UNIVERSITATEA „POLITEHNICA“ din BUCUREȘTI

Facultatea de Automatică şi Calculatoare

Catedra de Calculatoare

Fenomene Emergente în Sistemele Bazate pe Agenţi

Rezumat

Autor: Ing. Șerban Iordache

Conducător de doctorat: Prof. dr. ing. Florica Moldoveanu

BUCUREȘTI 2011

1. INTRODUCERE

1.1. Motivaţie

Emergența reprezintă unul dintre cele mai interesante fenomene care se manifestă în sistemele complexe. Esența sa constă în apariția unor proprietăți care nu se regăsesc în elementele ce compun sistemul considerat. Avem de a face cu un fenomen emergent atunci când interacțiunea dintre componente este guvernată de reguli relativ simple, dar care la nivel macroscopic generează un comportament complex. In multe cazuri acest comportament este greu sau chiar imposibil de dedus printr-o simplă analiză a proprietăților elementelor ce compun sistemul. Conceptul de emergenţă este adesea exprimat concis prin fraza: „întregul este mai mare decât suma părţilor sale”. Emergența se manifestă în sisteme descentralizate, în care comportamentul observat nu este rezultatul unui control la nivel global, ci apare ca efect al interacțiunii dintre entități independente, pe care le putem numi generic agenți.

Un exemplu clasic din natură este cel al unei colonii de aşa numite „insecte sociale” (furnici, termite, albine etc.). O astfel de colonie poate fi privită ca un macroorganism cu comportament complex, capabil să se adapteze situațiilor neprevăzute, deşi fiecare insectă în parte este lipsită de inteligență şi reacționează numai pe baza unor reguli simple la stimulii din mediul înconjurător. De exemplu, anumite specii de termite îşi clădesc un muşuroi a cărui înălțime poate atinge câțiva metri. În interiorul acestuia, datorită unui sistem complicat de canale ce asigură o răcire pasivă, temperatura se menține constantă, indiferent de fluctuațiile din mediul exterior. Arhitecții şi inginerii s-au inspirat din acest model pentru a construi clădiri care nu necesită instalații de aer condiționat pentru reglarea temperaturii [1]. In cazul termitelor însă, muşuroiul nu este construit pe baza planurilor unor arhitecți şi nici sub coordonarea unor ingineri. Fiecare termită acționează după un algoritm simplu, dar la nivelul întregii colonii rezultatul este uimitor.

Fenomenele emergente pot fi observate în cele mai diverse sisteme. Exemplele includ crearea ambuteiajelor, variaţia preţurilor pe piaţa liberă, apariţia striaţiilor în dunele de nisip, formarea unui fulg de zăpadă sau a unui uragan. Cele mai surprinzătoare exemple pot fi găsite însă în sistemele biologice: o celulă vie ia naştere din interacţiunea moleculelor sale; sistemul imunitar, capabil să protejeze organismul împotriva bolilor, este rezultatul acţiunii combinate a câtorva tipuri de limfocite; celulele din creier se auto-organizează într-o reţea neuronală complexă, care duce la apariţia inteligenţei şi chiar a conştiinţei.

Întrega materie din univers este compusă din particule elementare. De aceea, chiar şi cele mai complexe fenomene reprezintă în ultimă instanţă rezultatul interacţiunii câtorva tipuri de particule elementare. Cum este posibil acest lucru? Cum pot aceste particule elementare să se auto-organizeze în structuri din ce în ce mai complexe? Cum poate viaţa să apară din materie lipsită de viaţă? Cum ajung fiinţele vii sa aibă scopuri proprii, când particulele din care sunt ele constituite sunt lipsite de orice scop? Cum pot să apară inteligenţa şi conştiinţa din particule lipsite de inteligenţă?

Deşi aceste întrebări sunt îndeajuns de importante pentru a justifica studiul emergenţei, cercetarea prezentată în lucrarea de faţă are o altă motivaţie principală: problema proiectării unui anumit comportament emergent. Ca urmare a creşterii continue a numărului de aplicaţii descentralizate, bazate pe agenţi, aceasta reprezintă o problemă importantă, pentru care nu există în momentul de faţă o metodologie unanim acceptată. Metodele tradiţionale de inginerie software nu pun în valoare comportamentul emergent manifestat de aplicaţiile bazate pe agenţi. Cele mai multe metodologii orientate pe agenţi consideră că fenomenele

emergente sunt indezirabile şi încearcă să prevină comportamentul „neaşteptat” impunând diverse constrângeri agenţilor. Ar fi bineînţeles preferabil să proiectăm aplicaţii care exploatează comportamentul emergent în loc să îl evite, dar aceasta este o sarcină dificilă, ţinând cont de aparenta impredictibilitate a fenomenelor emergente.

În ultimii ani, unii cercetători s-au inspirat din natură pentru a proiecta algoritmi care produc un anumit comportament emergent dorit. Multe sisteme naturale sunt capabile să se adapteze la schimbările rapide din mediu şi pot efectua în mod eficient anumite operaţii pentru care algoritmii convenţionali cunoscuţi sunt impractici. De aceea, este tentantă ideea de a imita astfel de sisteme naturale pentru a obţine un comportament similar. Un exemplu în acest sens îl oferă optimizarea cu colonii de furnici [11], care este o metodă euristică inspirată de modul în care furnicile îşi caută hrana. În drumul lor către o sursă de hrană şi înapoi spre cuib, furnicile depun mici cantităţi de substanţe chimice numite feromoni. Celelalte furnici au capacitatea de a percepe aceşti feromoni şi urmează cu predilecţie drumurile pe care concentraţia de feromoni este mai ridicată. Această simplă strategie de căutare a hranei conduce în cele din urmă la descoperirea drumului cel mai scurt până la sursa de hrană. Cu alte cuvinte, drumul cel mai scurt apare ca urmare a interacţiunii dintre furnici şi mediul înconjurător. Optimizarea cu colonii de furnici este inspirată de acest fenomen emergent şi a fost folosită iniţial pentru a rezolva problema comis voiajorului. Ulterior, ea a fost aplicată la un număr mare de alte probleme de optimizare combinatorială şi a fost de asemenea adaptată pentru probleme de optimizare continuă.

Deşi natura constituie o bogată sursă de inspiraţie, nu întotdeauna poate fi găsit un sistem natural care manifestă un anumit comportament emergent. În plus, nu este de ajuns să fie identificat un sistem corespunzător, ci este necesar să se înţeleagă modul în care acesta funcţionează, pentru a îl putea imita. De aceea, o întrebare importantă care apare este cum putem proiecta un sistem bazat pe agenţi în cazul în care nu găsim o sursă de inspiraţie în natură. În lucrarea de faţă am atacat această problemă atât prin abordări teoretice, cât şi practice.

Punctul de interes principal în această teză este proiectarea unui comportament complex şi adaptiv, asemănător cu cel manifestat de organismele vii, deoarece este foarte dificil să se obţină un astfel de comportament folosind tehnici tradiţionale de inginerie software. Una din ipotezele centrale de la care am pornit este că pentru a proiecta un astfel de comportament, este preferabil să considerăm sisteme eterogene cu agenţi simpli, în loc de a ne axa pe sisteme omogene cu agenţi complecşi.

1.2. Contribuţii originale

Teza de faţă aduce trei contribuţii originale importante:

- **Un formalism matematic al emergenţei în sistemele bazate pe agenţi.** Fenomenele emergente sunt adesea descrise ca neaşteptate, surprinzătoare sau greu predictibile. De aceea, multe definiţii ale emergenţei implică un anumit grad de subiectivitate. Alte definiţii oferă o descriere riguroasă a proprietăţilor ce caracterizează fenomenele emergente, dar nu reuşesc să surprindă toate aspectele legate de emergenţă, sau necesită calule extrem de complexe pentru a putea decide dacă un anumit comportament este emergent sau nu. Pentru a oferi o definiţie care este atât obiectivă, cât şi adecvată pentru scopuri practice, am adoptat un punct de vedere diferit asupra emergenţei. Formalismul propus este bazat pe ideea că o definiţie a fenomenelor emergente trebuie să ia în considerare numai modul în care acestea apar şi nu trebuie să discute proprietăţile acestor fenomene.

- **MetFrEm – un meta-framework pentru studiul emergenţei.** Pentru a putea studia emergenţa într-o manieră riguroasă, am introdus un meta-framework numit MetFrEm, care permite modelarea unor framework-uri algoritmice variate, bazate pe populaţii de agenţi care interacţionează. MetFrEm favorizează modelarea sistemelor descentralizate, puternic eterogene, cu agenţi ghidaţi de reguli simple.
- **Metaeuristica de căutare bazată pe consultanţi (Consultant-Guided Search - CGS).** Metaeuristica propusă face parte din clasa "swarm intelligence" şi pune în valoare comportamentul emergent manifestat de o populaţie de persoane virtuale care interacţionează. Am aplicat această metaeuristică la o serie de probleme de optimizare combinatorială, iar rezultatele experimentale arată că CGS atinge o performanţă de cel mai ridicat nivel în cazul fiecăreia dintre aceste clase de probleme:
  - *problema comis voiajorului (TSP - Traveling Salesman Problem)* – Experimentele efectuate cu şi fără căutare locală arată că CGS depăşeşte în performanţă cei mai buni algoritmi de optimizare cu colonii de furnici pentru TSP: Ant Colony System [10] şi MAX-MIN Ant System [32].
  - *problema asignării pătratice (QAP - Quadratic Assignment Problem)* – Algoritmul CGS pentru QAP se dovedeşte mai performant decât MAX-MIN Ant System [31], care este în prezent cel mai bun algoritm de optimizare cu colonii de furnici pentru această clasă de probleme.
  - *problema generalizată a comis voiajorului (GTSP - Generalized Traveling Salesman Problem)* – Rezultatele experimentale arată că din punct de vedere statistic nu există diferenţe semnificative între algoritmul nostru şi algoritmul memetic al lui Gutin şi Karapetyan [12], care reprezintă cea mai performantă euristică pentru GTSP publicată până în momentul de faţă.

Pe lângă aceste contribuţii, cercetarea prezentată în această teză a condus la crearea a trei produse software open source, importante din punct de vedere practic pentru comunitatea ştiinţifică:

- **AgSysLib** (http://agsyslib.sourceforge.net/). AgSysLib este un pachet software pentru rezolvarea problemelor folosind metode bazate pe agenţi. El asistă utilizatorii în toate aspectele legate de proiectarea, implementarea, depanarea şi ajustarea configuraţiei algoritmilor bazaţi pe agenţi. AgSysLib este atât un framework, cât şi o bibliotecă şi dispune de o arhitectură bazată pe componente, care permite construirea algoritmilor într-o manieră modulară şi uşurează experimentarea şi analiza diverselor variante ale unui algoritm.
- **SwarmTSP** (http://swarmtsp.sourceforge.net/). SwarmTSP este o bibliotecă Java de algoritmi din clasa swarm intelligence pentru problema comis voiajorului (traveling salesman problem - TSP) şi pentru problema generalizată a comis voiajorului (generalized traveling salesman problem – GTSP). SwarmTSP implementează toţi algoritmii de tipul CGS (căutare ghidată de consultant) propuşi în această teză pentru TSP şi GTSP, precum şi o serie de algoritmi de optimizare cu colonii de furnici: Ant System, Ant Colony System, MAX-MIN Ant System, Elitist Ant System, Rank-Based Ant System şi Best-Worst Ant System.
- **SwarmQAP** (http://swarmqap.sourceforge.net/). SwarmQAP este o bibliotecă Java de algoritmi din clasa swarm intelligence pentru problema asignării pătratice (quadratic assignment problem - QAP). SwarmQAP implementează toţi algoritmii de tipul CGS (căutare ghidată de consultant) propuşi în această teză pentru QAP, precum şi algoritmul MAX-MIN Ant System.

