Ant colony optimization (ACO)

BABEŞ-BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

March 21, 2022

Agenda

- Aspecte teoretice
- Algoritm
- Exemplu
- Proprietati
- Aplicatii

State-of-the-art

- Furnicile au un comportament social ce influenteaza in mod direct cautarea unui drum intre cuib si sursa de hrana
- Acest comportament se remarca prin:
  - o activitate/munca colectiva aproape perfect coordonata indiferent de dimensiunea populatiei (cateva sau milioane de furnici)
  - diviziunea muncii
  - interactiune sociala complexa
- Colorni si Dorigo (1991) propun un algoritm inspirat din acest comportament pentru rezolvarea probemelor de optimizari ca si alternativa a algoritmilor genetici (AG)

State-of-the-art

- ACO face parte din familia de algoritmi ce implica o cautare **cooperativa**, ghidata de calitatea **relativa** a indivizilor.
- ACO presupune de asemenea si cautare **constructiva** - se porneste de la o solutie initiala, si se adauga elemente in solutie la fiecare iteratie.

Elemente speciale

- ACO are nevoie de transformarea problemei de optimizare intr-o problema de identificare a drumului optim intr-un graf.
- **De ce ?**
  - Furnicile vor construi solutia plimbandu-se prin graf si depunand pe muchii
- **Cum ?**
  - Colonii de furnici care cauta solutia optima prin intermediul cooperarii
  - Diferente cu **AG**: avem populatii de cromozomi care sunt in competitie.
  - **cooperare** vs **competitie**

Elemente speciale

- **Ce face o furnica ?**
  - Se deplaseaza in spatiul de cautare, lasand in urma sa o cantitate de feromon
  - Retine drumul parcurs (memorie proprie)
  - Alege drumul pe care sa-l urmeze in functie de:
    - feromonul existent pe drum
    - euristica asociata acelui drum
  - Foloseste feromonul ca si mecanism de comunicare si cooperare astfel incat:
    - acesta are o intensitate corelata cu calitatea solutiei
    - se evapora odata cu trecerea timpului

Furnici naturale

Ne putem imagina cum:

- O colonie de furnici pleaca in cautarea hranei
- In drumul lor apare in obstacol ce implica aparitia a doua rute, **A** respectiv **B**.
- Conform imaginii de mai jos ruta A este mai scurta, astfel ca furnicile de pe acel drum il vor parcurge de mai multe ori (ture), deci vor lasa mai multe feromon.
- Concentratia de fernomon de pe ruta A va creste mai accelerat decat pe ruba B, astfel ca dupa cateva parcurgeri furnicile de pe ruta B vor alege ruta A
- Dupa un anumit numar de parcurgeri datorita concentratiei de feromon care creste pe ruta A si descreste pe ruta B, toate furnicile vor folosi ruta A.

Furnici artificiale

Asemanarile furnicilor artificiale cu cele reale:

- navigheaza de la o sursa catre o destinatie
- descopera drumul cel mai scurt pe baza concentratiei de feromon, tinand cont ca:
  - poate executa miscari aleatoare
  - depoziteaza feromon pe drumul parcurs
  - poate detecta drumul urmat de o anumita furnica (sefa), incliand sa-l urmeze
  - persistand feromon pe un drum, automat creste probablitatea ca acel drum sa fie urmat si de alte furnici

Furnici artificiale

Imbunatatiri ale furnicilor artificiale fata de cele reale:

- au memorie (retin actiunile efectuate sub forma unui istoric)
- nu sunt complet oarbe, pentru ca pot aprecia calitatea spatiului vecin
- depun feromon si in functie de calitatea solutiei identificate

Feromon

In reprezentarea artificiala feromonul are rolul unei memorii **colective dinamice distribuita** intre furnici, unde sunt inmagazinate cele mai recente experiente de cautare a solutiei de catre furnici.
Putem spune ca furnicile **pot comunica indirect si se pot influenta reciproc prin acest mecanism** in vederea obtinerii solutiei dorite.

Descriere generala

1. Initializare (cu valori random)
2. Cat timp nu s-a parcurs numarul necesar de pasi pentru identificarea solutiei, fiecare furnica:
   - executa o mutare
   - modifica local urma de fermonom corespunzator ultimului element adaugat in solutie
3. Se modifica urma de feromon de pe drumurile parcurse, fie de toate furnicile, fie de cea mai buna furnica
4. Se returneaza solutia gasita de cea mai buna furnica

Versiuni ACO

Exista 3 versiuni principale in functie de:

- Regulile de deplasare a furnicilor
- Momentul la care furnicile depun feromon (**pe parcursul constructiei solutiei** vs **la sfarsitul crearii unei solutii**)
- Cine depune feromonul: **toate furnicile** vs **cea mai buna furnica**

Versiuni ACO

1. **AS** - Ant system
   - **Toate** furnicile depun feromon **dupa** construirea unei solutii **complete** (modificare globala colectiva)
2. **MMAS** - MaxMin Ant System
   - doar **cea mai buna** furnica depune feromon **dupa** construirea unei solutii **complete** (modificare globala a liderului)
   - feromonul este **limitat** la un interval dat
3. **ACO** - Ant Colony System
   - **toate** furnicile depun feromon **la fiecare pas** in constructia solutiei (modificare locala colectiva)
   - doar **cea mai buna** furnica depune feromon dupa construirea unei solutii complete (modificarea globala a liderului)

Problema comisului voiajor

Sa se gaseasca un drum care sa treaca prin n orase, astfel incat costul sa fie minim si fiecare oras sa fie vizitat o singura data.

Structura de date: graf unde nodurile reprezinta orasele, iar drumurile dintre orase sunt encodate ca si muchii intre nodurile corespondente.

Initializare

- t := 0 (timpul)
- fiecare muchie (i, j) este initializata cu:

$$\gamma_{ij}^{(t)} = c$$

(intensitatea urmei de feromon pe muchia (i, j) la momentul t)

$$\Delta\gamma_{ij} = 0$$

(cantitatea de feromon lasata pe muchia (i, j) de catre toate furnicile)

- pornim cu *m* furnici plasate aleaturi pe cele *n* noduri (m $\leq$ n)
- fiecare furnica isi modifica memoria (lista de noduri vizitate), adaugand in lista nodul din care pleaca in cautare.

Parcurgere

Pentru un numar de pasi **n**, fiecare furnica **k** din colonie:

- isi mareste solutia partiala cu un element (executa o mutare), alegand un oras (i) fiind pozitionata pe orasul i astfel:

$$\mathrm{j} = \begin{cases} argmax_{l \in permis_k}\{[\gamma_{il}]^{\alpha}[\eta_{il}]^{\beta}\}, q \leq \mathrm{q}_0 \\ J, altfel \end{cases}$$

- unde:
  - q - numar aleator uniform distribuit in [0, 1]s
  - q_0 - parametru, $0 \leq q_0 \leq 1$ ($q_0 = 0$ pentru AS/MMAS, altfel pentru ACO)
  - J este un oras selectat cu probabilitatea:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\gamma_{ij}^{(t)}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{s-permis_k(t)}[\gamma_{is}^{(t)}]^{\alpha}[\eta_{is}]^{\beta}}, j - permis \\ 0, altfel \end{cases}$$

Parcurgere

Pentru un numar de pasi **n**, fiecare furnica **k** din colonie:

- J este un oras selectat cu probabilitatea:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\gamma_{ij}^{(t)}]^{\alpha}[\eta_{ij}^{(t)}]^{\beta}}{\sum_{s-permis_k(t)}[\gamma_{ij}^{(t)}]^{\alpha}[\eta_{ij}^{(t)}]^{\beta}}, j - permis \\ 0, altfel \end{cases}$$

- unde:
  - $p_{ij}^{k}(t)$ - probabilitatea de tranizitie a furnicii k situatia in orasul i spre orasul j
  - $\eta_{ij} = \frac{1}{d_{ij}}$ - vizilitatea din orasul i spre orasul j (distanta)
  - $permis_k$- orasele pe care le mai poate vizita a k furnica la momentul t
  - α - controleaza importanta urmei/feromonului (cate furnici au mai trecut prin muchia respectiva)
  - β - controleaza importanta vizibilitatii (cat de aproape se afla urmatorul oras

Persistarea feromonului

Dupa ce o furnica k isi mareste solutia partiala cu un element (executa o mutare), se modifica local urma de feromon lasata de fiecare furnica pe ultimul element adaugat in solutie.

$$\gamma_{ij}^{(t+1)} = (1 - \varphi)\gamma_{ij}^{(t)} + \varphi * \gamma_0$$

unde:

- φ - coeficient de degradare a feromonului cu valori intre [0, 1]. $\varphi = 0 \rightarrow$ AS/MMAS, altfel ACO
- γ_0 - valoarea initiala a feromonului
- (i, j) - ultima muchie parcursa de furnica

Persistarea feromonului

Feromonul se modifica:

- pe drumurile parcurse de toate furnicile (**AS**)
- pe cel mai bun drum (**ACO)**
- pe cel mai bun drum pacurs de cea mai buna furnica (**MMAS**)

Persistarea feromonului - AS

(**AS**) - pe drumurile parcurse de toate furnicile, pentru fiecare muchie:

- se calculeaza canitatea unitara de feromon lasat de a k furnica pe muchia ij:
  $$\Delta\gamma_{ij}^{k} = \begin{cases} \frac{Q}{L_k}, \text{daca a k furnica a folosit muchia (i, j)} \\ 0, altfel \end{cases}$$
- unde
  - Q - canitatea de feromon lasata de o furnica
  - L_k - lungimea (costului) turul efectual de a k-a furnica
- canitatea totala de feromon de pe muchia (ij) se calculeaza: $\Delta\gamma_{ij} = \sum_{k=1}^{m} \Delta\gamma_{ij}^{k}$
- intensitatea urmei de feromoni se calculeaza ca suma intre evaporarea feromonilor vechi si feromonul nou lasat:

$$\gamma_{ij}^{(t+n)} = (1-\varphi) * \gamma_{ij}^{(t)} + \Delta\gamma_{ij}$$

Persistarea feromonului - AS

(**ACO-MMAS**) - cel mai bun drum:

- se calculeaza cantitatea unitara de feromon lasat de cea mai buna furnica pe muchia ij:
  $\Delta\gamma_{ij} = \frac{1}{L_{best}}$
- unde
  - L_{best} - lungimea (costum) celui mai bun drum:
    - din iteratia curenta
    - din toate iteratiile executate pana atunci
- cantitatea totala de feromon de pe muchia (ij) se calculeaza:
  $\Delta\gamma_{ij} = \sum_{k=1}^{m} \Delta\gamma_{ij}^{k}$
- intensitatea urmei de feromoni:

$$\gamma_{ij}^{(t+n)} = [(1-\varphi) * \gamma_{ij}^{(t)} + \varphi * \Delta\gamma_{ij}^{best}]_{\gamma_{min}}^{\gamma_{max}}$$

- unde φ $(0 < \varphi < 1)$ - coeficientul de evaporare a urmei de feromon intre 2 cururi complete
- $\gamma_{min} si \gamma_{max} -$limitele feromonului, $\gamma_{min} \rightarrow -\infty$, $\gamma_{max} \rightarrow \infty pentruACO$, $altfelMMAS$

Proprietati

- Algoritm iterativ, stocastic
- Algoritm care construieste progresiv solutia pe baza informatiilor euristice si a urmei de feromoni

Avantaje

- solutie compatibila cu modificari dinamice in timp real a datelor de intrare
- Euristica greedy ajusta la gasirea unei solutii acceptabile inca din primele stadii ale cautarii
- calculul distribuit evita convergenta prematura, prin interactiunea colectiva a indivizilor

Dezavantaje

- Converge incet fata de alte cautari euristice
- Eficienta scazuta pentru exemple cu mai mult de 75 de orase in problema comisului-voiajor
- Inexistenta unui mecanism central de ghidare catre solutiile bune

Aplicatii

- Probleme de identificare al unui drum optim in grafe
- Probleme de optimizari in retele
- Probleme de transport

Referinte

Informatiile prezentate au fost colectate din diferite surse, cea mai importanta dintre ele fiind:

- Prof. Dr. Diosan Laura - Inteligenta artificiala 2021-2022

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Introducere în Inteligenţa artificială

Laura Dioşan

Sumar

- Scurte întrebări despre IA
- Câteva repere istorice ale IA
- Exemple de utilizare a IA

Obiective

- Familiarizare cu domeniul AI
- Familiarizare cu vocabularul specific AI
- Descrierea principalelor repere in istoria AI

Câteva exemple

- Quick Draw https://quickdraw.withgoogle.com/
- Botsify https://botsify.com/
- Akinator https://en.akinator.mobi/
- Odizzi https://www.oddizzi.com/
- Splashmath https://www.splashmath.com/

Scurte întrebări despre IA

- Ce este IA?
  - Știința de a crea mașini care

Gândesc precum oamenii (științele cognitive, neuroștiințele)

Gândesc rațional (Aristotle -> codificarea gândirii)

Acționează precum oamenii (Turing test)

Acționează rațional

- Termenul **rațional**
  - A fi rațional = a maximiza atingerea anumitor scopuri
  - Raționalitatea privește deciziile luate (nu procesul de luare a acestora)
  - Obiectivele se pot exprima prin utilitatea rezultatelor/efectelor
  - Deci, **a fi rațional = a maximiza utilitatea așteptată**

Scurte întrebări despre IA

- Ce este IA?
  - Science Fiction AI

Scurte întrebări despre IA

- Ce este IA?
  - Întrebare dificilă (pentru că IA este foarte tânără)
  - ***AI is a branch of science which deals with helping machines find solutions to complex problems in a more human-like fashion.***
  - Pe scurt: determinarea maşinilor de a efectua lucruri inteligente
  - *Strong AI*
    - Calculatoarele pot fi programate să gândească la un nivel cel puţin egal cu cel uman şi chiar să fie conştiente de acţiunile lor
  - *Weak AI*
    - Calculatoarele pot efectua anumite sarcini de gândire – ceea ce deja se întâmplă

Scurte întrebări despre IA

- De ce avem nevoie de IA?
  - Ordonarea paginilor web

Scurte întrebări despre IA

- De ce avem nevoie de IA?
  - Ordonarea paginilor web
  - Recunoaşterea/analiza
    - vocii – http://www.indiegogo.com/projects/feed-your-pet-from-your-phone-with-pintofeed

Scurte întrebări despre IA

- De ce avem nevoie de IA?
  - Recunoaşterea/analiza
    - vocii
    - imaginilor

Scurte întrebări despre IA

- De ce avem nevoie de IA?
  - Recunoaşterea/analiza
    - vocii
    - imaginilor
    - scrisului de mână

Scurte întrebări despre IA

- De ce avem nevoie de IA?
  - Traducerea automată

Scurte întrebări despre IA

- De ce avem nevoie de IA?
  - Ordonarea paginilor web
  - Recunoaşterea/analiza
    - vocii
    - imaginilor
    - scrisului de mână
  - Traducerea automată
  - Diagnosticare medicală
  - Planificarea sarcinilor
  - Manipularea roboţilor în medii neprietenoase
  - Filtrarea spam-urilor
  - Evitarea air-craft-urilor

Scurtă istorie a IA

- See the movie!
- 1940-1950: Early days (nașterea IA)
  - 1943: McCulloch & Pitts: modelul de circuit boolean pentru creier
  - 1950: Turing's “Computing Machinery and Intelligence”
- 1950—1970: Excitement: Look, Ma, no hands! (Epoca de aur)
  - 1950s: primele programe de IA (Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine)
  - 1956: întâlnirea din Dartmouth: a fost adoptat termenul “Artificial Intelligence”
  - 1965: algoritmul complet al lui Robinson pentru deducții logice
- 1970—1990: Knowledge-based approaches
  - 1969—79: primele dezvoltări ale sistemelor bazate pe cunoștințe
  - 1980—88: Explozia industrială a sistemelor experte
  - 1988—93: o pauză a sistemlor expert industriale - “AI Winter”
- 1990—present: Statistical approaches (Meta AI)
  - Revenirea la probabilități, cu accent pe incertitudine
  - Creștere generală a dezvoltărilor tehnice
  - Agenți și sisteme de învățare automată “AI Spring”?
- 2020: Unde suntem acum?

Repere istorice ale IA

- Etape majore:
  - Naşterea IA (1943-1956)
  - Epoca de aur (1956-1974)
  - Prima iarnă (1974-1980)
  - Boom (1980-1987)
  - A 2-a iarnă (1987-1993)
  - IA meta-modernă (după 1993)

Câteva repere istorice ale IA – Naşterea IA (1943-1956)

- Care sunt originile IA?
  - Matematică, logică, informatică, filozofie, psihologie, ştiinţe cognitive, biologie
- Primele noţiuni de IA
  - 1943 → Walter Pitts şi Warren McCulloch propun neuronul artificial
  - 1950 → Alan Turing → testul Turing
    - Maşinile pot gândi? Putem să ne dăm seama într-o conversaţie dacă interlocutorul este o maşină sau nu?
    - Demo → ALICE http://www.alicebot.org
  - 1951 → primele programe pentru jocuri (dame şi şah)
  - 1955 → Allen Newell şi Herbert Simon → primul program pentru demonstrara automată a teoremelor

1950 1960 1970 1980 1990 2000

Testul Turing

Câteva repere istorice ale IA –Naşterea IA (194

- Conceptul de IA
  - 1956 → John McCarthy, şcoala de vară de la Dartmouth, SUA, propune termenul de IA
  - 1956 → John McCarthy face prima demonstraţie cu rularea unui program de IA la CMU (Carnegie Mellon University)

Câteva repere istorice ale IA – Epoca de aur (1956-1974)

- calculatoarele pot executa o anumită sarcină X
  - X = rezolvarea puzzle-urilor, demonstrarea teoremelor geometrice, jucarea jocului de dame
    - multe dintre aceste probleme → *toy problems*
  - 1958 → John McCarthy propune limbajul LISP la MIT (Massachusetts Institute of Technology)
  - 1965 → ELIZA
  - 1969 → robotul Shakey combină locomoţia, percepţia şi rezolvarea problemelor (Stanford Research Institute)
  - 1970 → "naşterea" algoritmilor evolutivi

Câteva repere istorice ale IA – Epoca de aur (1956-1974)

- 1966 – 1973 → o doză de realism
  - Necesitatea cunoştinţelor din domeniul problemei
    - Abordarea sintactică nu este suficientă → translatorul automat rusă-engleză (US suspendă finanţarea)
  - Control dificil → complexitate exponenţială
    - Guvernul britanic suspendă finanţarea IA → raportul lui Lighthill → opinie pesimistă asupra cercetării în domeniul IA
  - Limite teoretice → perceptronul nu poate rezolva problema XOR-ului
    - cercetarea reţelelor neuronale este suspendată

Câteva repere istorice ale IA – Epoca de aur (1956-1974)

- 1969 – 1988 → sisteme bazate pe cunoştinţe
  - Ghidarea căutării pe baza cunoştinţelor specifice domeniului problemei
    - Cyc → o bază de cunoştinţe → http://cyc.com
    - Sisteme expert dezvoltate de majoritatea companiilor

Câteva repere istorice ale IA – Prima iarnă (1974 – 1980)

- Probleme
  - Puterea limitată a calculatoarelor
  - Creşterea exponenţială a timpului necesar rezolvării unei probleme cu tehnici ale IA
  - Necesitatea unei baze de cunoştinţe specifice domeniului problemei
  - Sistarea finaţării

Câteva repere istorice ale IA – Prima iarnă (1974 – 1980)

- Sisteme expert
  - Investiţii masive
  - Promisiuni extravagante
- Crah financiar
  - S-a limitat finanţarea IA
- 1979 – primul vehicul autonom controlat de computer (the Stanford Cart)

Câteva repere istorice ale IA – Boom (1980 – 1987)

- Se dezvoltă puternic sistemele expert
  - MYCIN – Standford University
    - diagnosticul bolilor infecţioase de sânge
  - XCON (e**X**pert **CON**figurer) - Carnegie Mellon University
    - Selectarea componentelor unui calculator în funcţie de opţiunile utilizatorului

Câteva repere istorice ale IA – Boom (1980 – 1987)

- 1986 – reţele neuronale artificiale
  - Perceptronul multistrat
  - Redescoperirea algoritmului de antrenare backpropagation
  - Noi dezvoltări
    - Modelele simbolice (Newell, Simon)
    - Modelele logistice (McMarthy)
  - Apariţia învăţării automate statistice

Câteva repere istorice ale IA – a 2-a iarnă (1987-1993)

- Puterea de calcul limitată
- Suspiciunea companiilor
  - Banii au fost dirijaţi spre alte domenii de cercetare (diferite de IA)

Câteva repere istorice ale IA – IA metamodernă (1993 – prezent)

- 1997 – Deep Blue
- 1998 – programarea genetică
- 2000 – roboţi pe post de animale de companie

Câteva repere istorice ale IA – IA metamodernă (1993 – prezent)

- Modele statistice
  - Învăţare automată
    - Modele Markov ascunse (HMM)
    - Maşini cu suport vectorial (SVM)
    - Procese Gaussiene
    - Modele grafice (reţele Bayesiene, câmpuri condiţionate aleatorii)
    - Modele inspirate de natură (calcul evolutiv, PSO, ACO, calcul cuantic)
  - Analiza datelor (data mining)

Percepția publicului asupra IA

- Artificial Intelligence AI (2001) – Steven Spielberg

Exemple de utilizare a IA

- 2005: DARPA grand challenge
  - http://www.youtube.com/watch?v=jSnw6UHBREA
  - Defense Advanced Research Projects Agency → http://www.darpa.mil
- 2011: Google self-driving cars
  - View 1 (http://www.smartplanet.com/blog/thinking-tech/googles-self-driving-car/5445)
  - View 2 (http://www.youtube.com/watch?v=eXeUu_Y6WOw)

Exemple de utilizare a IA

- Testul Turing
  - ALICE
    - http://www.pandorabots.com/pandora/talk?botid=f5d922d97e345aa1
  - Laura: My name is Laura
    ALICE: Glad to know you, Laura.
    Laura: Which film do you like?
    ALICE: My favorite movie is Imagine. Have you seen it?
    Laura: No.
    ALICE: I would recommend you check it out.
    Laura: What is it about?
    ALICE: Science fiction.
    Laura: Do you remember my name?
    ALICE: You said your name is Laura?
    Laura: Yes, I said.
    ALICE: Give me a break.
    Laura: Do you know Turing?
    ALICE: A brilliant and tragic figure in the history of computer science.
    Laura: Who else do you know?
    ALICE: No I don't know. I am chatting with several other people at once.
    Laura: But Loebner?
    ALICE: The Loebner Prize is an annual Turing Test sponsored by New York philanthropist Hugh Loebner. Dr. Wallace and ALICE won it in 2000. The ALICE program was ranked "most human computer" by the panel of judges.

Exemple de utilizare a IA

Procesarea limbajului natural - *chatbot*

- Valerie: CMU Robot Receptionist in Newell-Simon hall.
- ALICE: 2004 Loebner Prize winner
- ELIZA: psychotherapist
- Jeopady (2011)
  - IBM's Watson – view

Exemple de utilizare a IA

Procesarea limbajului natural – recunoaşterea vocii

- Ton (vorbit sau prin contact) pentru numărul de card
  - Vocabular redus, dar nevoia unei acurateţi mărită
- Transmiterea mesajelor
  - Vocabular foarte larg şi nevoia unei acurateţi mari
- Dictare
  - Vocabular foarte larg şi nevoia unei acurateţi mari
  - Exemple
    - IBM Via Voice
    - Dragon Naturally Speaking
    - Siri

- Dpdv teoretic
  - Modele Markov ascunse,
  - Căutare de tip A*

Exemple de utilizare a IA

Procesarea limbajului natural – traducere automată

- de la
  - Georgetown-IBM experiment
- până la
  - Yahoo! Babel Fish → Systran – view (http://www.systranet.com/translate)
  - Free translation → SDL Language Weaver
  - Google translate → Google

"Il est impossible aux journalistes de rentrer dans les régions tibétaines"

Bruno Philip, correspondant du "Monde" en Chine, estime que les journalistes de l'AFP qui ont été expulsés de la province tibétaine du Qinghai "n'étaient pas dans l'illégalité".

Les faits Le dalaï-lama dénonce l'"enfer" imposé au Tibet depuis sa fuite, en 1959

Vidéo Anniversaire de la rébellion

"It is impossible for journalists to enter Tibetan areas"

Philip Bruno, correspondent for "World" in China, said that journalists of the AFP who have been deported from the Tibetan province of Qinghai "were not illegal."

Facts The Dalai Lama denounces the "hell" imposed since he fled Tibet in 1959

Video Anniversary of the Tibetan rebellion: China on guard

- Dpdv teoretic
  - Modele bazate pe reguli
  - Modele statistice de traducere (IBM)
  - Modele bazate pe exemple

Exemple de utilizare a IA

Percepția vizuală

- de la
  - Recunoașterea cifrelor scrise de mână
- până la
  - Recunoașterea fețelor și a emoțiilor
  - Recunoașterea obstacolelor în trafic
  - Segmentări semantice în domeniul medical

- Dpdv teoretic
  - Modelare geometrică
  - Modele de regresie
  - Modele de clasificare

Exemple de utilizare a IA

Jocuri – jocul de şah

- IBM Deep Blue vs. Kasparov, mai 1995
  - 6 jocuri: K, DB, remiză, remiză, remiză, DB
  - IBM → 18 bilioane de dolari
- Alpha Go - https://deepmind.com/research/case-studies/alphago-the-story-so-far

- Din punct de vedere teoretic
  - Joc cu
    - 2 jucători
    - Sumă zero
    - Stări discrete
    - Cu informaţie perfectă
    - Final finit
  - Modelare
    - *temporal difference learning (TD) algorithm*
    - *distributional reinforcement learning*

Exemple de utilizare a IA

www – căutări pe web

- Selectarea/aranjarea automată a ştirilor provenite din diferite surse
  - vs.
- Organizarea manuală a ştirilor → CNN

- Dpdv teoretic
  - învăţare nesupervizată (clustering)

Exemple de utilizare a IA

www – orientare (pe hartă)

- Din UBB → străzile din New York

- Dpdv teoretic
  - strategii de căutare

Exemple de utilizare a IA

www – extragerea informaţiei

- Extragerea informaţiilor despre un job
  - Flipdog → http://www.flipdog.com/
- Dpdv teoretic
  - învăţare automată → învăţare supervizată (clasificare)

Exemple de utilizare a IA

www – filtrare colaborativă

- Recomandări bazate pe comportamentul utilizatorilor
  - Amazon → view (http://www.amazon.com/Intelligent-Systems-Approach-Reference-Library/dp/3642210031)

- Dpdv teoretic
  - învăţare automată → învăţare nesupervizată (clustering)

Exemple de utilizare a IA

Robotică – pantofi inteligenţi

- Adaptarea pernei de aer la viteza de deplasare, suprafaţa drumului, etc

- Dpdv teoretic
  - regresie simplă

Exemple de utilizare a IA

Robotică – fotbal cu roboţi

- Robocup → http://www.robocup.org/

  - View the movie
  - http://www.youtube.com/watch?v=-Y4H3Sox_4I

- Dpdv teoretic
  - învăţare automată → învăţare cu întărire

Exemple de utilizare a IA

Robotică – roboţi umanoizi

- Roboţi bipezi, care se deplasează similar oamenilor
  - Asimo (Honda) → view
  - QRIO (Sony)

Exemple de utilizare a IA

Robotică – telescopul Hubble

- Planificare: cine şi când merge să vadă ceva?
  - 30000 observaţii pe an
  - multe contrângeri

- Dpdv teoretic
  - problemă de satisfacere a constrângerilor

Exemple de utilizare a IA

Robotică – vehiculele de pe Marte

- Conducerea automată a vehiculelor pe marte

- Dpdv teoretic
  - planificarea mişcării roboţilor

Exemple de utilizare a IA

Artă

- AARON
  - view

- Dpdv teoretic
  - învăţare automată

Exemple de utilizare a IA

Dispozitive mobile

- Text-to-Picture
  - Proiect pentru dezvoltarea de aplicaţii care să genereze propoziţii simple pe baza gesturilor observate
    - Ajutor pentru oamenii cu mai puţine abilităţi de comunicare verbală

- Dpdv teoretic
  - Învăţare supervizată şi nesupervizată

IA astăzi, pe scurt

- Sumar
  - Nu se ştie (încă) cum pot fi realizate 98% din lucrurile inteligente
  - Dar restul de 2% pot fi făcute chiar foarte bine
  - Nu este nimic magic în IA. Totul este despre:
    - Optimizare
    - Probabilităţi şi statistică
    - Logică
    - Algoritmi

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENȚĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare neinformată

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Sumar

- Probleme
- Rezolvarea problemelor
  - Paşi în rezolvarea problemelor
- Rezolvarea problemelor prin căutare
  - Paşi în rezolvarea problemelor prin căutare
  - Tipuri de strategii de căutare

Materiale de citit şi legături utile

- capitolele I.1, I.2, II.3 şi II.4 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*

- capitolele 1 - 4 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- capitolele 2.1 – 2.5 din http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/

Probleme

- Două mari categorii de probleme:
  - Rezolvabile în mod determinist
    - Calculul sinusului unui unghi sau a rădăcinii pătrate dintr-un număr
  - Rezolvabile în mod stocastic
    - Probleme din lumea reală → proiectarea unui ABS
    - Presupun căutarea unei soluţii → metode ale IA

Probleme

- Tipologie
  - Probleme de căutare/optimizare
    - Planificare, proiectarea sateliţilor

  - Probleme de modelare
    - Predicţii, clasificări

  - Probleme de simulare
    - Teoria jocurilor economice

Rezolvarea problemelor

- Constă în identificarea unei soluţii
  - În informatică (IA) → proces de căutare
  - În inginerie şi matematică → proces de optimizare

- Cum?
  - Reprezentarea soluţiilor (parţiale) → puncte în spaţiul de căutare
  - Proiectarea unor operatori de căutare → transformă o posibilă soluţie în altă soluţie

Paşi în rezolvarea problemelor

- Definirea problemei
- Analiza problemei
- Alegerea unei tehnici de rezolvare
  - căutare
  - reprezentarea cunoştinţelor
  - abstractizare

Rezolvarea problemelor prin căutare

- bazată pe urmărirea unor obiective
- compusă din acţiuni care duc la îndeplinirea unor obiective
  - fiecare acţiune modifică o anumită stare a problemei
- succesiune de acţiuni care transformă starea iniţială a problemei în stare finală

Paşi în rezolvarea problemelor prin căutare – Definirea problemei

- Definirea problemei implică stabilirea:
  - unui spaţiu de stări
    - toate configuraţiile posibile, fără enumerarea obligatorie a tuturor configuraţiilor
    - reprezentare
      - explicită – construirea (în memorie) a tuturor stărilor posibile
      - implicită – utilizarea unor structuri de date şi a unor funcţii (operatori)
  - unei/unor stări iniţiale
  - unei/unor stări finale - obiectiv
  - unui/unor drum(uri)
    - succesiuni de stări
  - unui set de reguli (acţiuni)
    - funcţii succesor (operatori) care precizează starea următoare a unei stări
    - funcţii de cost care evaluează
      - trecerea dintr-o stare în alta
      - un întreg drum
    - funcţii obiectiv care verifică dacă s-a ajuns într-o stare finală

Paşi în rezolvarea problemelor prin căutare – Definirea problemei

- Exemple
  - Joc puzzle cu 8 piese
    - Spaţiul stărilor – configuraţii ale tablei de joc cu 8 piese
    - Starea iniţială – o configuraţie oarecare
    - Starea finală – o configuraţie cu piesele aranjate într-o anumită ordine
    - Reguli → acţiuni albe
      - Condiţii: mutarea în interiorul tablei
      - Transformări: spaţiul alb se mişcă în sus, în jos, la stânga sau la dreapta
    - Soluţia – secvenţa optimă de acţiuni albe

Paşi în rezolvarea problemelor prin căutare
Definirea problemei

- Exemple
  - Joc cu *n* dame
    - Spaţiul stărilor – configuraţii ale tablei de joc cu *n* regine
    - Starea iniţială – o configuraţie fără regine
    - Starea finală – o configuraţie cu *n* regine care nu se atacă
    - Reguli → amplasarea unei regine pe tablă
      - Condiţii: regina amplasată nu este atacată de nici o regină existentă pe tablă
      - Transformări: amplasarea unei noi regine într-o căsuţă de pe tabla de joc
    - Soluţia – amplasarea optimă a reginelor pe tablă

Paşi în rezolvarea problemelor prin căutare
Analiza problemei

- Se poate descompune problema?
  - Sub-problemele sunt independente sau nu?
- Universul stărilor posibile este predictibil?
- Se doreşte obţinerea oricărei soluţii sau a unei soluţii optime?
- Soluţia dorită constă într-o singură stare sau într-o succesiune de stări?
- Sunt necesare multiple cunoştinţe pentru a limita căutarea sau chiar pentru a identifica soluţia?
- Problema este conversaţională sau solitară?
  - Este sau nu nevoie de interacţiune umană pentru rezolvarea ei?

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

- Rezolvarea prin utilizarea regulilor (în combinaţie cu o strategie de control) de deplasare în spaţiul problemei până la găsirea unui drum între starea iniţială şi cea finală

- Rezolvare prin căutare
  - Examinarea sistematică a stărilor posibile în vederea identificării
    - unui drum de la o stare iniţială la o stare finală
    - unei stări optime
  - Spaţiul stărilor = toate stările posibile + operatorii care definesc legăturile între stări

Paşi în rezolvarea problemelor prin căutare – Alegerea unei tehnici de rezolvare

- Rezolvare prin căutare
  - Strategii de căutare multiple ➔ cum alegem o strategie?
    - Complexitatea computaţională (temporală şi spaţială)
    - Completitudine → algoritmul se sfârşeşte întotdeauna şi găseşte o soluţie (dacă ea există)
    - Optimalitate → algoritmul găseşte soluţia optimă (costul optim al drumului de la starea iniţială la starea finală)

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

- Rezolvare prin căutare
  - Strategii de căutare multiple ➔ cum alegem o strategie?

