema de contingență:
    - *percepțiile* oferă informații despre starea curentă;
    - *soluția* este un plan contingent sau o strategie;
    - *căutarea și execuția* se îmbină.

3.1.1 Formularea problemei

Procesul de formulare a problemei, în detaliu, se referă la cunoștințele acumulate, statutul și acțiunile agentului. Lucruri care depind de modul de conectare la mediu, la percepții și iarăși acțiuni. Conform (1) există patru tipuri de probleme:

i. *single-state problems* (problemele simple);
ii. *multiple–state problems* (probleme multiple);
iii. *contingency-problems* (probleme urgente);
iv. *exploration-problems* (probleme exploratorii).

Introducerea noțiunilor de *determinism/nedeterminism* aduce în discuție faptul că acțiunile agenților inteligenți pot fi de ignorare/~~ignorare~~ a efectelor acestor acțiuni. Probabilitățile de înfăptuire a unui scenariu sau altuia devin, astfel, elementul care trebuie introdus în formulele de calcul a deciziilor. Senzorii cu care sunt echipați roboții (mașinile, etc.) trimit informații precise (*accurate perceptions*), dar nu trimit nici un fel de logică privind utilizarea acestora. Problema nu mai este una simplă (*single-problems*) ea este deja una complexă (*multiple-problems*). Prin urmare se trece la următorul nivel de abordare: *agenți de urgență*, care sunt nevoiți să despartă informațiile în funcție de caracterul lor. Design-ul deja este multi-layer, iar intercalarea (ierarhizarea) diferitelor layere specializate reprezintă cheia rezolvării problemelor propuse, probleme concrete punctuale. Un robot nu poate rezolva orice problemă. Un exemplu de algoritm stabilește ca prioritate situația cea mai dificilă. Urmând ca prin repetarea într-o buclă închisă a datelor să le transforme într-o hartă a mediului de lucru.

Problema începe să fie cunoscută de agent specificând ce informații are nevoie pentru a putea defini o problemă într-un singur spațiu de stare. Elementele de bază sunt stările și acțiunile și pentru a folosi datele se urmărește algoritmul:

- starea inițială;
- setul de acțiuni posibile;
- testul obiectiv;
- funcția de cost.

Instanțele acestui tip de date vor fi datele de intrare pentru algoritm. Ieșirea este căutarea soluției, care, devine o stare inițială pentru testul obiectiv. Apar în acest fel stări multiple, automat probleme multiple, care necesită intervenții în algoritmul de analiză. Operatorul introduce stările corespunzătoare pentru a putea analiza doar acele date pe care le solicită controler-ul.

Realitatea este diferită de abstract, de aceea acțiunile în domeniul abstract trebuie să fie cât mai simple și să fie cât mai simple față de acțiunea reală.

datatype: PROBLEM
components: INITIAL-STATE, OPERATORS, GOAL-TEST, PATH-COST-FUNCTION

Fig. 3-2 Reprezentarea problemei.

3.1.2 Spații de căutare

Spațiile de căutare sunt asimilate cu mulțimea de stări (10). Mulțimea operatorilor de transformare indică modul în care are loc transformarea $\langle starea\ initiala \rightarrow starea\ finala \rangle$:

- *starea inițială* – descrie condițiile inițiale ale problemei;
- *starea finală* – reprezintă soluția problemei, a cărei reprezentare poate fi explicită sau implicită, sau traseul prin care se ajunge dintr-o stare în alta.

Mulțimea stărilor investigate reprezintă ***spațiul de căutare***. Spre exemplu (în cazul jocului de șah) problema celor 8 regine poziționate pe tabla de joc, astfel încât, să nu se poată ataca reciproc. Acest lucru înseamnă că cerința se traduce, astfel:

- nici o linie, coloană sau diagonală să nu conțină mai mult de o regină;
- starea inițială o reprezintă configurația inițială a tablei de șah, fără regine;
- starea finală o reprezintă configurația tablei cu reginele amplasate.