1.3. Publicaţii ştiinţifice

Iordache, S. *Consultant-Guided Search - A New Metaheuristic for Combinatorial Optimization Problems*. In: GECCO 2010: Proceedings of the 12th Genetic and Evolutionary Computation Conference, Portland, Oregon, USA, ACM Press, 2010, pp. 225-232 [19] (**nominalizată pentru "best paper award"**).

Iordache, S. *Consultant-Guided Search Algorithms with Local Search for the Traveling Salesman Problem*. In: 11th International Conference Parallel Problem Solving from Nature - PPSN XI. LNCS 6239, Krakow, Poland, Springer, 2010, pp. 81-90 [17].

Iordache, S. *Consultant-Guided Search Algorithms for the Quadratic Assignment*. In: Hybrid Metaheuristics - 7th International Workshop, HM 2010. LNCS 6373, Vienna, Austria. Springer, 2010, pp. 148-159 [15].

Iordache, S., Moldoveanu, F. *AgSysLib - A Software Tool for Agent-Based Problem Solving*. In: Scientific Bulletin of "Politehnica" University of Bucharest, C Series (Electrical Engineering), vol. 73, issue 2, ISSN 1454-234x, 2011 [20].

Iordache, S. *A Framework for the Study of the Emergence of Rules in Multiagent Systems*, In: Katalinic, B. (Ed.), Proceedings of the 20th International DAAAM Symposium, Vienna, Austria, ISSN 1726-9679, 2009, pp, 1285-1286 [14].

Iordache, S., Pop, P.C. *An Efficient Algorithm for the Generalized Traveling Salesman Problem*. In: A. Quesada-Arencibia et al. (Eds.), Proceedings of the 13-th International Conference on Computer Aided Systems Theory (EUROCAST 2011), Las Palmas de Gran Canaria, Spain, ISBN: 978-84-693-9560-8, 2011, pp. 264-267 [21].

Iordache, S. *Consultant-Guided Search algorithms for the quadratic assignment problem*. In: GECCO 2010 companion: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation. ACM Press, 2010, pp. 2089-2090 [16].

Iordache, S. *Consultant-Guided Search combined with local search for the traveling salesman problem*. In: GECCO 2010 companion: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation. ACM Press, 2010, pp. 2087-2088 [18].

Pop, P.C., **Iordache, S**. *A Hybrid Heuristic Approach for Solving the Generalized Traveling Salesman Problem*. In: GECCO 2011: Proceedings of the Genetic and Evolutionary Computation Conference, Dublin, Ireland, ACM Press, 2011 (acceptată) [28].

1.4. Structura tezei

Capitolul 2 prezintă principalele concepte şi tehnici relevante pentru conţinutul acestei teze. După o scurtă introducere a sistemelor bazate pe agenţi, sunt prezentate diferite aspecte ale noţiunii de emergenţă şi sunt discutate diverse fenomene emergente care se manifestă în automatele celulare. Apoi, este descrisă optimizarea cu colonii de furnici, ca un exemplu de tehnică de rezolvare a problemelor care pune în valoare fenomenele emergente.

În capitolul 3 este dezvolat un formalism matematic pentru studiul emergenţei, care pune emergenţa într-un context orientat pe agenţi, în concordanţă cu cadrul impus de problema proiectării unui comportament emergent.

În capitolul 4 este propus un meta-framework pentru studiul emergenţei, numit MetFrEm, care poate fi utilizat pentru a descrie diverse framework-uri algoritmice bazate pe populaţii de

agenţi care interacţionează. Scopurile avute în vedere pentru proiectarea acestui meta-framework reflectă obiectivele şi ipotezele principale ale acestei teze de doctorat. După o descriere intuitivă a conceptelor şi a structurii lui MetFrEm, este oferită şi o descriere formală a acestui meta-framework. Apoi, cu ajutorul unor studii de caz, este ilustrat modul în care pot fi modelate diverse sisteme în MetFrEm.

Capitolul 5 introduce căutarea ghidată de consultant (Consultant-Guided Search – CGS), o nouă metaeuristică pentru probleme de optimizare combinatorială, inspirată de posibilitatea de a privi interacţiunile din MetFrEm din perspectiva unor clienţi care primesc recomandări de la consultanţi. Această metaeuristică este aplicată la câteva clase de probleme (problema comis voiajorului, problema asignării pătratice şi problema generalizată a comis voiajorului), iar rezultatele experimentale sunt comparate cu cele obţinute de cei mai performanţi algoritmi pentru aceste probleme.

În Anexa A este descris AgSysLib, un produs software pe care l-am dezvoltat pentru a oferi ajutor în rezolvarea problemelor folosind metode bazate pe agenţi. După identificarea problemelor întâmpinate pe parcursul proiectării, implementării, depanării şi ajustării configuraţiei unui nou algoritm bazat pe agenţi, este prezentat modul în care ele pot fi depăşite folosind acest produs software. AgSysLib este atât un framework, cât şi o bibliotecă, şi a jucat un rol important în dezvoltarea algoritmilor prezentaţi în capitolul 5.

2. FUNDAMENTE TEORETICE

2.1. Sisteme bazate pe agenţi

În esenţă, un sistem bazat pe agenţi reprezintă un sistem compus din agenţi care interacţionează. De aceea, pentru a putea defini un sistem bazat pe agenţi, este necesar să precizăm semnificaţia noţiunii de *agent*. La ora actuală, nu există însă o definiţie unanim acceptată a acestui termen. Sistemele bazate pe agenţi joacă un rol important în diverse domenii şi, de obicei, sensul acordat noţiunii de *agent* diferă de la un domeniu la altul şi chiar în cadrul aceluiaşi domeniu.

În majoritatea cazurilor, cercetarea legată de sistemele bazate pe agenţi poate fi încadrată în domeniul *inteligenţei artificiale*. Acesta este un domeniu de cerectare foarte vast şi de aceea este dificil de definit conceptul de *agent* într-un context atât de general. Cu toate acestea, Russel şi Norvig [30] oferă următoarea definiţie: "Un *agent* este orice lucru despre care se poate considera că îşi percepe mediul prin intermediul unor senzori şi acţionează asupra lui print intermediul unor efectori".

În contextul *modelării şi simulării bazate pe agenţi* (agent-based modeling and simulation – ABMS), Macal şi North [24] identifică următoarele caracteristici definitorii ale unui agent:

- un agent este autonom.
- un agent este modular.
- un agent este social, în sensul că interacţionează cu alţi agenţi.

Majoritatea sistemelor considerate în ABMS sunt sisteme adaptive complexe (complex adaptive systems – CAS) [25]. Acestea sunt sisteme care „îşi modifică şi îşi reorganizează părţile componente pentru a se adapta la problemele impuse de mediul înconjurător" [13]. În contextul CAS, "*agenţii* sunt unităţi semi-autonome care evoluează în timp în încearcarea de a-şi maximiza fitness-ul" [8].

Un număr mare de definiţii ale conceptului de agent au fost oferite în domeniul *sistemelor multi-agent* (multi-agent systems – MAS). Ele diferă, uneori în mod substanţial, prin caracteristicile pe care le asociază unui agent. În general, aceste definiţii impun capabilităţi mai ridicate decât în cazul ABMS şi, adesea, principala abstracţie folosită este cea de *agent inteligent*. Wooldridge [34] propune următoarea definiţie:

> Un *agent* este un sistem computaţional care este situat într-un mediu înconjurător şi care este capabil de acţiuni autonome în acest mediu, pentru a-şi putea atinge obiectivele.

În absenţa unei definiţii general acceptate, este necesar să specificăm ce înţelegem prin *agent* în contextul prezentei teze. Întrucât suntem interesaţi în principal de comportamentul complex care apare din interacţiunea agenţilor guvernaţi de reguli simple, avem nevoie de o definiţie care să impună un număr mic de restricţii agenţilor. În lucrarea de faţă, privim ca agenţi chiar şi enităţi foarte simple precum automatele celulare. Singura cerinţă pe care o avem este ca un agent să acţioneze în mod autonom, adică să decidă el însuşi ce acţiuni să execute, fără a primi comenzi de la entităţi externe. În capitolul 3 dezvoltăm un formalism pentru studiul emergenţei, care ne permite să definim riguros principalele concepte legate de sistemele bazate pe agenţi.

2.2. Emergenţa

Emergenţa constă în apariţia la nivelul sistemului a unor caracteristici care nu se regăsesc la nivelul elementelor care compun sistemul considerat. Exemplele includ apariţia vieţii din materie lipsită de viaţă sau apariţia conştiinţei ca urmare a interacţiunii unui număr mare de neuroni. Un alt exemplu îl constituie o colonie de furnici, unde urme de feromoni corespunzând drumurilor cele mai scurte între cuib şi sursele de hrană apar ca urmare a comportamentului colectiv al furnicilor.

Persoane diferite atribuie sensuri diferite acestui termen, dar în general ele pot fi grupate în două categorii: emergenţa puternică şi emergenţa slabă. Chalmers [4] consideră că un sistem manifestă *emergenţă puternică* dacă „fenomene de nivel înalt sunt produse de domeniul de nivel coborât, dar adevărurile referitoare la aceste fenomene nu sunt *deductibile* nici măcar în principiu din adevărurile referitoare la domeniul de nivel coborât”. Exemplul tipic de fenomen în jurul căruia se fac speculaţii legate de emergenţa puternică este conştiinţa. Existenţa emergenţei puternice este însă un subiect de discuţie în filozofie, iar majoritatea oamenilor de ştiinţă resping ideea acestui tip de emergenţă. Chalmers [4] consideră că un sistem manifestă *emergenţă slabă* dacă „fenomene de nivel înalt sunt produse de domeniul de nivel coborât, dar adevărurile referitoare la aceste fenomene sunt *neaşteptate*, date fiind principiile care guvernează domeniul de nivel coborât”.

În lucrarea de faţă suntem interesaţi să proiectăm sisteme bazate pe agenţi care produc un anumit comportament emergent dorit. În acest context, suntem preocupaţi în principal de emergenţa slabă, dar ataşăm acestui termen o semnificaţie mai largă, deoarece pentru scopurile noastre nu este relevant dacă percepem comportamentul manifestat ca neaşteptat sau nu. Interesul nostru se îndreaptă numai asupra sarcinii dificile de a găsi un set de reguli simple pentru agenţi, care să permită obţinerea unui comportament dorit. Propria noastră definiţie a emergenţei este oferită în capitolul 3, în care dezvoltăm un formalism matematic pentru studiul emergenţei în sistemele bazate pe agenţi.

2.3. Automate celulare

Fenomene emergente pot fi observate chiar şi în sisteme computaţionale extrem de simple, precum automatele celulare. Acestea sunt sisteme deterministe, discrete din punct de vedere spaţial şi temporal, caracterizate numai de interacţiuni locale. Ele sunt constituite dintr-o reţea regulată de celule, fiecare dintre acestea aflată într-o stare dintr-o mulţime finită de posibile stări. Fiecare celulă are aceeaşi regulă după care îşi actualizează starea, bazată numai pe starea unui anumit număr de celule învecinate. Modificarea stării are loc simultan în toate celulele, la fiecare cuantă de timp.

Există un număr infinit de automate celulare, ce pot fi obţinute combinând valorile diverşilor factori ce caracterizează un automat celular. Câţiva dintre aceşti factori sunt:

- **numărul de dimensiuni**: cel mai des studiate sunt automatele celulare uni- şi bidimensionale, care permit şi o vizualizare comodă.
- **numărul de stări**: cele mai simple automate celulare au numai două stări (notate de obicei 0 şi 1), care sunt adesea reprezentate grafic prin două culori diferite (de obicei alb şi negru).
- **vecinătatea**: numărul de vecini ai unei celule şi modul în care aceştia sunt identificaţi. Trebuie de asemenea precizat dacă vecinătatea unei celule o include şi pe ea însăşi.
- **regula de actualizare**: determină noua stare a celulei în funcţie de starea celulelor din vecinătatea ei. Este adesea reprezentată în formă tabulară.