    - Complexitatea computaţională (temporală şi spaţială)
      - Performanţa strategiei depinde de:
        - Timpul necesar rulării
        - Spaţiul (memoria) necesară rulării

        } Factori interni

        - Mărimea intrărilor algoritmului
        - Viteza calculatorului
        - Calitatea compilatorului

        } Factori externi
    - Se măsoară cu ajutorul complexităţii → Eficienţă computaţională
      - Spaţială → memoria necesară identificării soluţiei
        - *S(n)* – cantitatea de memorie utilizată de cel mai bun algoritm A care rezolvp o problemă de decizie *f* cu *n* date de intrare
      - Temporală → timpul necesar identificării soluţiei
        - *T(n)* – timpul de rulare (numărul de paşi) al celui mai bun algoritm A care rezolvă o problemă de decizie *f* cu *n* date de intrare

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

- Rezolvarea problemelor prin căutare poate consta în:
  - Construirea progresivă a soluţiei

  - Identificarea soluţiei potenţiale optime

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

- Rezolvarea problemelor prin căutare poate consta în:
  - Construirea progresivă a soluţiei
    - Componentele problemei
      - Stare iniţială
      - Operatori (funcţii succesor)
      - Stare finală
      - Soluţia = un drum (de cost optim) de la starea iniţială la starea finală
    - Spaţiul de căutare
      - Mulţimea tuturor stărilor în care se poate ajunge din starea iniţială prin aplicarea operatorilor
      - stare = o componentă a soluţiei
    - Exemple
      - Problema comisului voiajor
    - Algoritmi
      - Ideea de bază: se începe cu o componentă a soluţiei şi se adaugă noi componente până se ajunge la o soluţie completă
      - Recursivi → se re-aplică până la îndeplinirea unei condiţii
      - Istoricul căutării (drumul parcurs de la starea iniţială la starea finală) este reţinut în liste de tip LIFO sau FIFO
    - Avantaje
      - Nu necesită cunoştinţe (informaţii inteligente)

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

- Rezolvarea problemelor prin căutare poate consta în:
  - Identificarea soluţiei potenţiale optime
    - Componentele problemei
      - Condiţii (constrângeri) pe care trebuie să le satisfacă (parţial sau total) soluţia
      - Funcţie de evaluare a unei soluţii potenţiale → identificareaa optimului
    - Spaţiul de căutare
      - mulţimea tuturor soluţiilor potenţiale complete
      - Stare = o soluţie completă
    - Exemple
      - Problema celor 8 regine
    - Algoritmi
      - Ideea de bază: se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluţie optimă şi se efectuează modificări pentru a elimina aceste violări
      - Iterativi → se memorează o singură stare şi se încearcă îmbunătăţirea ei
      - Istoricul căutării nu este reţinut
    - Avantaje
      - Simplu de implementat
      - Necesită puţină memorie
      - Poate găsi soluţii rezonabile în spaţii de căutare (continue) foarte mari pentru care alţi algoritmi sistematici nu pot fi aplicaţi

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

- Rezolvarea problemelor prin căutare presupune
  - algoritmi cu o complexitate ridicată (probleme NP-complete)
  - căutarea într-un spaţiu exponenţial

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

- Tipologia strategiilor de căutare
  - În funcţie de modul de **generare** a soluţiei
    - Căutare **constructivă**
      - Construirea progresivă a soluţiei
      - Ex. TSP
    - Căutare **perturbativă**
      - O soluţie candidat este modificată în vederea obţinerii unei noi soluţii candidat
      - Ex. SAT - Propositional Satisfiability Problem
  - În funcţie de modul de **traversare** a spaţiului de căutare
    - Căutare **sistematică**
      - Traversarea completă a spaţiului
        - Ideintificarea soluţiei dacă ea există ➔ algoritmi compleţi
    - Căutare **locală**
      - Traversarea spaţiului de căutare dintr-un punct în alt punct vecin ➔ algoritmi incompleţi
      - O stare a spaţiului poate fi vizitată de mai multe ori
  - În funcţie de elementele de **certitudine** ale căutării
    - Căutare **deterministă**
      - Algoritmi de identificare exactă a soluţiei
    - Căutare **stocastică**
      - Algoritmi de aproximare a soluţiei
  - În funcţie de stilul de **explorare** a spaţiului de căutare
    - Căutare **secvenţială**
    - Căutare **paralelă**

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

Tipologia strategiilor de căutare

- În funcţie de **scopul** urmărit
  - Căutare **uni-obiectiv**
    - Soluţia trebuie să satisfacă o signură condiţie/constrângere
  - Căutare **multi-obiectiv**
    - Soluţia trebuie să satisfacă mai multe condiţii (obiective)
- În funcţie de **numărul de soluţii optime**
  - Căutare **uni-modală**
    - Există o singură soluţie optimă
  - Căutare **multi-modală**
    - Există mai multe soluţii optime
- În funcţie de tipul de **algoritm** folosit
  - Căutare de-a lungul unui **număr finit de paşi**
  - Căutare **iterativă**
    - Algoritmii converg către soluţie
  - Căutare **euristică**
    - Algoritmii oferă o aproximare a soluţiei
- În funcţie de **mecanismul** căutării
  - Căutare **tradiţională**
  - Căutare **modernă**
- În funcţie de **locul** în care se desfăşoară căutarea în spaţiul de căutare
  - Căutare **locală**
  - Căutare **globală**

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

Tipologia strategiilor de căutare

- În funcţie de tipul **(linearitatea) constrângerilor**
  - Căutare **liniară**
  - Căutare **neliniară**
    - Clasică (deterministă)
      - Directă – bazată doar pe evaluarea funcţiei obiectiv
      - Indirectă – bazată şi pe derivata (I si/sau II) a funcţiei obiectiv
    - Enumerativă
      - În funcţie de modul de stabilire a soluţiei
        - Ne-informată → soluţia se ştie doar când ea coincide cu starea finală
        - Informată → se lucrează cu o funcţie de evaluare a unei soluţii parţiale
      - În funcţie de tipul spaţiului de căutare
        - Completă – spaţiu finit (dacă soluţia există, ea poate fi găsită)
        - Incompletă – spaţiu infinit
    - Stocastică
      - Bazată pe elemente aleatoare
- În funcţie de **agenţii** implicaţi în căutare
  - Căutare realizată de un **singur agent** → fără piedici în atingerea obiectivelor
  - Căutare **adversială** → adversarul aduce o incertitudine în realizarea obiectivelor

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

Exemplu

- Tipologia strategiilor de căutare
  - În funcţie de modul de generare a soluţiei
    - **Căutare constructivă**
    - Căutare perturbativă
  - În funcţie de modul de traversare a spaţiului de căutare
    - **Căutare sistematică**
    - Căutare locală
  - În funcţie de elementele de certitudine ale căutării
    - **Căutare deterministă**
    - Căutare stocastică
  - În funcţie de stilul de explorare a spaţiului de căutare
    - **Căutare secvenţială**
    - Căutare paralelă

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

Exemplu

- Problema "capra, varza şi lupul"
  - Se dau:
    - o capră, o varză şi un lup pe malul unui râu
    - o barcă cu barcagiu
  - Se cere
    - Să se traverseze toţi pasagerii pe malul celălalt al râului
    - cu următoarele condiţii
      - în barcă există doar 2 locuri
      - nu pot rămâne pe acelaşi mal
        - capra şi varza
        - lupul şi capra

Paşi în rezolvarea problemelor prin căutare
Alegerea unei tehnici de rezolvare

Strategii de căutare – Elemente fundamentale

- Tipuri abstracte de date (TAD)
  - TAD listă → structură liniară
  - TAD arbore → structură arborescentă (ierarhică)
  - TAD graf → structură de graf

- TAD
  - Domeniu şi operaţii
  - Reprezentare

Strategii de căutare – Elemente fundamentale – TAD Listă

- Domeniu
  - *D = {l | l = (el1, el2, …),* unde el_i, *i=1,2,3…*, sunt de acelaşi tip *TE* (*tip element*) şi fiecare element el_i, *i=1,2,3…,* are o poziţie unică în *l* de tip *TP* (*TipPoziţie*)}
- Operaţii
  - •Creare(l)
  - •Prim(l)
  - •Ultim(l)
  - •Următor(l,p)
  - •Anterior(l,p)
  - •Valid(l,p)
  - •getElement(l,p)
  - •getPoziţie(l,e)
  - •Modifică(l,p,e)
  - •AdăugareLaÎnceput(l,e)
  - •AdăugareLaSfârşit(l,e)
  - •AdăugareDupă(l,p,e)
  - •AdăugareÎnainte(l,p,e)
  - •Eliminare(l,p)
  - •Căutare(l,e)
  - •Vidă(l)
  - •Dimensiune(l)
  - •Distrugere(l)
  - •getIterator(l)
- Reprezentare
  - Vectorială
  - Liste (simplu sau dublu) înlănţuite, etc
- Cazuri particulare
  - Stivă – LIFO
  - Coadă – FIFO
  - Coadă cu priorităţi

Strategii de căutare – Elemente fundamentale – TAD Listă

- Domeniu
  - *D = {l | l = (el1, el2, …),* unde el_i, *i=1,2,3…*, sunt de acelaşi tip *TE* (*tip element*) şi fiecare element el_i, *i=1,2,3…,* are o poziţie unică în *l* de tip *TP* (*TipPoziţie*)}
- Operaţii
  - •Creare(l)
  - •Prim(l)
  - •Ultim(l)
  - •Următor(l,p)
  - •Anterior(l,p)
  - •Valid(l,p)
  - •getElement(l,p)
  - •getPoziţie(l,e)
  - •Modifică(l,p,e)
  - •AdăugareLaÎnceput(l,e)
  - •AdăugareLaSfârşit(l,e)
  - •AdăugareDupă(l,p,e)
  - •AdăugareÎnainte(l,p,e)
  - •Eliminare(l,p)
  - •Căutare(l,e)
  - •Vidă(l)
  - •Dimensiune(l)
  - •Distrugere(l)
  - •getIterator(l)
- Reprezentare
  - Vectorială
  - Liste (simplu sau dublu) înlănţuite, etc
- Cazuri particulare
  - Stivă – LIFO
  - Coadă – FIFO
  - Coadă cu priorităţi

Strategii de căutare – Elemente fundamentale – TAD Listă

- Domeniu
  - *D = {l | l = (el1, el2, …),* unde el_i, *i=1,2,3…*, sunt de acelaşi tip *TE* (*tip element*) şi fiecare element el_i, *i=1,2,3…,* are o poziţie unică în *l* de tip *TP* (*TipPoziţie*)}
- Operaţii
  - •Creare(l)
  - •Prim(l)
  - •Ultim(l)
  - •Următor(l,p)
  - •Anterior(l,p)
  - •Valid(l,p)
  - •getElement(l,p)
  - •getPoziţie(l,e)
  - •Modifică(l,p,e)
  - •AdăugareLaÎnceput(l,e)
  - •AdăugareLaSfârşit(l,e)
  - •AdăugareDupă(l,p,e)
  - •AdăugareÎnainte(l,p,e)
  - •Eliminare(l,p)
  - •Căutare(l,e)
  - •Vidă(l)
  - •Dimensiune(l)
  - •Distrugere(l)
  - •getIterator(l)
- Reprezentare
  - Vectorială
  - Liste (simplu sau dublu) înlănţuite, etc
- Cazuri particulare
  - Stivă – LIFO
  - Coadă – FIFO
  - Coadă cu priorităţi

Strategii de căutare – Elemente fundamentale – TAD Graf

- Domeniu – container de noduri si legături între noduri
  - $D = \{nod_1, nod_2, \ldots, nod_n, leg_1, leg_2, \ldots, leg_m$, unde nod_i, cu $i=1,2,\ldots,n$ sunt noduri, iar leg_i, cu $i=1,2,\ldots,m$ sunt muchii între noduri$\}$

- Operaţii
  - creare
  - creareNod
  - traversare
  - getIterator
  - distrugere

- Reprezentare
  - Lista muchilor
  - Lista de adiacenţă (Tradiţională şi Modernă)
  - Matricea de adiacenţă (Tradiţională şi Modernă)
  - Matricea de incidenţă

- Cazuri particulare
  - Grafuri orientate şi neorientate
  - Grafuri simple sau multiple
  - Grafuri conexe sau nu
  - Grafuri complete sau nu
  - Grafuri cu sau fără cicluri (aciclice → păduri, arbori)

Strategii de căutare – Elemente fundamentale – TAD Arbore

- Domeniu – container de noduri si legături între noduri
  - $D = \{nod_1, nod_2, ..., nod_n, leg_1, leg_2, ..., leg_m$, unde nod_i, cu $i=1,2,...,n$ sunt noduri, iar leg_i, cu $i=1,2,...,m$ sunt muchii între noduri astfel încât să nu se formeze cicluri$\}$

- Operaţii
  - creare
  - creareFrunză
  - adăugareSubarbore
  - getInfoRădăcină
  - getSubarbore
  - traversare
  - getIterator
  - distrugere

- Reprezentare
  - Vectorială
  - Liste înlănţuite ale descendenţilor

- Cazuri particulare
  - Arbori binari (de căutare)
  - Arbori n-ari

Strategii de căutare – Elemente fundamentale – parcurgerea grafelor

- Drumuri
  - drum (*path)*
    - nodurile nu se pot repeta
  - *trail*
    - muchiile nu se pot repeta
  - *walk*
    - fără restricţii
  - drum închis
    - nodul iniţial = nodul final
  - circuit
    - un *trail* închis
  - ciclu
    - un *path* închis

Strategii de căutare neinformate (SCnI)

- Caracteristici
  - nu se bazează pe informaţii specifice problemei
  - sunt generale
  - strategii oarbe
  - strategii de tipul forţei brute

- Topologie
  - În funcţie de ordinea expandării stărilor în spaţiul de căutare:
    - SCnI în structuri liniare
      - căutare liniară
      - căutare binară
    - SCnI în structuri ne-liniare
      - căutare în lăţime (breadth-first)
        - căutare de cost uniform (branch and bound)
      - căutare în adâncime (depth-first)
        - căutare în adâncime limitată (limited depth-first)
        - căutare în adâncime iterativă (iterative deepening depth-first)
      - căutare bidirecţională

SCnI în structuri liniare
Căutare liniară

- **Aspecte teoretice**
  - Se verifică fiecare element al unei liste până la identificarea celui dorit
  - Lista de elemente poate fi ordonată sau nu

- **Exemplu**
  - Lista = (2, 3, 1, ,7, 5)
  - Elem = 7

- **Algoritm**

```
bool LS(elem, list){
      found = false;
      i = 1;
      while ((!found) && (i <= list.length)){
            if (elem = list[i])
               found = true;
            else
               i++;
      }  //while
      return found;
}
```

SCnI în structuri liniare
Căutare liniară

- **Analiza căutării**
  - Complexitate temporală
    - Cel mai bun caz: *elem = list[1] => O(1)*
    - Cel mai slab caz: *elem ∉ list => T(n) = n +1 => O(n)*
    - Cazul mediu: *T(n) = (1 + 2 + … + n + (n+1))/(n+1) => O(n)*
  - Complexitate spaţială
    - *S(n) = n*
  - Completitudine
    - da
  - Optimalitate
    - da
- **Avantaje**
  - Simplitate, complexitate temporală bună pentru structuri mici
  - Structura nu trebuie sortată în prealabil
- **Dezavantaje**
  - complexitate temporală foarte mare pentru structuri mari
- **Aplicaţii**
  - Căutări în baze de date reale

SCnI în structuri liniare
Căutare binară

- **Aspecte teoretice**
  - Localizarea unui element într-o listă ordonată
  - Strategie de tipul *Divide et Conquer*

- **Exemplu**
  - List = (2, 3, 5, 6, 8, 9, 13,16, 18), Elem = 6
    - List = (2, 3, 5, 6, 8, 9, 13,16, 18)
    - List = (2, 3, 5, 6)
    - List = (5, 6)
    - List = (6)

- **Algoritm**

```
bool BS(elem, list){
      found = false;
      left = 1;
      right = list.length;
      while((left < right) && (!found)){
            middle = left + (right - left)/2;
            if (element == list[middle])
                found = true;
            else
                  if (element < list[middle])
                      right = middle - 1;
                  else
                      left = middle + 1;
      }  //while
      return found;
}
```

SCnI în structuri liniare
Căutare binară

- **Analiza căutării**
  - Complexitate temporală $T(n) = 1$, pt n = 1 şi $T(n) = T(n/2) + 1$, altfel

    Pp. că $n = 2k \Rightarrow k = \log_2 n$

    Pp. că $2k < n < 2k+1 \Rightarrow k < \log 2n < k + 1$

    $T(n) = T(n/2) + 1$

    $T(n/2) = T(n/22) + 1$

    …

    $T(n/2k-1) = T(n/2k) + 1$

    $T(n) = k + 1 = \log_2 n + 1$
  - Complexitate spaţială – $S(n) = n$
  - Completitudine – da
  - Optimalitate – da
- **Avantaje**
  - Complexitate temporală redusă faţă de căutarea liniară
- **Dezavantaje**
  - Lucrul cu vectori (trebuie accesate elemente indexate) sortaţi
- **Aplicaţii**
  - Jocul ghicirii unui număr
  - Căutare într-o carte de telefon/dicţionar

SC în structuri arborescente

- Noţiuni necesare
  - *f(n)* – funcţie de evaluare pentru estimarea costului soluţiei prin nodul (starea) *n*
  - *h(n)* – funcţie euristică pentru estimarea costului drumului de la nodul *n* la nodul obiectiv
  - *g(n)* – funcţie de cost pentru estimarea costului drumului de la nodul de start până la nodul *n*
  - *f(n) = g(n) + h(n)*

SCnI în structuri arborescente
căutare în lățime (breadth-first search – BFS)

- **Aspecte teoretice**
  - Toate nodurile aflate la adâncimea *d* se expandează înaintea nodurile aflate la adâncimea *d+1*
  - Toate nodurile fii obţinute prin expandarea nodului curent se adaugă într-o *listă* de tip *FIFO (coadă)*

- **Exemplu**

  - Ordinea vizitării: A, B, C, D, E, F, G, H, I, J, K

- **Algoritm**

```
bool BFS(elem, list){
       found = false;
       visited = Φ;
       toVisit = {start};          //FIFO list
       while((toVisit != Φ) && (!found)){
              node = pop(toVisit);
              visited = visited U {node};
              if (node == elem)
                  found = true;
              else{
                  aux = Φ;
                  for all (unvisited) children of node do{
                      aux = aux U {child};
                  }
              }
              toVisit = toVisit U aux;
       }  //while
       return found;
}
```

Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	C, D, E, F
A, B, C	D, E, F, G
A, B, C, D	E, F, G, H, ,I, J
A, B, C, D, E	F, G, H, I, J
A, B, C, D, E, F	G, H, I, J
A, B, C, D, E, F, G	H, I, J
A, B, C, D, E, F, G, H	I, J
A, B, C, D, E, F, G, H, I	J, K
A, B, C, D, E, F, G, H, I, J	K
A, B, C, D, E, F, G, H, I, J, K	Φ

SCnI în structuri arborescente
căutare în lățime (breadth-first search – BFS)

- **Analiza căutării:**
  - Complexitate temporală:
    - *b* – factor de ramificare (nr de noduri fii ale unui nod)
    - *d* - lungimea (adâncimea) soluţiei
    - $T(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Complexitate spaţială
    - $S(n) = T(n)$
  - Completitudine
    - Dacă soluţia există, atunci BFS o găseşte
  - Optimalitate
    - nu
- **Avantaje**
  - Găsirea drumului de lungime minimă până la nodul obiectiv (soluţia cea mai puţin adâncă)
- **Dezavantaje**
  - Generarea şi stocarea unui arbore a cărui mărime creşte exponenţial cu adâncimea nodului obiectiv
  - Complexitate temporală şi spaţială exponenţială
  - Experimentul Russel&Norvig????
  - Funcţional doar pentru spaţii de căutare mici
- **Aplicaţii**
  - Identificarea tuturor componentelor conexe într-un graf
  - Identificarea celui mai scurt drum într-un graf
  - Optimizări in reţele de transport → algoritmul Ford-Fulkerson
  - Serializarea/deserializarea unui arbore binar (vs. serializarea în mod sortat) permite reconstrucţia eficientă a arborelui
  - Copierea colecţilor (garbage collection) → algoritmul Cheney

Vizitate deja	De vizitat
Φ	B
B	A, E, F
B, A	E, F, C
B, A, E	F, C
B, A, E, F	C
B, A, E, F, C	Φ

SCnI în structuri arborescente
căutare de cost uniform (uniform cost search – UCS)

- **Aspecte teoretice**
  - BFS + procedură specială de expandare a nodurilor (bazată pe costurile asociate legăturilor dintre noduri)
  - Toate nodurile de la adâncimea *d* sunt expandate înaintea nodurilor de la adâncimea *d+1*
  - Toate nodurile fii obţinute prin expandarea nodului curent se adaugă într-o *listă ORDONATĂ* de tip *FIFO*
    - Se expandează mai întâi nodurile de cost minim
    - Odată obţinut un drum până la nodul ţintă, acesta devine candidat la drumul de cost optim
  - Algoritmul *Branch and bound*

- **Exemplu**
  - Ordinea vizitării nodurilor: A, C, B, D, G, E, F, I, H, J, K

- **Algoritm**

```
bool UCS(elem, list){
      found = false;
      visited = Φ;
      toVisit = {start};      //FIFO sorted list
      while((toVisit != Φ) && (!found)){
            node = pop(toVisit);
            visited = visited U {node};
            if (node== elem)
               found = true;
            else
               aux = Φ;
            for all (unvisited) children of node do{
               aux = aux U {child};
            }  // for
            toVisit = toVisit U aux;
            TotalCostSort(toVisit);
      }  //while
      return found;
}
```

visited	toVisit
Φ	A
A	C(3), B(7), D(9)
A, C	B(7), D(9), G(3+7)
A, C, B	D(9), G(10), E(7+10), F(7+15)
A, C, B, D	G(10), I(9+3), J(9+4) ,H(9+5), E(17), F(22)
A, C, B, D, G	I(12), J(13) ,H(14), E(17), F(22)
A, C, B, D, G, I	J(13) ,H(14), E(17), F(22), **K(9+3+7)**
A, C, B, D, G, I, J	H(14), E(17), F(22), **K(19)**
A, C, B, D, G, I, J, H	E(17), F(22), **K(19)**
A, C, B, D, G, I, J, H, E	F(22), **K(19)**
A, C, B, D, G, I, J, H, E, F	K(19)
A, C, B, D, G, I, J, H, E, F, K	Φ

SCnI în structuri arborescente
căutare de cost uniform (uniform cost search – UCS)

- **Analiza complexităţii**
  - Complexitate temporală:
    - *b* – factor de ramificare (nr de noduri fii ale unui nod)
    - *d* - lungimea (adâncimea) soluţiei
    - $T(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Complexitate spaţială
    - $S(n) = T(n)$
  - Completitudine
    - Da – dacă soluţia există, atunci UCS o găseşte
  - Optimalitate
    - Da

- **Avantaje**
  - Găsirea drumului de cost minim până la nodul obiectiv

Vizitate deja	De vizitat
Φ	A(0)
A(0)	B(5), C(10)
A(0), B(5)	F(8), C(10), E(14)
A(0), B(5), F(8)	C(9), E(10)
A(0), B(5), F(8), C(9)	E(10), H(14)
A(0), B(5), F(8), C(9), E(10)	H(14)

- **Dezavantaje**
  - Complexitate temporală şi spaţială exponenţială
- **Aplicaţii**
  - Cel mai scurt drum → algoritmul Dijkstra

SCnI în structuri arborescente
căutare în adâncime (depth-first search – DFS)

- **Aspecte teoretice**
  - Expandarea intr-un nod fiu şi înaintarea în adâncime până când
    - Este găsit un nod ţintă (obiectiv) sau
    - Nodul atins nu mai are fii
  - Cu revenirea în cel mai recent nod care mai poate fi explorat
  - Toate nodurile fii obţinute prin expandarea nodului curent se adaugă

  într-o listă de tip *LIFO (stivă)*
  - Similar cu BFS, dar nodurile se plasează într-o stivă (în loc de coadă)

- **Exemplu**
  - Ordinea vizitării nodurilor: A, B, E, F, C, G, D, H, I, K, J

- **Algoritm**

```
bool DFS(elem, list){
      found = false;
      visited = Φ;
      toVisit = {start};       //LIFO list
      while((toVisit != Φ) && (!found)){
            node = pop(toVisit);
            visited = visited U {node};
            if (node== elem)
               found = true;
            else{
                  aux = Φ;
                  for all (unvisited) children of node do{
                     aux = aux U {child};
                  }
                  toVisit = aux U toVisit;
            }
      }  //while
      return found;
}
```

Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	K, J
A, B, E, F, C, G, D, H, I, K	J
A, B, E, F, C, G, D, H, I, K, J	Φ

SCnI în structuri arborescente
căutare în adâncime (depth-first search – DFS)

- **Analiza complexităţii**
  - Complexitate temporală:
    - b – factor de ramificare (nr de noduri fii ale unui nod)
    - d^{max} - lungimea (adâncimea) maximă a unui arbore explorat
    - $T(n) = 1 + b + b^2 + \ldots + b^{dmax} \Rightarrow O(b^{dmax})$
  - Complexitate spaţială
    - $S(n) = b * d_{max}$
  - Completitudine
    - Nu → algoritmul nu se termină pt drumurile infinite (neexistând suficientă memeorie pt reţinerea nodurilor deja vizitate)
  - Optimalitate
    - Nu → căutarea în adâncime poate găsi un drum soluţie mai lung decât drumul optim

- **Avantaje**
  - Găsirea drumului de lungime minimă până la nodul obiectiv cu consum minim de memorie
    - versiunea recursivă

- **Dezavantaje**
  - Se poate bloca pe anumite drumuri greşite (nenorocoase) fără a putea reveni
    - Ciclu infinit
    - Găsirea unei soluţii mai "lungi" decât soluţia optimă

- **Aplicaţii**
  - Problema labirintului (maze)
  - Identificarea componentelor conexe
  - Sortare topologică
  - Testarea planarităţii

SCnI în structuri arborescente
căutare în adâncime (depth-first search – DFS)

```
bool DFS_edges(elem, list){
   discovered = Φ;
   back = Φ;
   toDiscover = Φ;     //LIFO
   for (all neighbours of start) do
        toDiscover = toDiscover U {(start, neighbour)}
   found = false;
   visited = {start};
   while((toDiscover != Φ) && (!found)){
        edge = pop(toDiscover);
        if (edge.out !є visited){
           discovered = discovered U {edge};
           visited = visited U {edge.out}
           if (edge.out == end)
                found = true;
           else{
                aux = Φ;
                for all neighbours of edge.out  do{
                aux = aux U {(edge.out, neighbour)};
           }
           toDiscover = aux U toDiscover;
        }
        else
           back = back U {edge}
   }  //while
   return found;
}
```

Muchia	Muchii vizitate deja	Muchii de vizitat	înapoi	Noduri vizitate
	Φ	AB, AF	Φ	A
AB	AB	BC, BK, AF	Φ	A, B
BC	AB, BC	CD, BK, AF	Φ	A, B, C
CD	AB. BC, CD	DE, BK, AF	Φ	A, B, C, D
DE	AB, BC, CD, DE	EF, EH, BK, AF	Φ	A, B, C, D, E
EF	AB, BC, CD, DE, EF	FI, FG, EH, BK, AF	Φ	A, B, C, D, E, F
FI	AB, BC, CD, DE, EF, FI	FG, EH, BK, AF	Φ	A, B, C, D, E, F, I
FG	AB, BC, CD, DE, EF, FI, FG	GA, EH, BK, AF	Φ	A, B, C, D, E, F, I, G
GA	AB, BC, CD, DE, EF, FI, FG	EH, BK, AF	GA	A, B, C, D, E, F, I, G
EH	AB, BC, CD, DE, EF, FI, FG	HJ, HN, BK, AF	GA	A, B, C, D, E, F, I, G, H
HJ	AB, BC, CD, DE, EF, FI, FG, HJ	HN, BK, AF	GA	A, B, C, D, E, F, I, G, H, J
HN	AB, BC, CD, DE, EF, FI, FG, HI, HN	BK, AF	GA	A, B, C, D, E, F, I, G, H, N

SCnI în structuri arborescente
căutare în adâncime limitată (depth-limited search – DLS)

- **Aspecte teoretice**
  - DFS + adâncime maximă care limitează căutarea (d_{lim})
  - Se soluţionează problemele de completitudine ale căutării în adâncime (DFS)

- **Exemplu**
  - $d_{lim} = 2$
  - Ordinea vizitării nodurilor: A, B, E, F, C, G, D, H, I, J

- **Algoritm**

```
bool DLS(elem, list, dlim){
      found = false;
      visited = Φ;
      toVisit = {start}; //LIFO list
      while((toVisit != Φ) && (!found)){
            node = pop(toVisit);
            visited = visited U {node};
            if (node.depth <= dlim){
                  if (node == elem)
                     found = true;
                  else{
                     aux = Φ;
                     for all (unvisited) children of node do{
                              aux = aux U {child};
                     }
                     toVisit = aux U toVisit;
                  }//if found
            }//if dlim
      } //while
      return found;
}
```

Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	J
A, B, E, F, C, G, D, H, I, K, J	Φ

SCnI în structuri arborescente
căutare în adâncime limitată (depth-limited search – DLS)

- **Analiza complexităţii**
  - Complexitate temporală:
    - b – factor de ramificare (nr de noduri fii ale unui nod)
    - d^{lim} – limita lungimii (adâncimii) permisă pentru un arbore explorat
    - $T(n) = 1 + b + b^2 + \ldots + b^{dlim} \Rightarrow O(b^{dlim})$
  - Complexitate spaţială
    - $S(n) = b * d_{lim}$
  - Completitudine
    - Da, dar ⇔ $d_{lim} > d$, unde d = lungimea (adâncimea) soluţiei optime
  - Optimalitate
    - Nu → căutarea în adâncime poate găsi un drum soluţie mai lung decât drumul optim
- **Avantaje**
  - Se soluţionează problemele de completitudine ale căutării în adâncime (DFS)
- **Dezavantaje**
  - Cum se alege o limită d_{lim} bună?
- **Aplicaţii**
  - Determinarea "podurilor" într-un graf

SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)

- **Aspecte teoretice**
  - U DLS(d_{lim}), unde d_{lim} = 1, 2, 3, ..., d_{max}
  - Se soluţionează problema stabilirii limitei d_{lim} optime din DLS
  - De obicei, se aplică acolo unde:
    - spaţiul de căutare este mare şi
    - se cunoaşte lungimea (adâncimea) soluţiei

- **Exemplu**

- **Algoritm**

```
bool IDS(elem, list){
     found = false;
     dlim = 0;
     while ((!found) && (dlim < dmax)){
          found = DLS(elem, list, dlim);
          dlim++;
     }
     return found;
}
```

SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)

- **Analiza complexităţii**
  - Complexitate temporală:
    - Nodurile situate la adâncimea d_{max} (în număr de b^{dmax}) se expandează o singură dată=> 1 * b^{dmax}
    - Nodurile situate la adâncimea d_{max}-1 (în număr de b^{dmax-1}) se expandează de 2 ori => $2 * (b^{dmax-1})$
    - ...
    - Nodurile situate la adâncimea 1 (în număr de b) se expandează de d_{max} ori => $d_{max} * b^1$
    - Nodurile situate la adâncimea 0 (în număr de 1 - rădăcina) se expandează de d_{max}+1 ori => $(d_{max}+1)*b^0$

$$T(n) = \sum_{i=0}^{d_{\max}} (i+1) b^{d_{\max}-1} \Rightarrow O(b^{d_{\max}})$$

  - Complexitate spaţială
    - $S(n) = b * d_{max}$
  - Completitudine
    - Da
  - Optimalitate
    - Da

- **Avantaje**
  - Necesită memorie liniară
  - Asigură atingerea nodului ţintă urmând un drum de lungime minimă
  - Mai rapidă decât BFS şi DFS
- **Dezavantaje**
  - Necesită cunoaşterea "adâncimii" soluţiei
- **Aplicaţii**
  - Jocul Tic tac toe

SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)

SCnI în structuri arborescente
căutare bidirecțională (bi-directional search – BDS)

- **Aspecte teoretice**
  - 2 căutări simultane
    - Înainte (*forward*): de la rădăcină spre frunze
    - Înapoi (*backward*): de la frunze spre rădăcină

    care se opresc atunci când ajung la un nod comun
  - într-o direcţie pot fi folosite oricare dintre streategiile de căutare anterioare
  - necesită
    - stabilirea succesorilor, respectiv a predecesorilor unui nod
    - stabilirea locului de întâlnire
- **Exemplu**

- **Algoritm**
  - În funcţie de strategia de căutare folosită

SCnI în structuri arborescente
căutare bidirecțională (bi-directional search – BDS)

- **Analiza complexităţii**
  - Complexitate temporală
    - b – factor de ramificare (nr de noduri fii ale unui nod)
    - d - lungimea (adâncimea) soluţiei
    - $O(b^{d/2}) + O(b^{d/2}) \Rightarrow O(b^{d/2})$
  - Complexitate spaţială
    - $S(n) = T(n)$
  - Completitudine
    - da
  - Optimalitate
    - da

- **Avantaje**
  - Complexitate spaţială şi temporală redusă
- **Dezavantaje**
  - Dificultăţi în formularea problemei astfel încât fiecare stare să poată fi inversată
    - Parcurgere dinspre cap spre coadă
    - Parcurgere dinspre coadă spre cap
  - Implementare dificilă
  - Trebuie determinaţi succesorii şi predecesorii tuturor stărilor
  - Trebuie cunoscută starea finală (obiectiv)

- **Aplicaţii**
  - Problema partiţionării
  - Cel mai scurt drum

SCnI în structuri arborescente

SCnI în structuri arborescente

Compararea performanţelor

Metoda de căutare	Complexitate temporală	Complexitate spaţială	Completitudine	Optimalitate
BFS	$O(b^d)$	$O(b^d)$	Da	Da
UCS	$O(b^d)$	$O(b^d)$	Da	Da
DFS	$O(b^{dmax})$	$O(b*d_{max})$	Nu	Nu
DLS	$O(b^{dlim})$	$O(b*d_{lim})$	Da dacă $d_{lim} > d$	Nu
IDS	$O(b^d)$	$O(b*d)$	Da	Da
BDS	$O(b^{d/2})$	$O(b^{d/2})$	Da	Da

Rezolvarea problemelor prin căutare

- Strategii de căutare (SC)
  - Topologie
    - În funcţie de informaţia disponibilă
      - SC ne-informate (oarbe)
      - SC informate (euristice)

Strategii de căutare informate (SCI)

- Caracteristici
  - se bazează pe informaţii specifice problemei încercând să restrângă căutarea prin alegerea inteligentă a nodurilor care vor fi explorate
  - ordonarea nodurilor se face cu ajutorul unei funcţii (euristici) de evaluare
  - sunt particulare

- Topologie
  - Strategii globale
    - Best-first search
      - Greedy best-first search
      - A* + versiuni ale A*
  - Strategii locale
    - Căutare tabu
    - *Hill climbing*
    - *Simulated annealing*

SC în structuri arborescente

- Noţiuni necesare
  - *f(n)* – funcţie de evaluare pentru estimarea costului soluţiei prin nodul (starea) *n*
  - *h(n)* – funcţie euristică pentru estimarea costului drumului de la nodul *n* la nodul obiectiv
  - *g(n)* – funcţie de cost pentru estimarea costului drumului de la nodul de start până la nodul *n*
  - *f(n) = g(n) + h(n)*

SCI – Best first search

- **Aspecte teoretice**
  - Best first search = mai întâi cel mai bun
  - Se determină costul fiecărei stări cu ajutorul funcţiei de evaluare *f*
  - Nodurile de expandant sunt reţinute în structuri (cozi) ordonate
  - Pentru expandare se alege starea cu cea mai bună evaluare
    - Stabilirea nodului care urmează să fie expandat
  - Exemple de SC care depind de funcţia de evaluare
    - Căutare de cost uniform (SCnI)
      - *f* = costul drumului
    - În SCI se folosesc funcţii euristice
  - Două categorii de SCI de tip best first search
    - SCI care încearcă să expandeze nodul cel mai apropiat de starea obiectiv
    - SCI care încearcă să expandeze nodul din soluţia cu costul cel mai mic