Din punct de vedere nedeterminist obținerea soluției se face printr-un proces de căutare fără aplicarea unei secvențe de transformări. Selectarea transformărilor și memorarea transformărilor efectuate constituie *strategia de control*. O strategie de control nu este doar o secvență de acțiuni, ci o modalitate de descriere a selecției unei acțiuni ca răspuns la un eveniment extern, cum ar fi rezultatul unui test, rezultatul aplicării unei proceduri complicate de calcul, care trebuie să fie *sistematică*:

- efectuează fiecare acțiune – cerința de *completitudine*;
- nu efectua de mai multe ori aceeași acțiune – cerința de *protejare*.

Aceste tip de căutare se mai numește *euristic*. Strategia utilizează metodologii/proceduri de căutare, iar informațiile sunt reprezentate sub forma unor funcții euristice asociate fiecărei stări.

Există strategii de căutare care nu folosesc funcții euristice, acestea se numesc strategii de căutare neinformate (în orb - *blind-search*). Dintre acestea strategii amintim:

- strategia de căutare pe nivel (Breadth-First);
- strategia de căutare în adâncime (Depth-First);
- strategia de căutare cu cost uniform (Uniform Cost);
- strategia de căutare limitată în adâncime (Depth-Limited);
- strategia de căutare iterativă în adâncime (Iterative-Deepening);
- strategia de căutare bidirecțională (Bidirectional search).

Aplicarea strategiilor *blind*-search caută să răspundă la următoarele întrebări:

i. care este complexitatea de timp a strategiei?;
ii. care este complexitatea de spațiu a strategiei?
iii. este strategia completă?
iv. este strategia optimală?
v. cum se implementează?
vi. ce avantaje și dezavantaje are?

Orice tip de căutare funcționează pe principiul unei cozi de așteptare, sau tip stivă/listă de sortare. Fiecare strategie are particularitatea ei.

Particularizările sunt date de modul în care se răspunde la întrebările de mai sus, întrebări care sunt accesate într-o ordine generată de tipul structurii de date în care se reține nodurile arborelui de căutare.

1. **Breadth-First**: cea mai simplă strategie de căutare este, pe nivel, explorează nodurile în ordinea nivelelor, altfel spus nodurile de pe nivelul x sunt explorate înaintea nodurilor de pe nivelul x+1, exemplu din Fig. 3-3 pentru un arbore binar (doi operatori).

Strategia parcurgerii în lățime (**BF- breadth first**) funcționează respectând mecanismul de tip **FIFO** (first in first out). Ea se bazează pe traversarea tuturor muchiilor disponibile din nodul curent către noduri nedescoperite, care vor fi astfel vizitate. După acest proces, nodul explorat este scos din coadă, prelucrându-se succesiv toate nodurile ajunse în vârful cozii.

Acest mecanism permite identificarea drumurilor de lungime minimă (ca număr de muchii) de la nodul de start către toate vârfurile accesibile lui din graf. Arborele **BF**, ce cuprinde muchii traversate în parcurgerea în lățime, are proprietatea de a fi format doar din drumuri de lungime minimă ce pornesc din nodul considerat ca rădăcină.

În parcurgerea **BF**, pentru fiecare nod se reține mai multe informații și anume:

- nodul predecesor în parcurgere: **TP[nod]**
- lungimea drumului minim către nodul de start: **DMIN[nod]**

- ordinea de parcurgere a nodurilor în **BF**: vectorul **C**
- muchiile arborelui **MABF**: matricea **a** cu **k** linii și **2** coloane