- **tipul de univers celular**: în cazul în care universul este finit, trebuie precizat cum se aplică regulile în cazul celulelor aflate la margine. De multe ori se consideră un aranjament circular sau toroidal al celulelor, evitând astfel problema marginilor chiar şi în cazul unui univers finit. O altă soluție este să se considere că toate celulele din afara universului au o anumită stare dată.

Automatele celulare elementare reprezintă cel mai simplu tip de automate celulare. Ele au numai două stări (notate 0 şi 1), iar universul celular este infinit, adică linia pe care sunt dispuse celulele se întinde în ambele direcții la infinit. Vecinătatea unei celule este constituită din celula însăşi şi cele două celule alăturate. Deşi extrem de simple, automatele celulare elementare pot manifesta un comportament complex şi s-a demonstrat [5] că unul dintre cele 256 de automate elementare posibile este echivalent din punct de vedere computațional cu o maşină Turing universală.

Un exemplu de proiectare a comportamentului emergent folosind automate celulare este oferit de Mitchell *et al*. [26], care folosesc algoritmi genetici pentru a evolua o populaţie de automate celulare în scopul rezolvării unei anumite probleme date.

2.4. Optimizarea cu colonii de furnici

Fenomene emergente pot fi observate în multe sisteme naturale, iar oamenii de ştiinţă se inspiră adesea din natură pentru a proiecta algoritmi care pun în valoare comportamentul emergent manifestat în aceste sisteme. Un astfel de exemplu este oferit de optimizarea cu colonii de furnici (ant colony optimization – ACO), care are ca sursă de inspiraţie comportamentul furnicilor în căutarea hranei.

În drumul lor către o sursă de hrană şi înapoi spre cuib, furnicile depun pe sol substanțe chimice numite feromoni. Celelalte furnici pot percepe prezența acestor feromoni şi au tendința de a urma calea pe care concentrația acestora este mai mare. Deşi extrem de simplă, această strategie le permite furnicilor să transporte în mod efectiv hrana către cuib. Interacţiunile dintre furnici sunt mediate de feromoni. Acest mod de comunicare indirectă, în care acțiunea unui individ asupra mediului influențează acțiunile altor indivizi sau chiar ale lui însuşi, a primit numele de *stigmergie* [33]. Algorimii ACO sunt bazaţi pe observaţia că o urmă de feromoni corespunzând celui mai scurt drum între cuib şi sursa de hrană apare ca urmare a interacţiunii furnicilor.

Problema comis voiajorului a fost prima problemă rezolvată folosind această tehnică. Algoritmii ACO folosesc o populaţie de furnici virtuale, care imită într-o anumită măsură comportamentul furnicilor reale. Fiecare furnică virtuală construieşte la fiecare iteraţie o soluţie a problemei. Pentru a evita vizitarea repetată a unui oraş, fiecare furnică menţine o listă a nodurilor deja vizitate în iteraţia curentă. De obicei, la începutul unei noi iteraţii fiecare furnică este plasată într-un oraş ales în mod aleator. La fiecare pas din construcţia soluţiei, o furnică virtuală trebuie să decidă care va fi următorul oraş vizitat. Alegerea următorului nod de vizitat se face în mod probabilistic, în funcţie de concentraţia de feromoni pe calea către nodul considerat şi în funcţie de anumite informaţii euristice. Cei mai mulţi algoritmi folosesc drept informaţie euristică inversul distanţei până la nodul considerat. La sfârşitul fiecărei iteraţii, concentraţia de feromoni pe fiecare cale între două oraşe este actualizată. Regula de actualizare este bazată pe doi factori care afectează concentraţia feromionilor în cazul furnicilor reale: evaporarea feromonilor şi acumularea de feromoni pe căile traversate.

Primul algoritm ACO, denumit Ant System, a fost introdus de Marco Dorigo [9]. O serie de extensii ale lui au fost ulterior propuse, dintre care în momentul de faţă cele mai performante sunt: Ant Colony System [10] şi MAX-MIN Ant System [32].

3. O VIZIUNE DIFERITĂ ASUPRA EMERGENŢEI

Cercetarea prezentată în această teză este motivată de problema proiectării unui comportament emergent. Aceasta este o problemă dificilă, deoarece mecanismele pe care se bazează emergenţa nu sunt pe deplin înţelese, iar fenomenele emergente sunt caracterizate de aparenta lor impredicitibilitate. În acest capitol dezvoltăm un formalism matematic pentru studiul emergenţei, cu scopul de a studia principiile ce stau la baza comportamentului emergent. Formalismul nostru plasează emergenţa într-un context orientat pe agenţi, care este consistent cu cadrul impus de problema proiectării unui comportament emergent.

Există o mulţime de definiţii ale emergenţei, care cuprind formulări intuitive sau formale şi care diferă prin aspectele emergenţei pe care reuşesc să le captureze. Multe dintre definiţiile formale pornesc de la o definiţie preliminară intuitivă, pe care încearcă apoi să o formuleze într-un mod riguros. Acest proces vizează de obicei acele aspecte care disting comportamentul sau proprietăţile emergente de cele ne-emergente. În anumite cazuri, această distincţie poate fi uşor formalizată, dar nu există o procedură generală care permite să se decidă dacă un fenomen este emergent sau nu. Un exemplu este definiţia lui Darley [7], care consideră că "un fenomen emergent adevărat este unul pentru care mijlocul optim de predicţie este simularea". În alte cazuri, caracteristicile definitorii ale unui fenomen emergent sunt mai greu de exprimat formal, iar definiţiile formalizează de fapt o procedură pentru detectarea emergenţei. Aceasta este de exemplu abordarea propusă de Bonabeau and Dessalles [2]. În multe definiţii, emergenţa este detectată prin intermediul unui observator. Uneori, acest lucru duce la o descriere subiectivă care plasează emergenţa "în ochii observatorului". Exemplele includ testul Turing pentru emergenţă [3] sau folosirea unui observator care se bazează pe elementul surpriză [29]. Există însă şi încercări de a oferi o definiţie obiectivă a emergenţei. Un exemplu în acest sens este abordarea lui Crutchfield [6], care introduce *emergenţa intrinsecă*, noţiune caracterizată de o creştere a capacităţii computaţionale intrinsece. Dincolo de importanţa lor teoretică, definiţiile obiectibe sunt atrăgătoare deoarece oferă în principiu o modalitate automată de a detecta fenomenele emergente. Cu toate acestea, ele implică de obicei calcule extrem de complexe, ceea ce le face neadecvate pentru scopuri practice.

În lucrarea de faţă adoptăm un punct de vedere diferit asupra emergenţei, care ne permite să oferim o definiţie care este atât obiectivă, cât şi adecvată pentru scopuri practice. Ideea pe care o susţinem este că o definiţie a fenomenelor emergente trebuie să ia în considerare numai modul în care apar aceste fenomene şi nu trebuie să ia în discuţie proprietăţile acestora. În viziunea noastră proprietăţile fenomenelor emergente trebuie să fie subiectul unui întreg domeniu de cercetare şi nu parte a definiţiei emergenţei. Din această perspectivă, o definiţie trebuie exprimată numai în termenii proceselor care dau naştere fenomenelor emergente. Ca punct de plecare în dezvoltarea formalismului nostru propunem următoarea definiţie:

Definition 3.1 *Numim comportament emergent comportamentul manifestat de un sistem descentralizat bazat pe agenţi.*

Evident, definiţia de mai sus este obiectivă, pentru că nu depinde de modul în care un observator percepe comportamentul. Ea este de asemenea adecvată pentru scopuri practice, deoarece elimină necesitatea detectării emergenţei. O obiecţie care ar putea fi însă adusă este că această definiţie este prea largă, întrucât consideră chiar şi comportamente "neinteresante" ca fiind emergente. Pentru a ne apăra punctul de vedere, apelăm la un exemplu din teoria complexităţii, pornind de la problema comis voiajorului (traveling salesman problme - TSP), care, după cum se ştie, este NP-completă. O instanţă a TSP a cărei matrice a costurilor conţine

numai zerouri poate fi rezolvată fără dificultate şi nu prezintă niciun interes pentru cineva care studiază această clasă de probleme, dar cu toate acestea reprezintă o instanţă a acestei clase de probleme NP-complete. Un cercetător ar putea fi interesat de instanţe ne-elementare ale TSP, pentru care există algoritmi exacţi care pot găsi soluţii într-un interval de timp rezonabil. Un alt cercetător ar putea considera interesante acele instanţe ale TSP care nu pot fi rezolvate folosind algoritmi exacţi, dar pentru care anumiţi algoritmi euristici obţin rezultate foarte bune. Iar un alt cercetător ar putea considera că o instanţă a TSP este interesantă dacă niciun algoritm exact sau euristic nu este capabil să găsească o soluţie satisfăcătoare într-un interval de timp rezonabil. În opinia noastră, a defini emergenţa pe baza caracteristicilor comportamentului manifestat este ca şi când cineva ar include în definiţia clasei de probleme TSP condiţia ca instanţele să fie "interesante". Din acest motiv, considerăm că decizia de a exclude proprietăţile comportamentului din definiţia comportamentului emergent reprezintă un pas necesar către elaborarea unei teorii a emergenţei.

Unul dintre cele mai interesante aspecte ale emergenţei este că până şi reguli foarte simple sunt capabile să dea naştere unui comportament complex. Se pune însă întrebarea cum ar putea fi exprimată în mod formal noţiunea de regulă simplă. Utilizarea unor noţiuni de complexitate algoritmică precum complexitatea Kolmogorov [22] nu este practic posibilă, deoarece complexitatea Kolmogorov nu este calculabilă. De aceea, propunem o abordare pragmatică, bazată pe arbori sintactici, pentru a putea evalua complexitatea regulilor agenţilor. Presupunând că a fost stabilită o metodă de reprezentare a regulilor agenţilor ca arbori sintactici, măsurăm complexitatea regulilor unui agent ca numărul de frunze din arborele sintactic corespunzător. Evident că aceasta nu este o măsură ideală, întrucât depinde de limbajul folosit pentru a exprima regulile şi pentru că aceeaşi regulă poate fi exprimată în mai multe moduri. Cu toate acestea, ea reprezintă o măsură utilă pentru scopuri practice.

Suntem acum în măsură să oferim o definiţie formală a modelelor bazate pe agenţi. În paragrafele ce urmează, indicii superiori denotă agenţii, iar indicii inferiori cuantele de timp.

Definition 3.2 *Un model bazat pe agenţi este un sistem dinamic discret descris de un tuplu* $(\mathrm{S}, \mathrm{L}, \mathrm{A}_0)$*, unde:*

- S este spaţiul stărilor.
- L este limbajul utilizat pentru a descrie regulile agenţilor, împreună cu o metodă bine definită de reprezentare folosinf arbori sintactici.
- A_0 este mulţimea iniţială finită de agenţi. Un agent este un tuplu $(s_0, \sigma, f, \varphi)$, unde:
  - $s_0 \in S$ este starea iniţială.
  - $\sigma : 2^S \to 2^{\mathcal{A}}$ este funcţia de selecţie, care returnează mulţimea de agenţi cu care agentul considerat interacţionează. ($\mathcal{A}$ denotă mulţimea tuturor posibililor agenţi, adică mulţimea tuturor tuplurilor posibile de forma (s, σ, f, φ). Notaţia $2^{\mathcal{M}}$ denotă mulţimea tuturor submulţimilor lui $\mathcal{M}$.)
  - $f : 2^S \to S$ este regula de interacţiune, care calculează noua stare a unui agent pe baza stărilor agenţilor cu care acesta interacţionează. f este descrisă în limbajul L.
  - $\varphi : S \to 2^{\mathcal{A}}$ este regula de transformare care decide dacă un agent trebuie să moară şi/sau dacă alţi agenţi trebuie să fie creaţi. Valoarea returnată de φ este chiar agentul însuşi, dacă agentul continuă să existe şi nu sunt creaţi agenţi noi; este mulţimea vidă, dacă agentul moare şi nu sunt creaţi agenţi noi; este o mulţime finită de agenţi (care poate conţine şi agentul însuşi), dacă sunt creaţi noi agenţi. Funcţia φ este descrisă în limbajul L.