- **Exemplu**
  - Detalii în slide-urile următoare

SCI – Best first search

- **Algoritm**

```
bool BestFS(elem, list){
     found = false;
     visited = Φ;
     toVisit = {start};   //FIFO sorted list (priority queue)
     while((toVisit != Φ) && (!found)){
          if (toVisit == Φ)
             return false
          node = pop(toVisit);
          visited = visited U {node};
          if (node == elem)
             found = true;
          else
             aux = Φ;
          for all unvisited children of node do{
             aux = aux U {child};
          }
          toVisit = toVisit U aux; //adding a node into the FIFO list based on its
                                   // evaluation (best one in the front of list)

     }  //while
     return found;
}
```

SCI – best first search

- **Analiza căutării**
  - Complexitate temporală:
    - *b* – factor de ramnificare (nr de noduri fii ale unui nod)
    - *d* - lungimea (adâncimea) maximă a soluţiei
    - $T(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Complexitate spaţială
    - $S(n) = T(n)$
  - Completitudine
    - Nu- drumuri infinite dacă euristica evaluează fiecare stare din drum ca fiind cea mai bună alegere
  - Optimalitate
    - Posibil – depinde de euristică
- **Avantaje**
  - Informaţiile despre domeniul problemei ajută căutarea (SCI)
  - Viteză mai mare de a ajunge la starea obiectiv
- **Dezavantaje**
  - Necesită evaluarea stărilor → efort computaţional, dar nu numai
  - Anumite path-uri pot "arăta" ca fiind bune conform funcţiei euristice
- **Aplicaţii**
  - *Web crawler (automatic indexer)*
  - Jocuri

SCI – Funcții euristice

- Etimologie: *heuriskein (gr)*
  - *a găsi, a descoperi*
  - *studiul metodelor şi regulilor de descoperire şi invenţie*

- Utilitate
  - Evaluarea potenţialului unei stări din spaţiul de căutare
  - Estimarea costului drumului (în arborele de căutare) din starea curentă până în starea finală (cât de aproape de ţintă a ajuns căutarea)

- Caracteristici
  - Depind de problema care trebuie rezolvată
  - Pentru probleme diferite trebuie proiectate sau învăţate diferite euristici
  - Se evaluează o anumită stare (nu operatorii care transformă o stare în altă stare)
  - Funcţii pozitive pentru orice stare *n*
    - $h(n) \geq 0$ pentru orice stare n
    - $h(n) = 0$ pentru starea finală
    - $h(n) = \infty$ pentru o stare din care începe un drum mort (o stare din care nu se poate ajunge în starea finală)

SCI – Funcții euristice

- Exemple
  - Problema misionarilor şi canibalilor
    - *h(n)* – nr. persoanelor aflate pe malul iniţial
  - 8-puzzle
    - *h(n)* – nr pieselor care nu se află la locul lor
    - *h(n)* – suma distanţelor Manhattan (la care se află fiecare piesă de poziţia ei finală)
  - Problema comisului voiajor
    - *h(n)* – cel mai apropiat vecin !!!
  - Plata unei sume folosind un număr minim de monezi
    - *h(n)* – alegerea celei mai valoroase monezi mai mică decât suma (rămasă) de plată

SCI - Greedy

- **Aspecte teoretice**
  - Funcţia de evalaure ***f(n) = h(n)***
    - estimarea costului drumului de la starea curentă la starea finală – *h(n)*
    - minimizarea costului drumului care mai trebuie parcurs

- **Exemplu**
  - A,D,E,H

- **Algoritm**

```
bool BestFS(elem, list){
       found = false;
       visited = Φ;
       toVisit = {start};          //FIFO sorted list (priority queue
       while((toVisit != Φ) && (!found)){
              if (toVisit == Φ)
                 return false
              node = pop(toVisit);
              visited = visited U {node};
              if (node == elem)
                 found = true;
              else
                 aux = Φ;
              for all unvisited children of node do{
                 aux = aux U {child};
              }
              toVisit = toVisit U aux; //adding a node onto the FIFO list based on its evaluation h(n)
                                       //(best one in the front of list)

       }  //while
       return found;
}
```

Vizitate deja	De vizitat
Φ	A
A	D, B, C
A, D	E, F, G, B, C
A, D, E	H, I, F, G, B, C
A, D, E, H	Φ

SCI - Greedy

- **Analiza căutării:**
  - Complexitate temporală → DFS
    - b – factor de ramnificare (nr de noduri fii ale unui nod)
    - d^{max} - lungimea (adâncimea) maximă a unui arbore explorat
    - $T(n) = 1 + b + b^2 + \ldots + b^{dmax} \Rightarrow O(b^{dmax})$
  - Complexitate spaţială → BFS
    - d - lungimea (adâncimea) soluţiei
    - $S(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Completitudine
    - Nu (există posibilitatea intrării în cicluri infinite)
  - Optimalitate
    - posibil

- **Avantaje**
  - Găsirea rapidă a unei soluţii (dar nu neapărat soluţia optimă), mai ales pentru probleme mici
- **Dezavantaje**
  - Suma alegerilor optime de la fiecare pas nu reprezintă alegerea globală optimă
    - Ex. Problema comisului voiajor

- **Aplicaţii**
  - Probleme de planificare
  - Probleme de sume parţiale
    - Plata unei sume folosind diferite tipuri de monezi
    - Problema rucsacului
  - Puzzle-uri
  - Drumul optim într-un graf

SCI – A*

- **Aspecte teoretice**
  - Combinarea aspectelor pozitive ale
    - căutării de cost uniform
      - optimalitate şi completitudine
      - utilizarea unei cozi ordonate
    - căutării Greedy
      - viteza mare
      - ordonare pe baza unei funcţii de evaluare
  - Funcţia de evalaure ***f(n)***
    - estimarea costului celui mai bun drum care trece prin nodul n
    - ***f(n) = g(n) + h(n)***
    - ***g(n)*** – funcţie folosită pentru stabilirea costului drumului de la starea iniţială până la starea curentă *n*
    - ***h(n)*** – funcţie euristică folosită pentru estimarea costului drumului de la starea curentă n la starea finală
  - Minimizarea costului total al unui drum

	o_1	o_2	o_3	o_4
p_i	10	18	32	14
w_i	1	2	4	3

- **Exemplu**
  - Problema rucsacului de capacitate *W* în care pot fi puse *n* obiecte $(o_1, o_2, \ldots, o_n)$ fiecare având o greutate w_i şi aducând un profit p_i, $i=1,2,\ldots,n$
    - Soluţia: pentru un rucsac cu W = 5 ➔ alegerea obiectelor o_1 şi o_3
  - $g(n) = \Sigma p_i$, pentru acele obiecte o_i care au fost selectate
  - $h(n) = \Sigma p_j$, pentru acele obiecte care nu au fost selectate şi $\Sigma w_j <= W - \Sigma w_i$
  - Fiecare nod din graf este un tuplu: *(p, w, p*, f)*, unde:
    - *p* – profitul adus de obiectele deja selectate (funcţia *g(n)*)
    - *w* – greutatea obiectelor selectate
    - *p** - profitul maxim care se poate obţine pornind din starea curentă şi ţinând cont de locul disponibil în rucsac (funcţia *h(n)*)

SCI – A*

- **Algoritm**

```
bool BestFS(elem, list){
     found = false;
     visited = Φ;
     toVisit = {start};  //FIFO sorted list (priority queue
     while((toVisit != Φ) && (!found)){
          if (toVisit == Φ)
            return false
          node = pop(toVisit);
          visited = visited U {node};
          if (node == elem)
            found = true;
          else
            aux = Φ;
          for all unvisited children of node do{
            aux = aux U {child};
          }
          toVisit = toVisit U aux; //adding a node onto the FIFO list
                                   // based on its evaluation f(n)= g(n)+ h(n)
                                   // (best one in the front of list)

     }  //while
     return found;
}
```

SCI – A*

- **Analiza căutării:**
  - Complexitate temporală:
    - b – factor de ramnificare (nr de noduri fii ale unui nod)
    - d^{max} - lungimea (adâncimea) maximă a unui arbore explorat
    - $T(n) = 1 + b + b^2 + \ldots + b^{dmax} \Rightarrow O(b^{dmax})$
  - Complexitate spaţială
    - d - lungimea (adâncimea) soluţiei
    - $T(n) = 1 + b + b^2 + \ldots + b^d \Rightarrow O(b^d)$
  - Completitudine
    - Da
  - Optimalitate
    - Da

- **Avantaje**
  - Algoritmul care expandează cele mai puţine noduri din arborele de căutare
- **Dezavantaje**
  - Utilizarea unei cantităţi mari de memorie

- **Aplicaţii**
  - Probleme de planificare
  - Probleme de sume parţiale
    - Plata unei sume folosind diferite tipuri de monezi
    - Problema rucsacului
  - Puzzle-uri
  - Drumul optim într-un graf

SCI – A*

- **Variante**
  - iterative deepening A* (IDA*)
  - memory-bounded A* (MA*)
  - simplified memory bounded A* (SMA*)
  - recursive best-first search (RBFS)
  - dynamic A* (DA*)
  - real time A*
  - hierarchical A*

- **Bibliografie suplimentară**
  - 02/A_IDA.pdf
  - 02/A_IDA_2.pdf
  - 02/SMA_RTA.pdf
  - 02/Recursive Best-First Search.ppt
  - 02/IDS.pdf
  - 02/IDA_MA.pdf
  - http://en.wikipedia.org/wiki/IDA*
  - http://en.wikipedia.org/wiki/SMA*

Rezolvarea problemelor prin căutare

- Tipologia strategiilor de căutare
  - În funcţie de modul de **generare** a soluţiei
    - Căutare **constructivă**
      - Construirea progresivă a soluţiei
      - Ex. TSP
    - Căutare **perturbativă**
      - O soluţie candidat este modificată în vederea obţinerii unei noi soluţii candidat
      - Ex. SAT - Propositional Satisfiability Problem
  - În funcţie de modul de **traversare** a spaţiului de căutare
    - Căutare **sistematică**
      - Traversarea completă a spaţiului
        - Ideintificarea soluţiei dacă ea există ➔ algoritmi compleţi
    - Căutare **locală**
      - Traversarea spaţiului de căutare dintr-un punct în alt punct vecin ➔ algoritmi incompleţi
      - O stare a spaţiului poate fi vizitată de mai multe ori
  - În funcţie de elementele de **certitudine** ale căutării
    - Căutare **deterministă**
      - Algoritmi de identificare exactă a soluţiei
    - Căutare **stocastică**
      - Algoritmi de aproximare a soluţiei
  - În funcţie de stilul de **explorare** a spaţiului de căutare
    - Căutare **secvenţială**
    - Căutare **paralelă**

Rezolvarea problemelor prin căutare

Poate consta în:

- Construirea progresivă a soluţiei
- Identificarea soluţiei potenţiale optime
  - Componentele problemei
    - Condiţii (constrângeri) pe care trebuie să le satisfacă (parţial sau total) soluţia
    - Funcţie de evaluare a unei soluţii potenţiale → identificareaa optimului
  - Spaţiul de căutare
    - mulţimea tuturor soluţiilor potenţiale complete
    - Stare = o soluţie completă
    - Stare finală (scop) → soluţia optimă
  - Exemple
    - Problema celor 8 regine,
      - Stările posibile: configuraţii ale tablei de sah cu câte 8 regine
      - Operatori: modificarea coloanei în care a fost plasată una din regine
      - Scopul căutării: configuraţia în care nu existe atacuri între regine
      - Funcţia de evaluare: numărul de atacuri
    - probleme de planificare,
    - proiectarea circuitelor digitale, etc

Rezolvarea problemelor prin căutare

Poate consta în:

- Construirea progresivă a soluţiei
- Identificarea soluţiei potenţiale optime
  - Algoritmi
    - Algoritmii discutaţi până acum explorau în mod **sistematic** spaţiul de căutare
      - De ex. A* ➔ 10^{100} stări ≈ 500 variabile binare
    - Problemele reale pot avea 10 000 – 100 000 variabile ➔ nevoia unei alte categorii de algoritmi care explorează **local** spaţiul de căutare (algoritmi iterativi)
    - Ideea de bază:
      - se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluţie optimă şi
      - se efectuează modificări pentru a elimina aceste violări (se deplasează căutarea într-o soluţie vecină cu soluţia curentă) astfel încât căutarea să se îndrepte spre soluţia optimă
    - Iterativi → se memorează o singură stare şi se încearcă îmbunătăţirea ei
      - versiunea inteligentă a algoritmului de forţă brută
    - Istoricul căutării nu este reţinut

```
bool LS(elem, list){
    found = false;
    crtState = initState
    while ((!found) && timeLimitIsNotExceeded){
       toVisit = neighbours(crtState)
       if (best(toVisit) is better than crtState)
       crtState = best(toVisit)
      if (crtState == elem)
       found = true;
    }  //while
    return found;
}
```

Rezolvarea problemelor prin căutare

Poate consta în:

- Construirea progresivă a soluţiei
- Identificarea soluţiei potenţiale optime
  - Avantaje
    - Simplu de implementat
    - Necesită puţină memorie
    - Poate găsi soluţii rezonabile în spaţii de căutare (continue) foarte mari pentru care alţi algoritmi sistematici nu pot fi aplicaţi
  - E utilă atunci când:
    - Se pot genera soluţii complete rezonabile
    - Se poate alege un bun punct de start
    - Există operatori pentru modificarea unei soluţii complete
    - Există o măsură pentru a aprecia progresul (avansarea căutării)
    - Există un mod de a evalua soluţia completă (în termeni de constrângeri violate)

Strategii de căutare locală

- Tipologie
  - Căutare locală simplă - se reţine o singură stare vecină
    - Hill climbing → alege cel mai bun vecin
    - Simulated annealing → alege probabilistic cel mai bun vecin
    - Căutare tabu → reţine lista soluţiilor recent vizitate
  - Căutare locală în fascicol (*beam local search*) – se reţin mai multe stări (o populaţie de stări)
    - Algoritmi evolutivi
    - Optimizare bazată pe comportamentul de grup (*Particle swarm optimisation*)
    - Optimizare bazată pe furnici (*Ant colony optmisation*)

Strategii de căutare locală

- Căutare locală simplă
  - elemente de interes special:
    - Reprezentarea soluţiei
    - Funcţia de evaluare a unei potenţiale soluţii
    - Vecinătatea unei soluţii
      - Cum se defineşte/generează o soluţie vecină
      - Cum are loc căutarea soluţiilor vecine
        - Aleator
        - Sistematic

    } dependente de problemă

    - Criteriul de acceptare a unei noi soluţii
      - Primul vecin mai bun decât soluţia curentă
      - Cel mai bun vecin al soluţiei curente mai bun decât soluţia curentă
      - Cel mai bun vecin al soluţiei curente mai slab decât soluţia curentă
      - Un vecin aleator

Strategii de căutare locală – Hill climbing (HC)

- **Aspecte teoretice**
  - Urcarea unui munte în condiţii de ceaţă şi amnezie a excursionistului :D
  - Mişcarea continuă spre valori mai bune (mai mari ➔ urcuşul pe munte)
  - Căutarea avansează în direcţia îmbunătăţirii valorii stărilor succesor până când se atinge un optim
  - Criteriul de acceptare a unei noi soluţii
    - cel mai bun vecin al soluţiei curente mai bun decât soluţia curentă

  - Îmbunătăţire prin
    - Maximizarea calităţii unei stări ➔ *steepest ascent HC*
    - Minimizarea costului unei stări ➔ *gradient descent HC*
  - HC $\neq$ *steepest ascent/gradient descent* (SA/GD)
    - HC optimizează f(x) cu $x \in R^n$ prin modificarea unui element al lui x
    - SA/GD optimizează f(x) cu $x \in R^n$ prin modificarea tuturor elementelor lui x

Strategii de căutare locală – Hill climbing (HC)

- **Exemplu**
  - Construirea unor turnuri din diferite forme geometrice
    - Se dau *n* piese de formă dreptunghiulară (de aceeaşi lăţime, dar de lungimi diferite) aşezate unele peste altele formând un turn (stivă). Să se re-aşeze piesele astfel încât să se formeze un nou turn ştiind că la o mutare se poate mişca doar o piesă din vârful stivei, piesă care poate fi mutată pe una din cele 2 stive ajutătoare.

    - Reprezentarea soluţiei
      - Stare *x* – vectori de *n* perechi de forma *(i,j)*, unde *i* reprezintă indexul piesei (*i=1,2,...,n*), iar *j* indexul stivei pe care se află piesa (*j=1,2,3*)
      - Starea iniţială – vectorul corespunzător turnului iniţial
      - Starea finală – vectorul corespunzător turnului final
    - Funcţia de evaluare a unei stări
      - *f1* = numărul pieselor corect amplasate → maximizare
        - conform turnului final – *f1 = n*
      - *f2* = numărul pieselor greşit amplasate → minimizare
        - conform turnului final – *f2 = 0*
      - *f = f1 – f2* → maximizare
    - Vecinătate
      - Mutări posibile
        - Mutarea unei piese *i* din vârful unei stive *j1* pe altă stivă *j2*
    - Criteriul de acceptare a unei noi soluţii
      - Cel mai bun vecin al soluţiei curente mai bun decât soluţia curentă

Strategii de căutare locală – Hill climbing (HC)

- **Exemplu**
  - Iteraţia 1
    - Starea curentă x = starea iniţială:
      - $x = s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$
      - Piesele 1, 2 şi 3 sunt corect aşezate
      - Piesele 4 şi 5 nu sunt corect aşezate
      - $f(s_1) = 3 - 2 = 1$

    - $x^* = x$
    - Vecinii stării curente x – un singur vecin → piesa 5 se mută pe stiva 2 ➔
      - $s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$
      - $f(s_2) = 4-1=3 > f(x)$ ➔ $x = s_2$

Strategii de căutare locală – Hill climbing (HC)

- **Exemplu**
  - Iteraţia 2
    - Starea curentă x = ((1,1), (2,1), (3,1), (4,1), (5,2))
      - f(x) = 3
    - Vecinii stării curente – doi vecini:
      - piesa 1 se mută pe stiva 2 ➔ s_3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) ➔ $f(s_3)$ = 3-2=1 < f(x)

      - piesa 1 se mută pe stiva 3 ➔ s_4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) ➔ $f(s_4)$ = 3-2=1 < f(x)

      - nu există vecin de-al lui x mai bun ca x ➔ stop
  - x* = x = ((1,1), (2,1), (3,1), (4,1), (5,2))
  - Dar x* este doar optim local, nu global

Strategii de căutare locală – Hill climbing (HC)

- **Exemplu**
  - Construirea unor turnuri din diferite forme geometrice
    - Funcţia de evaluare a unei stări
      - f1 = suma înălţimilor stivelor pe care sunt amplasate corect piese (conform turnului final – f1 = 10) → maximizare
      - f2 = suma înălţimilor stivelor pe care sunt amplasate incorect piese (conform turnului final – f2 = 0) → minimizare
      - f = f1 – f2 → maximizare
    - Vecinătate
      - Mutări posibile
        - Mutarea unei piese i din vârful unei stive j1 pe altă stivă j2

Strategii de căutare locală – Hill climbing (HC)

- **Exemplu**
  - Iteraţia 1
    - Starea curentă x = starea iniţială s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))
      - Toate piesele nu sunt corect aşezate ➔ f1 = 0, f2 = 3+2 + 1 + 0 + 4 = 10
      - $f(s_1) = 0 - 10 = -10$

    - x* = x
    - Vecinii stării curente x– un singur vecin → piesa 5 se mută pe stiva 2 ➔ s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))

      - $f(s_2) = 0 - (3+2+1+0) = -6 > f(x)$ ➔ $x = s_2$

Strategii de căutare locală – Hill climbing (HC)

- **Exemplu**
  - Iteraţia 2
    - Starea curentă x = ((1,1), (2,1), (3,1), (4,1), (5,2))
      - f(x) = -6
    - Vecinii stării curente – doi vecini:
      - piesa 1 se mută pe stiva 2 ➔ s3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) ➔ f(s3) = 0 – (0+2+3+0)=-5 > f(x)

      - piesa 1 se mută pe stiva 3 ➔ s4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) ➔ f(s4) = 0 - (1+2+1) = -4 > f(x)

      - cel mai bun vecin al lui x este s4 ➔ x = s4
  - Iteraţia 3
    - ...
  - Se evită astfel blocarea în optimele locale

Strategii de căutare locală – Hill climbing (HC)

- **Algoritm**

```
Bool HC(S){
   x = s1    //starea iniţială
   x*=x      // cea mai bună soluţie găsită
  (până la un moment dat)
   k = 0     // numarul de iteraţii
   while (not termiantion criteria){
      k = k + 1
      generate all neighbours of x (N)
      Choose the best solution s from N
      if f(s) is better than f(x) then x = s
      else stop
   }  //while
   x* = x
   return x*;
}
```

Strategii de căutare locală – Hill climbing (HC)

- **Analiza căutării**
  - Convergenţa spre optimul local
- **Avantaje**
  - Simplu de implementat → se poate folosi usor pentru a aproxima soluţia unei probleme când soluţia exactă este dificil sau imposibil de găsit
    - Ex. TSP cu forate multe oraşe
  - Nu necesită memorie (nu se revine în starea precedentă)
- **Dezavantaje**
  - Funcţia de evaluare (eurisitcă) poate fi dificil de estimat
  - Dacă se execută foarte multe mutări algoritmul devine ineficient
  - Dacă se execută prea puţine mutări algoritmul se poate bloca
    - Într-un optim local (nu mai poate "coborî" din vârf)
    - Pe un platou – zonă din spaţiul de căutare în care funcţia de evaluare este constantă
    - Pe o creastă – saltul cu mai mulţi paşi ar putea ajuta căutarea
- **Aplicaţii**
  - Problema canibalilor
  - 8-puzzle, 15-puzzle
  - TSP
  - Problema damelor

Strategii de căutare locală – Hill climbing (HC)

- **Variante**
  - HC stocastic
    - Alegerea aleatoare a unui succesor
  - HC cu prima opţiune
    - Generarea aleatoare a succesorilor până la întâlnirea unei mutări neefectuate
  - HC cu repornire aleatoare → *beam local search*
    - Repornirea căutării de la o stare iniţială aleatoare atunci când căutarea nu progresează

Strategii de căutare locală – Simulated Annealing

- **Aspecte teoretice**
  - Inspirată de modelarea proceselor fizice
    - Metropolis et al. 1953, Kirkpatrick et al. 1982;
  - Succesorii stării curente sunt aleşi şi în mod aleator
    - Dacă un succesor este mai bun decât starea curentă
      - atunci el devine noua stare curentă
      - altfel succesorul este reţinut doar cu o anumită probabilitate
  - Se permit efectuarea unor mutări "slabe" cu o anumită probabilitate p
    - ➔ evadarea din optimele locale
  - Probabilitatea p = $e^{\Delta E/T}$
    - Proporţională cu valoarea diferenţei (energia) ΔE
    - Modelată de un parametru de temperatură T
  - Frecvenţa acestor mutări "slabe" şi mărimea lor se reduce gradual prin scăderea temperaturii
    - T = 0 ➔ hill climbing
    - T → ∞ ➔ mutările "slabe" sunt tot mai mult executate
  - Soluţia optimă se identifică dacă temperatura se scade treptat ("slowly")
  - Criteriul de acceptare a unei noi soluţii
    - Un vecin aleator mai bun sau mai slab (cu probabilitatea p) decât soluţia curentă

Strategii de căutare locală – Simulated Annealing

- **Exemplu** – Problema celor 8 regine
  - Enunţ
    - Să se amplaseze pe o tablă de şah 8 regine astfel încât ele să nu se atace reciproc
  - Reprezentarea soluţiei
    - Stare *x* – permutare de n elemente x = (x_1,x_2,..,x_n), unde xi – linia pe care este plasată regina de pe coloana j
      - Nu există atacuri pe verticală sau pe orizontală
      - Pot exista atacuri pe diagonală
    - Starea iniţială – o permutare oarecare
    - Starea finală – o permutare fără atacuri de nici un fel
  - Funcţia de evaluare a unei stări
    - f – suma reginelor atacate de fiecare regină → minimizare
  - Vecinătate
    - Mutări posibile
      - Mutarea unei regine de pe o linie pe alta (interschimbarea a 2 elemente din permutare)
  - Criteriul de acceptare a unei noi soluţii
    - Un vecin oarecare al soluţiei curente
      - mai bun decât soluţia curentă
      - mai slab decât soluţia curentă – cu o anumită probabilitate $P(\Delta E) = e^{-\frac{\Delta E}{T}}$ unde:
        - ΔE – diferenţa de energie (evaluare) între cele 2 stări (vecină şi curentă)
        - T – temperatura, T(k) = 100/k, unde k este nr iteraţiei

Strategii de căutare locală – Simulated Annealing

- **Exemplu** – Problema celor 8 regine
  - Iteraţia 1 (k = 1)
    - Starea curentă x = starea iniţială

      s_1 = (8,5,3,1,6,7,2,4)
      - $f(s_1)$ = 1+1 = 2

    - x* = x
    - T = 100/1 = 100

    - Un vecin al stării curente x → regina de pe linia 5 se interschimbă cu regina de pe linia 7 ➔ s_2 = (8,7,3,1,6,5,2,4)
      - $f(s_2)$ = 1+1+1=3 > f(x)
      - $\Delta E = f(s_2) - f(s_1) = 1$
      - $P(\Delta E)=e^{-1/100}$
      - r = random(0,1)
      - Dacă r < $P(\Delta E)$ ➔ x = s_2

Strategii de căutare locală – Simulated Annealing

- **Algoritm**

```
bool SA(S){
     x = s1          //starea iniţială
     x*=x  // cea mai bună soluţie găsită (până la un moment dat)
     k = 0 // numarul de iteraţii
     while (not termiantion criteria){
         k = k + 1
         generate a neighbour s of x
         if f(s) is better than f(x) then x = s
         else
           pick a random number r (in (0,1) range)
           if r < P(ΔE) then x = s
     }  //while
     x* = x
     return x*;
}
```

- Criterii de oprire
  - S-a ajuns la soluţie
  - S-a parcurs un anumit număr de iterţii
  - S-a ajuns la temepratura 0 (îngheţ)
- Cum se alege o probabilitate mică?
  - p = 0.1
  - p scade de-a lungul iteraţiilor
  - p scade de-a lungul iteraţiilor şi pe măsură ce "răul" |f(s) – f(x)| creşte
    - p = exp(- |f(s) – f(x)|/T)
    - Unde T – temepratura (care creşte)
      - Pentru o T mare se admite aproape orice vecin v
      - Petnru o T mică se admite doar un vecin mai bun decât s
    - Dacă "răul" e mare, atunci probabilitatea e mică

Strategii de căutare locală – Simulated Annealing

- **Analiza căutării**
  - Convergenţa (complet, optimal) lentă spre optimul global

- **Avantaje**
  - Algoritm fundamentat statistic ➔ garantează găsirea soluţiei optime, dar necesită multe iteraţii
  - Uşor de implementat
  - În general găseşte o soluţie relativ bună (optim global)
  - Poate rezolva probleme complexe (cu zgomot şi multe constrângeri)
- **Dezavantaje**
  - Algoritm încet – convergenţa la soluţie durează foarte mult timp
    - Compromis între calitatea soluţiei şi timpul necesar calculării ei
  - Depinde de anumiţi parametri (temperatura) care trebuie reglaţi
  - Nu se ştie dacă soluţia oferită este optimă (local sau gloabal)
  - Calitatea soluţiei găsite depinde de precizia variabilelor implicate în algoritm

- **Aplicaţii**
  - Probleme de optimizare combinatorială → problema rucsacului
  - Probleme de proiectare → Proiectarea circuitelor digitale
  - Probleme de planificare → Planificarea producţiei, planificarea meciurilor în turniruirle de tenis US Open

Strategii de căutare locală – Căutare tabu

- **Aspecte teoretice**
  - "Tabu" → interdicţie socială severă cu privire la activităţile umane sacre şi interzise
  - Propusă în anii 1970 de către F. Glover

  - Ideea de bază
    - se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluţie optimă şi
    - se efectuează modificări pentru a elimina aceste violări (se deplasează căutarea în cea mai bună soluţie vecină cu soluţia curentă) astfel încât căutarea să se îndrepte spre soluţia optimă
    - se memorează
      - Starea curentă
      - Stările vizitate până la momentul curent al căutării şi mutările efectuate pentru a ajunge în fiecare din acele stări de-a lungul căutării (se memorează o listă limitată de stări care nu mai trebuie revizitate)
  - Criteriul de acceptare a unei noi soluţii
    - Cel mai bun vecin al soluţiei curente mai bun decât soluţia curentă şi nevizitat încă

  - 2 elemente importante
    - Mutări tabu (T) – determinate de un proces non-Markov care se foloseşte de informaţiile obţinute în timpul căutării de-a lungul ultimelor generaţii
    - Condiţii tabu – pot fi inegalităţi liniare sau legături logice exprimate în funcţie de soluţia curentă
      - Au rol în alegerea mutărilor tabu

Strategii de căutare locală – Căutare tabu

- **Exemplu**
  - Enunţ
    - Plata sumei S folosind cât mai multe dintre cele n monezi de valori v_i (din fiecare monedă există b_i bucăţi)
  - Reprezentarea soluţiei
    - Stare x – vector de n întregi $x = (x_1, x_2, ..., x_n)$ cu $xi \in \{0, 1, 2, ..., b_i\}$
    - Starea iniţială – aleator
  - Funcţia de evaluare a unei stări
    - f_1 = Diferenţa între S şi valaorea monezilor alese → minimă
      - Dacă valoarea monezilor depăşeşte S ➔ penalizare de 500 unităţi
    - f_2 = Numărul monezilor selectate → maxim
    - $f = f_1 - f_2$ → minimizare
  - Vecinătate
    - Mutări posibile
      - Includerea în sumă a monezii i în j exemplare ($plus_{i,j}$)
      - Excluderea din sumă a monezii i în j exemplare ($minus_{i,j}$)
    - Lista tabu reţine mutările efectuate într-o iteraţie
      - Mutare – moneda adăugată/eliminată din sumă

Strategii de căutare locală – Căutare tabu

- **Exemplu**
  - S = 500, penalizare = 500, n = 7

S=500	m_1	m_2	m_3	m_4	m_5	m_6	m_7
v_i	10	50	15	20	100	35	5
b_i	5	2	6	5	5	3	10

Stare curentă	Val. f	Listă tabu	Stări vecine	Mutări	Val. f
2 0 1 0 0 2 1	384	∅	2 0 1 3 0 2 1	$plus_{4,3}$	321
			2 0 1 0 0 3 1	$plus_{6,1}$	348
			0 0 1 0 0 2 1	$minus_{1,2}$	406
2 0 1 3 0 2 1	321	$plus_{4,3}$	2 0 1 3 5 2 1	$plus_{5,5}$	316
			2 0 1 1 0 2 1	$minus_{4,2}$	363
			2 1 1 3 0 2 1	$plus_{2,1}$	270
2 1 1 3 0 2 1	270	$plus_{4,3}$ $plus_{2,1}$	...		

  - Soluţia finală: 4 1 5 4 1 3 10 (f = -28)

Strategii de căutare locală – Căutare tabu

- **Exemplu**
  - S = 500, penalizare = 500, n = 7

S=50	m_1	m_2	m_3	m_4	m_5	m_6	m_7
v_i	10	50	15	20	100	35	5
b_i	5	2	6	5	4	3	10

Stare curentă	Val. f	Listă tabu	Stări vecine	Mutări	Val. f
2 0 1 0 0 2 1	384	∅	1 0 1 4 0 2 1	$minus_{1,1}$,$plus_{4,4}$	311
			2 0 4 0 1 2 1	$plus_{3,3}$,$minus_{5,1}$	235
			2 0 1 0 4 2 6	$plus_{5,4}$, $plus_{7,5}$	450
2 0 4 0 1 2 1	235	$plus_{3,3}$, $minus_{5,1}$	2 0 5 0 5 2 1	$plus_{3,1}$, $plus_{5,4}$	315
			5 0 4 0 4 2 1	$plus_{1,3}$, $plus_{5,3}$	399
			2 2 4 0 5 2 1	$plus_{2,2}$, $plus_{5,4}$	739
2 0 4 0 1 2 1	235	$plus_{3,3}$, $minus_{5,1}$	...		