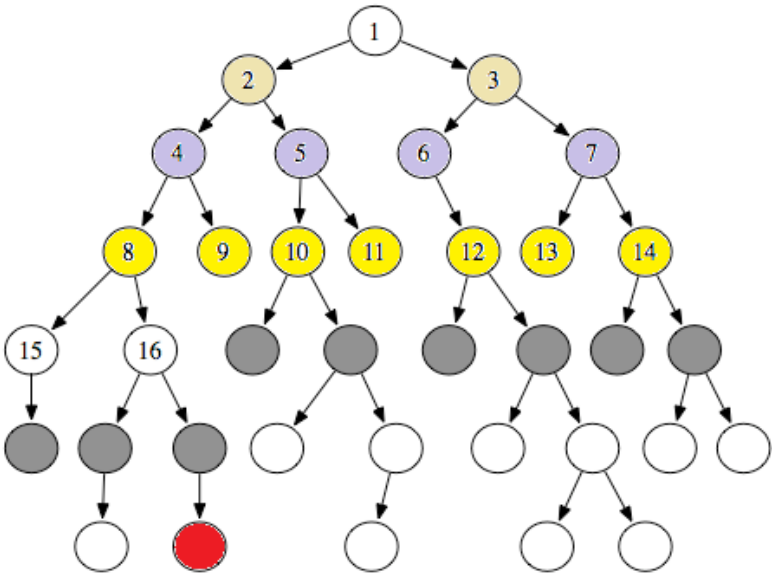


Fig. 3-3 Reprezentarea strategiei de căutare Breadth-First.

Vectorul va avea forma:

$$\begin{aligned}&\text{COADA} = (1,2,4,5,3)\\&\text{DMIN} = (0,1,2,1,1)\\&\text{TP} = (0,1,2,1,1)\\&\text{MABF} = [1,2],[1,4],[1,5],[2,3]\end{aligned}$$

Complexitatea algoritmului este **O(n+m)**. Programul următor realizează implementarea algoritmului **BF** pentru un graf reprezentat prin liste de adiacență alocate dinamic.

```
#include<iostream.h>
#include<ifstream.h>
#define N 50

typedef struct nod{
        int inf;
        nod *leg;
}AD;
AD *L[N];  //listele vecinilor
int VIZ[N],C[N],DMIN[N],TATA[N];  // C-vectorul coada
int n,k,prim,ultim;
int a[2*N][3];  //a retine muchiile arborelui BF
void citire()
{
        int i,j; AD *x;
        Ifstream f("graf.in");
        f>>n;
        for(i=1;i<=n;i++)  //aloca adrese pentru listele de vecini
        {
                L[i]=new AD; L[i]->leg=0;
        }

        while(!feof(f))
        {
                f>>i>>j;
                x=new AD; x->inf=j; x->leg=L[i]->leg; L[i]->leg=x;
                //adaug j in lista vecinilor lui i
```

```
                x=new AD; x->inf=i; x->leg=L[j]->leg; L[j]->leg=x;
                //adaug i in lista vecinilor lui j
        }
        f.close();
}
void BF(int nod)
{
        AD *x;
        VIZ[nod]=1; TATA[nod]=0; DMIN[nod]=0;
        prim=ultim=1; C[prim]=nod;
        while(prim<=ultim)
        {
                nod=C[prim];    //extrag nodul din varful cozii
                x=L[nod]->leg;  //lista vecinilor lui nod
                while(x)
                {
                        if(!VIZ[x->inf])
                        {
                                TATA[x->inf]=nod; VIZ[x->inf]=1;
                                DMIN[x->inf]=DMIN[nod]+1;
                                C[++ultim]=x->inf;  //adaug x->inf in coada
                                k++; a[k][1]=nod; a[k][2]=x->inf;
                                //memorez muchia de arbore [nod,x->inf]
                        }
                        x=x->leg;
                }
                prim++;  //avansez la urmatorul nod din coada
        }
}
void afisare_coada()  //afiseaza nodurile dintr-un arbore BF
{
        int i;
        for(i=1;i<=ultim;i++) cout<<C[i]<<" ";
        cout<<"\n";
}
void drum(int i,int nod)  //afiseaza nodurile dintr-un lant minim
{
        if(i!=nod) drum(TATA[i],nod);
        cout<<i<<" ";
}
void parcurgere_bf()
{
        int nod,i;
        for(nod=1;nod<=n;nod++) VIZ[nod]=0;
        for(nod=1;nod<=n;nod++)
                if(!VIZ[nod])
                {
                        k=0; BF(nod); afisare_coada();
                        cout<<"Muchiile arborelui BF: ";
                        for(i=1;i<=k;i++)
            cout<<"[ a["<<i<<"][1],a["<<i<<"][2]] ";
                        cout<<"\n";
                        for(i=1;i<=ultim;i++)
                                if(C[i]!=nod)
                                {
                                cout<<"de la <<nod<<" la "<<C[i]<<" lantul este: ";

                                        drum(TATA[C[i]],nod);
                                        cout<< C[i],<<" lungime \n"<< DMIN[C[i]]<<" ";
                                }

                }
```

```
}
void main()
{
        int i;
        citire();
        cout<<"Parcurgere BF: \n";
            parcurgere_bf();
        cout<<"TATA_DF=("; for(i=1;i<=n;i++) cout<<TATA[i]<< " ",;        cout<<")\n";
}
```