- Interacţiunea între agenţi este reflexivă, în sensul că dacă un agent a interacţionează cu un agent b la un anumit moment de timp t, atunci şi b interacţionează cu a la momentul t:

  $$\forall t \in \mathbb{N},\ \forall a, b \in A_t,\ \ b \in \sigma^a(s_t^a) \Leftrightarrow\ a \in \sigma^b(s_t^b)$$

- Starea unui agent evoluează în timp conform următoarei reguli:
  $s'_{t+1} = f(\{s_t\} \cup \{s_t^a \mid a \in \sigma(s_t)\})$
  În ecuaţia de mai sus, s'_{t+1} denotă starea preliminară a unui agent la momentul $t+1$, adică starea înainte de aplicarea regulii de transformare. Configuraţia definitivă a modelului bazat pe agenţi la momentul $t+1$ este:
  $A_{t+1} = \bigcup_{a \in A_t} \varphi^a(s'^a_{t+1})$

În continuare, oferim o definiţie a unui model descentralizat bazat pe agenţi, care reprezintă o formalizare a definiţiei **3.1**:

Definition 3.3 *Un model bazat pe agenţi este descentralizat dacă:*

$$\sigma^a(s_t^a) \cup \{a\} \subset A_t, \forall t \in \mathbb{N}, \forall a \in A_t.$$

În studiul emergenţei este intersant de analizat şi comparat comportamentul manifestat de o gamă variată de modele, de la cele centralizate până la cele puternic descentralizate. De aceea, este util să avem o măsură a gradului de descentralizare a unui model bazat pe agenţi. Să observăm mai întâi că putem exprima condiţia impusă de definiţia 3.3 şi sub forma:

$$|\sigma^a(s_t^a)| + 1 < |A_t|, \forall t \in \mathbb{N}, \forall a \in A_t.$$

Bazat pe această observaţie, oferim următoarea definiţie:

Definition 3.4 *Nivelul de centralizare al unui model bazat pe agenţi este o cantitate γ definită după cum urmează:*

$$\gamma = \max_{t \in \mathbb{N}} \max_{a \in A_t} \frac{|\sigma^a(s_t^a)| + 1}{|A_t|}$$

Folosind măsura de mai sus, putem defini un model bazat pe agenţi puternic descentralizat ca fiind un model cu un nivel de centralizare extrem de scăzut:

Definition 3.5 *Un model bazat pe agenţi este puternic descentralizat dacă* $\gamma \ll 1$.

Fenomenele emergente pot apărea şi în sisteme care sunt numai parţial descentralizate. Pentru acest concept oferim următoarea definiţie:

Definition 3.6 *Un model bazat pe agenţi este parţial descentralizat dacă:*

$$\exists\, T \subseteq \mathbb{N}, a.\hat{\imath}. |T| = \infty \text{ şi } \sigma^a(s_t^a) \cup \{a\} \subset A_t, \forall t \in T, \forall a \in A_t$$

Aşa cum am mai menţionat, în această lucrare suntem interesaţi în principal de fenomenele emergente manifestate în sisteme cu agenţi care urmează numai reguli simple. În consecinţă, avem nevoie de o măsură a complexităţii regulilor agenţilor.

Definition 3.7 *Dându-se o funcţie* g *descrisă în limbajul* L *care specifică o metodă bine definită de reprezentare folosind arbori sintactici, complexitatea regulii* g *relativă la limbajul* L, *notată* $\rho_L(g)$, *este dată de numărul de frunze al arborelui sintactic al lui* g *în reprezentarea asociată cu limbajul* L.

Folosind această masură, introducem următoarea definiţie a complexităţii regulilor unui model bazat pe agenţi:

Definition 3.8 *Complexitatea regulilor unui model bazat pe agenţi este o cantitate r dată de formula:*

$$r = \max_{t\in\mathbb{N}} \max_{a\in A_t}(\rho_L(f^a) + \rho_L(\varphi^a))$$

Există o clasă largă de modele bazate pe agenţi pentru care mulţimea de agenţi nu se modifică în decursul timpului: agenţii nu mor şi nu se creează noi agenţi. În astfel de modele, regula de transformare a fiecărui agent este funcţia de identitate, care are o complexitate a regulii egală cu 1. Deoarece mulţimea de agenţi nu se modifică în timp, complexitatea regulilor unui astfel de model este determinată de mulţimea iniţială a agenţilor:

$$r = \max_{a\in A_0}(\rho_L(f^a) + 1)$$

Definition 3.9 *Dinamica unui model descentralizat bazat pe agenţi, exprimată în termenii proprietăţilor de nivel înalt, poartă numele de comportament emergent pur.*

Definition 3.10 *Dinamica unui model parţial descentralizat bazat pe agenţi, exprimată în termenii proprietăţilor de nivel înalt, poartă numele de comportament parţial emergent*.

Formalismul introdus în acest capitol reprezintă un prim pas către o *Teorie a Comportamentului Emergent* (Emergent Behavior Theory – EBT). El este în concordanţă cu punctul nostru de vedere conform căruia fenomenele emergente trebuie definite numai în termenii modelelor de calcul capabile să le producă, în timp ce caracteristicile acestor fenomene trebuie să reprezinte obiectul de studiu al unui întreg domeniu din cadrul EBT. Din această perspectivă, EBT trebuie să identifice şi să analizeze diferite clase de comportament emergent, cum ar fi:

- comportament emergent caracterizat de o creştere a complexităţii;
- comportament emergent caracterizat de apariţia unor tipare;
- comportament emergent haotic;
- comportament emergent caracterizat de atractori;
- comportament emergent pentru care cel mai rapid mod de predicţie este simularea.

Unul dintre motivele principale pentru care am dezvoltat acest formalism matematic a fost posibilitatea de a demonstra într-un mod riguros aserţiuni referitoare la proprietăţile emergenţei. Metoda pe care o propunem în această teză este una empirică: astfel de aserţiuni trebuie mai întâi identificate efectuând experimente computaţionale, iar apoi ele pot fi exprimate, analizate şi demonstrate folosind formalismul nostru matematic, pentru a obţine noi rezultate teoretice. Această metodă presupune existenţa unui framework experimental pentru studiul emergenţei, care trebuie să fie compatibil cu formalismul nostru matematic. În următorul capitol prezentăm un meta-framework pe care l-am proiectat pentru a întruni aceste cerinţe.

4. METFREM – UN META-FRAMEWORK PENTRU STUDIUL EMERGENŢEI

Pentru a studia în mod riguros fenomenele emergente, avem nevoie de un cadru de lucru care să ofere o modalitate comună de a modela sistemele în care se manifestă acest fenomen. În acest scop, am conceput un **Met**a-**Fr**amework pentru studiul **Em**ergenţei, numit *MetFrEm*, care poate fi folosit pentru a descrie diverse modele algoritmice care implică o populaţie de agenţi. MetFrEm este un meta-framework, deoarece este în general folosit pentru a modela algoritmi abstracţi, de nivel înalt, cum ar fi metaeuristicile, care sunt la rândul lor framework-uri ce permit descrierea unor algoritmi specifici.

Modelând diferite specificaţii algoritmice de nivel înalt în acelaşi meta-framework, avem posibilitatea de a face o comparaţie riguroasă a metodelor considerate. MetFrEm oferă un set de concepte şi de reguli care trebuie folosite pentru a modela sistemele dorite şi impune o structură algoritmică generală pentru aceste sisteme. Obiectivele principale ale MetFrEm sunt:

- să permită modelarea oricărui framework algoritmic bazat pe agenţi.
- să favorizeze modelarea sistemelor puternic descentralizate.
- să favorizeze modelarea sistemelor cu agenţi ghidaţi numai de reguli simple.
- să faciliteze modelarea framework-urilor algoritmice cu agenţi eterogeni.
- să ofere o metodă unitară de modelare a comunicaţiei directe şi a celei bazate pe stigmergie.

4.1. Conceptele şi structura MetFrEm

Univers
Denumim *univers* un model bazat pe agenţi reprezentat în MetFrEm. Acesta este un model de discret în timp, care nu are un mediu înconjurător. Elementele de mediu pot fi reprezentate ca agenţi obişnuiţi, numiţi *mecanisme* în terminologia MetFrEm. Un metamodel în MetFrEm conţine un singur univers, care la rândul lui conţine toate mecanismele metamodelului.

Mecanism
Folosim termenul *mecanism* pentru a desemna un agent în MetFrEm. O caracteristică importantă a mecanismelor din MetFrEm este că ele ştiu cum să interacţioneze cu alte mecanisme, fără a avea nevoie să ştie ce tipuri de mecanisme există în univers.

Propritetate
Starea internă a unui mecanism este caracterizată de valorile *proprietăţilor sale interne*. Mulţimea proprietăţilor care caracterizează un mecanism este o submulţime a unei *mulţimi finite globale de proprietăţi* definite în universul considerat. Fiecare proprietate are o valoare numerică variabilă în timp. În general, un mecanism nu îşi expune complet starea în faţa celorlalte mecanisme. În plus, în contexte diferite, un mecanism expune diferite proprietăţi, folosind vederea (view) corespunzătoare contextului respectiv.

Vedere (View)
A *vedere (view)* reprezintă un grup de proprietăţi pe care un mecanism le expune altor mecanisme. Acest grup de proprietăţi poate fi o submulţime a proprietăţilor interne sau poate include proprietăţie ale căror valori combină valorile anumitor proprietăţi interne. Proprietăţile expuse de o vedere trebuie să reprezinte o submulţime a mulţimii globale de

proprietăţi din univers. Fiecare mecanism oferă trei vederi: *vedeera observabilă, vederea activă* şi *vederea reactivă*. Fiecăreia dintre cele trei vederi în corespunde o *funcţie de vedere,* care este folosită pentru a calcula valorile proprietăţilor expuse. Acestea sunt respectiv: *funcţia de vedere observabilă* υ_O, *funcţia de vedere activă* υ_A şi *funcţia de vedere reactivă* υ_R. Numim *stare observabilă* a unui mecanism mulţimea de valori returnate de funcţia de vedere observabilă υ_O.

Vecinătate
La fiecare moment de timp, un mecanism poate iniţia o interacţiune cu o submulţime de mecanisme. Această submulţime este selectată dintr-o familie de potenţiale submulţimi, care reprezintă *vecinătatea* mecanismului la momentul respectiv de timp. Fiecare mecanism are asociată o *funcţie de vecinătate* η, care returnează vecinătatea unui mecanism la un moment de timp dat, pe baza *stărilor observabile* ale celorlalte mecanisme din univers.

Funcţie de evaluare
Pentru a alege o submulţime de mecanisme cu care să iniţieze o interacţiune la un moment de timp dat, un mecanism trebuie să evalueze fiecare submulţime din vecinătatea sa. Această operaţie este efectuată cu ajutorul unei *funcţii de evaluare* ψ, care returnează o valoare numerică indicând cât de adecvată este submulţimea respectivă. Gradul de adecvare al unei submulţimi este calculat pe baza valorilor proprietăţilor expuse de *vederea observabilă* a fiecărui mecanism din submulţime.

Funcţie de selecţie
Pe baza valorilor gradelor de adecvare returnate de funcţia de evaluare, un mecanism selectează cu ajutorul *funcţiei de selecţie* σ setul de mecanisme cu care va iniţia o interacţiune.