  - Soluţia finală: 4 1 5 4 1 3 10 (f = -28)

Strategii de căutare locală – Căutare tabu

- **Algoritm**

```
bool TS(S){
    Select x∈S   //S - spaţiul de căutare
    x*=x         //cea mai bună soluţie (până la un mom. dat)
    k = 0        // numarul de iteraţii
    T = ∅        // listă de mutări tabu
    while (not termiantion criteria){
       k = k + 1
       generate a subset of solutions in the neighborhood N-T of x
       choose the best solution s from N-T and set x=s.
       if f(x)<f(x*) then x*=x
       update T with moves of generating x
    }  //while
    return x*;
}
```

- Criterii de terminare
  - Număr fix de iteraţii
  - Număr de iteraţii fără îmbunătăţiri
  - Apropierea suficientă de soluţie (dacă aceasta este cunoscută)
  - Epuizarea elementelor nevizitate dintr-o vecinătate

Strategii de căutare locală – Căutare tabu

- **Analiza căutării**
  - Convergenţa rapidă spre optimul global
- **Avantaje**
  - Algoritm general şi simplu de implementat
  - Algoritm rapid (poate oferi soluţia optimă globală în scurt timp)
- **Dezavantaje**
  - Stabilirea stărilor vecine în spaţii continue
  - Număr mare de iteraţii
  - Nu se garantează atingerea optimului global

Strategii de căutare locală – Căutare tabu

- **Aplicaţii**
  - Determinarea structurii tridimensionale a proteinelor în secvenţe de aminoacizi (optimizarea unei funcţii de potenţial energetic cu multiple optime locale)
  - Optimizarea traficului în reţele de telecomunicaţii
  - Planificare în sisteme de producţie
  - Proiectarea reţelelor de telecomunicaţii optice
  - Ghidaj automat pentru vehicule
  - Probleme în grafuri (partiţionări)
  - Planificări în sistemele de audit
  - Planificări ale task-urilor paralele efectuate de procesor (multiprocesor)
  - Optimizarea structurii electromagnetice (imagistica rezonanţei magnetice medicale)
  - Probleme de asignare quadratică (proiectare VLSI)
  - Probleme de combinatorică (ricsac, plata sumei)
  - Problema tăierii unei bucăţi în mai multe părţi
  - Controlul structurilor spaţiale (NASA)
  - Optimizarea proceselor cu decizii multi-stagiu
  - Probleme de transport
  - Management de portofoliu
  - Chunking

Recapitulare

- SCI best first search
  - Nodurile mai bine evaluate (de cost mai mic) au prioritate la expandare
  - SCI de tip greedy
    - minimizarea costului de la starea curentă la starea obiectiv – h(n)
    - Timp de căutare < SCnI
    - Ne-completă
    - Ne-optimală
  - SCI de tip A*
    - minimizarea costului de la starea iniţială la starea curentă – g(n) – şi a costului de la starea curentă la starea obiectiv – h(n)
    - Evitarea repetării stărilor
    - Fără supraestimarea lui h(n)
    - Timp şi spaţiu de căutare mare → în funcţie de euristica folosită
    - Complet
    - Optimal

Recapitulare

- SC locale
  - Algoritmi iterativi
    - Lucrează cu o soluţie potenţială → soluţia optimă
    - Se pot bloca în optime locale

	Alegerea stării următoare	Criteriul de acceptare	Convergenţa
HC	Cel mai bun vecin	Vecinul este mai bun decât strarea curentă	Optim local sau gloabl
SA	Un vecin oarecare	Vecinul este mai bun sau mai slab (acceptat cu probabilitatea p) decât starea curentă	Optim global (lentă)
TS	Cel mai bun vecin nevizitat încă	Vecinul este mai bun decât strarea curentă	Optim global (rapidă)

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- capitolul 14 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- Informaţiile prezentate au fost colectate din diferite surse bibliografice, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – https://mihaioltean.github.io/
  - Lect. Dr. Crina Groşan - http://people.brunel.ac.uk/~cssrcdg/
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare informată locală

Algoritmi Evolutivi

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Sumar

- Rezolvarea problemelor prin căutare
  - Strategii de căutare informate (euristice) – SCI
    - Strategii locale
      - Algoritmi evolutivi

Materiale de citit şi legături utile

- capitolul 14 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Căutare locală

- Tipologie
  - Căutare locală simplă - se reţine o singură stare vecină
    - Hill climbing → alege cel mai bun vecin
    - Simulated annealing → alege probabilistic cel mai bun vecin
    - Căutare tabu → reţine lista soluţiilor recent vizitate
  - Căutare locală în fascicol (beam local search) – se reţin mai multe stări (o populaţie de stări)
    - Algoritmi evolutivi
    - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
    - Optimizare bazată pe furnici (Ant colony optmisation)

Algoritmi inspirați de natură

- Care este cea mai bună metodă de rezolvare a unei probleme?
  - Creierul uman
    - a creat roata, maşina, oraşul, etc
  - Mecanismul evoluţiei
    - a creat creierul (mintea) umană
- Simularea naturii
  - Cu ajutorul maşinilor → reţelele neuronale artificiale simulează mintea umană
    - maşini de zbor, computere bazate pe ADN, computere cu membrane
  - Cu ajutorul algoritmilor
    - algoritmii evolutivi simulează evoluţia naturii
    - algoritmii inspiraţi de comportamentul de grup simulează adaptarea colectivă şi procesele sociale dintr-un colectiv (*Particle Swarm Optimisation*)
    - algoritmii inspiraţi de furnici (*Ant Colony Optimisation*)

Algoritmi evolutivi – aspecte teoretice

- Simularea naturii
  - Zborul liliecilor

  - Leonardo da Vinci – schema unei maşini de zbor

  - Zborul păsărilor şi al avioanelor

  - Zborul păsărilor şi turbinele eoliene

Algoritmi evolutivi – aspecte teoretice

- Care sunt caracteristicile de bază ale AE?
  - Implică procese iterative şi paralele
  - Folosesc populaţii de potenţiale soluţii
  - Se bazează pe o căutare aleatoare
  - Sunt inspiraţi de biologie – implică mecanisme precum:
    - selecţia naturală
    - reproducerea
    - recombinarea
    - mutaţia

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- Jean Baptise de Lamark (1744-1829)
  - A propus în 1809 o explicaţie pentru originea speciilor în cartea *Zoological Philosophy*:
    - Nevoile unui organism determină caracteristicile care evoluează
    - Caracteristicile utile dobândite în cursul vieţii unui organism se pot transfera urmaşilor acestuia
  - Legea utilizării şi neutilizării
    - *use and disuse*

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- Charles Darwin (1807-1882)
  - În cartea *Origin of Species* demostrează că toate organismele au evoluat din alte organisme pe baza:

    - variaţiei
      - supraproducţia de descendenţi
    - selecţiei naturale
      - competiţia (generaţii constante ca dimensiune)

      - supravieţuirea pe baza calităţii/adaptării la mediul de viaţă (fitness)
      - reproducerea
      - apariţia de specii noi

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- Teoria evolutivă modernă
  - Îmbogăţeşte teoria Darwiniană cu mecanismul moştenirii genetice
  - Variaţia genetică se produce prin:
    - mutaţie – spontană – şi
    - reproducere sexuală
  - L. Fogel 1962 (San Diego, CA) → programare evolutivă – PE – (*Evolutionary Programming)*
  - J. Holland 1962 (Ann Arbor, MI) → algoritmi genetici – AG – (*Genetic Algorithms*)
  - I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany)→ strategii evolutive – SE – (*Evolution Strategies)*
  - J. Koza 1989 (Palo Alto, CA) → programare genetică – PG – (*Genetic Programming*)

Algoritmi evolutivi – aspecte teoretice

- Metafora evolutivă

Evoluţia naturală		Rezolvarea problemelor
Individ	↔	Soluţie potenţială
Populaţie	↔	Mulţime de soluţii potenţiale
Cromozom	↔	Codarea unei soluţii potenţiale
Genă	↔	Parte a codării
Fitness	↔	Calitate
Încrucişare şi mutaţie	↔	Operatori de căutare
Mediu	↔	Problemă

Algoritmi evolutivi - algoritm

- Schema generală
- Proiectare

Algoritmi evolutivi – algoritm

- Schema generală a unui AE
  - Generaţional
  - Steady-state

Algoritmi evolutivi – algoritm

- Proiectare
  - Alegerea unei reprezentări a cromozomilor
  - Alegerea unui model de populaţie
  - Stabilirea unei funcţii de evaluare
  - Stabilirea operatorilor genetici
    - Selecţie
    - Mutaţie
    - Recombinare
  - Stabilirea unui criteriu de stop

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- 2 nivele de existenţă pentru o soluţie candidat
  - Nivel exterior → fenotip
    - Individ - obiectul original în contextul dat de problemă
    - Aici are loc evaluarea unei potenţiale soluţii
    - Furnică, rucsac, elefant, oraşe, ...

  - Nivel interior → genotip
    - Cromozom – codul asociat unui obiect
      - format din gene, poziţionate în locuri (fixe) – loci – şi având anumite valori – alele
    - Aici are loc căutarea unei noi potenţiale soluţii
    - Vector unidimensional (numeric, boolean, string), matrice, ...

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentarea trebuie să fie relevantă pentru:
  - problemă,
  - funcţia de evaluare şi
  - operatorii genetici

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

Tipologia reprezentării cromozomilor

- Liniară
  - Discretă
    - Binară → problema rucsacului
    - Ne-binară
      - Întreagă
        - Oarecare → procesarea imaginilor
        - Permutări → problema comisului voiajor
      - Categorială → problema colorării hărţilor
  - Continuă (reală) → optimizări de funcţii
- Arborescentă → probleme de regresie

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi
  - Fenotip
    - Elemente de tip Boolean
      - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului

Genotip

1	0	1	0	0	0	1	1

Fenotip

$ob_1+ob_3+ob_7+ob_8$

Au fost alese obiectele 1, 3, 7 şi 8

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi
  - Fenotip
    - Elemente de tip Boolean
      - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului
    - Numere întregi

Genotip

1	0	1	0	0	0	1	1

Fenotip

= 163

$1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 =$

$128 + 32 + 2 + 1 = 163$

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi
  - Fenotip
    - Elemente de tip Boolean
      - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului
    - Numere întregi
    - Numere reale într-un anumit Interval (ex. [2.5, 20.5])

Genotip Fenotip

1	0	1	0	0	0	1	1

= 13.9609

$$x = 2.5 + \frac{163}{256}(20.5 - 2.5) = 13.9609$$

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

Transformarea valorilor reale reprezentate pe biţi

- Fie $z \in [x,y] \subseteq \mathcal{R}$ reprezentat ca $\{a_1,\ldots,a_L\} \in \{0,1\}^L$
- Funcţia $[x,y] \rightarrow \{0,1\}^L$ trebuie să fie inversabilă (un fenotip corespunde unui genotip)
- Funcţia Γ: $\{0,1\}^L \rightarrow [x,y]$ defineşte reprezentarea

$$\Gamma(a_1,\ldots,a_L) = x + \frac{y-x}{2^L-1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j\right) \in [x,y]$$

- Observaţii
  - Se pot reprezenta doar 2^L valori
  - L indică precizia maximă a soluţiei
  - Pentru o precizie cât mai bună → cromozomi lungi ➔ evoluţie încetinită

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă oarecare
  - Genotip
    - şir de numere întregi dintr-un anumit interval
  - Fenotip
    - Utilitatea numerelor în problemă
  - Ex. Problema plăţii unei sume folosind diferite monezi
    - Genotip → şir de nr întregi de lungime egală cu numărul de monezi diferite, fiecare număr din intervalul [0,suma/valoarea monezii curente]
    - Fenotip → câte monezi din fiecare tip trebuie considerate

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă de tip permutare
  - Genotip
    - Permutare de *n* numere (*n* – numărul de gene)
  - Fenotip
    - Utilitatea permutării în problemă
  - Ex. Problema comisului voiajor
    - Genotip → permutare de *n* elemente
    - Fenotip → ordinea de vizitare a oraşelor, ştiind că fiecărui oraş îi corespunde un număr din mulţimea {1,2,...,*n*}

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară categorială
  - Similară cu cea întreagă, dar în loc de numere se folosesc etichete
  - Genotip
    - şir de etichete dintr-o anumită mulţime
  - Fenotip
    - Interpretarea etichetelor
  - Ex. Problema colorării hărţilor
    - Genotip → şir de etichete (culori) de lungime egală cu numărul de ţări, fiecare etichetă aparţinând unei mulţimi de culori date
    - Fenotip → cu ce culoare trebuie haşurată fiecare hartă a unei ţări

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară continuă (reală)
  - Genotip
    - Şir de numere reale
  - Fenotip
    - Utilitatea numerelor în problemă
  - Ex. Problema optimizării funcţiilor $f: R^n \rightarrow R$
    - Genotip → tuplu de numere reale $X=[x_1, x_2, \ldots, x_n]$, $x_i \in R$
    - Fenotip → valorile asociate argumentelor funcţiei f

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare arborescentă
  - Genotip
    - Arbori care codează S-Expresii
    - Nodurile interne ale arborelui → funcţii (F)
      - Matematice
        - Operatori aritmetici
        - Operatori de tip Boolean
      - Instrucţiuni
        - Într-un limbai de programare
        - Alt tip de instrucţiuni
    - Frunzele arborelui → terminale (T)
      - Valori reale sau Booleene, constante sau variabile
      - Subprograme
  - Fenotip
    - Interpretarea S-expresiilor
  - Ex. Calculul ariei unui cerc

$$\pi * r^2$$

Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Populaţie – concept
  - Scop
    - reţine o colecţie de soluţii candidat
      - se permit repetiţii
    - este folosită în întregime în procesul de selecţie pentru reproducere
  - Proprietăţi
    - dimensiune (de obicei) fixă μ
    - diversitate
      - Nr de fitness-uri/fenotipuri/genotipuri diferite
  - Observaţii
    - Reprezintă unitatea de bază care evoluează
      - populaţia întreagă evoluează, nu indivizii!!!

Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Populaţie – iniţializare
  - Uniformă (dacă e posibil) în spaţiul de căutare
    - Stringuri binare
      - generarea de 0 şi 1 cu probabilitatea 0.5
    - Şiruri de numere reale generate uniform (într-un anumit interval)
    - Permutări
      - generarea permutării identice şi efectuarea unor schimbări

Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Populaţie – iniţializare
  - Uniformă (dacă e posibil) în spaţiul de căutare
    - Arbori
      - Metoda *Full* – arbori compleţi
        - Nodurile de la adâncimea d < D_{max} se iniţializează aleator cu o funcţie din setul de funcţii *F*
        - Nodurile de la adâncimea d = D_{max} se iniţializează aleator cu un terminal din setul de terminale *T*
      - Metoda *Grow* – arbori incompleţi
        - Nodurile de la adâncimea d < D_{max} se iniţializează aleator cu un element din *F U T*
        - Nodurile de la adâncimea d = D_{max} se iniţializează aleator cu un terminal din setul de terminale *T*
      - Metoda *Ramped half and half*
        - ½ din populaţie se creează cu metoda *Full*
        - ½ din populaţie se creează cu metoda *Grow*
        - Folosind diferite adâncimi

Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Modele de populaţii – algoritm evolutiv:
  - Generaţional
    - În fiecare generaţie se crează μ descendenţi
    - Fiecare individ supravieţuieşte o singură generaţie
    - Mulţimea părinţilor este înlocuită în întregime cu mulţimea descendenţilor
  - Steady-state
    - În fiecare generaţie se obţine un singur descendent
    - Un singur părinte (cel mai slab) este înlocuit cu descendentul obţinut

- Discrepanţa între generaţii (*Generation Gap*)
  - Proporţia populaţiei înlocuite
  - $1 = \mu/\mu$, pentru modelul generaţional
  - $1/\mu$, pentru modelul steady-state

Algoritmi evolutivi – algoritm
Proiectare – funcția de evaluare

- Scop
  - Reflectă condiţiile la care trebuie să se adapteze populaţia
  - Funcţie de calitate sau funcţie obiectiv
  - Asociază o valoare fiecărei soluţii candidat
    - Consecinţe asupra selecţiei → cu cât sunt mai multe valori diferite, cu atât e mai bine

- Proprietăţi
  - Etapa cea mai costisitoare
    - Nu se re-evaluează indivizii nemodificaţi

- Tipologie:
  - După nr de obiective urmărite:
    - Uni-obiectiv
    - Multi-obiectiv → fronturi Pareto
  - După direcţia optimizării
    - De maximizat
    - De minimizat
  - După gradul de exactitate
    - Exactă
    - Euristică

Algoritmi evolutivi – algoritm
Proiectare – funcția de evaluare

- Exemple
  - Problema rucsacului
    - reprezentare → liniară discretă binară
    - fitness → abs(greutatea rucsacului – greutatea obiectelor alese) → minimizare
  - Problema plăţii unei sume folosind diferite monezi
    - reprezentare → liniară discretă întreagă
    - fitness → abs(suma de plată – suma monezilor selectate) → minimizare
  - Problema comisului voiaior
    - reprezentare → liniară discretă întreagă sub formă de permutare
    - fitness → costul drumului parcurs → minimizare
  - Problema optimizării funcţiilor
    - Reprezentare → liniară continuă reală
    - fitness → valoarea funcţiei → minimizare/maximizare
  - Calculul ariei unui cerc
    - reprezentare → arborescentă
    - fitness → suma pătratelor erorilor (diferenţelor între valoarea reală şi cea calculată pe un set de exemple) → minimizare

Algoritmi evolutivi – algoritm Proiectare – selecția

- Scop:
  - acordă şanse de reproducere/supravieţuire mai mari indivizilor mai buni
    - şi indivizii mai slabi trebuie să aibă şansa să se reproducă/supravieţuiască pentru că pot conţine material genetic util
  - direcţionează populaţia spre îmbunătăţirea calităţii

- Proprietăţi
  - lucrează la nivel de populaţie
  - se bazează doar pe fitnessul indivizilor (este independentă de reprezentare)
  - aiută la evadarea din optimele locale datorită naturii sale stocastice

Algoritmi evolutivi – algoritm Proiectare – selecția

- Tipologie
  - în funcţie de scop:
    - Selecţia părinţilor (din generaţia curentă) pentru reproducere
    - Selecţia supravieţuitorilor (din părinţi şi descendenţi) pentru generaţia următoare
  - în funcţie de modul de decidere al câştigătorului
    - Deterministă – cel mai bun câştigă
    - Stocastică – cel mai bun are cele mai mari şanse să câştige
  - în funcţie de mecanism
    - Selecţia pentru reproducere
      - Selecţie proporţională (bazată pe fitness)
      - Selecţie bazată pe ranguri
      - Selecţie prin turnir ----→ Bazată pe o parte din populaţie

      } Bazate pe întreaga populaţie
    - Selecţia pentru supravieţuire
      - Bazată pe vârstă
      - Bazată pe calitate (fitness)

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţie proporţională (bazată pe fitness) – SP
  - Ideea de bază
    - Algoritmul ruletei la nivelul întregii populaţii
    - Estimarea numărului de copii ale unui individ
      $E(n_i) = \mu \frac{f(i)}{\langle f \rangle}$, unde:
      - μ = dimensiunea populaţiei,
      - *f(i)* = fitnessul individului *i*,
      - $\langle f \rangle$ = fitnessul mediu al populaţiei
    - Indivizii mai buni
      - au alocat mai mult spaţiu în ruletă
      - au şanse mai mari să fie selectaţi
    - Ex. O populaţie cu μ = 3 indivizi

	f(i)	$P_{selSP}(i)$
A	1	1/10=0.1
B	5	5/10=0.5
C	4	4/10=0.4
Suma	10	1

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

Selecţie proporţională (bazată pe fitness)

- Avantaie
  - Algoritm simplu
- Dezavantaie
  - Convergenţa prematură
    - cromozomii foarte buni tind să domine populaţia
  - Presiune de selecţie foarte mică atunci când fintessurile indivizilor sunt foarte apropiate (la sfârşitul rulării)
  - Susceptibilă de traspoziţia funcţiei
  - Rezultatele reale ale unei astfel de selcţii diferă de distribuţia probabilistică teoretică
  - Lucrează cu întreaga populaţie
- Soluţii
  - scalarea fitnessului
    - Windowing
      - $f'(i) = f(i) - \beta^t$, unde β este un parametru care depinde de istoria recentă a evoluției
        - ex. β este fitnessul celui mai slab individ din populaţia curentă (a *t*-a ge
    - Scalare de tip sigma (de tip Goldberg)
      - $f'(i) = \max\{f(i) - (\langle f \rangle - c * \sigma_f), 0.0\}$, unde:
        - c este o constantă (de obicei 2)
        - $\langle f \rangle$ - fitnessul mediu al populaţei
        - σ_f – deviaţia standard a fitnessului populaţiei
    - Scalare prin normalizare
      - Se începe cu fitnessurile absolute (iniţiale)
      - Se standardizează astfel încât Se aiustează fitnessurile a.î.:
        - ele să aparţină [0,1]
        - cel mai bun fitness să fie cel mai mic (egal cu 0)
        - suma lor să fie 1
  - alt mecanism de selecţie

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia bazată pe ranguri – SR
  - Ideea de bază
    - Se ordonează întreaga populaţie pe baza fitnessului
      - Creşte puţin complexitatea algoritmului, dar se poate negliia această creştere comparativ cu timpul necesar evaluării unui individ
    - Se acordă ranguri fiecărui individ
    - Se calculează probabilităţile de selecţie pe baza rangurilor
      - Cel mai slab individ are rangul 1
      - Cel mai bun individ are rangul μ
    - Încearcă să rezolve problemele selecţiei proporţionale prin folosirea fitnessurilor relative (în locul celor absolute)

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia bazată pe ranguri – SR
  - Modalităţi de acordare a rangurilor
    - Liniară (RL) $P_{lin_rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$
      - *s* – presiunea de selecţie
        - măsoară avantaiele celui mai bun individ
        - $1.0 < s \leq 2.0$
        - în algoritmul genetic generaţional *s* este numărul de copii ai unui individ
      - Ex. pentru o populaţie cu μ = 3 indivizi

	f(i)	$P_{selSP}(i)$	Rang	$P_{selRL}(i)$ pt. s=2	$P_{selRL}(i)$ pt. s=1
A	1	1/10=0.1	1	0.33	0.33
B	5	5/10=0.5	3	1.00	0.33
C	4	4/10=0.4	2	0.67	0.33
Suma	10	1			

    - Exponenţială (RE) $P_{\exp_rank}(i) = \frac{1-e^{-i}}{c}$
      - Cel mai bun individ poate avea mai mult de 2 copii
      - c – factor de normalizare
        - depinde de dimensiunea populaţiei (μ)
        - trebuie ales a.î. suma probabilităţilor de selecţie să fie 1

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia bazată pe ranguri – SR
  - Avantaie
    - Păstrează presiunea de selecţie constantă
  - Dezavantaie
    - Lucrează cu întreaga populaţie
  - Soluţii
    - Alt mecanism de selecţie

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia prin turnir
  - Ideea de bază
    - Se aleg aleator *k* indivizi → eşantion de *k* indivizi (*k* – mărimea turnirului)
    - Se selectează cel mai bun individ dintre cei aleşi anterior
    - Probabilitatea alegerii unui individ în eşantion depinde de
      - Rangul individului
      - Dimensiunea eşantionului (*k*)
        - Cu cât k este mai mare, cu atât creşte şi presiunea de selcţie
      - Modul în care se face alegerea – dacă se realizează cu înlocuire (model steady-state) sau nu
        - Alegerea fără înlocuire creşte presiunea de selecţie
      - Pt k = 2 timpul necesar ca cel mai bun individ să domine populaţia este acelaşi cu cel de la selecţia pe bază de ranguri liniare cu s = 2 * *p*, *p* – probabilitatea alegerii celui mai bun individ din populaţie

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia prin turnir
  - Avantaje
    - Nu implică lucrul cu întrega populaţie
    - Uşor de implementant
    - Uşor de controlat presiunea de selcţie prin intermediul parametrului k
  - Dezavantaje
    - Rezultatele reale ale unei astfel de selecţii diferă de distribuţia probabilistică teoretică (similar selecţiei prin mecanismul ruletei)

Algoritmi evolutivi
Proiectare – selecția

- Selecţia pentru supravieţuire (înlocuire)
  - Pe baza vârstei
    - eliminarea celor mai "bătrâni" indivizi
  - Pe baza calităţii (fitness-ului)
    - selecţiei proporţională
    - selecţie bazată pe ranguri
    - selecţie prin turnir
    - elitism
      - Păstrarea celor mai buni indivizi de la o generaţie la alta (dacă descendenţii sunt mai slabi ca părinţii se păstrează părinţii)
    - GENITOR (înlocuirea celui mai slab individ)
      - Eliminarea celor mai slabi λ indivizi

Algoritmi evolutivi - algoritm
Proiectare – operatori de variaţie

- Scop:
  - Generarea unor soluţii potenţiale noi
- Proprietăţi
  - lucrează la nivel de individ
  - se bazează doar pe reprezentarea indivizilor (independent de fitness)
  - Aiută la explorarea şi exploatarea spaţiului de căutare
  - Trebuie să producă indivizi valizi
- Tipologie
  - În funcţie de aritate
    - Aritate 1 → operatori de mutaţie
    - Aritate > 1 → operatori de recombinare/încrucişare

Algoritmi evolutivi – algoritm Proiectare – mutația

- Scop
  - Reintroducerea în populaţie a materialului genetic pierdut
  - Operator unar de căutare (spaţiul continuu)
  - Introducerea diversităţii în populaţie (în spaţiul discret – binar)

- Proprietăţi
  - Acţionează la nivel de genotip
  - Bazată pe elemente aleatoare

  - Responsabilă cu explorarea unor noi regiuni promiţătoare ale spaţiului de căutare
  - Este responsabilă de evadarea din optimele locale

  - Trebuie să producă mici schimbări stocastice ale individului
  - Mărimea mutaţiei trebuie să fie controlabilă
  - Se produce cu o anumită probabilitate (p_m) la nivelul fiecărei gene a unui cromozom

Algoritmi evolutivi – algoritm Proiectare – mutația

- Tipologie
  - Reprezentare binară
    - Mutaţie tare - bit-flipping
    - Mutaţie slabă
  - Reprezentare întreagă
    - Random resetting
    - Creep mutation
  - Reprezentare permutare
    - Mutaţie prin inserţie
    - Mutaţie prin interchimbare
    - Mutaţie prin inversare
    - Mutaţie prin amestec
    - Mutaţie k-opt
  - Reprezentare reală
    - Mutaţie uniformă
    - Mutaţie neuniformă
      - Mutaţie Gaussiană
      - Mutaţie Cauchy
      - Mutaţie Laplace
  - Reprezentare arborescentă → într-un curs viitor
    - Mutaţie grow
    - mutaţie shrink
    - Mutaţie switch
    - Mutaţie cycle
    - Mutaţie tip Koza
    - Mutaţie pentru terminalele numerice

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. binară)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{0,1\}$, pt. $i=1,2,\ldots,L$

- Mutaţie tare – *bit flipping*
  - Ideea de bază
    - Schimbarea cu probabilitatea p_m (rată de mutaţie) a unor gene în complementul lor
      - 1 → 0
      - 0 → 1
    - Ex. Un cromozom cu L = 8 gene, p_m = 0.1

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. binară)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde g_i, $g_i' \in \{0,1\}$, pt. $i=1,2,\ldots,L$

- Mutaţie slabă
  - Ideea de bază
    - Schimbarea cu probabilitatea p_m (rată de mutaţie) a unor gene în 0 sau 1
      - 1 → 0/1
      - 0 → 1/0
    - Ex. Un cromozom cu L = 8 gene, p_m = 0.1

1	0	1	0	0	0	1	1

→

1	1	1	0	0	0	0	1

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. întreagă)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,\ldots,val_k\}$, pt. $i=1,2,\ldots,L$

- Mutaţie *random resetting*
  - Ideea de bază
    - Valoarea unei gene este schimbată (cu probabilitatea p_m) într-o altă valoare (din setul de valori posibile)

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. întreagă)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \ldots, val_k\}$, pt. $i=1,2,\ldots,L$

- Mutaţie *creep*
  - Ideea de bază
    - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
      - valoarea → face parte dintr-o distribuţie simetrică faţă de zero
      - modificarea produsă este fină (mică)

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ cu $g_i \neq g_i$ pentru orice $i \neq i$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,\ldots,val_L\}$, pt. $i=1,2,\ldots,L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i$.

- Mutaţie prin interschimbare (*swap mutation*)
  - Ideea de bază
    - Se aleg aleator 2 gene şi se interschimbă valorile lor

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ cu $g_i \neq g_i$ pentru orice $i \neq i$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,\ldots,val_L\}$, pt. $i=1,2,\ldots,L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i$.

Mutaţie prin inserţie

- Ideea de bază
  - Se aleg 2 gene oarecare g_i şi g_j cu $j > i$
  - Se inserează gj după gi a.î. $g_i'=g_i$, $g_{i+1}'=g_j$, $g_{k+2}'=g_{k+1}$, pentru k=i, i+1, i+2, ...

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ cu $g_i \neq g_i$ pentru orice $i \neq i$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,\ldots,val_L\}$, pt. $i=1,2,\ldots,L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i$.

- Mutaţie prin inversare
  - Ideea de bază
    - Se aleg aleator 2 gene şi se inversează ordinea genelor situate între ele (substringul dintre gene)

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ cu $g_i \neq g_i$ pentru orice $i \neq i$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,\ldots,val_L\}$, pt. $i=1,2,\ldots,L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i$.

- Mutaţie prin amestec (*scramble mutation*)
  - Ideea de bază
    - Se alege aleator un subşir (continuu sau discontinuu) de gene şi se rearanjează acele gene

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ cu $g_i \neq g_i$ pentru orice $i \neq i$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,\ldots,val_L\}$, pt. $i=1,2,\ldots,L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i$.

Mutaţie k-opt

- Ideea de bază
  - Se aleg 2 substringuri disjuncte şi de lungime *k*
  - Se interchimbă 2 elemente ale acestor substringuri de gene

k=2

2	5	3	4	8	6	1	7

↓

2	5	6	4	8	3	1	7

Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. reală)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Mutaţie uniformă
  - Ideea de bază
    - g_i' este schimbată cu probabilitatea p_m la o valoare aleasă aleator uniform din $[LI_i, LS_i]$

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. reală)

- Un cromozom $c=(g_1,g_2,\ldots,g_L)$ devine $c'=(g_1',g_2',\ldots,g_L')$, unde $g_i, g_i' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Mutaţie neuniformă
  - Ideea de bază
    - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
      - valoarea → face parte dintr-o distribuţie
        - $N(\mu, \sigma)$ (Gaussiană) cu $\mu = 0$
        - Cauchy (x_0, γ)
        - Laplace (μ, b)
      - şi readusă la $[LI_i, LS_i]$ (dacă este necesar) – *clamping*

Algoritmi evolutivi – algoritm Proiectare - recombinarea

- Scop
  - Amestecarea informaţiilor preluate din părinţi

- Proprietăţi
  - Descendentul trebuie să moştenească ceva de la fiecare dintre părinţi
    - Alegerea informaţilor care se amestecă este aleatoare
  - Operator de exploatare probabilistică (p_c) a spaţiilor deja descoperite
  - Descendenţii pot să fie mai buni, la fel de buni sau mai slabi decât părinţii lor
  - Efectele sale se reduc pe măsură ce căutarea converge

Algoritmi evolutivi – algoritm Proiectare - recombinarea

- Tipologie – în funcţie de reprezentarea indivizilor
  - Reprezentare binară şi întreagă
    - Cu puncte de tăietură
    - Uniformă
  - Reprezentare cu permutări
    - Încrucişare prin ordonare (versiunea 1 şi versiunea 2)
    - Încrucişare transformată parţial (Partially Mapped Crossover)
    - Încrucişare ciclică
    - Încrucişare bazată pe legături (muchii)
  - Reprezentare reală
    - Discretă
    - Intermediară (aritmetică)
      - Aritmetică singulară
      - Aritmetică simplă
      - Aritmetică completă
    - Geometrică
    - Încrucişare amestecată
    - Încrucişare binară simulată
  - Reprezentare cu arbori
    - Încrucişare de sub-arbori → într-un curs viitor

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. binară şi întreagă)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in \{0,1\} / \{val_1, val_2, \ldots, val_k\}$, pt. $i=1,2,\ldots,L$

Încrucişare cu *n* puncte de tăietură

- Ideea de bază
  - Se aleg *n* puncte de tăietură ($n < L$)
  - Se taie cromozomii părinţi prin aceste puncte
  - Se lipesc părţile obţinute, alternând părinţii

n=2

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. binară şi întreagă)

- Încrucişare cu *n* puncte de tăietură
  - Proprietăţi
    - Media valorilor codate de părinţi = media valorilor codate de descendenţi
      - Ex. Reprezentarea binară pe 4 biţi a numerelor întregi – XO cu n = 1 dupa bitul 2
        - p_1 = (1,0,1,0), p_2 = (1,1,0,1)
        - d_1 = (1,0, 0,1), d_2 = (1,1,1,0)
        - val(p_1) = 10, val(p_2) = (13) ➔ (val(p_1) + val(p_2))/2 = 23/2=11.5
        - val(d_1) = 9, val(d_2) = (14) ➔ (val(d_1) + val(d_2))/2 = 23/2=11.5
      - Ex. Reprezentare binară pe 4 biţi pentru problema rucsacului de capacitate K = 10 cu 4 obiecte de greutate şi valoare ((2,7), (1,8), (3,1), (2,3))
        - p_1 = (1,0,1,0), p_2 = (1,1,0,1)
        - d_1 = (1,0, 0,1), d_2 = (1,1,1,0)
        - val(p_1) = 8, val(p_2) = 18 ➔ (val(p_1) + val(p_2))/2 = 26/2=13
        - val(d_1) = 10, val(d_2) = 16 ➔ (val(d_1) + val(d_2))/2 = 26/2=13
    - Probabilitatea apariţiei unui factor de răspândire $\beta \approx 1$ este mai mare decât probabilitatea oricărui alt factor

$$\beta = \left| \frac{val(d_1) - val(d_2)}{val(p_1) - val(p_2)} \right|$$

      - Încrucişare prin contracţie $\beta < 1$
        - Valorile descendenţilor se află între valorile părinţilor
      - Încrucişare prin extensie $\beta > 1$
        - Valorile părinţilor se află între valorile descendenţilor
      - Încrucişare staţionară $\beta = 1$
        - Valorile descendenţilor coincid cu valorile părinţilor

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. binară şi întreagă)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in \{0,1\} / \{val_1, val_2, \ldots, val_k\}$, pt. $i=1,2,\ldots,L$

- Încrucişare uniformă
  - Ideea de bază
    - Fiecare genă a unui descendent provine dintr-un părinte ales aleator şi uniform:
      - Pentru fiecare genă în parte se generează un număr aleator *r* care respectă legea uniformă
      - Dacă numărul generat *r* < probabilitatea *p* (de obicei *p*=0.5), c_1 va lua gena respectivă din p_1 şi c_2 va lua gena respectivă din p_2,
      - Altfel c_1 va lua gena respectivă din p_2 şi c_2 va lua gena respectivă din p_1

p=0.5

1	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0

10*r →

8	6	3	2	6	4	7	2

→

0	0	1	1	0	0	0	1
1	0	1	0	0	1	1	0

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2,\ g_i',\ g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)
    - Se reia procedeul pentru al doilea descendent d_2.