2. **Depth-First**:

Strategia de căutare în adâncime frontiera acționează ca o ultimă intrare/ieșire dintr-o stivă. Elementele sunt adăugate în stivă, una câte una, iar cel selectat pentru extragere, este și ultimul intrat.

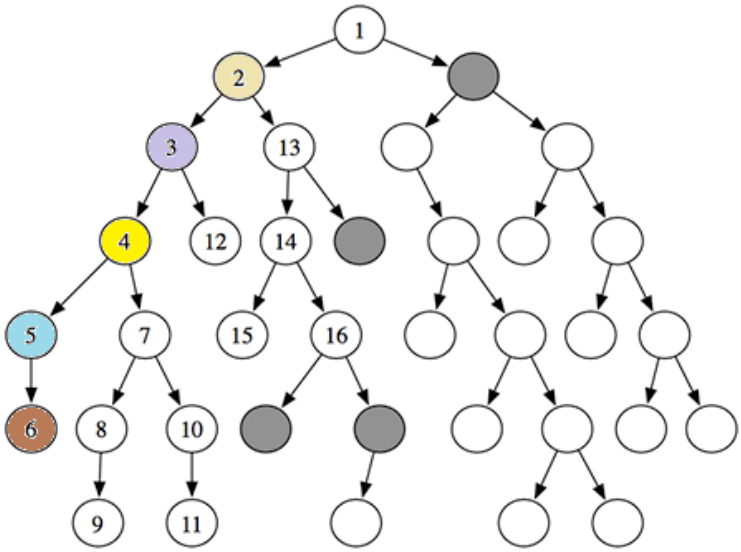


Fig. 3-4 Reprezentarea strategiei de căutare Depth-First.

Strategia de căutare în adâncime, este similară cu a grafurilor, care presupune traversarea unei muchii a nodului curent spre un nod ce urmează să fie descoperit. Când se termină de explorat muchiile nodului curent se revine în nodul din care s-a produs explorarea nodului curent. Procesul iterativ continuă până când toate nodurile au fost explorate. Strategia DF (Depth-First) respectă mecanismul **LIFO** (Last in First Out). Nodurile care sunt eliminate din stivă, nu mai au nici o muchie disponibilă pentru traversare. Muchiile grafului se împart în:

- muchii de avans – muchii de arbore;
- muchii de întoarcere – unesc un nod cu predecesorul.

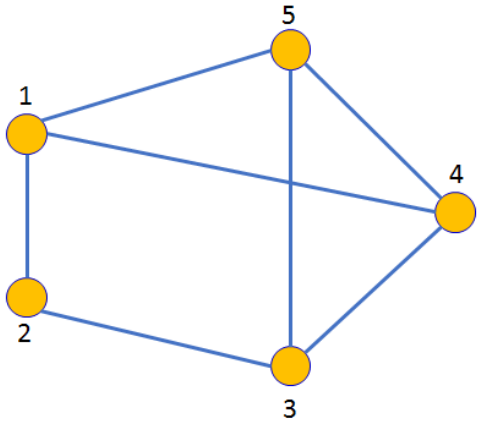


Fig. 3-5 Reprezentarea strategiei cu mecanism LIFO, parcurgerea în ordinea cronologică.