Funcţie de interacţiune
În MetFrEm, unul din mecanismele implicate într-o interacţiune joacă un rol activ şi este numit *iniţiatorul* acţiunii. Celelalte mecanisme, care sunt determinate de funcţia de selecţie σ, reprezintă *ţinta* interacţiunii şi joacă un rol pasiv. În timpul interacţiunii, iniţiatorul îşi expune starea în faţa celorlalte mecanisme prin intermediul *vederii active*, iar un mecanism ţintă îşi expune starea în faţa iniţiatorului şi a celorlalte mecanisme ţintă prin intermediul *vederii reactive*. În general, starea unui mecanism se modifică în urma interacţiunii. Această modificare este reflectată de modificarea valorilor proprietăţilor interne ale mecanismului. Noua stare internă este calculată cu ajutorul unei *funcţii de interacţiune*. Fiecare mecanism are asociate două funcţii de interacţiune: o *funcţie de interacţiune activă* f_A şi o *funcţie de interacţiune reactivă* f_R. În funcţie de rolul jucat de mecanism în interacţiune, se foloseşte una sau cealaltă dintre aceste două funcţii: mecanismul iniţiator îşi calculează noua stare cu ajutorul *funcţiei de interacţiune active*, iar mecanismele ţintă cu ajutorul *funcţiei de interacţiune reactive.*

Funcţie de transformare
Ca urmare a interacţiunii, un mecanism poate continua să existe, poate să dispară sau poate să creeze noi mecanisme. Aceste operaţii sunt efectuate de o *funcţie de transformare* φ, care înlocuieşte mecanismul dat cu o mulţime de alte mecanisme. Dacă mecanismul continuă să existe şi nu se creează alte mecanisme, această mulţime are ca unic element mecanismul însuşi. Dacă mecanismul dispare şi nu se creează noi mecanisme, funcţia de transformare returnează mulţimea vidă. Dacă se creează noi mecanisme, funcţia de transformare returnează o mulţime finită de mecanisme, care conţine şi mecanismul însuşi, în cazul în care acesta continuă să existe.

4.2. O descriere formală a MetFrEm

În această secţiune, oferim o descriere formală a conceptelor prezentate în paragrafele de mai sus. Începem prin a introduce *mulţimea globală de proprietăţi* din univers:

$\mathcal{P} = \{p_1, p_2, p_3, \dots\ p_z\}$

unde $z \in \mathbb{N}$ este numărul de proprietăţi globale.

Fiecare element p_i din mulţimea $\mathcal{P}$ identifică o proprietate măsurabilă. De exemplu, un univers care modelează particule microscopice poate include proprietăţi precum: masa, poziţia şi viteza. În MetFrEm, valoarea unei proprietăţi este o cantitate adimensională exprimată printr-un număr real.

Notăm cu $\mathcal{M}$ mulţimea infinită a tuturor mecanismelor posibile. Un *univers* în MetFrEm este un tuplu $(\mathcal{P}, M_0)$, unde $\mathcal{P}$ este mulţimea globală de proprietăţi, iar $M_0 \subset \mathcal{M}$ este mulţimea iniţială finită de mecanisme.

Un *mechanism* este definit ca un tuplu $(P_I, P_O, P_A, P_R, s_0, \upsilon_o, \upsilon_A, \upsilon_R, \eta, \psi, \sigma, \varphi, f_A, f_R)$, unde:

- $P_I \subseteq \mathcal{P}$ este mulţimea proprietăţilor interne*;*
- $P_O \subseteq \mathcal{P}$ este mulţimea proprietăţilor observabile.*;*
- $P_A \subseteq \mathcal{P}$ este mulţimea proprietăţilor active;
- $P_R \subseteq \mathcal{P}$ este mulţimea proprietăţilor reactive;
- $s_0 \in \mathbb{R}^{|P_I|}$ este starea internă iniţială, adică mulţimea valorilor iniţiale ale proprietătilor interne;
- $\upsilon_o : \mathbb{R}^{|P_I|} \to \mathbb{R}^{|P_O|}$ este funcţia de vedere observabilă, care calculează mulţimea valorilor ce constituie starea observabilă a mecanismului;
- $\upsilon_A : \mathbb{R}^{|P_I|} \to \mathbb{R}^{|P_A|}$ este funcţia de vedere activă, care calculează mulţimea valorilor expuse în timpul unei interacţiuni initiate de acest mecanism;
- $\upsilon_R : \mathbb{R}^{|P_I|} \to \mathbb{R}^{|P_R|}$ este funcţia de vedere reactivă, care calculează mulţimea valorilor expuse în timpul unei interacţiuni în care acest mecanism joacă rolul de ţintă;
- $\eta : 2^{\mathbb{R}} \to 2^{2^{\mathcal{M}}}$ este funcţia de vecinătate, care returnează o familie de submulţimi de mecanisme cu care mecanismul dat poate iniţia o interacţiune. Valoarea de retur este calculată pe baza stărilor observabile ale tuturor mecanismelor din univers.
- $\psi : 2^{\mathbb{R}} \to \mathbb{R}$ este funcţia de vedere evaluare, care calculează gradul de adecvare al unei submulţimi de mecanisme din familia de submulţimi potenţiale, pe baza stărilor observabile ale acestora.
- $\sigma : 2^{\mathbb{R}} \to 2^{\mathcal{M}}$ este funcţia de evaluare, care alege o submulţime de mecanisme cu care acest mecanism va iniţia o interacţiune, pe baza gradului de adecvare al fiecărei potenţiale submulţimi.
- $\varphi : \mathcal{M} \to 2^{\mathcal{M}}$ este funcţia de transformare, care returnează o mulţime de mecanisme ce reflectă transformarea suferită de acest mecanism la sfârşitul unei interacţiuni: el poate continua să existe, poate să dispară şi/sau poate crea noi mecanisme.
- $f_A : 2^{\mathbb{R}} \to 2^{\mathbb{R}}$ este funcţia de interacţiune folosită de mecanismul iniţiator pentru a-şi calcula noua stare internă, pe baza stării sale curente şi pe baza valorilor proprietăţilor reactive ale mecanismelor ţintă.
- $f_R : 2^{\mathbb{R}} \to 2^{\mathbb{R}}$ este funcţia de interacţiune folosită de un mecanism ţintă pentru a-şi calcula noua stare internă, pe baza stării sale curente, a valorilor proprietăţilor active ale mecanismului iniţiator şi a valorilor proprietăţilor reactive ale celorlalte mecanisme ţintă.

În următoarele paragrafe, vom arăta cum dinamica universului rezultă din aplicarea funcţiilor asociate mecanismelor. Vecinătatea unui mecanism μ, adică mulţimea potenţialelor submulţimi de mecanisme cu care acesta poate iniţia o interacţiune la momentul t, es te calculată astfel:

$$\mathcal{N}_t^{\mu} = \eta^{\mu}(\bigcup_{m \in M_t} \upsilon_O(s_t^m)) \tag{4.1}$$

unde M_t este mulţimea tuturor mecanismelor care există la momentul t, iar s_t^m reprezintă starea internă a mecanismului m la momentul t.

Mulţimea valorilor gradelor de adecvare corespunzătoare fiecărei potenţiale submulţimi din vecinătate este dată de:

$$B_t^{\mu} = \bigcup_{N \in \mathcal{N}_t} \psi^{\mu}(\bigcup_{m \in N} \upsilon_O^{\mu}(s_t^m)) \tag{4.2}$$

Submulţimea de mecanisme cu care un mecanism μ iniţiază efectiv interacţiunea este:

$$V_t^{\mu} = \sigma^{\mu}(B_t^{\mu}) \tag{4.3}$$

În MetFrEm, un mecanism poate interacţiona cu alte mecanisme în mod activ sau reactiv. Mulţimea tuturor mecanismelor cu care un mecanism interacţionează trebuie deci să includă mecanismele implicate în ambele tipuri de interacţiune. Pentru un mecanism dat μ, această multime este:

$$W_t^{\mu} = V_t^{\mu} \cup \{m \in M_t \mid \mu \in V_t^m\} \tag{4.4}$$

În MetFrEm, este posibil ca la un anumit moment t, un mecanism să fie implicat în mai multe interacţiuni: el poate juca rolul de iniţiator într-una dintre acestea şi rolul de ţintă în celelalte. Noua stare în urma unei interacţiuni în care mecanismul joacă rolul de iniţiator este calculată folosind funcţia de interacţiune activă f_A. Noua stare în urma unei interacţiuni în care mecanismul joacă rolul de ţintă este calculată folosind funcţia de interacţiune reactivă f_R. Starea finală a mecanismului este obţinută prin aplicarea succesivă a acestor funcţii pentru fiecare interacţiune a mecanismului considerat. Dacă notăm cu μ mecanismul iniţiator, starea sa preliminară în urma interacţiunii, adică starea înaintea aplicării regulii de transformare, este dată de:

$$s'^{\mu}_{t+1} = f_A^{\mu}(\{s_t^{\mu}\} \cup \{\upsilon_R^m(s_t^m) \mid m \in V_t^{\mu}\}) \tag{4.5}$$

Starea preliminară a unui mecanism ţintă λ în urma interacţiunii este:

$$s'^{\lambda}_{t+1} = f_R^{\lambda}(\{s_t^{\lambda}\} \cup \{\upsilon_A^{\mu}(s_t^{\mu})\} \cup \{\upsilon_R^m(s_t^m) \mid m \in V_t^{\mu}, m \neq \lambda\}) \tag{4.6}$$

Configuraţia definitivă a universului la momentul $t + 1$ este:

$$M_{t+1} = \bigcup_{m \in M_t} \varphi^m(s'^m_{t+1}) \tag{4.7}$$

4.3. Structura algoritmică

Evoluţia unui univers în MetFrEm este prezentată în pseudocod în figura de mai jos:

```
1  procedure MetFrEm ()
2      M ← initializeUniverse ()
3      while (termination condition not met) do
4          executePreliminaryActions (M)      // optional
5          foreach initiator ∈ M do
6              neighborhood ← getNeighborhood (initiator)
7              suitabilities ← evaluateNeighborhood(neighborhood)
8              targets ← selectTargets (neighborhood, suitabilities)
9              initiator.state ← activeInteraction (initiator, targets)
10             replacement ← transform(initiator)
11             M ← (M \ {initiator)} ∪ replacement
12             foreach target ∈ targets do
13                 target.state ← reactiveInteraction (target, initiator, targets)
14                 replacement ← transform(target)
15                 M ← (M \ {target)} ∪ replacement
16             end foreach
17         end foreach
18         executeFinalActions (M)          // optional
19     end while
20 end procedure
```

Figure 4-1. Descrierea în pseudocod a evoluţiei unui univers în MetFrEm.

În faza de iniţializare (linia 2), se creează mulţimea iniţială de mecanisme şi se configurează starea fiecărui mecanism. Algoritmul intră apoi în bucla principală (linile 3-19). Pentru a permite modelarea sistemelor care nu se încadrează perfect în structura formală specificată de MetFrEm, algoritmul oferă 2 proceduri opţionale: *executePreliminaryActions* (linia 4) şi *executeFinalActions* (linia 18). De obicei ele implementează operaţii globale care nu pot fi modelate ca acţiuni descentralizate. De exemplu, în anumiţi algoritmi de optimizare cu colonii de furnici, numai cea mai performantă furnică are dreptul să depună feromoni. Această operaţie necesită informaţii globale despre soluţiile construite de toate furnicile existente şi poate fi deci implementată de una dintre procedurile opţionale.