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare parţial transformată (*partially mapped XO*)
  - Ideea de bază
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se iau pe rând elementele *i* din substringul din p_2 care nu apar în substringul din p_1 şi se determină care element *j* a fost copiat în locul lui din p_1
    - Se plasează *i* în d_1 în poziţia ocupată de *j* în p_2 (dacă locul este liber)
    - Dacă locul ocupat de *j* în p_2 a fost deja completat în d_1 cu elementul *k*, *i* se pune în locul ocupat de *k* în p_2
    - Restul elementelor se copiază din p_2 în d_1
    - Pentru descendentul d_2 se procedează similar, dar inversând părinţii

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare parţial transformată (*partially mapped XO*)
  - Ideea de bază
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se iau pe rând elementele *i* din substringul din p_2 care nu apar în substringul din p_1 şi se determină care element *j* a fost copiat în locul lui din p_1
    - Se plasează *i* în d_1 în poziţia ocupată de *j* în p_2 (dacă locul este liber)
    - Dacă locul ocupat de *j* în p_2 a fost deja completat în d_1 cu elementul *k*, *i* se pune în locul ocupat de *k* în p_2
    - Restul elementelor se copiază din p_2 în d_1
    - Pentru descendentul d_2 se procedează similar, dar inversând părinţii

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare parţial transformată
  - Ideea de bază
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se iau pe rând elementele *i* din substringul din p_2 care nu apar în substringul din p_1 şi se determină care element *j* a fost copiat în locul lui din p_1
    - Se plasează *i* în d_1 în poziţia ocupată de *j* în p_2 (dacă locul este liber)
    - Dacă locul ocupat de *j* în p_2 a fost deja completat în d_1 cu elementul *k*, *i* se pune în locul ocupat de *k* în p_2
    - Restul elementelor se copiază din p_2 în d_1
    - Pentru descendentul d_2 se procedează similar, dar inversând părinţii

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ciclică
  - Ideea de bază
    1. iniţial k = 1
    2. Se formează un ciclu
       - Se adaugă în ciclu gena de pe poziţia k din p1 (g_k^1)
       - Se consideră gena de pe poziţia k din p2 (g_k^2)
       - Se alege gena din p1 cu valoarea egală cu g_k^2 (g_r^1) şi se include în ciclu
       - Se consideră gena de pe poziţia r din p2 (g_r^2)
       - Se repetă paşii anteriori până când se ajunge la gena de pe poziţia k din p1
    3. Se copiază genele din ciclu în d1 (respectând poziţiile pe care apar în p1)
    4. Se incrementează k şi se formează un nou ciclu dar cu genele din p2
    5. Se copiază genele din ciclu în d1 (respectând poziţiile pe care apar în p2)
    6. Se repetă paşii 2-5 până când k = L

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ciclică
  - Ideea de bază
    1. iniţial k = 1
    2. Se formează un ciclu
       - Se adaugă în ciclu gena de pe poziţia k din p1 (g_k^1)
       - Se consideră gena de pe poziţia k din p2 (g_k^2)
       - Se alege gena din p1 cu valoarea egală cu g_k^2 (g_r^1) şi se include în ciclu
       - Se consideră gena de pe poziţia r din p2 (g_r^2)
       - Se repetă paşii anteriori până când se ajunge la gena de pe poziţia k din p1
    3. Se copiază genele din ciclu în d1 (respectând poziţiile pe care apar în p1)
    4. Se incrementează k şi se formează un nou ciclu dar cu genele din p2
    5. Se copiază genele din ciclu în d1 (respectând poziţiile pe care apar în p2)
    6. Se repetă paşii 2-5 până când k = L

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ciclică
  - Ideea de bază
    1. iniţial k = 1
    2. Se formează un ciclu
       - Se adaugă în ciclu gena de pe poziţia k din p1 (g_k^1)
       - Se consideră gena de pe poziţia k din p2 (g_k^2)
       - Se alege gena din p1 cu valoarea egală cu g_k^2 (g_r^1) şi se include în ciclu
       - Se consideră gena de pe poziţia r din p2 (g_r^2)
       - Se repetă paşii anteriori până când se ajunge la gena de pe poziţia k din p1
    3. Se copiază genele din ciclu în d1 (respectând poziţiile pe care apar în p1)
    4. Se incrementează k şi se formează un nou ciclu dar cu genele din p2
    5. Se copiază genele din ciclu în d1 (respectând poziţiile pe care apar în p2)
    6. Se repetă paşii 2-5 până când k = L

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare bazată pe muchii
  - A se consulta: *Whitley, Darrell, Timothy Starkweather, D'Ann Fuquay (1989). "Scheduling problems and traveling salesman: The genetic edge recombination operator".International Conference on Genetic Algorithms. pp. 133–140* link

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare discretă
  - Ideea de bază
    - Fiecare genă a unui descendent este luată (cu aceeaşi probabilitate, *p = 0.5*) dintr-unul din părinţi
    - Similar încrucişării uniforme de la reprezentarea binară/întreagă
    - Nu se modifică valorile efective ale genelor (nu se creează informaţie nouă)

p=0.5

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

10*r →

8	6	3	2	6	4	7	2

→

-2.1	1.3	3.2	2.4	1.1	0.6	1.0	-1.7
0.3	-1.5	0.2	-1.4	-1.1	-0.3	2.0	1.7

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare intermediară (aritmetică)
  - Ideea de bază
    - Se creează copii aflaţi (ca valoare) între părinţi ➔ încrucişare aritmetică
      - $z_i = \alpha x_i + (1 - \alpha) y_i$ unde α : $0 \le \alpha \le 1$.
    - Parametrul α poate fi:
      - Constant → încrucişare aritmetică uniformă
      - Variabil → ex. dependent de vârsta populaţiei
      - Aleator pt fiecare încrucişare produsă
    - Apar noi valori ale genelor

- Tipologie
  - Încrucişare aritmetică singulară
  - Încrucişare aritmetică simplă
  - Încrucişare aritmetică completă

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare intermediară (aritmetică) singulară
  - Se alege câte o genă (de acelaşi index *k*) din cei doi părinţi şi se combină
    - $g_k'= \alpha g_k^1 + (1-\alpha)g_k^2$
    - $g_k''= (1-\alpha)g_k^1 + \alpha g_k^2$
  - Restul genelor rămân neschimbate
    - $g_i'= g_i^1$
    - $g_i''= g_i^2$, pentru $i = 1,2,\ldots,L$ şi $i \neq k$

0.6*3.2+(1-0.6)*0.2=2.0

(1-0.6)*3.2+0.6*0.2=1.4

[LI,LS] = [-2.5, +3]
k=3
α = 0.6

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

→

0.3	-1.5	2.0	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	1.4	-1.4	1.1	-0.3	1.0	1.7

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare intermediară (aritmetică) simplă
  - Se alege o poziţie *k* şi se combină toate genele de după acea poziţie
    - $g_i' = \alpha g_i^1 + (1-\alpha) g_i^2$
    - $g_i'' = (1-\alpha) g_i^1 + \alpha g_i^2$, pentru $i=k, k+1, \ldots, L$
  - Genele de pe poziţii < *k* rămân neschimbate
    - $g_i' = g_i^1$
    - $g_i'' = g_i^2$, pentru $i = 1,2,\ldots,k-1$

0.6*0.6+(1-0.6)*(-0.3)=0.24

(1-0.6)*0.6+0.6*(-0.3)=0.06

[LI,LS] = [-2.5, +3]
k=6
α = 0.6

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

→

0.3	-1.5	3.2	2.4	-1.1	0.24	1.6	-0.34
-2.1	1.3	0.2	-1.4	1.1	0.06	1.4	0.34

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- **Încrucişare intermediară (aritmetică) completă**
  - Toate genele (de pe poziţii corespunzătoare) se combină
    - $g_i'= \alpha g_i^1 + (1-\alpha)g_i^2$
    - $g_i''= (1-\alpha)g_i^1 + \alpha g_i^2$, pentru $i=1,2,\ldots,L$

0.6*0.3+(1-0.6)*(-2.1)=-0.66

(1-0.6)*0.3+0.6*(-2.1)=-1.14

[LI,LS] = [-2.5, +3]

α = 0.6

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

-0.66	4.3	2.0	0.48	-0.22	0.24	1.6	-0.34
-1.14	0.18	1.4	0.12	0.22	0.06	1.4	0.34

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare geometrică
  - Ideea de bază
    - Fiecare genă a unui descendent reprezintă produsul genelor părinţilor, fiecare cu un anumit exponent ω, respectiv $1-\omega$ (unde ω număr real pozitiv subunitar)
    - $g_i' = (g_i^1)^{\omega} (g_i^2)^{1-\omega}$
    - $g_i'' = (g_i^1)^{1-\omega} (g_i^2)^{\omega}$

$0.3^{0.7}+2.1^{1-0.7}=1.68$

$0.3^{1-0.7}+2.1^{0.7}=2.38$

[LI,LS] = [-2.5, +3]

$\omega = 0.7$

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7

1.68	2.41	2.87	2.95	2.10	1.40	2.62	2.62
2.38	2.33	1.74	2.57	2.10	1.29	2.23	2.62

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţine 1 descendent
  - $c_1=(g_1',g_2',\ldots,g_L')$,
  - unde $g_i^1,g_i^2, g_i' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$
- Încrucişare amestecată (*blend crossover – BLX*)
  - Ideea de bază
    - Se generează un singur descendent
    - Genele g_i' ale descendentului sunt alese aleator în intervalul $[Min_i-I*a, Max_i+I*a]$, unde:
      - $Min_i = min\{g_i^1, g_i^2\}$, $Max_i = max\{g_i^1, g_i^2\}$
      - $I = Max - Min$, a – parametru din [0,1]

[LI,LS] = [-2.5, +3]
a = 0.7

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7

1.25	1.45	-1.11	2.37	1.10	0.11	0.70	1.70

Min	0.3	1.3	0.2	1.4	1.1	0.3	1.0	1.7
Max	2.1	1.5	3.2	2.4	1.1	0.6	2.0	1.7
I	0.8	0.2	3.0	1.0	0	0.3	1.0	0.0

Min-Ia	-0.26	1.16	-1.90	0.70	1.10	0.09	0.30	1.70
Max+Ia	2.66	1.50	3.20	2.40	1.10	0.60	2.00	1.70

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare binară simulată
  - Ideea de bază
    - Fiecare genă a unui descendent reprezintă o combinaţie a genelor părinţilor

$$d_1=\frac{p_1+p_2}{2}-\beta\frac{p_2-p_1}{2},\quad d_2=\frac{p_1+p_2}{2}+\beta\frac{p_2-p_1}{2}$$

    - a.î. să se respecte cele 2 proprietăţi de la încrucişarea cu n puncte de tăietură (pt. reprezentarea binară)
      - media valorilor codate în părinţi = media valorilor codate în descendenţi
      - probabilitatea apariţiei unui factor de răspândire $\beta \approx 1$ este mai mare decât a oricărui alt factor

Algoritmi evolutivi – algoritm
Proiectare – recombinarea

- Recombinarea multiplă
  - Bazată pe frecvenţa valorilor din părinţi (încrucişare uniformă generală)
  - Bazată pe segmentare şi recombinare (încrucişare generală cu puncte de tăietură diagonală)
  - Bazată pe operaţii numerice specifice valorilor reale (încrucişare bazată pe centrul de masă, încrucişare generală aritmetică)

Algoritmi evolutivi – algoritm
Proiectare – mutaţie sau recombinare?

- Dezbateri aprinse
  - Întrebări:
    - care operator este mai bun?
    - care operator este necesar?,
    - care operator este mai important?
  - Răspunsuri:
    - Depinde de problemă, dar
    - În general, este bine să fie folosiţi ambii operatori
    - Fiecare având alt rol
    - Sunt posibili AE doar cu mutaţie, dar nu sunt posibili AE doar cu încrucişare
- Aspecte ale căutării:
  - Explorare → descoperirea regiunilor promiţătoare ale spaţiului de căutare (acumulând informaţie utilă despre problemă)
  - Exploatare → optimizarea într-o regiune promiţătoare (folosind informaţia existentă)
  - Trebuie să existe cooperare şi competiţie între aceste 2 aspecte
- Încrucişarea
  - Operator exploatativ, realizând un mare salt într-o regiune undeva între regiunile asociate părinţilor
    - Efectele exploatative se reduc pe măsură ce AE converge
  - Operator binar (n-ar) care poate combina informaţia din 2 (sau mai mulţi) părinţi
  - Operator care nu schimbă frecvenţa valorilor din cromozomi la nivelul întregii populaţii
- Mutaţia
  - Operator explorativ, realizând mici diversiuni aleatoare, rămânând în regiunea apropiată părintelui
    - Evadarea din optimele locale
  - Operator care poate introduce informaţie genetică nouă
  - Operator care schimbă frecvenţa valorilor din cromozomi la nivelul întregii populaţii

Algoritmi evolutivi – algoritm Proiectare – criteriu de oprire

- Stabilirea unui criteriu de stop
  - S-a identificat soluţia optimă
  - S-au epuizat resursele fizice
    - S-a efectuat un anumit număr de evaluaări ale funcţiei de fitness
  - S-au epuizat resursele utilizatorului (timp, răbdare)
    - S-au "născut" câteva generaţii fără îmbunătăţiri

Algoritmi evolutivi - algoritm

- Evaluarea performanţelor unui AE
  - După mai multe rulări se calculează:
    - Măsuri statistice
      - media soluţiilor,
      - mediana soluţiilor,
      - cea mai bună soluţie,
      - cea mai slabă soluţie,
      - deviaţia standard – pentru comparabilitate
    - Calculate pentru un număr suficient de mare de rulări independente

Algoritmi evolutivi

- Analiza complexităţii
  - Partea cea mai costisitoare → calculul fitnessului

Algoritmi evolutivi

- Avantaje
  - Schema AE universală pentru toate problemele
    - se modifică doar
      - reprezentarea
      - funcţia de fitness
  - AE sunt capabili să producă rezultate mai bune decât metodele convenţionale de optimizare pentru că:
    - nu necesită liniarizare
    - nu implică anumite presupuneri (continuitate, derivabilitate, etc. a funcţiei obiectiv)
    - nu ignoră anumite potenţiale soluţii
  - AE sunt capabili să exploreze mai multe potenţiale soluţii decât poate explora omul

Algoritmi evolutivi

- Dezavantaje
  - Timp de rulare îndelungat

Algoritmi evolutivi

- Aplicaţii
  - Proiectări vehicule
    - Componenţa materialelor
    - Forma vehiculelor
  - Proiectări inginereşti
    - Optimizarea structurală şi organizatorică a construcţiilor (clădiri, roboţi, sateliţi, turbine)
  - Robotică
    - Optimizarea proiectării, funcţionării componentelor
  - Evoluare de hardware
    - Optimizarea de circuite digitale
  - Optimizarea telecomunicaţiilor
  - Generarea de glume şi jocuri de cuvinte
  - Invenţii biomimetice (inspirate de arhitecturi naturale)
  - Rutări pentru trafic şi transporturi
  - Jocuri de calculator
  - Criptări
  - Profilul expresiv al genelor
  - Analiza chimcă a cinecticii
  - Strategii financiare şi marketing

Algoritmi evolutivi

- Tipuri de algoritmi evolutivi
  - Strategii evolutive
  - Programare evolutivă
  - Algoritmi genetici
  - Programare genetică

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- capitolul 16 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *James Kennedy, Russel Eberhart, Particle Swarm Optimisation*, *Proceedings of IEEE International Conference on Neural Networks*. **IV**. pp. 1942–1948, 1995 (04_ACO_PSO/PSO_00.pdf)
- *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (04_ACO_PSO/Dorigo05_ACO.pdf)

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare informată

algoritmi evolutivi

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit şi legături utile

- capitolul 14 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Căutare locală

- Tipologie
  - Căutare locală simplă - se reţine o singură stare vecină
    - Hill climbing → alege cel mai bun vecin
    - Simulated annealing → alege probabilistic cel mai bun vecin
    - Căutare tabu → reţine lista soluţiilor recent vizitate
  - Căutare locală în fascicol (beam local search) – se reţin mai multe stări (o populaţie de stări)
    - Algoritmi evolutivi
    - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
    - Optimizare bazată pe furnici (Ant colony optmisation)

Algoritmi evolutivi

- Tipuri de algoritmi evolutivi
  - Algoritmi genetici
  - Strategii evolutive
  - Programare evolutivă
  - Programare genetică

Algoritmi genetici

- Aspecte teoretice
- Algoritm
  - Schema generală a unui AGS
  - Reprezentare şi operatori
- Exemplu
- Proprietăţi
- Aplicaţii

Algoritmi genetici – aspecte teoretice

- Propuşi
  - J. Holland → AG simpli (AGS)
- Căutare
  - **Concurenţială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
  - Selecţia
  - Încrucişarea **ŞI** mutaţia
- Elemente speciale
  - Accent deosebit pe încrucişare

Algoritmi genetici – schema generală

Algoritm generaţional

Iniţializare P(0)
Evaluare(P(0))
g = 0;
while (*not* condiţie_stop) do
    repeat
        Selectarea a 2 părinţi p_1 şi p_2 din *P(g)*
        Încrucişare(p_1, p_2) ➔ o_1 şi o_2
        Mutaţie(o_1) ➔ o_1*
        Mutaţie(o_2) ➔ o_2*
        Evaluare(o_1*)
        Evaluare(o_2*)

        Adăugare o_1* şi o_2* în *P(g+1)*
    until P(g+1) este plină
    g++
endWhile

Algoritm steady-state

Iniţializare P
Evaluare(P)

while (*not* condiţie_stop) do
    For i = 1 to |P|
        Selectarea a 2 părinţi p_1 şi p_2 din *P(g)*
        Încrucişare(p_1, p_2) ➔ o_1 şi o_2
        Mutaţie(o_1) ➔ o_1*
        Mutaţie(o_2) ➔ o_2*
        Evaluare(o_1*)
        Evaluare(o_2*)
        B = Best(o_1*, o_2*)
        W = Worst(o_1*, o_2*)
        Dacă B e mai bun ca W, W ← B
    EndFor

endWhile

Algoritmi genetici – schema generală

Algoritm generaţional

1. Generarea aleatoare a unei populaţii (generaţia 0) cu *n* cromozomi
2. Evaluarea tuturor cromozomilor
3. Crearea unei noi populaţii (generaţii) prin repetarea următorilor 4 paşi
   - Selecţia, bazată pe fitness, a 2 părinţi
   - Încrucişarea părinţilor pentru obţinerea unui descendent cu o anumită probabilitate; dacă încrucişarea nu are loc, descendentul va fi:
     - Unul dintre părinţi
     - Cel mai bun dintre părinţi
   - Mutaţia cu o anumită probabilitate a fiecărui element al descendentului
   - Acceptarea descendentului şi plasarea lui în noua populaţie (generaţie)
4. Înlocuirea vechii populaţii cu noua populaţie (schimbul de generaţii)
5. Testarea condiţiilor de terminare a căutării; dacă ele sunt satisfăcute, se returnează cea mai bună soluţie din populaţia (generaţia) curentă
6. Ciclarea algoritmului – întoarcerea la pasul 2

Algoritmi genetici – schema generală

Algoritm steady-state

1. Generarea aleatoare a unei populaţii cu *n* cromozomi
2. Evaluarea tuturor cromozomilor
3. Crearea unei noi populaţii prin repetarea următorilor 4 paşi
   - Selecţia, bazată pe fitness, a 2 părinţi
   - Încrucişarea părinţilor pentru obţinerea unui descendent cu o anumită probabilitate; dacă încrucişarea nu are loc, descendentul va fi:
     - Unul dintre părinţi
     - Cel mai bun dintre părinţi
   - Mutaţia cu o anumită probabilitate a fiecărui element al descendentului
   - Alegerea celui mai bun descendent B
   - Dacă B este mai bun decât cel mai slab individ al populaţiei W, atunci B îl înlocuieşte pe W
4. Testarea condiţiilor de terminare a căutării; dacă ele sunt satisfăcute, se returnează cea mai bună soluţie din populaţia (generaţia) curentă
5. Ciclarea algoritmului – întoarcerea la pasul 2

Algoritmi genetici – reprezentare şi operatori

- Reprezentare
  - Stringuri binare (iniţial), de numere întregi, de numere reale, de alte elemente
- Populaţia
  - μ părinţi, μ descendeţi
- Selecţia pentru recombinare
  - Propoţională cu fitness-ul
- Recombinarea
  - Cu n puncte de tăietură sau uniformă cu o probabilitate p_c fixată ce acţionează la nivel de cromozom
- Mutaţia
  - *Bitwise bit-flipping* cu o probabilitate p_m fixată pentru fiecare genă (bit)
- Selecţia pentru supravieţuire
  - Toţi descendenţii înlocuiesc părinţii

Algoritmi genetici – exemplu

- Să se determine valoarea maximă a funcţiei

$$f: \{0,1,\ldots,31\} \rightarrow Z,\ f(x) = x^2$$

- Configurarea AG
  - Stringuri binare de lungime 5, ex. c=(10101)→x=21
  - O populaţie cu μ = 4 cromozomi
  - Selecţie proporţională prin ruletă
  - Încrucişare cu 1 punct de tăietură
  - Mutaţie tare

- Evaluare → optimizare prin maximizare

Algoritmi genetici – exemplu

Iniţializare

No cromo-zom	Cromo zom
1	01101
2	11000
3	01000
4	10011
sumă	

Algoritmi genetici – exemplu

Evaluare

No cromo-zom	Cromo zom	Valoa-rea x	Fitness $f(x^2)$
1	01101	13	169
2	11000	24	576
3	01000	8	64
4	10011	19	361
sumă			1170

Algoritmi genetici – exemplu

Selecţie

No cromo-zom	Cromo zom	Valoar ea x	Fitness $f(x^2)$	$P_{selSP}(i)$	$\Sigma P_{selSP}(i)$
1	01101	13	169	169/1170=0.14	0.14
2	11000	24	576	576/1170=0.49	0.63
3	01000	8	64	64/1170=0.06	0.69
4	10011	19	361	361/1170=0.31	1.00
sumă			1170		

Algoritmi genetici – exemplu

Selecţie

No cromo-zom	Cromo zom	Valoa-rea x	Fitness	$P_{selSP}(i)$	$\Sigma P_{selSP}(i)$	$r_1=0.5$	$r_2=0.8$
1	01101	13	169	169/1170=0.14	0.14		
2	11000	24	576	576/1170=0.49	0.63	X	
3	01000	8	64	64/1170=0.06	0.69		
4	10011	19	361	361/1170=0.31	1.00		x
sumă			1170				

$p_1=c_2$ = (11000) şi p_2 = c_4 = (10011)

Algoritmi genetici – exemplu

Încrucişare

No cromozom	Cromozomi părinţi	Cromozomi fii	Valoarea x (pt. fii)	Fitness (pt. fii)
2	11000	11011	27	729
4	10011	10000	16	256

Mutaţie

No cromozom	Cromozomi fii	Cromozomi fii*	Valoarea x (pt. fii*)	Fitness (pt. fii*)
o1	11011	10011	19	361
o2	10000	10010	18	324

Algoritmi genetici – exemplu

Adăugarea în următoarea generaţie

No cromo-zom	Cromo zom
1	10011
2	10010
3	
4	

Algoritmi genetici – exemplu

Selecţie

No cromo-zom	Cromo zom	Valoar ea x	Fitness $f(x^2)$	$P_{selSP}(i)$	$\Sigma P_{selSP}(i)$	$r_1=0.1$	$r_2=0.7$
1	01101	13	169	169/1170=0.14	0.14	x	
2	11000	24	576	576/1170=0.49	0.63		
3	01000	8	64	64/1170=0.06	0.69		
4	10011	19	361	361/1170=0.31	1.00		x
sumă			1170				

Algoritmi genetici – exemplu

Selecţie

No cromo-zom	Cromo zom	Valoa-rea x	Fitness $f(x^2)$	$P_{selSP}(i)$	$\Sigma P_{selSP}(i)$	$r_1=0.5$	$r_2=0.8$
1	01101	13	169	169/1170=0.14	0.14	x	
2	11000	24	576	576/1170=0.49	0.63		
3	01000	8	64	64/1170=0.06	0.69		
4	10011	19	361	361/1170=0.31	1.00		x
sumă			1170				

$p_1=c_1$ = (01101) şi p_2 = c_4 = (10011)

Algoritmi genetici – exemplu

Încrucişare

No cromozom	Cromozomi părinţi	Cromozomi fii	Valoarea x (pt. fii)	Fitness (pt. fii)
1	01|101	01011	11	121
4	10|011	10101	21	441

Mutaţie

No cromozom	Cromozomi fii	Cromozomi fii*	Valoarea x (pt. fii*)	Fitness (pt. fii*)
o1	01011	00011	3	9
o2	10101	10111	23	529

Algoritmi genetici – exemplu

Adăugarea în următoarea generaţie

No cromo-zom	Cromo zom
1	10011
2	10010
3	00011
4	10111

Algoritmi genetici – proprietăţi

- Cromozomi liniari de aceeaşi dimensiune
- Evidenţiază avantajele combinării informaţiilor de la părinţi buni prin încrucişare
- Numeroase variante
- Numeroşi operatori (selecţie, încrucişare, mutaţie)
- Nu sunt foarte rapizi
- Euristici bune pentru probleme de combinatorică

Algoritmi genetici – aplicații

- Probleme de combinatorică
- Optimizări în proiectarea compoziţiei materialelor şi a formei aerodinamice a vehiculelor (auto, aeriene, navale, trenuri)
- Optimizări în proiectarea structurală şi funcţională a clădirilor (locuinţe, fabrici, etc)
- Optimizări în robotică
- Optimizări în proiectarea circuitelor digitale

Strategii evolutive

- Aspecte teoretice
- Algoritm
  - Schema generală
  - Reprezentare şi operatori
- Exemplu
- Proprietăţi
- Aplicaţii

Strategii evolutive – aspecte teoretice

- Propuse
  - în anii ’60-’70 în Germania de către Bienert, Rechenberg şi Schwefel
- Căutare
  - **Concurenţială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
  - Selecţia
  - Încrucişarea **ŞI** mutaţia
- Elemente speciale
  - Auto-adaptarea parmetrilor (în special a parametrilor mutaţiei)

Strategii evolutive – schema generala

Iniţializare P(0)

Evaluare(P(0))

g = 0;

while (*not* condiţie_stop) do

  repeat

    Selectarea a 2 părinţi p_1 şi p_2 din *P(g)*

    Încrucişare(p_1, p_2) ➔ o_1

    Mutaţie(o_1, *param*) ➔ o_1*, *param'*

    Evaluare(o_1*)

    Adăugare o_1* în *P(g+1)*

  until P(g+1) este plină

  g++

endWhile

Strategii evolutive – reprezentare şi operatori

- Reprezentare
  - Reală
  - Codează şi rata de mutaţie
- Populaţia
  - μ părinţi, λ descendeţi
- Selecţia pentru recombinare
  - Uniformă aleatoare
- Recombinarea
  - Discretă sau intermediară
- Mutaţia
  - Perturbare Gaussiană
  - Auto-adaptare a pasului de mutaţie
- Selecţia pentru supravieţuire
  - (μ,λ) sau (μ+λ)

Strategii evolutive – reprezentare şi operatori

- Pp. că dorim minimizarea funcţiei $f: R^n \rightarrow R$

- Reprezentare
  - 3 părţi:
    - Variabile obiect: $x_1, x_2, \ldots, x_n$ cu $x_i \in R$ ➔ reprezentare reală
    - Parametri posibili ai SE:
      - Paşi de mutaţie: $\sigma_1, \ldots, \sigma_{n(\sigma)}$
      - Unghiuri de rotaţie $\omega_1, \ldots, \omega_{n(\alpha)}$
  - Completă
    - $n(\sigma)=n$, $n(\alpha) = n(n-1)/2$ – nr de perechi (i,j), $i, j = 1,2,\ldots,n$

Strategii evolutive – reprezentare şi operatori

- Selecţia părinţilor (pentru reproducere)
  - Uniformă aleatoare
  - Fiecare individ are aceeaşi probabilitate de a fi selectat

Strategii evolutive – reprezentare şi operatori

- Reproducerea
  - Combină doi sau mai mulţi părinţi
  - Crează un singur descendent

	2 părinţi	Câte 2 părinţi pentru fiecare element x_i al unui cromozom
$z_i = (x_i + y_i)/2$	Intermediară locală	Intermediară globală
z_i este fie x_i, fie y_i (alegerea fiind aleatoare)	Discretă locală	Discretă globală

Strategii evolutive – reprezentare şi operatori

- Mutaţia
  - Parametrii σ se coevoluează cu soluţia x
  - Mutaţie Gaussiană
    - σ este evoluat în σ'
    - $x_i' = x_i + N(0, \sigma')$
  - Nu este necesar ca parametrii σ să evolueze (să se modifice) cu aceeaşi frecvenţă ca soluţia (x)
  - Noua pereche (x',σ') se evaluează de 2 ori:
    - x' este bună dacă evaluarea $f(x')$ este bună
    - σ' este bun dacă x' este bună

$$x_{i,t+1} = x_{i,t} + N(0,\sigma)$$

Strategii evolutive – reprezentare şi operatori

- Cum este evoluat pasul de mutaţie din σ în σ' ? ➔ diferite metode
  - Regula succesului 1/5
    - Se determină procentajul p_s al mutaţiilor "folositoare" (care au îmbunătăţit potenţiala soluţie) din ultimele k iteraţii
    - Se modifică σ după fiecare k iteraţii astfel:
      - $\sigma = \sigma / c$, daca $p_s > 1/5$
      - $\sigma = \sigma * c$, dacă $p_s < 1/5$
      - $\sigma = \sigma$, dacă $p_s = 1/5$,
    - unde $0.8 \leq c \leq 1$
  - Regula auto-adaptării
    - Mutaţie necorelată cu un singur parametru σ
    - Mutaţie necorelată cu n parametri σ
    - Mutaţie corelată

Strategii evolutive – reprezentare şi operatori

- Mutaţie necorelată cu un singur parametru σ
  - Cromozomi de forma: $(x_1, x_2, \ldots, x_n, \sigma)$
  - Mutaţie
    - $\sigma' = \sigma * exp(\tau * N(0,1))$
    - $x_i' = x_i + \sigma' * N(0,1)$
    - Unde τ - rata de învăţare
      - de obicei $\tau = 1/(n^{1/2})$
    - Dacă $\sigma' < \varepsilon_0$ ➔ $\sigma' = \varepsilon_0$

Strategii evolutive – reprezentare şi operatori

- Mutaţie necorelată cu *n* parametri σ
  - Cromozomi de forma: $(x_1, x_2, \ldots, x_n, \sigma_1, \sigma_2, \ldots \sigma_n)$
  - Mutaţie
    - $\sigma'_i = \sigma_i * exp(\tau' * N(0,1) + \tau * N_i(0,1))$
    - $x_i' = x_i + \sigma'_i * N_i(0,1)$
    - unde:
      - τ' - rata globală de învăţare
      - τ - rata individuală de învăţare
      - de obicei $\tau' = 1/((2n)^{1/2})$ şi $\tau = 1/((2n^{1/2})^{1/2})$
    - dacă $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$

Strategii evolutive – reprezentare şi operatori

- Mutaţie corelată cu *n+k* parametri
  - Cromozomi de forma: $(x_1, x_2, \ldots, x_n, \sigma_1, \sigma_2, \ldots \sigma_n, \omega_1, \omega_2, \ldots, \omega_k)$,
    - unde $k=n(n-1)/2$
    - Matricea de covariaţie ***C*** este definită prin:

$$c_{ij} = \begin{cases} \sigma_i^2, & \text{dacă } i = j \\ 0, & \text{dacă } i \text{ şi } j \text{ nu sunt corelate} \\ \frac{1}{2}\left(\sigma_i^2 - \sigma_j^2\right) * \tan(2\omega_{ij}), & \text{dacă } i \text{ şi } j \text{ sunt corelate} \end{cases}$$

  - Mutaţie
    - $\sigma'_i = \sigma_i * exp(\tau'*N(0,1)+\tau*N_i(0,1))$
    - $\omega'_{ij} = \omega_{ij}+\beta*N(0,1)$
    - $\boldsymbol{x}' = \boldsymbol{x}+\boldsymbol{N}(0,\boldsymbol{C}')$
    - unde:
      - $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
      - $\boldsymbol{C}'$ – matricea de covariaţie ***C*** după mutarea valorilor ω
      - τ' - rata globală de învăţare
      - τ - rata inteligentă de învăţare
      - de obicei $\tau' = 1/((2n)^{1/2})$ şi $\tau = 1/((2n^{1/2})^{1/2})$ şi $\beta \approx 5°$
    - dacă $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$
    - dacă $|\omega'_{ij}| > \pi \rightarrow \omega'_{ij} = \omega'_{ij} - 2\pi \, \text{sign}(\omega'_{ij})$

Strategii evolutive – reprezentare şi operatori

- Selecţia de supravieţuire
  - Aplicată după crearea a λ descendeţi din μ părinţi prin recombinare şi mutaţie
  - Alegerea celor mai buni μ indivizi din
    - Mulţimea copiilor – SE (μ,λ)
      - Selecţie "uitucă"
      - Are performanţe mai bune
    - Mulţimea părinţilor şi copiilor – SE(μ+λ)
      - Selecţie elitistă
  - De obicei, $\lambda = 7 * \mu$ (➔ presiune de selecţie mare)

Strategii evolutive – proprietăţi

- Caracteristici
  - cromozomi liniari de aceeaşi dimensiune
  - oferă viteză de lucru
  - lucrează cu vectori de numere reale
  - se bazează pe o teorie matematică fundamentată
  - evoluează şi parametrii algoritmului în sine (auto-adaptează parametrii mutaţiei)

- SE iniţiale
  - SE(μ+λ), cu μ=1, λ=1
  - Căutare locală de tip Hill Climbing
  - Dar, cromozomul codează şi:
    - rata de mutaţie
    - strategie de modificare pentru deviaţia standard a distribuţiei mutaţiei

Strategii evolutive - aplicații

- Probleme de optimizare numerică
- Optimizarea formei lentilelor necesare refracţiei luminii
- Distribuţia lichidului într-o reţea sangvină
- Curba Brachystochrone
- Rezolvarea cubului Rubik

Programare evolutivă

- Aspecte teoretice
- Algoritm
  - Schema generală
  - Reprezentare şi operatori
- Proprietăţi
- Aplicaţii

Programare evolutivă – aspecte teoretice

- Propusă
  - în SUA în anii 1960 de către D. Fogel
- Căutare
  - **Concurenţială**, ghidată de calitatea **relativă** a indivizilor → selecţia de supravieţuire
- Operatori de căutare
  - Selecţia
  - **DOAR** mutaţia
- Elemente speciale
  - AE fără recombinare
  - Auto-adaptarea parametrilor (similar SE)

Programare evolutivă – schema generală

Iniţializare P(0)

Evaluare(P(0))

g = 0;

while (*not* condiţie_stop) do

Pentru fiecare cromozom c_i din P(g)

Mutaţie(c_i, *param*) ➔ o_i, *param'*

Evaluare(o_i)

Alegerea probabilistică a μ cromozomi dintre $c_1,\ldots,c_\mu$, $o_1,\ldots,o_\mu$ şi adăugarea lor în *P(g+1)*

g++

endWhile

Programare evolutivă – reprezentare şi operatori

- Reprezentare
  - Reală
  - Codează şi parametrii mutaţiei (pasul de mutaţie)
- Populaţia
  - μ părinţi, $\lambda = \mu$ descendeţi
- Selecţia pentru mutaţie
  - Deterministă
- Mutaţia
  - Perturbare Gaussiană
  - Auto-adaptare a parametrilor
- Selecţia pentru supravieţuire
  - $(\mu+\mu)$ probabilistică

Programare evolutivă – reprezentare şi operatori

- Pp că dorim optimizarea funcţiei $f: R^n \rightarrow R$

- Reprezentarea cromozomilor
  - 2 părţi:
    - Variabile obiect: $x_1, x_2, \ldots, x_n$
    - Paşi de mutaţie: $\sigma_1, \ldots, \sigma_n$
  - Completă
    - $(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$

Programare evolutivă – reprezentare şi operatori

- Selecţia părinţilor (pentru mutaţie)
  - Fiecare părinte produce prin mutaţie un descendent ➔ selecţie
    - deterministă
    - ne-bazată pe calitatea (fitnessul) indivizilor

Programare evolutivă – reprezentare şi operatori

- Mutaţia
  - Singurul operator care introduce variaţie în PE
  - Cromozomul $(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$
  - Modificări de tip Gaussian
    - $\sigma_i' = \sigma_i * (1 + \alpha * N_i(0,1))$
    - $x_i' = x_i + \sigma_i' * N_i(0,1)$
    - $\alpha \approx 0.2$ – rata de învăţare
  - Limitări
    - dacă $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$

Programare evolutivă – reprezentare şi operatori

- Selecţia de supravieţuire
  - Populaţia (la momentul *t*) are *μ* părinţi care produc *μ* descendeţi
  - Campionat "fiecare cu fiecare" (*round-robin)*
    - Fiecare soluţie s_i , $i = 1,2,\ldots,\mu^2$ din cei *μ* părinţi şi *μ* descendeţi este comparată cu alte *q* soluţii (diferite de *s*) alese aleator (din aceeaşi mulţime a părinţilor şi urmaşilor)
    - Pentru fiecare soluţie s_i se stabileşte de câte ori a câştigat un meci jucat

$$p_i = \sum_{l=1}^{q} p_{il},$$

$$p_{il} = \begin{cases} 1, & f(s_i) \text{ e mai bun ca } f(s_l) \\ 0, & \text{altfel} \end{cases}$$

    - Se aleg cele mai bune *μ* soluţii (cu cele mai multe jocuri câştigate - p_i)
  - Parametrul *q* reglează presiunea de selecţie
    - de obicei *q* = 10
  - ➔ procesul de căutare este ghidat de calitatea *relativă* a indivizilor

Programare evolutivă – proprietăţi

- Cromozomi liniari de aceeaşi dimensiune
- Algoritmi evolutivi fără recombinare
- Auto-adaptare a paremtrilor (similar *SE*)
- Cadru foarte permisiv: orice reprezentare şi mutaţie poate funcţiona bine
  - Mutaţie uniformă
  - Mutaţie Cauchy
  - Mutaţie Lévy