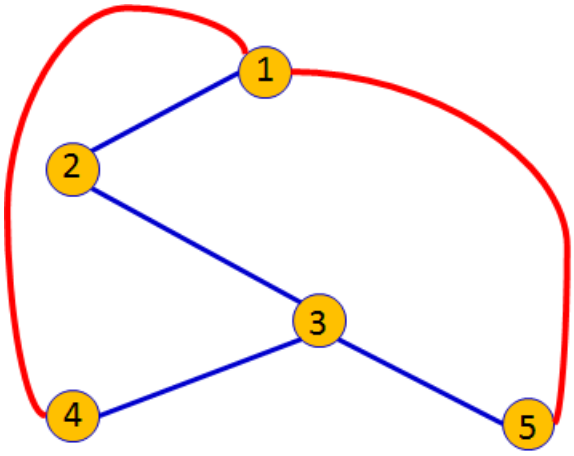


Fig. 3-6 Reprezentarea arborelui/grafului parțial.

În cadrul reprezentării din Fig. 3-6, muchiile de întoarcere sunt $(1 \leftrightarrow 4)$ și $(1 \leftrightarrow 5)$:

- nodul predecesor TP [nod];
- introducerea nodului curent în stivă TI [nod];
- finalul explorării TF [nod];
- muchii de avansare MA, matrice cu k – linii și 2 coloane;
- muchii de întoarcere MI, matrice cu t – linii și 2 coloane.

Cei trei vectori vor avea formele:

$$\begin{vmatrix} \text{TI} = (1,2,3,4,6) \\ \text{TF} = (10,9,8,5,7) \\ \text{TP} = (0,1,2,3,3) \\ \text{MA} = [1,2],[2,3],[3,4],[3,5], \\ \text{MI} = [1,4],[1,5] \end{vmatrix}$$

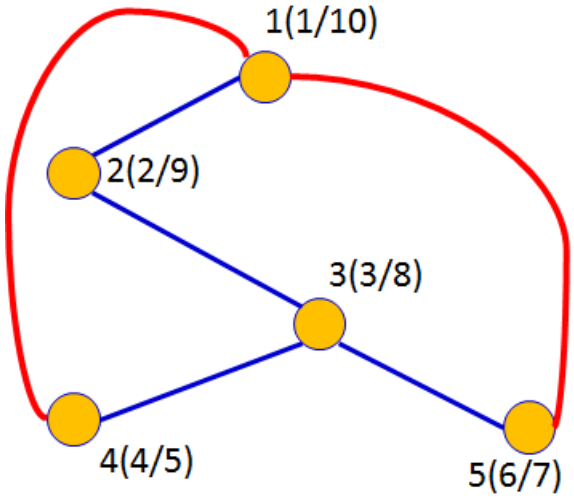


Fig. 3-7 Reprezentarea vectorilor de căutare.

Determinarea muchiilor arborelui BF (de avansare), precum și a celor de întoarcere. Graful este G(V,E), unde V – mulțimea celor n noduri, E este mulțimea muchiilor și graful privind complexitatea algoritmului este O(n+m).

Este posibil să determinăm și muchiile arborelui **BF** (de avansare), precum și muchiile de întoarcere.