În timp ce acţiunile efectuate de procedurile opţionale nu sunt specificate explicit, funcţionarea celorlalte proceduri din pseudocodul de mai sus este complet determinată de funcţiile şi vederile aferente introduse în descrierea formală a MetFrEm. Procedura *getNeighborhood* (linia 6) foloseşte funcţia de vecinătate η pentru a determina mulţimea potenţialelor submulţimi de mecanisme cu care iniţiatorul poate interacţiona, în concordanţă cu formula 4.1. Procedura *evaluateNeighborhood* (linia 7) foloseşte funcţia de evaluare ψ pentru a calcula valoarea gradului de adecvare pentru fiecare mulţime de mecanisme din vecinătate, în concordanţă cu formula 4.2. Procedura *selectTargets* (linia 8) foloseşte funcţia de selecţie σ pentru a alege mulţimea de mecanisme ţintă, în concordanţă cu formula 4.3. Procedura *activeInteraction* (linia 9) foloseşte funcţia de interacţiune activă f_A pentru a calcula starea iniţiatorului în urma interacţiunii, în concordanţă cu formula 4.5. Procedura *reactiveInteraction* (linia 13) foloseşte funcţia de interacţiune reactivă f_R pentru a calcula starea unui mecanism ţintă în urma interacţiunii, în concordanţă cu formula 4.6. Procedura *transform* (linile 10 şi 14) foloseşte funcţia de transformare φ pentru a determina mulţimea de mecanisme care înlocuieşte un mecanism dat, în concordanţă cu formula 4.7.

5. CĂUTAREA GHIDATĂ DE CONSULTANT

Meta-framework-ul introdus în capitolul anterior a fost proiectat pentru a permite modelarea sistemelor puternic eterogene, cu agenţi care ştiu cum să interacţioneze cu cu orice tip de agent, fără a avea nevoie să ştie ce tipuri de agenţi există în sistem. Din acest motiv, o interacţiune în MetFrEm este realizată în doi paşi. Mai întâi, iniţiatorul alege o submulţime de mecanisme ţintă dintr-o famile de potenţiale submulţimi, iar apoi interacţionează cu ţintele alese. Să considerăm pentru simplicitate cazul în care mulţimea de mecanisme ţintă conţine un singur element, adică iniţiatorul interacţionează cu un singur mecanism. Ca urmare a interacţiunii, iniţiatorul îşi modifică starea. Noua stare este calculată pe baza valorilor expuse de vederea reactivă a mecanismului ţintă. Întrucât vederea reactivă oferă şi valori care nu sunt în general disponibile în vederea observabilă, aceste valori suplimentare pot fi interpretate ca informaţii private oferite de mecanismul ţintă. Iniţiatorul primeşte aceste informaţii numai datorită faptului că a ales să interacţioneze cu acest mecanism ţintă. Putem privi mecanismul ţintă ca pe un consultant care are cunoştinte de specialitate, pe care le pune la dispoziţia mecanismelor care sunt dispuse să interacţioneze cu el. În mod similar, iniţiatorul poate fi privit ca un client care alege unul dintre consultanţii disponibili, pentru a primi de la acesta informaţii preţioase. Văzută din această perspectivă, interacţiunea mecanismelor în MetFrEm ne-a condus la ideea unei metode euristice, pe care am denumit-o Căutarea Ghidată de Consultant (Consultant-Guided Search – CGS), şi care constituie subiectul acestui capitol.

5.1. Metaeuristica CGS

Căutarea Ghidată de Consultant (CGS) este o metaeuristică bazată pentru probleme de optimizare combinatorială. Ea este bazată pe populaţii şi se inspiră din modul în care oamenii iau decizii pe baza recomandărilor primite de la consultanţi. Un individ dintr-o populaţie a CGS este o persoană virtuală, care poate juca simultan rolul de client şi de consultant. În postura de client, o persoană virtuală construieşte la fiecare iteraţie o soluţie a problemei. În postura de consultant, o persoană virtuală oferă recomandări clienţilor, în concordanţă cu ***strategia*** sa. În general, la fiecare pas din construcţia soluţiei există mai multe variante din care un client poate alege. Varianta recomandată de consultant are o probabilitate mai mare de a fi aleasă, dar clientul poate opta şi pentru una dintre celelalte variante, pe care o va alege pe baza unei anumite euristici.

La începutul fiecărei iteraţii, un client îşi alege un consultant pe baza ***preferinţei sale personale*** şi pe baza ***reputaţiei*** consultantului. Reputaţia unui consultant creşte cu numărul de succese repurtate de clienţii săi. Spunem că un client a repurtat un ***succes***, dacă a construit o soluţie mai bună decât toate soluţiile găsite până în acel moment de vreun client ghidat de acelaşi consultant. De fiecare dată când un client de-al său repurtează un succes, consultantul îşi ajustează strategia pentru a reflecta secvenţa de decizii luate de client. Întrucât reputaţia unui consultant scade cu trecerea timpului, este important ca în mod constant clienţii săi să obţină succese. În cazul în care reputaţia unui consultant scade sub o anumită valoare limită, acesta intră într-o ***perioadă sabatică***, pe parcursul căreia nu îşi va mai oferi serviciile de consultant, ci va căuta o nouă strategie pe care să o aplice în viitor.

Pseudocodul care formalizează metaeuristica CGS este prezentat în Fig. 5.1. În faza de iniţializare (liniile 2-5), se creează persoanele virtuale, care sunt apoi plasate în modul sabatic. În funcţie de modul său, o persoană virtuală construieşte la fiecare iteraţie fie o soluţie a

problemei (linia 13), fie o strategie (linia 9). O procedură de optimizare locală (linia 17) poate fi apoi aplicată pentru a îmbunătăţi această soluţie sau strategie.

După faza de construcţie, o persoană aflată în perioada sabatică verifică dacă a găsit o strategie mai bună decât cele anterioare (liniile 20-22), iar o persoană în stare normală verifică dacă a obţinut un succes, în care caz consultantul acesteia îşi va ajusta în mod corespunzător strategia (liniile 24-29).

```
 1 procedure CGSMetaheuristic()
 2     create the set 𝒫 of virtual persons
 3     foreach p ∈ 𝒫 do
 4         setSabbaticalMode(p)
 5     end foreach
 6     while (termination condition not met) do
 7         foreach p ∈ 𝒫 do
 8             if actionMode[p] = sabbatical then
 9                 currStrategy[p] ← constructStrategy(p)
10             else
11                 currCons[p] ← chooseConsultant(p)
12                 if currCons[p] ≠ null then
13                     currSol[p] ← constructSolution(p, currCons[p])
14                 end if
15             end if
16         end foreach
17         applyLocalOptimization()                    // optional
18         foreach p ∈ 𝒫 do
19             if actionMode[p] = sabbatical then
20                 if currStrategy[p] better than bestStrategy[p] then
21                     bestStrategy[p] ← currStrategy[p]
22                 end if
23             else
24                 c ← currCons[p]
25                 if c ≠ null and currSol[p] is better than all solutions
26                                                 found by a client of c since last sabbatical then
27                     successCount[c] ← successCount[c] + 1
28                     strategy[c] ← adjustStrategy(c, currSol[p])
29                 end if
30             end if
31         end foreach
32         updateReputations()
33         updateActionModes()
34     end while
35 end procedure
```

Fig. 5.1. Metaeuristica CGS.

Reputaţiile sunt ajustate pe baza ***rezultatelor*** obţinute de clienţi (line 32): reputaţia unui consultant este incrementată ori de câte ori unul dintre clienţii săi obţine un succes şi, pe deasupra, primeşte un bonus în cazul în care unul dintre clienţi găseşte o soluţie mai bună decât toate cele descoperite până în prezent. Pentru fiecare consultant, algoritmul memorează cel mai bun rezultat obţinut de unul dintre clienţii săi şi menţine un clasament al consultanţilor, bazat pe aceste rezultate. Pentru un anumit număr de consultanţi aflaţi în fruntea acestui clasament, CGS fixează o limită sub care reputaţia acestora nu poate să scadă.

În final, se actualizează modul de acţiune al fiecărei persoane (linia 33): consultanţii a căror reputaţie a scăzut sub limita minimă intră în perioada sabatică, iar consultanţii a căror perioadă sabatică a luat sfârşit intră în starea normală.

5.2. Metaeuristica CGS aplicată la Problema Comis Voiajorului

În această secţiune propunem un algoritm CGS pentru problema comis voiajorului (traveling salesman problem – TSP), pe care îl denumim CGS-TSP. Pentru a aplica CGS la o anumită clasă de probleme, trebuie definite diversele concepte folosite de această metaeuristică (strategie, rezultat, preferinţă personală etc.) în contextul respectivei clase de probleme. Apoi, trebuie decis cum vor fi implementate acţiunile pentru care metaeuristica nu oferă o sepcificaţie (*constructStrategy*, *constructSolution*, *chooseConsultant* etc.).

Construirea unei soluţii pentru TSP înseamnă realizarea unui circuit închis care conţine fiecare nod al grafului o singură dată. Pentru a evita vizitarea repetată a unui nod, fiecare persoană virtuală din CGS-TSP menţine o listă a nodurilor deja vizitate în interaţia curentă. ***Strategia*** unui consultant este reprezentată de un tur pe care acesta îl recomandă clienţilor săi. ***Rezultatul*** unui tur este calculat ca inversul lungimii sale. Deoarece atât construcţia soluţiei cât şi a strategiei implică realizarea unui tur, acelaşi tip de decizie trebuie luat de o persoană virtuală în ambele cazuri: ea trebuie să aleagă următorul oraş pe care îl vizitează. Totuşi, regulile folosite în luarea acestei decizii diferă în cele două cazuri.

Pentru a restrânge numărul de variante disponibile la fiecare pas al construcţiei, CGS-TSP foloseşte *liste candidat* care conţin pentru fiecare oraş *i* cele mai apropiate *cand* oraşe, unde *cand* este un parametru. În acest fel, vecinătatea efectivă a unei persoane *k* aflate în oraşul *i* reprezintă mulţimea oraşelor din lista candidat a lui *i* pe care persoana *k* nu le-a vizitat încă.

În timpul ***perioadei sabatice***, consultanţii construiesc strategii folosind o euristică bazată numai pe distanţele dintre oraşul curent şi potenţialele oraşe succesor: cu o probabilitate a_0 (unde a_0 este un parametru constant), o persoană virtuală se deplasează la cel mai apropiat oraş din vecinătatea sa efectivă, iar cu o probabilitate $(1 - a_0)$ ea alege pseudo-aleator unul dintre oraşele învecinate, ţinând cont de distanţa până la oraşul *i*.

La fiecare pas din construcţia soluţiei, un client primeşte de la consultantul său o recomandare privind următorul oraş de vizitat. Această recomandare este bazată pe turul ce constituie strategia consultantului. Fie *i* oraşul vizitat de clientul *k* la un anumit pas al construcţiei din iteraţia curentă. Pentru a decide ce oraş să recomande pentru următorul pas, consultantul localizează poziţia pe care oraşul *i* apare în turul promovat de el şi identifică oraşul care îl precede şi cel care îi urmează lui *i* în acest tur. Dacă niciunul dintre aceste două oraşe nu a fost încă vizitat de client, consultantul îl recomandă pe cel mai apropiat de oraşul *i*. Dacă numai unul dintre cele două oraşe este nevizitat, acesta va fi recomandat. Dacă ambele oraşe au fost deja vizitate, consultantul nu poate face o recomandare pentru următorul pas.

Clientul nu urmează întotdeauna recomandarea consultantului. O regulă pseudo-aleatoare bazată pe doi parametri constanţi q_0 şi b_0 este folosită pentru a alege oraşul vizitat la următorul pas: dacă s-a primit o recomandare de la consultant, aceasta va fi urmată cu probabilitatea q_0; cu o probabilitate $b_0(1 - q_0)$ clientul se deplasează către cel mai apropiat oraş din vecinătatea sa efectivă; cu o probabilitate $(1 - b_0)(1 - q_0)$ clientul alege pseudo-aleator unul dintre oraşele învecinate, influenţat de distanţa până la oraşul *i*.

Un client ţine cont de doi factori în alegerea unui consultant: ***reputaţia*** consultantului şi ***preferinţa personală*** a clientului. În CGS-TSP preferinţa personală este dată de rezultatul turului promovat de consultant, adică de inversul lungimii acestuia.

De fiecare dată când un client repurtează un succes (adică găseşte un tur mai scurt decât cel promovat de consultantul său), consultantul îşi actualizează strategia, înlocuindu-şi turul promovat cu cel construit de client.