Programare evolutivă – aplicații

- Învăţare automată cu maşini cu stări finite
- Optimizare numerică
- Distribuţia şi planificarea traficului în reţele
- Proiectarea farmaceutică
- Epidemiologie
- Detecţia cancerului
- Planificare militară
- Procesarea semnalelor

Programare genetică

- Aspecte teoretice
- Algoritm
  - Schema generală
  - Reprezentare şi operatori
- Exemplu
- Aplicaţii

Programare genetică – aspecte teoretice

- Propusă
  - În SUA în anii 1990 de către J. Koza
  - Evoluarea de programe ➔ evaluarea unui individ implică execuţia programului codat în cromozom
- Căutare
  - **Concurenţială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
  - Selecţia
  - Recombinarea **SAU** mutaţia
- Elemente speciale
  - Cromozomi ne-liniari (arbori sau grafe) şi de dimensiuni diferite
  - Pot folosi mutaţia (dar nu e neapărat necesar)

Programare genetică – schema generală

Iniţializare P(0)
Evaluare(P(0))
g = 0;
while (*not* condiţie_stop) do
    repeat
        Selectarea a 2 părinţi p_1 şi p_2 din *P(g)*
        Încrucişare(p_1, p_2) ➔ o_1 şi o_2
        Mutaţie(o_1) ➔ o_1*
        Mutaţie(o_2) ➔ o_2*
        Evaluare(o_1*)
        Evaluare(o_2*)
        Adăugare o_1* şi o_2* în *P(g+1)*
    until P(g+1) este plină
    g++
endWhile

Programarea genetică – reprezentare şi operatori

- Reprezentare
  - Structuri arborescente de dimensiune variabilă
- Populaţia
  - μ părinţi, μ descendeţi
- Selecţia pentru recombinare
  - Propoţională cu fitness-ul
- Recombinarea
  - Schimbul de sub-arbori
- Mutaţia
  - Schimbări aleatoare în arbore
- Selecţia pentru supravieţuire
  - Schema generaţională - toţi descendenţii înlocuiesc părinţii
  - Schema steady-state – cu elitism

Programarea genetică – reprezentare şi operatori

- Reprezentare
  - Potenţialele soluţii sub forma unor arbori ➔ implicaţii:
    - Indivizi adaptivi
      - Dimensiunea cromozomilor nu este prefixată
      - Dimensiunea cromozomilor depinde de adâncimea şi factorul de ramificare al arborilor
    - Gramatici specifice domeniului problemei de rezolvat
      - Necesitatea definirii exacte a unei gramatici reprezentative pentru problema abordată
      - Gramatica trebuie să permită reprezentarea oricărei soluţii posibile/potenţiale

Programarea genetică – reprezentare şi operatori

- Reprezentare
  - Gramatica conţine:
    - Setul de terminale specifică toate variabilele şi constantele problemei
    - Setul de funcţii conţine toţi operatorii care pot fi aplicaţi terminalelor:
      - Operatori aritmetici (+,-,*,/,sin, cos, log, ...)
      - Operatori de tip Boolean (and, or, not, ...)
      - Operatori de tip instrucţiune (if-then, for, while, set,...)
    - Regulile care asigură obţinerea unor soluţii potenţiale valide
  - De ex. arbori care codifică
    - Formule logice
    - Formule aritmetice
    - Programe

Programarea genetică – reprezentare şi operatori

- Reprezentare
  - exemplu de evoluare a unei expresii logice
    - Problemă: să se determine expresia logică identificată prin datele:

	x1	x2	Output
	0	0	0
	0	1	1
	1	0	1
	1	1	0

- Setul de funcţii $F = \{AND, OR, NOT\}$
- Setul de terminale $T = \{x_1, x_2\}$, cu $x_1, x_2 \in \{True, False\}$

- Soluţie: $(x_1\ AND\ NOT\ x_2)\ OR\ (NOT\ x_1\ AND\ x_2)$

Programarea genetică – reprezentare şi operatori

- Reprezentare
  - exemplu de evoluare a unei expresii aritmetice
    - Problemă: să se determine expresia aritmetică identificată prin datele:

X	a	z	Output
1.5	2	0.7	0.52690
0.8	0.25	2	-2.48536
2	1	0.3	-1.21638

- Setul de funcţii $F = \{+,-,*,/, sin, exp, ln\}$
- Setul de terminale $T = \{x,a,z,3.14\}$, cu $x, a,z \in \boldsymbol{R}$

- Soluţie: $y = x*ln(a)+sin(z)/exp(-x)-3.4$

Programarea genetică – reprezentare şi operatori

- Iniţializarea cromozomilor
  - Aleatoare, respectând
    - O limită a adâncimii maxime
    - Semantica dată de gramatică
  - Problema "bloat" – supravieţuirea arborilor foarte mari
  - Metode
    - Metoda *Full* – arbori compleţi
      - Nodurile de la adâncimea $d < D_{max}$ se iniţializează aleator cu o funcţie din setul de funcţii *F*
      - Nodurile de la adâncimea $d = D_{max}$ se iniţializează aleator cu un terminal din setul de terminale *T*
    - Metoda *Grow* – arbori incompleţi
      - Nodurile de la adâncimea $d < D_{max}$ se iniţializează aleator cu un element din *F U T*
      - Nodurile de la adâncimea $d = D_{max}$ se iniţializează aleator cu un terminal din setul de terminale *T*
    - Metoda *Ramped half and half*
      - ½ din populaţie se creează cu metoda *Full*
      - ½ din populaţie se creează cu metoda *Grow*
      - Folosind diferite adâncimi

Programarea genetică – reprezentare şi operatori

- Evaluarea cromozomilor
  - Necesitatea datelor de antrenament (cazuri de testare)
  - Calculul diferenţei între ce trebuie obţinut şi ceea ce se obţine de fapt
    - Expresii de tip Boolean → numărul ieşirilor corect prezise
    - Expresii aritmetice → media pătratelor diferenţelor între ieşirile corecte şi ieşirile prezise
    - Programe → numărul datelor de test corect procesate
  - Criteriul de optim → minimizare
  - Evaluarea poate penaliza:
    - Soluţiile invalide
    - Dimensiunea (prea mare a) arborilor

Programarea genetică – reprezentare şi operatori

- Evaluarea cromozomilor – exemplu
  - Problemă: să se determine expresia logică identificată prin datele:
    - *C = (x_1 AND x_2) OR (NOT x_1 AND x_2)*

	x1	x2	Output real	Output calculat	Eroare = |output real – output calculat|
	0	0	0	0	0
	0	1	1	1	0
	1	0	1	0	1
	1	1	0	1	1
suma					2

Programare genetică – reprezentare şi operatori

- Selecţia pentru reproducere
  - Bazată pe fitness
    - Selecţie proporţională (bazată pe fitness)
    - Selecţie bazată pe ranguri
    - Selecţie prin turnir
  - În populaţii foarte mari
    - Se acordă ranguri indivizilor (pe bază de fitness) şi se stabilesc mai multe grupe
      - Grupa 1: cei mai buni x% din populaţie
      - Grupa 2: restul de (100-x)% din populaţie
    - Alegerea va fi făcută din:
      - grupa 1 în 80% din cazuri
      - grupa 2 în 20% din cazuri
      - Ex.
        - $\mu = 1000$, $x = 32\%$
        - $\mu = 2000$, $x = 16\%$
        - $\mu = 4000$, $x = 8\%$
        - $\mu = 8000$, $x = 4\%$

Programare genetică – reprezentare şi operatori

- Recombinarea (încrucişarea)
  - Cu punct de tăietură

$p_1=(x+y)*(z-sin(x))$

$p_2=xyz+x^2$

$f_1=(x+y)yz$

$f_2=(z-sin(x))x+x^2$

Programare genetică – reprezentare şi operatori

- Mutaţie
  - Mutaţie de tip *Koza* → Înlocuirea unui nod (intern sau frunză) cu un nou sub-arbore

p=(x+y)(z-sin(x))*

*f=(x+y)*sin(x+4)*

Programarea genetică – proprietăţi

- Folosirea cromozomilor ne-liniari
- Necesită lucrul cu populaţii foarte numeroase
  - Algoritmi înceţi

- Comparaţie AG şi PG
  - Forma cromozomilor
    - AG – cromozomi liniari
    - PG – cromozomi ne-liniari
  - Dimensiunea cromozomilor
    - AG – fixă
    - PG – variabilă (în adâncime sau lăţime)
  - Schema de creare a descendenţilor
    - AG – încrucişare şi mutaţie
    - PG – încrucişare sau mutaţie

Programare genetică – aplicații

- Învăţare automată
  - probleme de regresie
    - Predicţii de curs valutar
    - Previziunea vremii
  - probleme de clasificare (învăţare supervizată)
    - Proiectarea circuitelor digitale
    - Recunoaşterea imaginilor
    - Diagnosticare medicală
  - probleme de clusterizare (învăţare nesupervizată)
    - Analiza secvenţelor de ADN
    - Cercetări şi studii de piaţă (segmentarea pieţei)
    - Analiza reţelelor sociale
    - Analiza rezultatelor căutărilor în Internet

Programare genetică – variante

- Linear Genetic Programming
- Gene Expression Programming
- Cartesian Genetic Programming
- Grammatical Evolution
- Multi Expression Programming
- Traceless Genetic Programming

- Mai multe detalii despre GP si variantele sale în cursurile dedicate învăţării automate

Recapitulare

Generational GA	Steady – state GA

Initialization(pop)
Evaluation(pop)
g = 0;
While (not stop_condition) do
Repeat
p1=Selection(pop)
p2=Selection(pop)
Crossover(p1,p2) =>o1 and o2
Mutation(o1) => o1*
Mutation(o2) => o2*
Evaluation(o1*)
Evaluation(o2*)
Add o1* and o2* into popAux
Until popAux is full.
pop ← popAux
EndWhile
	Initialization(pop)
	Evaluation(pop)
	
	While (not stop_condition) do
	
	p1=Selection(pop)
	p2=Selection(pop)
	Crossover(p1,p2) =>o1 and o2
	Mutation(o1) => o1*
	Mutation(o2) => o2*
	Evaluation(o1*)
	Evaluation(o2*)
	Best(o1*,o2*) replaces Worst(pop)
	
	EndWhile

SE	PE

Initialization(pop)
Evaluation(pop)
g = 0;
While (not stop_condition) do
Repeat
p1=Selection(pop)
p2=Selection(pop)
Crossover(p1,p2) =>o1
Mutation(o1,param) => o1*, param*
Evaluation(o1*)
Add o1* into popAux
Until popAux contains λ cromozoms
pop ← Bestμ(popAux) //SE(μ,λ)
pop ← Bestμ(popUpopAux) //SE(μ+λ)
EndWhile
	Initialization(pop)
	Evaluation(pop)
	g = 0;
	While (not stop_condition) do
	For all cromozoms c from pop
	
	Mutation(c,param) => o1*, param*
	Evaluation(o1*)
	Add o1* into popAux
	
	pop ← RoundRobin(popAux)
	
	EndWhile

Recapitulare

- Reprezentare şi fitness
  - Dependente de problemă
- Operatori de căutare
  - Selecţia pentru reproducere şi pentru supravieţuire
    - Dependentă de fitness
    - Independentă de reprezentare
  - Încrucişarea şi mutaţia
    - Dependente de reprezentare
    - Independente de fitness
    - Probabilitatea de încrucişare
      - Acţionează la nivel de cromozom
    - Probabilitatea de mutaţie
      - Acţionează la nivel de genă

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare adversială

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit şi legături utile

- capitolul II.5 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 6 din *H.F. Pop, G. Şerban, Inteligenţă artificială, Cluj Napoca, 2004*
- documentele din directorul *06_adversial_minimax*

Conținut

Jocuri

- Un pic de istorie
- Câteva repere teoretice
- Jocurile şi căutarea
  - Definiţii, componente, clasificare
  - Strategii şi algoritmi de căutare

Jocuri – un pic de istorie

- Provocări ale inteligenţei
  - Dame (Mesopotamia, cc. 3000 îHr.)
  - Go (China, sec. VI î.Hr.)
  - Sah (India, sec. VI d.Hr.)

- Tradiţional, jocurile erau văzute formal, ca o extensie a algoritmilor de căutare
  - Căutarea clasică: un singur agent, încearcă fără piedici să îşi atingă obiectivul
  - Jocurile: căutare în prezenţa unui adversar (agent ostil) + aduce incertitudine

- Jocurile şi IA
  - Proiectarea şi testarea algoritmilor
  - Unul dintre cele mai vechi domenii ale IA

Jocuri – un pic de istorie

- Jucarea jocurilor
  - De către oameni
    - Activitate inteligentă
    - Compararea aptitudinilor
  - De către calculatoare
    - Mediu propice pentru dezvoltarea tehnicilor de IA
      - Joc = problemă
        - structurată (succes sau eşec)
        - rezolvabilă prin căutare (simplă sau euristică)
    - Limite

Jocuri – câteva repere teoretice

- Probleme dificile cu o structură iniţială minimă de cunoştiinţe
- Stări şi acţiuni uşor de reprezentat
- Puţine cunoştiinţe necesare despre mediu
- Jocurile sunt un caz particular al problemelor de căutare
- Deseori au spaţii de căutare foarte mari
  - Complexitatea jocurilor → incertitudine
    - datorată insuficienţei de timp pentru a calcula consecinţele tuturor mutărilor şi
    - nu lipsei de informaţie
- Jocurile sunt interesante ☺

Jocurile şi căutarea

- Formalizare
- Reprezentarea jocurilor
- Tipuri de jocuri
- Spaţiul de căutare
  - Reprezentare
  - Metode de explorare

Jocurile şi căutarea – formalizare

- Rezolvarea unei probleme de căutare
  - Etapa x: definirea spaţiului de căutare
    - Spaţii liniare
    - Spaţii arborescente
      - Arbori
      - Grafe
  - Etapa x + 1: alegerea unei strategii de căutare
    - Care mutare/strategie?
      - Care determină câştigarea jocului
      - Care poate fi determinată cât mai repede (complexitate temporală redusă)
      - Care poate fi determinată cu efort fizic cât mai mic (complexitate spaţială redusă)

- Problema:
  - cum se poate determina cea mai bună mutare următoare într-un timp cât mai scurt?
- O soluţie:
  - căutarea între mutările posibile şi consecinţele lor

Jocurile şi căutarea – formalizare

- Căutarea adversială este folosită în jocurile în care unii jucători încearcă să-şi maximizeze scorul, dar se confruntă cu unul sau mai mulţi adversari

- Reprezentarea jocurilor ca probleme de căutare
  - **Stări**
    - configuraţiile (tablei) de joc + jucătorul care trebuie să mute
    - totalitatea stărilor posibile → arborele de căutare
  - **Operatori (acţiuni, funcţie succesor)**
    - mutările permise
  - **Starea iniţială**
    - configuraţia iniţială a jocului
  - **Starea scop (finală)**
    - configuraţia (câştigătoare) care termină jocul
  - **Funcţie de utilitate (funcţie scor)**
    - asociază o valoare numerică unei stări

- Dificultăţi
  - Jocurile pot fi probleme de căutare foarte dificile
    - totuşi usor de formalizat
  - Găsirea soluţiei optime poate fi nefezabilă
    - o soluţie care învinge adversarul este acceptabilă
    - o soluţie „inacceptabilă” nu numai că induce costuri mai mari, dar provoacă înfrângerea

Jocurile şi căutarea – formalizare

- Definiţii cheie
  - Situaţie conflictuală
    - Situaţie în care acţionează mai multe părţi care au scopuri contrare
    - Consecinţa acţiunii unei părţi depinde de reacţia celorlalte părţi
  - Joc
    - Modelare simplificată a unei situaţii conflictuale
    - Înşiruire de decizii (acţiuni, mutări) luate de părţi cu interese contrastante
  - Mutare
    - Funcţie H : {poziţiile jocului} → {poziţiile jocului}
    - Dacă p – o poziţie în joc, H(p) – o nouă poziţie
  - Reguli
    - Sistem de condiţii privind mutările posibile
  - Strategie
    - Ansamblu de reguli care definesc mutările libere în funcţie de situaţia concretă ivită

Jocurile şi căutarea – reprezentare

- Componente
  - Jucători
  - Mutări (strategii)
  - Criterii de câştig

- Forme de reprezentare pentru un joc
  - Forma strategică → matrice
    - Jucătorii
    - Strategiile
    - Câştigurile asociate fiecărui jucător şi fiecărei strategii
  - Forma extinsă → arbore
    - Nivel → jucător
    - Nod → alegere (mutare)

Jocurile şi căutarea – reprezentare

- Forma strategică → matrice
  - Ex. Dilema prizonierului: doi prizonieri sunt chestionaţi de poliţie. Poliţia ştie ceva despre ceea ce au făcut ei, dar nu are toate informaţiile. Pentru a afla, cei doi prizonieri sunt închişi în două celule şi sunt interogaţi. Prizonierii au 2 opţiuni:
    - Pot spune toată povestea (pot să se trădeze unul pe altul)
    - Pot să nu spună nimic (pot să coopereze)

    Nici un prizonier nu ştie ce va răspunde celălalt. În funcţie de răspunsurile lor, pedepsele sunt următoarele:
    - Dacă amândoi tac (cooperează), sentinţa este uşoară (1 an fiecare)
    - Dacă unul vorbeşte (trădează) şi unul tace (cooperează), trădătorul este eliberat, iar pârâtul primeşte 10 ani
    - Dacă amândoi vorbesc (se trădează reciproc), sentinţa este de 5 ani fiecare

    Ce vor face cei doi prizonieri?

J1 \ J2	Cooperare	Trădare
Cooperare	(1,1)	(10,0)
Trădare	(0,10)	(5,5)

Jocurile şi căutarea – reprezentare

- Forma strategică → matrice
  - Ex. Vânătoarea de cerbi: doi indivizi merg la vânătoare. Fiecare poate alege să vâneze:
    - un cerb sau
    - un iepure,
  - fără însă să ştie ce a ales celălalt individ. Vânarea
    - cerbilor, mai profitoare (câştig=4), se poate face doar în doi,
    - iepurilor, mai puţin valoroasă (câştig=1), se poate face individual.
  - Ce vor alege să vâneze cei doi indivizi?

J1 \ J2	Cerb	Iepure
Cerb	(4,4)	(1,3)
Iepure	(3,1)	(3,3)

Jocurile şi căutarea – reprezentare

- Forma strategică → matrice
  - Ex. economic
    - Reclama făcută de 2 firme pe aceeaşi piaţă
    - Înţelegerile la nivel de cartel pentru stabilirea preţurilor
  - Ex. sportiv
    - Doi ciclişti aflaţi în faţa plutonului poartă consecutiv trena (cooperează) pentru a nu fi ajunşi din urmă. De multe ori, doar unul duce trena (cooperează), iar la linia de sosire este trădat de adversar.

  - Ex. sociologic
    - Cand cunoaştem o nouă persoană, tindem să fim foarte atenţi pentru a avea o poziţie de siguranţă (competiţie). Ambele persoane pot semnala dorinţa de a se muta de la poziţiile defensive către
      - interacţiune şi
      - recunoaşterea unei intenţii comune.

Jocurile şi căutarea - tipologie

- Numărul jucătorilor
  - 1
  - 2
  - Mai mulţi
- Comunicarea între jucători
  - Cooperative
  - Ne-cooperative
- Strategiile de joc urmate
  - Simetrice
  - Asimetrice

Jocurile şi căutarea – tipologie

- Câştigurile jucătorilor
  - Sumă zero
    - câştigul unui jucător = pierderea celorlaţi jucători → un jucător trebuie să câştige sau jocul se sfârşeşte cu remiză
  - Sumă diferită de zero
    - câştigul unui jucător ≠ pierderea celorlaţi jucători

- Natura mutărilor efectuate
  - Cu mutări libere
    - mutarea este aleasă conştinet dintr-o mulţime de acţiuni posibile → jocuri deterministe
  - Cu mutări întâmplătoare
    - factor aleator (zar, cărţi de joc, monede) → jocuri non-deterministe

Jocurile şi căutarea – tipologie

- Informaţiile despre joc
  - Cu informaţie perfectă
    - un jucător, înainte de a executa o mutare, cunoaşte rezultatele tuturor mutărilor precedente (ale lui şi ale adversarilor);
    - de obicei, jocul e cu mutări secvenţiale
  - Cu informaţie imperfectă
    - un jucător nu cunoaşte toate efectele mutărilor precedente

Jocurile şi căutarea – tipologie

	Deterministic	Non-deterministic (aleator)
Informaţie perfectă		
Informaţie imperfectă	Vaporaşe/Avioane/ Submarine	

Jocurile şi căutarea – spaţiul de căutare

- Reprezentare
  - Spaţiu liniar
  - Spaţiu arborescent → Arborele jocului
    - Identificarea strategiei de câştig → explorarea întregului arbore
    - foarte mare pt anumite jocuri
      - Şah („drosofila IA”)
        - factor de ramificare ≈ 35
        - ≈ 50 de mutări pe jucător
        - ≈ 35^{100} (10^{154}) noduri
        - 10^{40} noduri distincte (dimensiunea grafului de căutare)
      - Go
        - ≈ 200 mutări/stare, 300 niveluri
        - 200^{300} (10^{690}) noduri în arbore
    - exemplu – XO (Tic Tac Toe)

Jocurile şi căutarea – spațiul de căutare

- Strategia de joc (a unui jucător)
  - Definire
    - ansamblul mutărilor unui jucător care
    - ţine cont de:
      - regulile jocului
      - starea curentă a jocului
    - ≠ mutare
  - Tipologie
    - Scop
      - Strategii pas cu pas
        - În fiecare etapă a jocului se identifică mutarea cea mai bună
      - Strategii complete
        - Se identifică o succesiune de mutări

Jocurile şi căutarea – spațiul de căutare

- Strategia de joc
  - Pas cu pas
    - Ex.: XO, Dame, Şah
    - Algoritmi – pot lucra cu structuri:
      - Liniare
        - Strategia simetriei
        - Strategia perechilor
        - Strategia parităţii
        - Programare dinamică
        - Alte strategii
      - Arborescente
        - Arbori AndOr
        - MiniMax (cu tăieturi Alpha-Beta)
  - Completă
    - Ex.: ieşirea dintr-un labirint, deplasarea pe o hartă între 2 locaţii date
    - Algoritmi pot să identifice
      - Un drum optim de la o locaţie la alta
      - O succesiune de acţiuni care să deplaseze jucătorul de la o locaţie la alta

Jocurile şi căutarea – spațiul de căutare

- Strategia de joc
  - Pas cu pas
    - Ex.: XO, Dame, Şah
    - Algoritmi – pot lucra cu structuri:
      - **Liniare**
        - Strategia simetriei
        - Strategia perechilor
        - Strategia parităţii
        - Programare dinamică
        - Alte strategii
      - Arborescente
        - Arbori AndOr
        - MiniMax (cu tăieturi Alpha-Beta)
  - Completă
    - Ex.: ieşirea dintr-un labirint, deplasarea pe o hartă între 2 locaţii date
    - Algoritmi pot să identifice
      - Un drum optim de la o locaţie la alta
      - O succesiune de acţiuni care să deplaseze jucătorul de la o locaţie la alta

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia …

- Exemplu: jocul *haşurează căsuţe*
  - Se dă:
    - O bandă de hârtie este împărţită în *n* căsuţe. Alternativ doi jucători A şi B haşurează câte maxim *k* căsuţe adiacente, nehaşurate (*n* şi *k* au aceaşi paritate). Cel care nu mai poate muta pierde. Iniţial mută jucătorul A.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Caz concret:
    - n = 10, k = 4

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia simetriei

- Exemplu: jocul *haşurează căsuţe*
  - Se dă:
    - O bandă de hârtie este împărţită în *n* căsuţe. Alternativ doi jucători A şi B haşurează câte maxim *k* căsuţe adiacente, nehaşurate (*n* şi *k* au aceaşi paritate). Cel care nu mai poate muta pierde. Iniţial mută jucătorul A.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Caz concret:
    - n = 10, k = 4

  - Soluţie
    - Dacă n – nr. par
      - Prima mutare → jucătorul A haşurează un nr. par de căsuţe din mijlocul benzii
      - La următoarele mutări jucătorul A îl imită pe jucătorul B simetric faţă de mijlocul benzii
    - Dacă n – nr. impar
      - Prima mutare → jucătorul A haşurează un nr. impar de căsuţe din mijlocul benzii
      - La următoarele mutări jucătorul A îl imită pe jucătorul B simetric faţă de mijlocul benzii

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia simetriei

- Aspecte teoretice
  - Jucătorul B imită mutările jucătorului A pe baza unei (unui) axe (centru) de simetrie
  - Dacă A mai poate muta, atunci şi B mai poate muta
    - Jocul se termină când A nu mai poate muta
  - E posibil ca prima mutare să nu aibă simetric

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia …

- Exemplu - Jocul *pătratelor alunecătoare*
  - Se dă:
    - În fiecare căsuţă a unei table pătratice *n***n* (*n* – nr. impar)se află un pătrat roşu sau negru astfel încât tabla se aseamănă cu o tablă de şah. Căsuţa din mijlocul tablei este goală (nu conţine un pătrat). Alternativ, doi jucători A (cu pătratele roşii) şi B (cu pătratele negre) mută câte unul din pătratele lor în căsuţa liberă de pe tablă. Pătratul mutat trebuie să se afle iniţial într-o căsuţă vecină (pe orizontală sau verticală) cu căsuţa liberă. Jucătorul care nu mai poate muta nici un pătrat de-al lui pierde jocul. Jucătorul B mută primul.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Caz concret:
    - n = 5 (jocul propus iniţial de către G.W.Lewthwaite)

a b c

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia perechilor

- Exemplu - Jocul *pătratelor alunecătoare*
  - Se dă:
    - În fiecare căsuţă a unei table pătratice *n*n* (*n* – nr. impar)se află un pătrat roşu sau negru astfel încât tabla se aseamănă cu o tablă de şah. Căsuţa din mijlocul tablei este goală (nu conţine un pătrat). Alternativ, doi jucători A (cu pătratele roşii) şi B (cu pătratele negre) mută câte unul din pătratele lor în căsuţa liberă de pe tablă. Pătratul mutat trebuie să se afle iniţial într-o căsuţă vecină (pe orizontală sau verticală) cu căsuţa liberă. Jucătorul care nu mai poate muta nici un pătrat de-al lui pierde jocul. Jucătorul B mută primul.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Caz concret:
    - n = 5 (jocul propus iniţial de către G.W.Lewthwaite)

a b c

  - Soluţie
    - Se împarte tabla în *Domino-uri*, căsuţa din mijlocul tablei (iniţial goală) nefacând parte din nici un *Domino*.
    - După fiecare mutare a jucătorului B, căsuţa liberă se află acoperită de acelaşi *Domino* care acoperă şi o piesă de culoare roşie. Această piesă o va muta A, având în acest fel întotdeauna ceva de mutat.
    - acoperirea cu *Domino-uri* în spirală, plecându-se din colţul stânga sus, spre dreapta

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia perechilor

- Aspecte teoretice
  - Generalizare a strategiei simetriei
  - Gruparea mutărilor în perechi:
    - (mutare jucător A, mutare jucător B)
  - Dacă A mai poate muta, atunci şi B mai poate muta
    - Jocul se termină când A nu mai poate muta
  - E posibil ca prima mutare să nu poată fi împerecheată

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia parităţii

- Exemplu - Jocul *NIM*
  - Se dau:
    - N stive, fiecare conţinând p_i obiecte. 2 jucători, A şi B, extrag, alternativ, dintr-o singură stivă, oricâte obiecte. Jucătorul care execută ultima extragere câştigă jocul. Jucătorul A mută primul.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Exemplu:
    - 4 stive cu 2, 2, 4 şi, respectiv, 7 obiecte

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia parităţii

- Exemplu - Jocul *NIM*
  - Se dau:
    - N stive, fiecare conţinând p_i obiecte. 2 jucători, A şi B, extrag, alternativ, dintr-o singură stivă, oricâte obiecte. Jucătorul care execută ultima extragere câştigă jocul. Jucătorul A mută primul.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Exemplu:
    - 4 stive cu 2, 2, 4 şi, respectiv, 7 obiecte

  - Soluţie
    - Nr de obiecte din fiecare stivă se reprezintă binar (ca sumă a puterilor lui 2)
      - fiecare număr natural admite o descompunere unică ca sumă de puteri distincte ale lui 2
    - Stare pară – toate puterile lui 2 din reprezentarea jocului apar de un număr par de ori (în toate stivele)
    - Stare impară – orice stare care nu este pară
      - T1 ➔ dintr-o stare impară se ajunge într-o stare pară printr-o mutare convenabilă
      - T2 ➔ dintr-o stare pară se ajunge într-o stare impară prin orice fel de mutare
    - Extragerea de pe stiva care conţine cel mai seminificativ bit pe poziţia k
      - K poziţia celui mai semnificativ bit in suma NIM a tuturor stivelor

Strategii de joc → Strategia parităţii

- Aspecte teoretice
  - 2 tipuri de poziţii în joc
    - Poziţie pară (singulară)
    - Poziţie impară (nesingulară)
  - Teoreme
    - T1: poziţia impară, printr-o mutare **convenabilă**, → poziţie pară (jucătorul A)
    - T2: poziţia pară, prin **orice** mutare, → poziţie impară (jucătorul B)
  - Câştigător
    - → poziţia finală = poziţie pară

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia parităţii

- Exemplu - Jocul *NIM*
  - Demonstrarea celor 2 teoreme T1 şi T2
    - S_1: $2 = 2^1$, S_2: $3 = 2^1 + 2^0$, S_3: $4 = 2^2$, S_4: $5 = 2^2 + 2^0$
    - $2^0 - 2$, $2^1 - 2$, $2^2 - 2$ → stare pară
    - Extragerea dintr-o stivă cu număr par de obiecte
      - A unui număr par de obiecte (ex. 2 obiecte din S_3) = >
        - S_1: $2 = 2^1$, S_2: $3 = 2^1 + 2^0$, S_3: $4-2 = 2^1$, S_4: $5 = 2^2 + 2^0$
        - $2^0 - 2$, $2^1 - 3$, $2^2 - 1$ ➔ stare impară
      - A unui număr impar de obiecte (ex. 1 obiect din S_1)
        - S_1: $2-1 = 2^0$, S_2: $3 = 2^1 + 2^0$, S_3: $4 = 2^2$, S_4: $5 = 2^2 + 2^0$
        - $2^0 - 3$, $2^1 - 1$, $2^2 - 2$ ➔ stare impară
    - Extragerea dintr-o stivă cu număr impar de obiecte
      - A unui număr par de obiecte (ex. 2 obiecte din S_4) = >
        - S_1: $2 = 2^1$, S_2: $3 = 2^1 + 2^0$, S_3: $4 = 2^2$, S_4: $5-2 = 2^1 + 2^0$
        - $2^0 - 2$, $2^1 - 3$, $2^2 - 1$ ➔ stare impară
      - A unui număr impar de obiecte (ex. 1 obiect din S_2)
        - S_1: $2 = 2^1$, S_2: $3-1 = 2^1$, S_3: $4 = 2^2$, S_4: $5 = 2^2 + 2^0$
        - $2^0 - 1$, $2^1 - 2$, $2^2 - 2$ → stare impară

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Strategia parităţii

- Exemplu - Jocul *NIM*
  - algoritm
    - Pp. următorul joc:
      - S_1: 3 = 2^1+2^0 → 011, S_2: 4 = 2^2 → 100, S_3: 5 = 2^2+2^0 → 101
    - Paşi:
      1. Se calculează suma Nim (S_{Nim}) a tuturor stivelor
         - S_{Nim} = S_1 XOR S_2 XOR S_3 = 3 XOR 4 XOR 5 = 011 XOR 100 XOR 101 = 010= 2
      2. Se identifică poziţia *k* a celui mai reprezentativ 1 în suma Nim
         - S_{Nim} = 2 = $010_{(2)}$ ➔ poziţia a 2-a (k = 2)
      3. Se caută stiva cu cel mai reprezentativ bit pe poziţia k (stiva care descreşte dacă se face XOR între ea si suma Nim)
         - S_1: 3 = 010, S_2: 4 = 100, S_3: 5 = 101 => S_1 sau
         - S_1 XOR S_{Nim} = 3 XOR 2 = 011 XOR 010 = 001 = 1
         - S_2 XOR S_{Nim} = 4 XOR 2 = 100 XOR 010 = 110 = 6
         - S_3 XOR S_{Nim} = 5 XOR 2 = 101 XOR 010 = 111 = 7 ➔ S_1
      4. Se extrage din această stivă un număr de obiecte a.î. restul stivelor să formeze o stare pară; nr de obiecte care rămân pe stivă este dat de XOR-ul între stiva respectivă şi suma Nim
         - S_1 XOR S_{Nim} = 3 XOR 2 = 011 XOR 010 = 001 = 1 ➔ se extrag 2 (=3 – 1) obiecte de pe stiva 1
      5. Se reia pasul 1

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Aspecte teoretice
  - Paşi în rezolvarea unei probleme cu PD:
    - Descompunerea problemei în sub-probleme
    - Rezolvarea sub-problemelor
    - Combinarea sub-soluţiilor pentru obţinerea soluţiei finale
  - Paşi în rezolvarea unui joc cu PD:
    - Descompunerea jocului în sub-jocuri
    - Găsirea unei strategii perfecte de câştig pentru fiecare sub-joc
    - Combinarea sub-strategiilor pentru obţinerea strategiei finale

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Aspecte teoretice
  - Cum se rezolvă un joc cu programare dinamică?
    - descompunerea jocului în sub-jocuri J_h (sub-jocuri de pe nivelul cel mai de jos) şi
      - etichetarea fiecărui joc J_h cu T (*true*) sau F (*false*) în funcţie de posibilitatea jucătorului (care trebuie să mute din acea poziţie) de a avea strategie sigură de câştig
    - un sub-joc J_k de pe un nivel interior ($k < h$) va fi etichetat cu:
      - T, dacă există cel puţin un sub-joc J_i, $k < i$, etichetat cu F, în care poate fi transformat jocul J_k
      - F, dacă toate sub-jocurile J_i, $k < i$, în care se poate ajunge printr-o mutare din jocul J_k sunt etichetate cu T

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *şirului de căsuţe*
  - Se dă:
    - În cele *n* căsuţe ale unui şir se află amplasate cuburi (maxim un cub în fiecare căsuţă). Scopul jocului este umplerea tuturor căsuţelor cu cuburi. Alternativ, doi jucători A şi B umplu cu cuburi (complet sau parţial) cel mai din stânga şir de căsuţe libere. Jucătorul care nu mai poate muta pierde. Jucătorul A mută primul.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Exemplu
    - n = 8

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *şirului de căsuţe*
  - Soluţie:

T

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *şirului de căsuţe*
  - Soluţie:

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *şirului de căsuţe*
  - Soluţie:

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *şirului de căsuţe*
  - Soluţie:

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc → Programare dinamică (PD)

- Exemplu - Jocul *şirului de căsuţe*
  - Soluţie:

Jocurile şi căutarea – spaţiul de căutare

- Strategia de joc
  - Pas cu pas
    - Ex.: XO, Dame, Şah
    - Algoritmi – pot lucra cu structuri:
      - Liniare
        - Strategia simetriei
        - Strategia perechilor
        - Strategia parităţii
        - Programare dinamică
        - Alte strategii
      - **Arborescente**
        - **Arbori AndOr**
        - MiniMax (cu tăieturi Alpha-Beta)
  - Completă
    - Ex.: ieşirea dintr-un labirint, deplasarea pe o hartă între 2 locaţii date
    - Algoritmi pot să identifice
      - Un drum optim de la o locaţie la alta
      - O succesiune de acţiuni care să deplaseze jucătorul de la o locaţie la alta

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc - Arbori And-Or (AAO)

- Aspecte teoretice
  - **Reprezentarea spaţiului de căutare** cu ajutorul AAO pentru un joc cu 2 jucători:
    - Partea (nodul) OR
      - → selectarea unei mutări pentru jucătorul curent A
      - → există o posibilitate (mutare) pentru jucătorul A să ajungă într-un nod AND
    - Partea (nodul) AND
      - → considerarea tuturor mutărilor posibile ale adversarului (jucătorul B)
      - → prin orice mutare jucătorul B ajunge într-un nod OR
    - Astfel:
      - Rădăcina → problema jucătorului câştigător (care începe jocul din starea iniţială)
      - Mutările posibile ale primului jucător (A) se reunesc prin disjuncţie (OR)
      - Mutările posibile ale celui de-al doilea jucător (B) se reunesc prin conjuncţie (AND)
    - Jocul poate fi câştigat dacă începe cu un nod OR
  - **Dificultăţi:**
    - Arbori foarte foarte mari

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc - Arbori And-Or (AAO)

- Aspecte teoretice

  - Paşi în rezolvarea unui joc cu AAO:

    - Descompunerea jocului în sub-jocuri
      - noduri pe mai multe nivele,
      - nodurile de pe un nivel corespunzând mutărilor posibile ale unui jucător

    - Găsirea unei strategii perfecte de câştig pentru fiecare sub-joc (nod)
      - Etichetarea nodurilor cu T sau F în funcţie de posibilitatea jucătorului A de a avea o strategie sigură de câştig pentru acel sub-joc
      - Reguli de etichetare:
        - Frunzele se etichtează cu T sau F în funcţie de configuraţia jocului
        - Nodurile interne se etichetează:
          - Pentru jucătorul A, cu T dacă **cel puţin un nod** fiu a fost etichetat cu T – regula OR
          - Pentru jucătorul B, cu T dacă **toate nodurile** fiu au fost etichetat cu T – regula AND

    - Combinarea sub-strategiilor pentru obţinerea strategiei finale

Strategii de joc - Arbori And-Or (AAO)

- Exemplu

A (OR)

B (AND)

A (OR)

B (AND)

A (OR)

Jocurile şi căutarea–spaţiul de căutare

Strategii de joc - Arbori And-Or (AAO)

- Algoritm

```
bool backAndOr(Node N, int level){
        //level = 0 for the node in the top of the tree.
    if (N is a terminal) {
        if (the first player won)
                return true;
        else
                return false;
    }
    else{
        if (level % 2){         //B is about to move; AND
                result = true;
                for each child Ni of N
                        result = result && backAndOr(Ni, level+1);
        }
        else{                   // A is about to move; OR
                result = false;
                for each child Ni of N
                        result = result || backAndOr(Ni, level+1);
        }
        return result;
    }
}
```

Strategii de joc - Arbori And-Or (AAO)

- Avantaje
  - Pot fi aplicaţi pentru rezolvarea oricărui joc
- Dezavantaje
  - Necesită multă memorie
  - Necesită mult timp de calcul
- ➔algoritmul mini-max

Jocurile şi căutarea – spațiul de căutare

- Strategia de joc
  - Pas cu pas
    - Ex.: XO, Dame, Şah
    - Algoritmi – pot lucra cu structuri:
      - Liniare
        - Strategia simetriei
        - Strategia perechilor
        - Strategia parităţii
        - Programare dinamică
        - Alte strategii
      - Arborescente
        - Arbori AndOr
        - **MiniMax** (cu tăieturi Alpha-Beta)
  - Completă
    - Ex.: ieşirea dintr-un labirint, deplasarea pe o hartă între 2 locaţii date
    - Algoritmi pot să identifice
      - Un drum optim de la o locaţie la alta
      - O succesiune de acţiuni care să deplaseze jucătorul de la o locaţie la alta

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Aspecte teoretice
  - Propus de John von Neuman în 1944
  - Ideea de bază: maximizarea poziţiei unui jucător în timp ce poziţia adversarului este minimizată
  - Arborele de căutare constă în alternarea nivelelor pe care un jucător încearcă să-şi maximizeze câştigul cu nivelele pe care adversarul minimizează câştigul (primului jucător)
    - Primul jucător va încerca să-şi maximizeze câştigul (jucătorul MAX → mută primul)
    - Al doilea jucător va încerca să minimizeze câştigul primului jucător (jucătorul MIN → adversarul)
  - Identificarea celor mai bune mutări
    - Se construieşte arborele tuturor mutărilor posibile (fiecare mutare se aplică unei stări a jocului)
    - Se evaluează frunzele (care jucător câştigă)
    - Se propagă (în sus în arbore) câştigurile
    - Explorarea arborelui este de tip *Depth first search*
  - Generarea tuturor mutărilor
    - Posibilă doar în jocuri simple (ex. Tic-Tac-Toe)
    - Imposibilă (cu resurse limitate) pentru jocuri complexe
      - reţinând minimele în MIN
      - reţinând maximele în MAX

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Algoritm
  - Se construieşte arborele corespunzător tuturor mutărilor
  - Se evaluează frunzele
  - Cât timp se mai poate alege un nod cu toţi descendeţii evaluaţi
    - Se alege un nod
    - Dacă nodul este de pe nivel Min, el va fi evaluat la cea mai mică valoare a unui descendent
    - Dacă nodul este de pe nivel Max, el va fi evaluat la cea mai mare valoare a unui descendent
  - Se returnează valoarea nodului rădăcină (nod de pe nivel Max)

```
int backMiniMax(Node N, int level){
        //level = 0 for the node in the top of the tree.
    if (level == MaxLevel)
        return the quality of N computed with a heuristic;
    else if (level < MaxLevel){
        if (level % 2){
                                //B is about to move; minimize
                result = MaxInt;
                for each child Ni of N
                                result = minim(result, backMiniMax(Ni, level+1));
        }
        else{
                                // A is about to move; Maximize
                result = -MaxInt;
                for each child Ni of N
                                result = maxim(result, backMiniMax(Ni, level+1));
        }
        return result;
    }
}
```

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – jocul *Tic-Tac-Toe*

  - Funcţia de evaluare a unui nod:
    - Dacă există o linie aproape completă (cu 2 semne la fel) → 200 puncte
    - Dacă există 2 linii aproape complete (cu 2 semne la fel) → 300 puncte
    - O linie completă → 600 puncte
    - Fiecare linie potenţială → 1 punct
    - Punctele jucătorului care urmează la mutare se adună
    - Punctele celuilalt jucător se scad

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – jocul *Tic-Tac-Toe*

Mută O → max

Mută X → min

Mută O → max

Mută X → min

Mută O → max

-399

Mută X → min

200+1+1-600-1

-399 101 0 -99

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – *Jocul NIM*
  - Se dă:
    - N stive conţin fiecare p_i obiecte. 2 jucători A şi B extrag, alternativ, dintr-o singură stivă oricâte obiecte. Jucătorul care execută ultima extragere câştigă jocul. Jucătorul A mută primul.
  - Se cere:
    - Să se determine dacă jucătorul A are strategie sigură de câştig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.
  - Exemplu:
    - 3 stive cu 1, 1 şi, respectiv, 2 obiecte

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – jocul NIM

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – jocul NIM

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – jocul NIM

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – jocul NIM

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Exemplu – jocul NIM

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax

- Dificultăţi
  - Nu explorează întregul arbore
    - Căutare depth-first
  - La începutul jocului se fixează:
    - O adâncime maximă
      - Care este adâncimea optimă?
        - Adâncime mare → căutare lungă
        - Adâncime mică → anumite drumuri pot fi ratate (sacrificii timpurii pentru câştiguri târzii)
    - O funcţie de evaluare
      - Fiecare nod (stare) trebuie evaluat(ă)
      - Cum? ➔ folosind euristici

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax

- Dezavantaje
  - Doar 100 noduri sunt explorate intr-o secundă
  - Sah – timp mutare = 150s → 150 000 de poziţii → doar 3 - 4 mutări în avans

- Soluţia
  - Evitarea anumitor noduri prin retezarea unor ramuri (redundante) ➔ retezare (reducere) alpha-beta ➔ MiniMax cu retezare α-β

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Minimax
  - Crează întreg arborele
  - Se propagă valorile de jos în sus
- MiniMax cu retezare α-β
  - Crearea şi propagarea în acelaşi timp
  - Dacă se ştie că un drum este rău, nu se mai consumă energie ca să se afle cât de rău este acel drum
    - Se pot elimina anumite evaluări inutile
    - Se pot elimina anumite expandări ale nodurilor

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Apecte teoretice
  - Descoperit de McCarthy în 1956, dar publicat pentru prima dată într-un raport tehnic de la MIT
  - Ideea de bază
    - Valoarea minimax a rădăcinii poate fi determinată fără examinarea tuturor nodurilor de pe frontiera căutării
    - Similar algoritmului *branch and bound*
  - Denumirea α-β
    - α - valoarea celei mai bune alegeri (celei mai mari) efectuate de-a lungul drumului urmat de MAX
      - dacă o valoare unui nod este mai slabă decât α atunci MAX o va evita şi va reteza sub-arborele cu rădăcina în acel nod
    - β - similar lui α, dar pentru jucătorul MIN

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Aspecte teoretice
  - α
    - valoarea cele mai bune (adică cea mai mare) alegeri găsită până la momentul curent în orice punct de-a lungul unui drum pentru MAX
    - scorul minim pe care jucătorul MAX îl poate obţine în mod garantat
    - dacă un nod *v* este mai slab (mai mic) decât α, MAX îl va evita prin eliminarea acelei ramuri ➔
      - Dacă s-a ajuns cu căutarea într-un nod MIN a cărui valoare $\leq \alpha$ atunci descendenţii acelui nod nu mai merită explorаţi pentru că oricum MAX îi va ignora
  - β
    - cea mai mică valoare găsită la orice punct de-a lungul unui drum pentru MIN
    - scorul minim pe care jucătorul MAX speră să-l obţină
    - dacă un nod *v* este mai bun (mai mare) decât β, MIN îl va evita prin eliminarea acelei ramuri ➔
      - Dacă s-a ajuns cu căutarea într-un nod MAX a cărui valoare $\geq \beta$ atunci descendenţii acelui nod nu mai merită explorаţi pentru că oricum MIN îi va ignora

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Aspecte teoretice
  - Valoarea α a unui nod
    - Iniţial, scorul acelui nod (dacă este frunză) sau -∞
    - Apoi,
      - Nod de tip MAX = cel mai mare scor al descendenţilor
      - Nod de tip MIN = valoarea α a predecesorului
  - Valoarea β a unui nod
    - Iniţial, scorul acelui nod (daca este frunză) sau +∞
    - Apoi,
      - Nod de tip MIN = cel mai mic scor al descendenţilor
      - Nod de tip MAX = valoarea β a predecesorului
  - Scorul unui nod
    - Nod de tip MAX valoarea α finală
    - Nod de tip MIN valoarea β finală

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Algoritm

```
int backMiniMaxAB(Node N, int A, int B){
    Set Alpha value of N to A and Beta value of N to B;
    if N is a leaf
        return the estimated score of this leaf
    else{
        if N is a Min node
            for each child Ni of N {
                Val = backMiniMaxAB(Ni, Alpha of N, Beta of N);
                Beta value of N = minim(Beta value of N, Val);
                if Beta value of N ≤ Alpha value of N then exit loop;
            }
            return Beta value of N;
        else // N is a Max node
            for each child Ni of N do {
                Val = MINIMAX-AB(Ni, Alpha of N, Beta of N);
                Alpha value of N = Max{Alpha value of N, Val};
                if Alpha value of N ≥ Beta value of N then exit loop;
            }
            Return Alpha value of N;
    }
}
```

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX

MIN

MAX

MIN

MAX

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Exemplu

MAX(α)

MIN(β)

MAX(α)

MIN(β)

MAX(α)

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- Proprietăţi
  - Reducerea nu afectează rezultatul final
  - O bună ordonare a mutărilor îmbunătăţeşte algoritmul de reducere
  - Dacă succesorii sunt puşi perfect în ordine (cei mai buni se află primii), atunci complexitatea temporara ar fi = $O(b^{d/2})$, în loc de $O(b^d)$ cat are minimax
  - Se poate transforma un program de la nivelul începător la nivelul expert

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → MiniMax cu retezare α-β

- MinMax *vs.* MinMax cu retezare α-β

	MinMax	**MinMax α-β**
Complexitate temporală	$O(b^m)$	$O(b^{m/2})$ cu ordonare perfectă a nodurilor
Complexitate spaţială	$O(b^m)$	$O(2b^{d/2})$ – cel mai bun caz (când unui nod Max îi este generat ca prim copil cel cu valoarea cea mai mare şi când lui Min îi este generat ca prim copil cel cu valoarea cea mai mică) $O(b^d)$ – cel mai rău caz (fără retezare)
Completitudine	Da	Da
Optimalitate	Da	Da

Jocurile şi căutarea – spaţiul de căutare

- Arbori AndOR, MiniMax, MiniMax cu retezare α-β
  - Complexitatea mărită ➔ necesitatea unor tehnici eficiente de căutare în rezolvarea jocurilor
    - Jocul de şah
      - b~35
      - d~100
      - $b^d \sim 35^{100} \sim 10^{154}$ noduri
    - Jocul Tic-Tac-Toe
      - ~ 5 mutări legale dintr-un total de 9 mutări
      - 5^9 = 1 953 125
      - 9! = 362 880 (dacă computerul mută primul)
      - 8! = 40 320 (dacă computerul mută al doilea)
    - Jocul Go
      - b > 361 (pentru o tablă de 19x19)

Jocurile şi căutarea – spaţiul de căutare

- În practică
  - Jocul de dame → Chinok
    - Chinok îl invinge pe Marion Tinsley în 1994 (după 40 de ani de confruntări)
    - Se foloseşte o bază cu finaluri de joc calculate pentru toate poziţiile unui joc perfect jucat cu cel mult 8 piese (444 bilioane de poziţii posibile)
  - Jocul de şah → Deep Blue
    - Deep Blue l-a învins pe Garry Kasparov în 1997
    - Se verifică 200 mil poziţii/sec
    - Factorul de ramificare se reduce la 6 cu retezarea α-β (faţă de 35-40)
  - Jocul Othello
    - Jucătorii umani refuză să joace cu computerele (pt că sunt prea bune) – Moor (1980), Logistello (1997)
  - Jocul Go
    - Jucătorii umani refuză să joace cu computerele (pt că sunt prea slabe)
    - Factorul de ramificare > 300 ➔ necesitatea unor algoritmi bazaţi pe pattern-uri
  - Jocul de table
    - BKG (logica fuzzy), TD-Gammon (reţele neuronale artificiale)
    - Computerul l-a învins pe campionul lumii, dar pt că a fost norocos

Jocurile şi căutarea – spațiul de căutare

- Strategia de joc
  - Pas cu pas
    - Ex.: XO, Dame, Şah
    - Algoritmi – pot lucra cu structuri:
      - Liniare
        - Strategia simetriei
        - Strategia perechilor
        - Strategia parităţii
        - Programare dinamică
        - Alte strategii
      - Arborescente
        - Arbori AndOr
        - MiniMax (cu tăieturi Alpha-Beta)
  - Completă
    - Ex.: ieşirea dintr-un labirint, deplasarea pe o hartă între 2 locaţii date
    - Algoritmi pot să identifice
      - **Un drum optim de la o locaţie la alta (pathfinding)**
      - O succesiune de acţiuni care să deplaseze jucătorul de la o locaţie la alta

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Pathfinding

- Aspecte teoretice
  - Problema
    - Identificarea pe o hartă (posibil cu obstacole) a unui drum optim de la o locaţie la alta
    - Unde?
      - pe o hartă
        - harta ca o matrice
        - harta ca un graf (simplu, *mesh, quad-tree*)
      - într-un mediu (cunoscut/necunoscut, static/dinamic)
    - De către cine?
      - un agent
      - 2 agenţi
      - mai mulţi agenţi
  - Soluţia
    - Utilizarea unei metode de căutare (optimizare)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Pathfinding

- Aspecte teoretice
  - Soluţia
    - Cum?
      - Utilizarea unei metode de căutare (optimizare)
    - Care soluţie este cea mai bună?
      - viteza de calcul,
      - lungimea drumului,
      - calitatea drumului,
      - informaţia disponibilă
    - Dificultăţi
      - soluţie oferită în timp real
      - resurse limitate
      - spaţiu de căutare imens
      - modelarea situaţiilor reale implică incertitudini şi elemente (teren, unităţi, agenţi) eterogene

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Pathfinding

- Algoritmi de căutare pentru hărţi reprezentate ca "matrici de dale" → labirinturi

  - De ce?
    - Simplitate
  - Caracteristici
    - Spaţiu de căutare discret
    - Dalele sunt pătrate
    - Dalelele pot fi traversabile sau blocate
    - Mişcări în linie dreaptă de pe o dală pe alta (orizontală, verticală, diagonală)
  - Metode
    - A*
      - Algoritmi de tip *Best-first search*
      - Reprezintă un standard în industria jocurilor
    - Euristici → Manhattan (gen. Minkowski)
      - Estimarea distanţei de la punctul curent la destinaţie
      - Se ignoră obstacolele

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Pathfinding

- Algoritmi de căutare pentru hărţi reprezentate ca un graf
  - De ce?
    - Reprezentarea matriceală → zone mari din hartă care au acelaşi cost
  - Caracteristici
    - Spaţiu de căutare discret
    - "Dalele" pot avea orice formă
    - Dalelele pot fi traversabile sau blocate
    - Mişcări oarecare de pe o dală pe alta
  - Metode
    - Deplasare pe muchii
    - Deplasare pe noduri
      - Puncte
      - Marginile poligonului
      - Mijlocul poligonului
    - Ex. Dijkstra, Floyd-Warshall, Kruskal, etc

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → Pathfinding

- Îmbunătăţiri ale algoritmilor de căutare
  - Euristici mai bune
  - Abstractizări ierarhice
  - Abordări descentralizate

Jocurile şi căutarea – spațiul de căutare

Strategii de joc → Pathfinding

- Îmbunătăţiri ale algoritmilor de căutare
  - Euristici mai bune
    - Euristici fără memorie
      - Ex. Manhattan
      - Foarte rapid de calculat
      - Nu necesită memorie suplimentară
      - Calitate bună uneori
    - Euristici bazate pe memorie
      - Ex. euristici bazate pe repere
      - Destul de rapide de calculat
      - Necesită memorie
      - Calitate bună (identificarea drumurilor moarte)
    - Informaţie perfectă
      - Foarte costisitoare la calculare şi stocare
      - Foarte rapidă la utilizat (după ce au fost calculate)
      - Ne-practicabile

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Pathfinding

- Îmbunătăţiri ale algoritmilor de căutare
  - Abstractizări ierarhice
    - Grafuri în care
      - sunt adnotate şi fluxurile de mişcare (deplasări în ambele sensuri)
      - sunt asociate relaţii de dominanţă între muchii
      - Sunt identificate nivele (ex. cameră, casă, oraş, ţară)
  - Abordări descentralizate
    - Ideea de bază:
      - Descompunerea problemei în sub-probleme care
        - pot fi rezolvate repede
        - pot fi sub-optimale
        - pot fi incomplete
    - Scop
      - toate elementele (identificare colaborativă) să ajungă la destinaţie
    - Dificultate
      - se măreşte spaţiul de căutare: $O(n) \rightarrow O(n^m)$
      - factorul de ramificare explodează (de la 4 sau 8 la 5^m sau 9^m)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Planning

- Aspecte teoretice
  - Problema
    - Planificarea (ordonarea) unor acţiuni
      - mutări, deplasări, alegeri, răsuciri, etc
    - Se dau
      - O stare iniţială

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Planning

- Aspecte teoretice
  - Problema
    - Planificarea (ordonarea) unor acţiuni
      - mutări, deplasări, alegeri, răsuciri, etc
    - Se dau
      - O stare iniţială
      - O stare obiectiv (finală)

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Planning

- Aspecte teoretice
  - Problema
    - Planificarea (ordonarea) unor acţiuni
      - mutări, deplasări, alegeri, răsuciri, etc
    - Se dau
      - O stare iniţială
      - O stare obiectiv (finală)
      - Un set de acţiuni posibile

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Planning

- Aspecte teoretice
  - Problema
    - Planificarea (ordonarea) unor acţiuni
      - mutări, deplasări, alegeri, răsuciri, etc
    - Se dau
      - O stare iniţială
      - O stare obiectiv (finală)
      - Un set de acţiuni posibile
    - Se cere
      - Să se identifice o secvenţă de acţiuni care transformă starea iniţială în starea finală

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Planning

- De ce?
  - jocuri cu NPC *(First-person shooter, Role-playing, Real-time strategy)*
  - Probleme reale de planificare
    - Telescop în spaţiu
    - Roboţei
    - UAV-uri

Jocurile şi căutarea – spaţiul de căutare

Strategii de joc → Planning

- Cum?
  - Algoritmi
    - Pentru alegerea unei acţiuni (shooting)
    - Pentru evaluarea mediului
  - Abordări
    - STRIPS → http://www.dis.uniroma1.it/~degiacom/didattica/dottorato-stavros-vassos/
    - HTN
  - Simularea comportamentului
    - Maşini cu stări finite (FSM)
    - Arbori de comportament (Halo 2)
    - GOAP (FEAR)

Recapitulare

- Definirea unui joc
  - Starea iniţială (cum sunt aşezate iniţial elementele în joc)
  - Acţiunile posibile (care sunt mutările permise)
  - Test terminal (care indică terminarea jocului)
  - Funcţie utilitate (care spune cine şi cât a câştigat)

- Strategii de rezolvare a jocurilor
  - Bazate pe explorarea (aproape) completă a arborelui de joc
    - AndOR ➔ cine câştigă
    - MiniMax ➔ cine şi cât câştigă
    - MiniMax cu retezare α-β ➔ cine şi cât câştigă şi ce mutări nu merită să fie efectuate
  - Bazate pe strategii inteligente
    - Strategia simetriei
    - Strategia perechilor
    - Strategia parităţii
    - Programare dinamică
    - A* (Pathfinding, Planning)

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- capitolul III din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 4 şi 5 din *H.F. Pop, G. Şerban, Inteligenţă artificială, Cluj Napoca, 2004*
- capitolul 2 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 6 şi 7 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare informată

algoritmi inspiraţi de natură

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit şi legături utile

- capitolul 16 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- *James Kennedy, Russel Eberhart, Particle Swarm Optimisation*, *Proceedings of IEEE International Conference on Neural Networks*. **IV**. pp. 1942–1948, 1995 (05_ACO_PSO/PSO_00.pdf)

- *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (05_ACO_PSO/Dorigo05_ACO.pdf)

Căutare locală

- Tipologie
  - Căutare locală simplă - se reţine o singură stare vecină
    - Căutare tabu → reţine lista soluţiilor recent vizitate
    - Hill climbing → alege cel mai bun vecin
    - Simulated annealing → alege probabilistic cel mai bun vecin
  - Căutare locală în fascicol (beam local search) – se reţin mai multe stări (o populaţie de stări)
    - Algoritmi evolutivi
    - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
    - Optimizare bazată pe furnici (Ant colony optmisation)

Algoritmi inspirați de natură

- Care este cea mai bună metodă de rezolvare a unei probleme?
  - Creierul uman
    - A creat roata, maşina, oraşul, etc
  - Mecanismul evoluţiei
    - A creata creierul (mintea) umană

- Simularea naturii
  - Cu ajutorul maşinilor → reţele neuronale artificiale simulează mintea umană
    - Maşini de zbor, computere bazate pe ADN, computere cu membrane
  - Cu ajutorul algoritmilor
    - algoritmii evolutivi simulează evoluţia naturii
    - algoritmii inspiraţi de comportamentul de grup simulează adaptarea colectivă si procesele sociale dintr-un colectiv
      - Particle Swarm Optimisation (PSO)
        - http://www.youtube.com/watch?feature=endscreen&v=JhZKc1Mgub8&NR=1
        - http://www.youtube.com/watch?v=ulucJnxT7B4&feature=related
        - https://www.youtube.com/watch?v=TWqx57CR69c
      - Ant Colony Optimisation (ACO)
        - http://www.youtube.com/watch?v=jrW_TTxP1ow

Algoritmi inspiraţi de natură

- Inteligenţa de grup (colectivă)
  - O populaţie de indivizi care interacţionează în scopul atingerii unor obiective prin adaptarea colectivă la un mediu global sau local
  - Metaforă computaţională inspirată de:
    - zborul păsărilor în formă de V
    - furnicile aflate în căutarea hranei
    - roiurile de albine care îşi construiesc cuibul
    - bancurile de peşti

  - deoarece
    - controlul este distribuit între mai mulţi indivizi
    - comunicarea între indivizi se realizează local
    - comportamentul sistemului transcede din comportamentul individual
    - sistemul este robust şi se poate adapta schimbărilor de mediu
  - Insecte sociale (2% din totalul insectelor):
    - Furnici
      - 50% din insectele sociale
      - 1 furnică are aprox. 1 mg ➔ Greutatea totală a furnicilor ≈ greutatea totală a oamenilor
      - Trăiesc de peste 100 milioane de ani (oamenii trăiesc de aprox. 50 000 de ani)
    - Termite
    - Albine

Algoritmi inspirați de natură

- Grup (roi - Swarm)
  - O colecţie aparent dezorganizată de indivizi care se mişcă tinzând să se grupeze, dar fiecare individ pare să se mişte într-o direcţie oarecare
  - În interiorul colecţiei apar anumite procese sociale
  - Colecţia este capabilă să efectueze sarcini complexe
    - fără nici o ghidare sau control extern
    - fără nici o coordonare centrală
  - Colecţia poate atinge performanţe care nu pot fi atinse de indivizi în izolare

- Adaptare colectivă ➔ auto-organizare
  - Mulţimea mecanismelor dinamice care generează un comportament global ca rezultat al interacţiunii componentelor individuale
  - Regulile care specifică interacţiunea sunt executate doar pe baza unor informaţii locale, fără referinţe globale
  - Comportamentul global este o proprietate emergentă a sistemului (şi nu una impusă din exterior)

PSO

- Aspecte teoretice
- Algoritm
- Exemplu
- Proprietăţi
- Aplicaţii

PSO – aspecte teoretice

- Propusă
  - de Kennedy şi Eberhart în 1995 http://www.particleswarm.info/
  - Inspirată de comportamentul social al stolurilor de păsări şi al bancurilor de peşti
- Căutare
  - **Cooperativă**, ghidată de calitatea **relativă** a indivizilor
- Operatori de căutare
  - Un fel de mutaţie

PSO – aspecte teoretice

- Elemente speciale
  - Metodă de optimizare bazată pe:
    - populaţii (≈ AG) de particule (≈ cromozomi) care caută soluţia optimă
    - cooperare (în loc de competiţie ca în cazul AG)
  - Fiecare particulă:
    - Se mişcă (deplasează în spaţiul de căutare) şi are o viteză (viteză ≈ mutare pt că timpul este discret)
    - Reţine locul (poziţia) unde a obţinut cele mai bune rezultate
    - Are asociată o vecinătate de particule
  - Particulele cooperează
    - Schimbă informaţii (legate de descoperirile făcute în locurile deja vizitate) între ele
    - Fiecare particulă ştie fitnessul vecinilor ei a.î. poate folosi poziţia celui mai bun vecin pentru a-şi ajusta propria viteză

PSO – aspecte teoretice

Ideea de bază: comportament cognitiv → un individ îşi aminteşte cunoştinţele acumultate în trecut (are memorie)

PSO – aspecte teoretice

Ideea de bază: comportament social → un individ se bazează şi pe cunoştinţele celorlalţi membri ai grupului

PSO – algoritm

- Schema generală
  1. Crearea populaţiei iniţiale de particule
     - Poziţii aleatoare
     - Viteze nule/aleatoare
  2. Evaluarea particulelor
  3. Pentru fiecare particulă
     - Actualizarea memoriei
       - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})
       - Stabilirea celei mai bune poziţii (cu cel mai bun fitness) în care a ajuns până atunci – p_{Best}
     - Modificarea vitezei
     - Modificarea poziţiei
  4. Dacă nu se îndeplinesc condiţiile de oprire, se revine la pasul 2, altfel STOP

PSO – algoritm

1. Crearea populaţiei iniţiale de particule
   - Fiecare particulă are asociată
     - o poziţie – potenţială soluţie a problemei
     - o viteză – modifică o poziţie în altă poziţie
     - o funcţie de calitate (fitness)
   - Fiecare particulă trebuie să poată:
     - interacţiona (schimba informaţii) cu vecinii ei
     - memora o poziţie precedentă
     - utiliza informaţiile pentru a lua decizii
   - Iniţializarea particulelor
     - poziţii aleatoare
     - viteze nule/aleatoare

PSO – algoritm

2. Evaluarea particulelor
   - dependentă de problemă

PSO – algoritm

3. Pentru fiecare particulă x
   - Actualizarea memoriei
     - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})

   - Vecinătate a unei particule
     - Întinderea vecinătăţii
       - Globală
       - Locală
     - Tipul vecinătăţii
       - Geografică
       - Socială
       - Circulară

PSO – algoritm

3. Pentru fiecare particulă x
   - Actualizarea memoriei
     - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})
     - Stabilirea celei mai bune poziţii (cu cel mai bun fitness) în care a ajuns până atunci – p_{Best}

PSO – algoritm

3. Pentru fiecare particulă $\boldsymbol{x_i} = (x_{i1}, x_{i2}, \ldots, x_{iD})$
   - Modificarea vitezei ***v*** şi a poziţiei ***x*** (pe fiecare dimensiune)
     - $v_{id} = w * v_{id} + c_1 * rand() * (p_{Best\ d} - x_{id}) + c_2 * rand() * (g_{Best\ d} - x_{id})$
     - $x_{id} = x_{id} + v_{id}$
     - unde:
       - $i=1,N$ (N – nr total de particule); $d = 1,D$
       - w – factor de inerţie (Shi, Eberhart)
         - $w * v_{id}$ – termen inerţial → forţează particula să se deplaseze în aceeaşi direcţie ca şi până acum (tendinţă curajoasă – *audacious*)
         - balansează căutarea între explorare globală (w mare) şi locală (w mic).
         - poate fi constantă sau descrescătoare (pe măsura „îmbătrânirii" grupului)
       - c_1 - factor de învăţare cognitiv
         - $c_1 * rand() * (p_{Best\ d} - x_{id})$ – termen cognitiv → forţează particula să se deplaseze spre cea mai bună poziţie atinsă până atunci (tendinţă de conservare)
       - c_2 - factor de învăţare social
         - $c_2 * rand() * (g_{Bestd} - x_{id})$ – termen social → forţează particula să se deplaseze spre cea mai bună poziţie a vecinilor; spirit de turmă, de urmăritor
       - Cei doi factori c_1 şi c_2 pot fi egali sau diferiţi ($c_1 > c_2$ *şi* $c_1 + c_2 < 4$ – *Carlise, 2001*)
     - Fiecare componentă a vectorului vitezelor este restricţionată la un interval: $[-v_{max}, v_{max}]$ pentru a asigura păstrarea particulelor în spaţiul de căutare.

PSO – proprietăți

- Principii în PSO:
  - proximitate – grupul trebuie să efectueze calcule în spaţiu şi timp
  - calitate – grupul trebui să fie capabil să răspundă la factorii calitativi ai mediului
  - stabilitate – grupul nu trebuie să îşi schimbe comportamentul la fiecare sesizare a mediului
  - adaptabilitate – grupul trebuie să fie capabil să îşi schimbe comportamentul atunci când costul schimbării nu este prohibit.

- Diferenţe faţă de EC:
  - nu există un operator de recombinare directă – schimbul de informaţie are loc în funcţie de experienţa particulei şi în funcţie de cea a celui mai bun vecin şi nu în funcţie de părinţii selectaţi pe baza fitness-ului.
  - Update poziţie ~ similar cu mutaţia
  - Nu se foloseşte selecţia – supravieţuirea nu este legată de fitness.