Considerând graful **G(V,E)**, unde **V** este mulțimea celor **n** noduri și **E** este mulțimea celor **m** muchii, și graful reprezentat prin listele de adiacență alocate dinamic, complexitatea algoritmului este **O(n+m)**. Vectorul **VIZ** indică la fiecare pas ce vârf a fost explorat. Inițial, toate vârfurile sunt nevizitate și acest vector

este inițializat cu 0. În vectorul **TI** se memorează pentru fiecare nod timpul la care începe explorarea, iar în vectorul **TF** când se încheie explorarea unui nod. În vectorul **TP** se memorează pentru fiecare nod predecesorul în parcurgerea **DF**. Muchile de avansare se memorează pe liniile matricei **a**, numărul acestor muchii fiind **k**. Când un nod **j** este parcurs plecând din nodul **i**, muchia **[i,j]** este de avansare. Când un vecin **j** al lui **i** este deja vizitat, muchia **[i,j]** este de întoarcere. Afișarea muchiilor de avansare, respectiv de întoarcere se realizează cu ajutorul a două funcții recursive care folosesc matricea de adiacență construită simultan cu construirea listele de adiacență. După afișarea unei muchii, aceasta se elimină din graf.

```
#include<iostream.h>
#include<ifstream.h>
#define N 50
typedef struct nod{
        int inf;
        nod *leg;
}AD;
AD *L[N];  //listele vecinilor
int VIZ[N],TI[N],TF[N],TP[N],n,timp,k,t;
int a[N][N];  //matricea de adiacenta
void citire()
{
        int i,j; AD *x;
        Ifstream f("graf.in");
        f>>n;
        for(i=1;i<=n;i++)  //aloca adrese pentru listele de vecini
        {
                L[i]=new AD; L[i]->leg=0;
        }

        while(!feof(f))
        {
                f>>i>>j;  a[i][j]=a[j][i]=1;
                x=new AD; x->inf=j; x->leg=L[i]->leg; L[i]->leg=x;
                //adaug j in lista vecinilor lui i
                x=new AD; x->inf=i; x->leg=L[j]->leg; L[j]->leg=x;
                //adaug i in lista vecinilor lui j
        }
        f.close();
}
void parcurge_df(int nod)
{
        AD *x=L[nod]->leg;
        VIZ[nod]=1; timp++; TI[nod]=timp;  //timpul de incepere
        cout<<"%d ",nod);
        while(x)
        {
                if(!VIZ[x->inf])
                {
                        TP[x->inf]=nod;
                        parcurge_df(x->inf);
                }
                x=x->leg;
        }
        timp++; TF[nod]=timp;  //timp de terminare
}
void muchii_intoarcere(int nod)
{
        int i;
        for(i=1;i<=n;i++)
                if(a[nod][i])
                        if(TATA[nod]!=i && nod!=TP[i])
                        {
```

```
                                cout<<"["<<nod<<","<<i<<"] ";
                                a[nod][i]=a[i][nod]=0;
                                muchii_intoarcere(i);
                        }
}
void muchii_avans(int nod)
{
        int i;
        for(i=1;i<=n;i++)
                if(a[nod][i])
                {
                        cout<<"["<<nod<<","<<i<<"] ";
                        a[nod][i]=a[i][nod]=0;
                        muchii_avans(i);
                }
}
void DF()
{
        int nod;
        for(nod=1;nod<=n;nod++) VIZ[nod]=0;
        timp=0;
        for(nod=1;nod<=n;nod++)
                if(!VIZ[nod])
                {
                        parcurge_df(nod); cout<<"\n";
                        cout<<"Muchii de intoarcere: "; muchii_intoarcere(nod);
                        cout<<"\n";
                        cout<<"Muchii de avans: "; muchii_avans(nod);
                        cout<<"\n";

                }
}
void main()
{
        int i;
        citire();
        cout<<"Parcurgerea DF: \n"); DF(); cout<<"\n";
        cout<<"\nTI=("); for(i=1;i<=n;i++) cout<<TI[i]<<" "; cout<<")";
        cout<<"\nTF=("); for(i=1;i<=n;i++) cout<<TF[i]<<" "; cout<<")";
        cout<<"\nTATA_DF=("; for(i=1;i<=n;i++) cout<<"%d ",TP[i];
    cout<<")\n";
}
```