Experimente cu şi fără căutare locală arată că CGS-TSP este capabil să depăşească în performanţă cei mai buni algoritmi de optimizare cu colonii de furnici.

5.3. Metaeuristica CGS aplicată la problema asignării pătratice

În această secţiune analizăm modul în care CGS poate fi aplicată la problema asignării pătratice (Quadratic Assignment Problem - QAP) şi propunem algoritmul CGS-QAP, care combină CGS cu o procedură de căutare locală. Dându-se o mulţime de fabrici şi o mulţime de locaţii, o matrice a fluxurilor între fiecare pereche de fabrici, precum şi o matrice a distanţelor între fiecare pereche de locaţii, problema asignării pătratice constă în găsirea unei repartizări a fabricilor la locaţiile exstente, astfel încât suma produselor dintre fluxuri şi distanţele corespunzătoare să fie minimă.

În CGS-QAP, ***strategia*** este implementată ca o soluţie promovată de consultant. Ea este reprezentată printr-o repartiţie a fabricilor la locaţii, care este construită în timpul ***perioadei sabatice***. În algoritmul propus, perioada sabatică durează o singură iteraţie. Pentru a construi o nouă strategie, un consultant generează în mod aleator o repartiţie pe care apoi o îmbunătăţeşte folosind o procedură de căutare locală. ***Preferinţa personală*** pentru un anumit consultant este determinată de costul repartiţiei promovate de acesta. Împreună cu ***reputaţia***, este folosită pentru a calcula probabilitatea de a alege un anumit consultant.

La fiecare pas al construcţiei, un client plasează o fabrică nealocată în una dintre locaţiile libere. În CGS-QAP, ordinea în care fabricile sunt repartizate este aleatoare. La fiecare pas, un client primeşte de la consultantul său o recomandare privind locaţia pe care să o aleagă. Locaţia recomandată este cea care corespunde fabricii considerate în repartiţia promovată de consultant. Pentru a decide dacă va urma recomandarea, clientul foloseşte o regulă pseudo-aleatoare: cu o probabilitate q_0, clientul plasează fabrica dată în locaţia recomandată de consultant, iar cu o probabilitate $(1 - q_0)$ plasează fabrica în mod aleator în una din locaţiile libere. Valoarea parametrului q_0 este critică pentru performanţa algoritmului. O valoare mare a lui q_0 duce la o căutare agresivă, concentrată în jurul repartiţiei promovate de consultant. O valoare mică a lui q_0 favorizează explorarea spaţiului de căutare, permiţând algoritmului să scape din regiunile optime locale.

Oro de câte ori un client repurtează un succes (adică găseşte o repartiţie mai bună decât cea promovată de consultantul său), consultantul îşi actualizează strategia, înlocuind repartiţia promovată cu cea construită de client.

La sfârşitul fiecărei iteraţii, se aplică o procedură de căutare locală, pentru a îmbunătăţi repartiţiile construite de clienţi. În mod asemănător cu alţi algoritmi pentru QAP, CGS-QAP poate folosi ca procedură de căutare locală metoda *2-opt* sau scurte execuţii ale algoritmilor *tabu search* sau *simulated annealing*.

Rezultatele experimentale arată că algoritmul nostru este mai performant decât MAX-MIN Ant System (MMAS) pentru instanţe nestructurate ale QAP. Una din temele de cercetare viitoare pe ne-o propunem este să investigăm dacă CGS-QAP este capabil să concureze cu MMAS şi pentru instanţe structurate ale QAP.

5.4. Metaeuristica CGS aplicată la problema generalizată a comis voiajorului

Problema generalizată a comis voiajorului (generalized traveling salesman problem – GTSP) este o problemă NP-completă care extinde problema clasică a comis voiajorului, prin introducerea unei partiţionări în clustere a nodurilor grafului. Problema constă în găsirea celui mai scurt circuit închis care vizitează un singur nod din fiecare cluster. Algoritmul pe care îl propunem combină metoda CGS cu o abordare local-globală pentru rezolvarea GTSP. Cele mai multe instanţe GTSP de interes practic sunt probleme simetrice cu distanţe euclidiene, unde clusterele sunt compuse din noduri apropiate în spaţiu unele de celelalte. Algoritmul nostru este proiectat pentru a beneficia de structura acestor instanţe.

Abordarea local-globală a fost introdusă de Pop [27] în cazul problemei generalizate a arborelui de acoperire minim (generalized minimum spanning tree problem). În cazul GTSP, abordarea local-globală face distincţie între *conexiunile globale* (conexiuni între clustere) şi conexiunile locale (conexiuni între noduri din clustere diferite). Fiind dată o secvenţă în care sunt vizitate clusterele (adică un circuit Hamiltonian global), există o serie de circuite Hamiltoniene generalizate corespunzătoare acesteia. Cel mai bun circuit Hamiltonian generalizat (din punctul de vedere al costului) poate fi determinat fie folosind o reţea pe niveluri, fie folosind metode de programare liniară cu numere întregi. În paragrafele ce urmează desemnăm prin *graf global* graful obţinut prin înlocuirea tuturor nodurilor unui cluster printr-un supernod care să reprezinte clusterul.

În algoritmul pe care îl propunem, un client construieşte la fiecare iteraţie un tur global, adică un ciclu Hamiltonian în graful global. Strategia unui consultant este reprezentată de asemenea printr-un tur, pe care consultantul îl recomandă clienţilor. Algoritmul aplică o procedură de căutare locală pentru a îmbunătăţi tururile globale ce reprezintă fie soluţia globală a unui client, fie strategia unui consultant aflat în perioada sabatică. Apoi, folosind o procedură de optimizare a clusterelor, algoritmul găseşte cel mai bun tur generalizat corespunzând turului global returnat de procedura de căutare locală.

Pentru a compara strategiile construite în perioada sabatică, un consultant ţine cont de costul turului generalizat corespunzător fiecărei strategii. În mod similar, succesul unui client este evaluat pe baza costului soluţiei generalizate construite de acesta.

Euristica folosită în perioada sabatică pentru a construi a nouă strategie este bazată pe distanţele virtuale dintre supernodurile din graful global. Calculăm distanţa virtuală dintre două supernoduri ca distanţa dintre centrele de masă ale celor două clustere corespunzătoare. Alegerea acestei euristici este justificată de clasa de probleme pentru care este proiectat algoritmul nostru: instanţe simetrice cu distanţe euclidiene, cu clustere compuse din noduri apropiate spaţial unele de celelalte. Datorită introducerii distanţelor virtuale dintre clustere, avem posibilitatea să utilizăm liste candidat pentru a restrânge numărul de variante disponibile la fiecare pas al construcţiei.

Rezultatele experimentale arată ca algoritmul propus în această secţiune poate concura cu cei mai performanţi algoritmi pentru GTSP.

ANEXA A. AGSYSLIB – UN PACHET SOFTWARE PENTRU IMPLEMENTAREA SOLUŢIILOR BAZATE PE AGENŢI

Multe fenomene complexe ce apar în sisteme fizice şi biologice pot fi descrise în mod natural folosind modele bazate pe agenţi. Aceste modele pot captura comportamentul complex care rezultă din interacţiunea unor agenţi ghidaţi de reguli simple. Este tentantă ideea de a folosi abordări bazate pe agenţi nu numai pentru a descrie fenomene existente, ci şi pentru a rezolva probleme complexe care nu pot fi atacate prin metode convenţionale. De aceea, în ultimii ani s-a înregistrat o creştere continuă a interesului pentru proiecatrea algoritmilor care pun în valoare comportamentul emergent manifestat în sistemele cu agenţi care interacţionează.

Diversele biblioteci software disponibile la ora actuală pentru modelarea şi simularea bazată pe agenţi (agent-based modeling and simulation - ABMS [24]) se pot dovedi utile şi în conceperea unui nou algoritm bazat pe agenţi. Cu toate acestea, software-ul ABMS nu poate oferi asistenţă în toate aspectele legate de proiectarea, implementarea şi ajustarea algoritmilor bazaţi pe agenţi. În acest capitol, identificăm dificultăţile întâmpinate pe parcursul acestor etape şi arătăm cu pot fi ele depăşite folosind AgSysLib, un pachet software pe care l-am dezvoltat pentru a uşura activităţile legate de implementarea soluţiilor bazate pe agenţi.

Principala problemă în proiectarea algoritmilor bazaţi pe agenţi este găsirea unui set de reguli care să producă un comportament emergent dorit. Deoarece nu există o metodologie unanim acceptată, proiectarea acestor algoritmi este adesea un proces bazat pe încercări, care ar putea beneficia de pe urma ajutorului oferit de un pachet software.

În timp ce ABMS permite în principal o înţelegere calitativă a fenomenelor, în cazul implementării soluţiilor bazate pe agenţi accentul cade pe obţinerea unor rezultate cât mai performante. Transformarea unei simulări cu caracter demonstrativ într-o implementare la cel mai înalt nivel al tehnicii este un proces anevoios, pentru care ABMS nu oferă în general sprijin. AgSysLib umple acest gol şi vine în ajutorul celor care doresc să implementeze soluţii bazate pe agenţi pentru probleme complexe.

AgSysLib este în acelaşi timp un framework şi o bibliotecă Java. El oferă o API care trebuie implementată de fiecare algoritm bazat pe agenţi şi pune la dispoziţie o serie de clase care uşurează efectuarea diverselor activităţi legate de rezolvarea problemelor folosind metode bazate pe agenţi. Pentru multe dintre interfeţele specificate de API, AgSysLib oferă implementări implicite sau clase abstracte care pot fi uşor instanţiate, extinse şi personalizate.

AgSysLib are o arhitectură bazată pe componente, care permite construirea modulară a algoritmilor şi facilitează experimentarea şi analiza diverselor variante ale unui algoritm. O nouă variantă a unui algoritm poate fi obţinută prin simpla înlocuire a unei anumite componente din implementarea de bază cu o altă componentă. AgSysLib permite şi procesarea în loturi, pentru a putea aplica în mod repetat un algoritm la aceeaşi problemă, la probleme diferite, sau folosind valori diferite ale parametrilor.

Aşa cum am mai menţionat, unele dintre funcţiile puse la dispoziţie de pachetele software ABMS sunt utile şi în soluţionarea problemelor prin metode bazate pe agenţi. AgSysLib nu îşi propune să ofere o nouă implementare a acestor funcţii, ci are în vedere acele aspecte legate de proiectarea, implementarea şi ajustarea algoritmilor bazaţi pe agenţi, care nu sunt acoperite de pachetele ABMS. Pentru a oferi însă în continuare posibilitatea de a beneficia de facilităţile disponibile în produsele ABMS, AgSysLib poate acţiona ca un container pentru aceste

biblioteci. În acest fel, AgSysLib poate transforma un pachet ABMS într-o unealtă software capabilă să asiste în soluţionarea problemelor prin metode bazate pe agenţi. Există în prezent un număr mare de pachete ABMS, cu capabilităţi şi scopuri diferite. AgSysLib poate integra multe dintre aceste pachete, dar în configuraţia standard el înglobează biblioteca MASON. MASON [23] este un mediu de simulare cu evenimente discrete implementat în Java, care permite execuţia rapidă şi de un număr mare de ori a modelelor cu un număr mare de agenţi.

Configurarea

Soluţionarea problemelor prin metode bazate pe agenţi necesită realizarea unor diverse tipuri de experimente. Unul dintre acestea implică evaluarea diverselor variante ale unui algoritm. Adesea, trecerea la o nouă variantă a unui algoritm se face comentând porţiuni din codul original şi înlocuindu-le cu noi porţiuni de cod, sau folosind flag-uri în diverse părţi ale programului pentru a activa porţiunile de cod corespunzătoare variantei de algoritm dorite. O astfel de practică face codul sursă greu de citit şi de întreţinut. O arhitectură bazată pe componente permite o separare clară a porţiunilor de cod specifice fiecărei variante a unui algoritm, dar necesită în continuare modificări în codul original pentru a preciza componentele ce trebuie utilizate. AgSysLib permite specificarea componentelor unui sistem bazat pe agenţi într-un fişier de configurare. În acest mod, nu mai sunt necesare modificări ale codului sau flag-uri suplimentare pentru a comuta între diversele variante ale unui algoritm.

Un alt tip de experimente presupune compararea rezultatelor obţinute de un algoritm pentru diverse valori ale parametrilor săi sau chiar pentru diverse formule utilizate în cadrul lui. Dacă valorile parametrilor sau formulele algoritmului fac parte din codul programului, se ajunge din nou la un cod greu de întreţinut. De aceea, AgSysLib oferă un număr mare de funcţii utilitare pentru citirea diverselor tipuri de valori, precum şi a formulelor matematice din fişiere de configurare. În plus, AgSysLib permite declararea de liste de valori ale parametrilor şi execuţia loturilor corespunzătoare fiecărei combinaţii de valori ale acestora. Este de asemenea posibil să se specifice că valoarea unui parametru din fişierul de configurare trebuie calculată pe baza unei formule care foloseşte valorile altor parametri.

Componente de urmărire a evoluţiei (listeners)

Pe parcursul experimentelor cu un algoritm bazat pe agenţi, este adesea necesar să se inspecteze evoluţia diverselor mărimi prelucrate de aplicaţie sau a unor valori agregate ale acestora, cu scopul de a înţelege comportamentul emergent la sistemului. Informaţii similare sunt necesare pe parcursul activităţilor de depanare şi de ajustare a configuraţiei. Instrucţiunile necesare pentru a afişa aceste informaţii "murdăresc" codul sursă. În plus, ele afectează performanţa algoritmului, deşi în general ele nu mai sunt necesare în versiunea finală a aplicaţiei. Pentru a putea ţine aceste instrucţiuni separate de codul sursă principal, interfaţa de programare (API) a AgSysLib introduce componente de urmărire a evoluţiei (listeners) şi permite în acelaşi timp activarea şi dezactivarea rapidă a acestor porţiuni de cod. O componentă de urmărire a evoluţiei oferă metode activate de următoarele evenimente:

- începerea procesării unui lot
- terminarea procesării unui lot
- începerea procesării unui sistem dintr-un lot
- terminarea procesării unui sistem dintr-un lot
- începerea unui pas din evoluţia unui sistem
- terminarea unui pas din evoluţia unui sistem
- terminarea operaţiilor efectuate de un agent la un pas din evoluţia sistemului.

Componentele de urmărire a evoluţiei active pe parcursul unei execuţii pot fi specificate într-un fişier de configurare. În acest fel, nu sunt necesare modificări în codul aplicaţiei pentru a adăuga sau a şterge o astfel de componentă.

Experimentarea şi ajustarea configuraţiei (tuning)

Aşa cum am mai menţionat, AgSysLib permite specificarea unor liste de valori în fişierele de configurare, pentru a putea executa un algoritm folosind toate combinaţiile posibile ale valorilor parametrilor acestuia. În acest mod se poate realiza o formă elementară de ajustare a configuraţiei (tuning). AgSysLib oferă însă şi o metodă de tuning mai elaborata, bazată pe algoritmi genetici şi permite de asemenea o integrare comodă cu pachetele software de configurare automată a parametrilor.

AgSysLib oferă o serie de clase ce permit evaluarea formulelor matematice care apar într-un fişier de configurare. Pentru evaluarea formulelor nu se foloseşte un interpretor, întrucât acesta ar avea un impact negativ asupra performanţei. În schimb, AgSysLib generează on-the-fly o clasă Java pentru fiecare formulă şi creează o instanţă a acestei clase. Clasa generată dinamic conţine o metodă care acceptă ca argumente variabilele prezente în formula dată şi returnează valoarea corespunzătoare evaluării acesteia. Generarea on-the-fly a clasei are loc o singură dată, la pornirea programului. În acest fel, evaluarea unei formule are loc la fel de rapid ca în cazul în care formula ar fi înscrisă direct în codul algoritmului.

Depanarea

Aplicaţiile bazate pe agenţi sunt în general foarte adaptive, de aceea se pretează la rezolvarea unor probleme reale complexe, în medii dinamice. Pe de altă parte însă, depistarea neregulilor în astfel de aplicaţii este mai dificilă, pentru că în multe cazuri chiar şi o implementare cu erori este capabilă să rezolve problema, chiar dacă nu atât de eficient pe cât ar fi posibil. În plus, pentru o euristică nou concepută nu se ştie în avans cât de eficientă ar putea fi o implementare a acesteia, de aceea bug-urile care afectează numai performanţa algoritmului pot ramâne foarte uşor neobservate. Deoarece mulţi algoritmi bazaţi pe agenţi sunt stohastici, reproducerea unui comportament neobişnuit se poate dovedi de asemenea dificilă.

Investigarea problemelor legate de aplicaţiile bazate pe agenţi presupune adesea o analiză detaliată a dinamicii stării interne a programului. AgSysLib oferă o componentă GUI care permite conectarea prin RMI (Remote Method Invocation) la o aplicaţie în curs de execuţie şi interogarea, urmărirea sau modificarea stării interne a acesteia. Componenta GUI este denumită teleconsolă de control. Deoarece ea poate stabili o conexiune nu doar la pornire, ci în orice moment al execuţiei unui program, teleconsola de control poate fi folosită şi pentru a investiga un comportament neaşteptat apărut pe parcursul execuţiei unui program care nu se afla de fapt în faza de depanare.

Accesul la starea internă a unui program este oferit prin intermediul unori script-uri în limbajul de programare Groovy. Astfel, se pot face interogări complexe ale stării interne a unui program sau se pot face modificări multiple ale acesteia, cum ar fi alterarea într-un anumit mod a stării fiecărui agent, folosind numai câteva linii de cod. Script-urile pot fi şi înregistrate, pentru a fi executate la fiecare pas. În plus, rezultatele interogării stării interne pot fi folosite pentru a configura puncte de întrerupere condiţionale foarte complexe.

BIBLIOGRAFIE

[1] Philip Ball, "For sustainable architecture, think bug," *New Scientist*, no. 2748, pp. 35-37, February 2010.

[2] Eric Bonabeau and Jean-Louis Dessalles, "Detection and emergence," *Intellectica*, vol. 2, no. 25, pp. 85--94, 1997.

[3] Fabio Boschetti and Randall Gray, "A Turing Test for Emergence," in *Advances in Applied Self-organizing Systems*, Lakhmi Jain, Xindong Wu, and Mikhail Prokopenko, Eds.: Springer London, 2008, pp. 349-364.

[4] David J Chalmers, "Strong and weak emergence," in *The Re-Emergence of Emergence*.: Oxford University Press, 2006, pp. 244-254.

[5] M. Cook, "Universality in elementary cellular automata," in *Complex Systems, Vol. 15, Issue 1*., 2004, pp. 1-40.

[6] James P Crutchfield, "The calculi of emergence: computation, dynamics and induction," *Physica D*, vol. 75, no. 1-3, pp. 11--54, August 1994.

[7] Vince Darley, "Emergent phenomena and complexity," *Artificial Life*, vol. 4, pp. 411--416, 1994.

[8] Kevin Dooley, "Complex adaptive systems: A nominal definition," *The Chaos Network*, no. 8, pp. 2-3, 1996.

[9] M Dorigo, A Colorni, and V Maniezzo, "Positive feedback as a search strategy," Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, Technical Report 91-016, 1991.

[10] Marco Dorigo and Luca Maria Gambardella, "Ant Colony System: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53-66, 1997.

[11] M. Dorigo and T. Stützle, *Ant Colony Optimization*.: The MIT Press, 2004.

[12] Gregory Gutin and Daniel Karapetyan, "A memetic algorithm for the generalized traveling salesman problem," *Natural Computing*, vol. 9, no. 1, pp. 47--60, March 2010.

[13] John H Holland, "Complex adaptive systems," in *A new era in computation*. Cambridge, MA, USA: MIT Press, 1993, pp. 17--30.

[14] Serban Iordache, "A Framework for the Study of the Emergence of Rules in Multiagent Systems," in *Proceedings of the 20th International DAAAM Symposium*, Vienna, Austria, 2009, pp. 1285-1286.

[15] Serban Iordache, "Consultant-Guided Search Algorithms for the Quadratic Assignment Problem," in *Hybrid Metaheuristics - 7th International Workshop, HM 2010. LNCS*, vol.

6373, Vienna, Austria, 2010, pp. 148-159.

[16] Serban Iordache, "Consultant-guided search algorithms for the quadratic assignment problem," in *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, ortland, Oregon, USA, 2010, pp. 2089-2090.

[17] Serban Iordache, "Consultant-Guided Search Algorithms with Local Search for the Traveling Salesman Problem," in *11th International Conference Parallel Problem Solving from Nature - PPSN XI. LNCS*, vol. 6239, Krakow, Poland, 2010, pp. 81-90.

[18] Serban Iordache, "Consultant-guided search combined with local search for the traveling salesman problem," in *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, Portland, Oregon, USA, 2010, pp. 2087-2088.

[19] Serban Iordache, "Consultant-guided search: a new metaheuristic for combinatorial optimization problems," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation, GECCO 2010. ACM Press*, Portland, Oregon, USA, 2010, pp. 225--232.

[20] Serban Iordache and Florica Moldoveanu, "AgSysLib - A Software Tool for Agent-Based Problem Solving," *Scientific Bulletin of "Politehnica" University of Bucharest, C Series (Electrical Engineering)*, vol. 73, no. 2, 2011.

[21] Serban Iordache and Petrica C. Pop, "An Efficient Algorithm for the Generalized Traveling Salesman Problem," in *Proceedings of the 13-th International Conference on Computer Aided Systems Theory (EUROCAST 2011)*, Las Palmas de Gran Canaria, Spain, 2011, pp. 264-266.

[22] Ming Li and Paul Vitányi, *An introduction to Kolmogorov complexity and its applications*, 2nd ed.: Springer-Verlag New York, Inc., 1997.

[23] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel C Balan, "MASON: A Multiagent Simulation Environment," *Simulation*, vol. 81, no. 7, pp. 517-527, 2005.

[24] C M Macal and M J North, "Agent-based Modeling and Simulation," in *Winter Simulation Conference*, Austin, TX, USA, 2009, pp. 86 -98.

[25] John H Miller and Scott E Page, *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*.: Princeton University Press, 2007.

[26] M. Mitchell, J. P. Crutchfield, and R. Das, "Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work," in *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96)*, Moscow, Russia, 1996.

[27] Petrica C. Pop, "The Generalized Minimum Spanning Tree Problem," The Netherlands, PhD thesis 2002.

[28] Petrica C. Pop and Serban Iordache, "A Hybrid Heuristic Approach for Solving the Generalized Traveling Salesman Problem," in *GECCO 2011: Proceedings of the Genetic and Evolutionary Computation Conference*, Dublin, Ireland, 2011.

[29] Edmund M Ronald, Moshe Sipper, and Mathieu S Capcarrère, "Design, Observation, Surprise! A Test of Emergence," *Artificial Life*, vol. 5, pp. 225-239, 1999.

[30] S.J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[31] Thomas Stützle and Marco Dorigo, "ACO algorithms for the quadratic assignment problem," *New ideas in optimization*, pp. 33-50, 1999.

[32] Thomas Stützle and Holger H Hoos, "MAX-MIN Ant system," *Future Gener. Comput. Syst.*, vol. 16, pp. 889--914, 2000.

[33] G Theraulaz and E Bonabeau, "A brief history of stigmergy," *ARTIFICIAL LIFE*, vol. 5, no. 2, pp. 97-116, 1999.

[34] Michael Wooldridge, *An Introduction to MultiAgent Systems*.: John Wiley & Sons, 2002.