- Versiuni ale algoritmului de tip PSO
  - PSO binar discret
  - PSO cu mai mulţi termeni de învăţare socială
  - PSO cu particule eterogene
  - PSO ierarhic

PSO – proprietăți

- PSO discret (binar)
  - Versiune a PSO pentru spaţiu de căutare discret
  - Poziţia unei particule
    - Potenţială soluţie a problemei ➔ string binar
    - Se modifică în funcţie de viteza particulei
  - Viteza unei particule
    - element din spaţiu continuu
    - se modifică conform principiilor de la PSO standard
    - se interpretează ca probabilitatea de modificare a bit-ului corespunzator din poziţia particulei

$$x_{ij} = \begin{cases} 1, & \text{dacă } \tau < s(v_{ij}) \\ 0, & \text{altfel} \end{cases}, \text{unde } s(v_{ij}) = \frac{1}{1+e^{-v_{ij}}}$$

PSO – proprietăți

- Pericole
  - Particulele tind să se grupeze în acelaşi loc
    - Converg prea repede şi nu reuşesc să evadeze dintr-un optim local
    - Soluţia:
      - Reiniţializarea unor particule

  - Deplasarea particulelor spre regiuni nefezabile

PSO – proprietăți

- Analiza algoritmilor de tip PSO
  - Comportamentul dinamic al grupului poate fi analizat cu ajutorul a 2 indici
    - Indicele de dispersie
      - Măsoară gradul de împrăştiere a particulelor în jurul celei mai bune particule din grup
      - Media distanţelor absolute (pe fiecare dimensiune) între fiecare particulă şi particula cea mai bună
      - Explică gradul de acoperire (întins sau restrâns) a spaţiului de căutare
    - Indicele vitezei
      - Măsoară viteza de mişcare a grupului într-o iteraţie
      - Media vitezelor absolute
      - Explică cum (agresiv sau lent) se mişcă grupul

PSO – aplicații

- Controlul şi proiectarea antenelor
- Aplicaţii biologice, medicale, farmaceutice
  - Analiza tremurului în boala Parkinson
  - Clasificare cancerului
  - Predicţia structurii proteinelor
- Comunicare în reţele
- Optimizare combinatorială
- Optimizări financiare
- Analiza imaginilor şi analiza video
- Robotică
- Planificare
- Securitatea reţelelor, detecţia intruşilor, criptografie, criptanaliză
- Procesarea semnalelor

ACO

- Aspecte teoretice
- Algoritm
- Exemplu
- Proprietăţi
- Aplicaţii

ACO – aspecte teoretice

- Propusă
  - de Colorni şi Dorigo în 1991 iniţial pentru rezolvarea problemelor de optimizare discretă – gen TSP – (ca o contrapartidă pentru AG) – http://iridia.ulb.ac.be/~mdorigo/ACO/about.html
  - inspirată de comportamentul social al furnicilor în căutarea unui drum între cuib şi o sursă de hrană
  - De ce furnici?
    - Munca în colonie (de la câteva furnici până la milioane de furnici)
    - Diviziunea muncii
    - Au comportament social complex
- Căutare
  - **Cooperativă**, ghidată de calitatea **relativă** a indivizilor
- Operatori de căutare
  - Constructuvi, adăugând elemente în soluţie

ACO – aspecte teoretice

- Elemente speciale
  - Problema de optimizare trebuie transformată într-o problemă de identificare a drumului optim într-un graf orientat
  - Furnicile construiesc soluţia plimbându-se prin graf şi depunând pe muchii feromoni
  - Metodă de optimizare bazată pe:
    - Colonii (≈AG) de furnici (în loc de cromozomi) care caută soluţia optimă
    - cooperare (în loc de competiţie ca în cazul AG)
  - Fiecare furnică:
    - Se mişcă (deplasează în spaţiul de căutare) şi depune o cantitate de feromon pe drumul parcurs
    - Reţine drumul parcurs
    - Alege drumul pe care să-l urmeze în funcţie de
      - Feromonul existent pe drum
      - Informaţia euristică asociată acelui drum
    - Cooperează cu celelalte furnici prin urma de feromon corespunzătoare unui drum care
      - depinde de calitatea soluţiei şi
      - se evaporă cu trecerea timpului

ACO – aspecte teoretice

- Furnici naturale
  - O colonie de furnici pleacă în căutarea hranei

ACO – aspecte teoretice

- Furnici naturale
  - O colonie de furnici pleacă în căutarea hranei
  - La un moment dat, în drumul lor apare un obstacol

ACO – aspecte teoretice

- Furnici naturale
  - O colonie de furnici pleacă în căutarea hranei
  - La un moment dat, în drumul lor apare un obstacol
  - Furnicile vor ocoli obstacolul fie pe ruta A, fie pe ruta B

ACO – aspecte teoretice

- Furnici naturale
  - O colonie de furnici pleacă în căutarea hranei
  - La un moment dat, în drumul lor apare un obstacol
  - Furnicile vor ocoli obstacolul fie pe ruta A, fie pe ruta B
  - Pentru că ruta A este mai scurtă, furnicile de pe acest drum vor face mai multe ture, deci vor lăsa mai mult feromon
  - Concentraţia de feromon va creşte mai accelerat pe ruta A decât pe ruta B a.î. furniciile de pe ruta B vor alege (pe bază de miros) ruta A
  - Pentru că pe ruta B nu vor mai merge furnici şi pentru că feromonii sunt volatili, urma furnicilor de pe ruta B va dispărea
  - Deci, furnicile se vor plimba doar pe cel mai scurt drum (ruta A)

ACO – aspecte teoretice

- Furnicile artificiale seamănă cu furnicile reale
  - navighează de la cuib spre sursa de hrană
  - descoperă drumul mai scurt pe baza urmei de feromon
    - fiecare frunică execută mişcări aleatoare
    - fiecare furnică depozitează feromon pe drumul parcurs
    - fiecare furnică detectează drumul urmat de "furnica şefă", înclinând să-l urmeze
    - creşterea cantităţii de feromon de pe un drum îi creşte acestuia probabilitatea de a fi urmat de tot mai multe furnici
- dar au anumite îmbunătăţiri:
    - au memorie
      - pentru a reţine acţiunile efectuate ➔ au stare proprie (cu istoricul acţiunilor efectuate)
      - se pot întoarce la cuib (si pe baza urmei de feromon)
    - nu sunt complet oarbe – pot aprecia calitatea spaţiului vecin
    - execută mişcări într-un timp discret
    - depun feromoni şi în funcţie de calitatea soluţiei identificate

ACO – aspecte teoretice

- Urma de feromon are rolul
  - unei memorii colective dinamice distribuită (în colonie)
  - unui depozit cu cele mai recente experienţe de căutare a hranei ale furnicilor din colonie

- Furnicile pot comunica indirect şi se pot influenţa reciproc
  - prin modificarea şi mirosirea acestui depozit chimic
  - în vederea identificării celui mai scurt drum de la cuib până la hrană

ACO – algoritm

- Cât timp nu s-a ajuns la nr maxim de iteraţii
  1. Iniţializare
  2. Cât timp nu s-a parcurs numărul necesar de paşi pentru identificarea soluţiei
     - Pentru fiecare furnică din colonie
       - Se măreşte soluţia parţială cu un element (furnica execută o mutare)
       - Se modifică local urma de feromon corespunzător ultimului element adăugat în soluţie
  3. Se modifică urma de feromon de pe drumurile parcurse de
     - Toate furnicile/cea mai bună furnică
  4. Se returnează soluţia găsită de cea mai bună furnică

ACO – algoritm

- 3 versiuni principale în funcţie de:
  - Regulile de tranziţie de la o stare la alta (regulile de deplasare a furnicilor)
  - Momentul la care furnicile depun feromon:
    - pe parcursul construcţiei soluţiei
    - la sfârşitul creării unei soluţii
  - Furnica deponentă de feromon
    - Toate furnicile
    - Doar cea mai bună furnică

- Versiuni:
  - Ant system (AS)
    - **Toate** furnicile depun feromon **după** construirea unei soluţii **complete** (modificare globală colectivă)
  - MaxMin Ant System (MMAS) ≈ AS, dar
    - doar **cea mai bună** frunică depune feromon **după** construirea unei soluţii **complete** (modificare globală a leader-ului)
    - feromonul depus este **limitat** la un interval dat
  - Ant Colony System (ACO) ≈ AS, dar
    - **toate** furnicile depun feromon **la fiecare pas** în construcţia soluţiei (modificare locală colectivă)
    - doar **cea mai bună** furnică depune feromon după construirea unei soluţii complete (modificare globală a leader-ului)

ACO – exemplu

- Problema comisului voiajor
  - *Travelling salesman problem - TSP*
  - să se găsească un drum care să treacă prin *n* oraşe (inclusiv între primul şi ultimul) astfel încât costul să fie minim şi fiecare oraş să fie vizitat o singură dată.

ACO – exemplu

1. Iniţializare:
   - t := 0 (timpul)
   - pentru fiecare muchie (i,j) se iniţializează
     - $\tau_{ij}^{(t)} = c$ (intensitatea urmei de feromon pe muchia (i,j) la momentul t)
     - $\Delta\tau_{ij} = 0$ (cantitatea de feromon lăsată pe muchia (i,j) de către toate furnicile)
   - se plasează aleator *m* furnici în cele *n* noduri-oraş ($m \leq n$)
   - fiecare furnică îşi modifică memoria (lista cu oraşele vizitate)
     - adaugă în listă oraşul din care pleacă în căutare

ACO – exemplu pentru TSP

2. Cât timp nu s-a parcurs numărul necesar de paşi pentru construcţia soluţiei (nr de paşi = *n*)
   - Pentru fiecare furnică din colonie
     - Se măreşte soluţia parţială cu un element (furnica execută o mutare)
       - fiecare furnică *k* (aflată în oraşul *i*) alege următorul oraş pe care îl vizitează (*j*) astfel:

$$j = \begin{cases} \arg\max_{l \in permis_k} \left\{ [\tau_{il}]^{\alpha} [\eta_{il}]^{\beta} \right\}, & \text{daca } q \le q_0 \\ J, & \text{altfel} \end{cases}$$

Regula aleatoare proporţională

       - unde:
         - q – număr aleator uniform distribuit în [0,1]
         - q_0 – parametru, $0 \le q_0 \le 1$ ($q_0 = 0$ → AS/MMAS, altfel ACO)
         - J este un oraş selectat cu probabilitatea

Regula pseudo-aleatoare proporţională

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}^{(t)}]^{\alpha} [\eta_{ij}]^{\beta}}{\sum_{s - permis_k(t)} [\tau_{is}^{(t)}]^{\alpha} [\eta_{is}]^{\beta}}, & j - permis \\ 0, & altfel \end{cases}$$

unde:
- p_{ij}^{k} probabilitatea de tranziţie a furnicii *k* situată în oraşul *i* spre oraşul *j*
- $\eta_{ij} = \dfrac{1}{d_{ij}}$ - vizibilitatea din oraşul *i* spre oraşul *j* (atractivitatea alegerii muchiei (*i*,*j*))
- $permis_k$ – oraşele pe care le mai poate vizita a *k*-a furnică la momentul *t*
- *α* – controlează importanţa urmei (câte furnici au mai trecut pe muchia respectivă)
- *β* - controlează importanţa vizibilităţii (cât de aproape se află următorul oraş)

ACO – exemplu pentru TSP

2. Cât timp nu s-a parcurs numărul necesar de paşi pentru construcţia soluţiei (nr de paşi = *n*)
   - Pentru fiecare furnică din colonie
     - Se măreşte soluţia parţială cu un element (furnica execută o mutare)
     - Se modifică local urma de feromon lăsată de fiecare furnică pe ultimul element adăugat în soluţie

$$\tau_{ij}^{(t+1)} = (1-\varphi)\tau_{ij}^{(t)} + \varphi * \tau_0$$

       - unde:
         - φ – coeficient de degradare a feromonului; $\varphi \in [0,1]$; pentru $\varphi = 0$ ➔ AS/MMAS, altfel ACO
         - τ_0 – valoarea iniţială a feromonului
         - *(i,j)* – ultima muchie parcursă de furnică

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe
   - drumurile parcurse de toate furnicile (AS)
   - cel mai bun drum (ACO)
   - cel mai bun drum parcurs de cea mai bună furnică (MMAS)

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe
   - **drumurile parcurse de toate furnicile** (AS)
   - Pentru fiecare muchie
     - Se calculează cantitatea unitară de feromoni lăsată de a *k*-a furnică pe muchia (*ij*)
       - $\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} \\ 0 \end{cases}$ - dacă a *k*-a furnică a folosit muchia (*i,j*)
       - *Q* – cantitatea de feromon lăsată de o furnică.
       - L_k – lungimea (costul) turului efectuat de a *k*-a furnică
     - Se calculează cantitatea totală de feromoni de pe muchia (*ij*) $\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$
     - Se calculează intensitatea urmei de feromoni ca sumă între evaporarea feromonilor vechi şi feromonul nou lăsat $\tau_{ij}^{(t+n)} = (1-\rho) * \tau_{ij}^{(t)} + \Delta\tau_{ij}$
       - unde ρ ($0<\rho<1$) – coeficientul de evaporare a urmei de feromon între 2 tururi complete

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe
   - **cel mai bun drum** (ACO/MMAS)
   - Pentru fiecare muchie a celui mai bun drum
     - Se calculează cantitatea unitară de feromoni lăsată de cea mai bună furnică pe muchia (*ij*)
       - L_{best} – lungimea (costul) celui mai bun drum
         - din iteraţia curentă
         - din toate iteraţiile executate până atunci

$$\Delta\tau_{ij} = \frac{1}{L_{best}}$$

     - Se calculează intensitatea urmei de feromoni ca sumă între evaporarea feromonilor vechi şi feromonul nou lăsat

$$\tau_{ij}^{(t+n)} = \left[(1-\rho)*\tau_{ij}^{(t)} + \rho*\Delta\tau_{ij}^{best}\right]_{\tau_{\min}}^{\tau_{\max}}$$

       - unde ρ ($0<\rho<1$) – coeficientul de evaporare a urmei de feromon între 2 tururi complete
       - τ_{min} şi τ_{max} – limitele (inferioară şi superioară) feromonului;
         - pentru $\tau_{min} = -\infty$ şi $\tau_{max} = +\infty$ ➔ ACO, altfel MMAS

ACO – proprietăți

- Proprietăţi
  - Algoritm iterativ
  - Algoritm care construieşte progresiv soluţia pe baza
    - Informaţiilor euristice
    - Urmei de feromon
  - Algoritm stocastic

- Avantaje
  - Rulare neîntreruptă şi adaptabilă schimbării în timp real a datelor de intrare
    - Ex. Pt TSP graful se poate modifica dinamic
  - Feedback-ul pozitiv ajută la descoperirea rapidă a soluţiei
  - Calculul distribuit evită convergenţa prematură
  - Euristica greedy ajută la găsirea unei soluţii acceptabile încă din primele stadii ale căutării
  - Interacţiunea colectivă a indivizilor

- Dezavantaje
  - Converge încet faţă de alte căutări euristice
  - Funcţionează relativ slab pentru instanţe cu mai mult de 75 de oraşe ale TSP
  - În AS nu există un proces central care să ghideze căutarea spre soluţiile bune

ACO – aplicații

- Probleme de identificare a drumului optim în grafe
  - Ex. Traveling Salesman Problem
- Probleme de atribuiri quadratice
- Probleme de optimizări în reţele
- Probleme de transport

Recapitulare

- PSO
  - Algoritm de căutare locală în fascicol
  - Potenţialele soluţii → particule caracterizate prin:
    - poziţie în spaţiul de căutare
    - Viteză
  - Căutare cooperativă şi perturbativă bazată pe
    - Poziţia celei mai bune particule din grup
    - Cea mai bună poziţie a particulei de până atunci (particula are memorie)
- ACO
  - Algoritm de căutare locală în fascicol
  - Potenţialele soluţii → furnici caracterizate prin:
    - Memorie – reţin paşii făcuţi în construirea soluţiei
    - Miros – iau decizii pe baza feromonului depus de celelalte furnici (comportament social, colectiv, colaborativ)
  - Căutare cooperativă şi constructivă

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- capitolul II.5 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 6 din *H.F. Pop, G. Şerban, Inteligenţă artificială, Cluj Napoca, 2004*
- documentele din directorul *06_adversial_minimax*

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Sisteme inteligente

Sisteme care învaţă singure

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit şi legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 şi 11 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997* link
- *capitolul 1 din C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006* link
- *capitolul 1 din S. Guido, A. C. Müller, Introduction to Machine Learning with Python, O'Reilly Media, 2016* link
- *capitolele 1 și 2 din A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, 2019* link

Conținut

- Sisteme inteligente
  - Sisteme care învaţă singure (SIS)
    - Instruire (învăţare) automata (Machine Learning - ML)
      - Problematică
      - Proiectarea unui sistem de învăţare automată
      - Tipologie
        - Învăţare supervizată
        - Învăţare nesupervizată
        - Învăţare cu întărire
        - Teoria învăţării
    - Exemple de sisteme

Sisteme inteligente

Sisteme bazate pe cunoştinţe

Inteligenţă computaţională

Sisteme inteligente – SIS – Învățare automată

- Problematica
  - "How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?"

- Aplicaţii
  - Recunoaştere de imagini şi semnal vocal
    - Recunoaşterea scrisului de mână
    - Detecţia feţelor
    - Înţelegerea limbajului vorbit
  - Computer vision
    - Detecţia obstacolelor
    - Recunoaşterea amprentelor
  - Supraveghere bio
  - Controlul roboţilor
  - Predicţia vremii
  - Diagnosticare medicală
  - Detecţia fraudelor

Sisteme inteligente – SIS – Învățare automată

- Definire
  - Arthur Samuel (1959)
    - "field of study that gives computers the ability to learn without being explicity programmed"
    - Înzestrarea computerelor cu abilitatea de a învăţa pe baza experienţei
  - Herbert Simon (1970)
    - "Learning is any process by which a system improves performance from experience."
  - Tom Mitchell (1998)
    - "a well-posed learning problem is defined as follows: He says that a computer program is set to learn from an experience E with respect to some task T and some performance measure P if its performance on T as measured by P improves with experience E"
  - Ethem Alpaydin (2010)
    - Programming computers to optimize a performance criterion using example data or past experience.
  - John L. Hennessy, President of Stanford (2000–2016)
    - Machine learning is the hot new thing
  - Bill Gates (Microsoft co-founder)
    - A breakthrough in machine learning would be worth ten Microsofts

- Necesitate
  - Sisteme computaţionale mai bune
    - Sisteme dificil sau prea costisitor de construit manual
      - Sisteme care se adaptează automat
        - Filtre de spam
      - Sisteme care descoperă informaţii în baze de date mari → data mining
        - Analize financiare
        - Analize de text/imagini
  - Înţelegerea organismelor biologice

Sisteme inteligente – SIS – Învățare automată

- Persoane importante și/sau interesante
  - Peter Norvig
  - Stuart Russell
  - Michael Jordan (Bayesian Nets), Andrew Ng
  - Elon Musk, Andrej Karpathy (TESLA)
  - Fei Fei Li (AI for social good)
  - Richard Sutton (Reinforcement Learning)
  - Jurgen Schmidhuber (LSTM)
  - Geoffrey Hinton , Yann LeCun, and Yoshua Bengio (CNN and deep CNN)
  - John Koza (Genetic Programming)
  - Rana el Kaliouby (Affectiva)
  - …alții…

Sisteme inteligente – SIS – Învățare automată

- Programarea tradițională

- Machine Learning

Sisteme inteligente – SIS – Învățare automată

- Proiectare
  - Îmbunătăţirea task-ului T
    - Stabilirea scopului (ceea ce trebuie învăţat) - funcţiei obiectiv – şi reprezentarea sa
    - Alegerea unui algoritm de învăţare care să realizeze inferenţa (previziunea) scopului pe baza experienţei
  - respectând o metrică de performanţă P
    - Evaluarea performanţelor algortimului ales
  - bazându-se pe experienţa E
    - Alegerea bazei de experienţă

  - Exemplu
    - T: jucarea jocului de dame
    - P: procentul de jocuri câştigate împotriva unui oponent oarecare
    - E: exersarea jocului împotriva lui însuşi

    - T: recunoaşterea scrisului de mână
    - P: procentul de cuvinte recunoscute corect
    - E: baze de date cu imagini cu cuvinte corect adnotate

    - T: separarea spam-urilor de mesajele obişnuite
    - P: procentul de email-uri corect clasificate (spam sau normal)
    - E: baze de date cu email-uri adnotate

Sisteme inteligente – SIS – Învățare automată

- Stabilirea scopului (ceea ce trebuie învăţat)
  - SI pentru predicţii / regresii
    - Scop: predicţia ieşirii pentru o intrare nouă folosind un model învăţat anterior
    - Ex.: predicţia vânzărilor dintr-un produs pentru un moment de timp viitor în funcţie de preţ, lună calendaristică, regiune, venit mediu pe economie

  - SI pentru regresii simbolice
    - Scop: estimarea formei unei funcţii uni sau multivariată folosind un model învăţat anterior
    - Ex.: estimarea funcţiei care modelează conturul unei suprafeţe

  - SI pentru clasificare
    - Scop: clasificarea unui obiect într-una sau mai multe categorii (clase) – cunoscute anterior sau nu - pe baza caracteristicilor (atributelor, proprietăţilor) lui
    - Ex.: sistem de diagnoză pentru un pacient cu tumoare: nevasculară, vasvculară, angiogenă

  - SI pentru planificare
    - Scop: generarea unei succesiuni optime de acţiuni pentru efectuarea unei sarcini
    - Ex.: planificarea deplasării unui robot de la o poziţie dată până la o sursă de energie (pentru alimentare)

Sisteme inteligente – SIS – Învățare automată

- Stabilirea scopului (ceea ce trebuie învăţat)
  - Probleme de predicție / regresie
    - Se dau date (de intrare și ieșire) trecute
      - Numărul de persoane infectate cu SARS-CoV-2 pentru ultimele 3 luni
    - Se cer predicții viitoare (pentru anumite date de intrare)
      - Numărul de persoane care se vor infecta cu SARS-CoV-2 in urmatoarele 7 zile / 4 saptamani / 2 luni

Sisteme inteligente – SIS – Învățare automată

- Stabilirea scopului (ceea ce trebuie învăţat)
  - Probleme de clasificare
    - Se dau date (de intrare și ieșire) trecute
      - Imagini RMN de la pacienți infectați cu SARS-C0V-2 și de la martori (sănătoși)
    - Se cer predicții viitoare (pentru anumite date de intrare)
      - Să se prezică, pe baza RMN-ului, dacă o persoană este infectată sau nu cu SARS-CoV-2

Sisteme inteligente – SIS – Învățare automată

- Stabilirea scopului (ceea ce trebuie învăţat)
  - Probleme de clusterizare
    - Se dau date (de intrare)
      - Localizarea geografică a unor persoane infectate cu SARS-CoV-2
    - Se cere identificarea anumitor structuri în aceste date
      - Modul de grupare a celor infectați pe regiuni (Densitatea acestor regiuni)

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea funcţiei obiectiv
  - Care este funcţia care trebuie învăţată?
    - Ex.: pentru jocul de dame → funcţie care:
      - alege următoarea mutare
      - evaluează o mutare
    - obiectivul fiind alegerea celei mai bune mutări
  - Reprezentarea funcţiei obiectiv
    - Diferite reprezentări
      - Tablou (tabel)
      - Reguli simbolice
      - Funcţie numerică
      - Funcţii probabilistice
    - Ex. Jocul de dame
      - Combinaţie liniară a nr. de piese albe, nr. de piese negre, nr. de piese albe compromise la următoarea mutare, r. de piese albe compromise la următoarea mutare
    - Există un compromis între
      - expresivitatea reprezentării şi
      - uşurinţa învăţării
    - Calculul funcţiei obiectiv
      - Timp polinomial
      - Timp non-polinomial

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învăţare
  - Algoritmul
    - folosind datele de antrenament
    - induce definirea unor ipoteze care
      - să se potirvească cu acestea şi
      - să generalizeze cât mai bine datele ne-văzute (datele de test)
  - Principiul de lucru de bază
    - Minimizarea unei erori (funcţie de cost – loss function)
  - Tipuri de algoritmi după metodologia de învățare automată
    - Învățare supervizată
      - Ex. regresie, clasificare
    - Învățare nesupervizată
      - Ex. clusterizare, reducerea numărului de dimensiuni
    - Învățare prin întărire
      - Ex. planning, gaming

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învăţare → Tipuri de algoritmi după metodologia de învățare automată
  - Învățare supervizată
    - Ex. regresie, clasificare
    - Caracteristici
      - Date etichetate (se cunosc o parte din datele de intrare și ieșire *)
      - Feedback direct în timpul învățării – algoritmul se adaptează la datele de intrare și ieșire
      - Predicție a datelor de ieșire (fiind cunoscute niște date de intrare diferite de cele din *)

)

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învăţare → Tipuri de algoritmi după metodologia de învățare automată
  - Învățare supervizată
  - Învățare nesupervizată
    - Ex. clusterizare, reducerea numărului de dimensiuni
    - Carcteristici
      - Date neetichetate (se cunosc o parte din datele de intrare**)
      - Fără feedback direct în timpul învățării – pentru că nu se cunosc datele de ieșire
      - Identificarea unor structuri în date (generarea de date de ieșire pentru datele de intrare din **)

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învăţare → Tipuri de algoritmi după metodologia de învățare automată
  - Învățare supervizată
  - Învățare nesupervizată
  - Învățare prin întărire
    - Ex.
    - Caracteristici
      - Predicția unor secvențe de decizii / de acțiuni
      - Sistem de recompense (pentru fiecare decizie / acțiune)
      - Se învață un model de acțiune (o serie de acțiuni care trebuie efectuate)

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învăţare
  - Algoritmul
    - folosind datele de antrenament
    - induce definirea unor ipoteze care
      - să se potirvească cu acestea şi
      - să generalizeze cât mai bine datele ne-văzute (datele de test)
  - Principiul de lucru de bază
    - Minimizarea unei erori (funcţie de cost – loss function) pentru datele de antrenament
      - Eroarea de predicție (cât de departe sunt valorile prezise față de valorile reale)
      - Eroarea de clasificare (câte exemple au fost clasificate corect)
      - Eroarea creării unor structuri (cât de ne-omogene sunt structurile produse)

- Proiectare → Evaluarea unui sistem de învăţare
  - Experimental
    - Compararea diferitelor metode pe diferite date (cross-validare)
    - Colectarea datelor pe baza performanţei
      - Acurateţe, timp antrenare, timp testare
    - Aprecierea diferenţelor dpdv statistic
  - Teoretic
    - Analiza matematică a algoritmilor şi demonstrarea de teoreme
      - Complexitatea computaţională
      - Abilitatea de a se potrivi cu datele de antrenament
      - Complexitatea eşantionului relevant pentru o învăţare corectă

Sisteme inteligente – SIS – Învăţare automată

- Proiectare → Evaluarea unui sistem de învăţare
  - Compararea performanţelor a 2 algoritmi în rezolvarea unei probleme
    - Indicatori de performanţă
      - Parametrii ai unei serii statistice (ex. media)
      - Proporţie calculată pentru serie statistică (ex. acurateţea)
    - Comparare pe baza intervalelor de încredere
      - Pp o problemă şi 2 algoritmi care o rezolvă
      - Performanţele algoritmilor: p_1 şi p_2
      - Intervalele de încredere corespunzătoare celor 2 performanţe $I_1=[p_1-\Delta_1,p_1+\Delta_1]$ şi $I_2=[p_2-\Delta_2,p_2+\Delta_2]$
      - Dacă $I_1 \cap I_2=\emptyset$ ➔ algoritmul 1 este mai bun decât algoritmul 2 (pt problema dată)
      - Dacă $I_1 \cap I_2 \neq \emptyset$ ➔ nu se poate spune care algoritm este mai bun
    - Interval de încredere pentru medie
      - Pentru o serie statistică de volum n, cu media (calculată) m şi dispersia σ să se determine intervalul de încredere al valorii medii μ
      - $P(-z \leq (m-\mu)/(\sigma/\sqrt{n}) \leq z) = 1 - a$ ➔ $\mu \epsilon [m-z\sigma/\sqrt{n}, m+z\sigma/\sqrt{n}]$
      - $P = 95\%$ ➔ $z = 1.96$
      - Ex. Problema rucsacului rezolvată cu ajutorul algoritmilor evolutivi
    - Interval de încredere pentru acurateţe
      - Pentru o performanţă p (acurateţe) calculată pentru n date să se determine intervalul de încredere
      - $P \in [p-z(p(1-p)/n)^{1/2}, p+z(p(1-p)/n)^{1/2}]$
      - $P = 95\%$ ➔ $z = 1.96$
      - Ex. Problemă de clasificare rezolvată cu ajutorul Maşinilor cu suport vectorial

$P=1-\alpha$	z
99.9%	3.3
99.0%	2.577
98.5%	2.43
97.5%	2.243
95.0%	1.96
90.0%	1.645
85.0%	1.439
75.0%	1.151

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea bazei de experienţă
  - Bazată pe
    - Experienţă directă
      - Perechi (intrare, ieşire) utile pt. funcţia obiectiv
      - Ex. Jocul de dame → table de joc etichetată cu mutare corectă sau incorectă
    - Experienţă indirectă
      - Feedback util (diferit de perechile I/O) pt funcţia obiectiv
      - Ex. Jocul de dame → secvenţe de mutări şi scorul final asociat jocului
  - Surse de date
    - Exemple generate aleator
      - Exemple pozitive şi negative
    - Exemple pozitive colectate de un "învăţător" benevol
    - Exemple reale
  - Compoziţie
    - Date de antrenament
    - Date de test
  - Caracteristici
    - Date independente
      - Dacă nu → clasificare colectivă
    - Datele de antrenament şi de test trebuie să urmeze aceeaşi lege de distribuţie
      - Dacă nu → învăţare prin transfer (*transfer learning/inductive transfer*)
        - recunoaşterea maşinilor → recunoaşterea camioanelor
        - analiza textelor
        - filtre de spam

Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea bazei de experienţă
  - Tipuri de atribute ale datelor
    - Cantitative → scară nominală sau raţională
      - Valori continue → greutatea
      - Valori discrete → numărul de computere
      - Valori de tip interval → durata unor evenimente
    - Calitative
      - Nominale → culoarea
      - Ordinale → intensitatea sunetului (joasă, medie, înaltă)
    - Structurate
      - Arbori – rădăcina e o generalizare a copiilor (vehicol → maşină, autobus, tractor, camion)
  - Transformări asupra datelor
    - Standardizare → atribute numerice
      - Înlăturarea efectelor de scară (scări şi unităţi de măsură diferite)
      - Valorile brute se transformă în scoruri z
        - $Z_{ij} = (x_{ij} - \mu_j)/\sigma_j$, unde x_{ij} – valoarea atributului al j-lea al instanţei i, μ_j (σ_j) este media (abaterea) atributelor j pt. toate instanţele
    - Selectarea anumitor atribute

Sisteme inteligente – SIS – Învățare automată

- Învăţare supervizată
  - Definire
  - Exemple
  - Proces
  - Calitatea învăţării
    - Metode de evaluare
    - Măsuri de performanţă
  - Tipologie

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Scop
  - Furnizarea unei ieşiri corecte pentru o nouă intrare

- Definire
  - Se dă un set de date (exemple, instanţe, cazuri)
    - date de antrenament – sub forma unor perechi (atribute_data$_i$, ieşire$_i$), unde
      - i =1,N (N = nr datelor de antrenament)
      - atribute_data$_i$= (atr$_{i1}$, atr$_{i2}$, ..., atr$_{im}$), m – nr atributelor (caracteristicilor, proprietăţilor) unei date
      - ieşire$_i$
        - o categorie dintr-o mulţime dată (predefinită) cu k elemente (k – nr de clase) ➔ problemă de clasificare
        - un număr real ➔ problemă de regresie
    - date de test - sub forma (atribute_datai), i =1,n (n = nr datelor de test).
  - Să se determine
    - o funcţie (necunoscută) care realizează corespondenţa atribute – ieşire pe datele de antrenament
    - ieşirea (clasa/valoarea) asociată unei date (noi) de test folosind funcţia învăţată pe datele de antrenament

- Alte denumiri
  - Clasificare (regresie), învăţare inductivă

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Proces → 2 etape
  - Antrenarea
    - Învăţarea, cu ajutorul unui algoritm, a modelului de predicție
  - Testarea
    - Testarea modelului folosind date de test noi (*unseen data*)

- Caracteristic
  - BD experimentală adnotată (pt. învăţare)

Învăţare supervizată

- Tip de probleme
  - regresie
    - Scop: predicţia output-ului pentru un input nou
    - Output continuu (nr real)
    - Ex.: predicţia ratei șomajului în funcție de produsul intern brut și rata inflației
  - clasificare
    - Scop: clasificarea (etichetarea) unui nou input
    - Output discret (etichetă dintr-o mulţime predefinită)
    - Ex.: detectarea tumorilor maligne în imagini RMN

- Exemple de probleme
  - Recunoaşterea scrisului de mână
  - Recunoaşterea pietonilor în imagini
  - Previziunea vremii
  - Detecţia spam-urilor

Sisteme inteligente – SIS – Învățare automată

- Învățare supervizată
  - Terminologie – e.g. Problema predicției consumului de înghețată pe baza temperaturii de afară și a sumei de bani avută la dispoziție
    - Exemplu (example, observation, instance, record)
      - o observație a datelor care trebuie procesate
      - dacă datele de intrare sunt tabelare, un exemplu este asociat cu o linie din tabel
      - format din proprietăți a datelor care trebuie procesate (de intrare și de ieșire)
    - Caracteristică (feature, property, attribute)
      - Proprietate cunoscută a unui exemplu, folosită drept dată de intrare pentru algoritmul de ML (variabilele independente din modelul de predicție)
      - dacă datele de intrare sunt tabelare, un o proprietate are asociate valorile dintr-o coloana a tabelului (pentru toate exemplele)
      - E.g. Temperatura, banii
    - Valoare țintă (target or real value/label, ground-truth)
      - Proprietate a unui exemplu folosită ca variabilă dependentă
      - Cunoscută pentru exemplele de antrenament
      - Ne-cunoscută pentru exemplele de testare
      - E.g. Nr de inghetate
    - Valoare calculată (computed value/label)
      - Proprietate a unui exemplu estimată cu ajutorul algoritmului de ML
      - Se dorește a fi cât mai aproape de valoarea target

Exemplu	Temperatura	Banii	Nr de inghetate
Ex1	30	25	2
Ex2	5	100	0
Ex3	19	55	2
Ex4	35	75	4

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Calitatea învăţării
  - Definire
    - o măsură de performanţă a algoritmului de ML
      - ex. acurateţea (Acc = nr de exemple corect clasificate / nr total de exemple)
    - Posibile măsuri:
      - Măsuri statistice
        - Eroarea de predicție
        - acurateţea
        - Precizia
        - Rapelul
        - Scorul F1

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Calitatea învăţării → Măsuri de performanţă → Măsuri statistice
  - Eroarea de predicție
    - Suma diferențelor absolute între valorile reale și cele calculate

$$Err = \frac{1}{noSamples} \sum_{i=1}^{noSamples} abs(real_i - computed_i)$$

    - Suma pătratelor diferențelor între valorile reale și cele calculate

$$Err = \sqrt{\frac{1}{noSamples} \sum_{i=1}^{noSamples} (real_i - computed_i)^2}$$

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Calitatea învăţării → Măsuri de performanţă → Măsuri statistice
  - Acurateţea
    - Nr de exemple corect clasificate / nr total de exemple
    - Opusul erorii
    - Calculată pe
      - Setul de validare
      - Setul de test
    - Uneori
      - Analiză de text
      - Detectarea intruşilor într-o reţea
      - Analize financiare

      este importantă doar o singură clasă (clasă pozitivă) → restul claselor sunt negative
  - Precizia (P)
    - nr. de exemple pozitive corect clasificate / nr. total de exemple clasificate ca pozitive
    - probabilitatea ca un exemplu clasificat pozitiv să fie relevant
    - *TP / (TP + FP)*
  - Rapelul (R)
    - nr. de exemple pozitive corect clasificate / nr. total de exemple pozitive
    - Probabilitatea ca un exemplu pozitiv să fie identificat corect de către clasificator
    - *TP/ (TP +FN)*
    - Matrice de confuzie → rezultate reale vs. rezultate calculat
  - Scorul F1
    - Combină precizia şi rapelul, facilitând compararea a 2 algoritmi
    - Media armonică a preciziei şi rapelului
    - *2PR/(P+R)*

		Rezultate reale	
		Clasa pozitivă	Clasa(ele) negativă(e)
Rezultate calculate	Clasa pozitivă	*True positiv (TP)*	*False positiv (FP)*
	Clasa(ele) negativă(e)	*False negative (FN)*	*True negative (TN)*

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Calitatea învăţării → Măsuri de performanţă → Măsuri statistice
  - Acurateţea
    - Nr de exemple corect clasificate / nr total de exemple
  - Precizia (P)
    - nr. de exemple pozitive corect clasificate / nr. total de exemple clasificate ca pozitive
    - probabilitatea ca un exemplu clasificat pozitiv să fie relevant
    - *TP / (TP + FP)*
  - Rapelul (R)
    - nr. de exemple pozitive corect clasificate / nr. total de exemple pozitive
    - Probabilitatea ca un exemplu pozi
    - *TP/ (TP +FN)*
    - Matrice de confuzie → rezultate re
  - Scorul F1
    - Combină precizia şi rapelul, facilit
      a 2 algoritmi
    - Media armonică a preciziei şi rape
    - *2PR/(P+R)*

		Rezultate reale	
		Clasa pozitivă	Clasa(ele) negativă(e)
Rezultate calculate	Clasa pozitivă	*True positiv (TP)*	*False positiv (FP)*
	Clasa(ele) negativă(e)	*False negative (FN)*	*True negative (TN)*

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Calitatea învăţării
  - Definire
    - o măsură de performanţă a algoritmului
      - ex. acurateţea (Acc = nr de exemple corect clasificate / nr total de exemple)
    - Posibile măsuri:
      - Măsuri statistice
        - acurateţea
        - Precizia
        - Rapelul
        - Scorul F1
      - Eficienţa
        - În construirea modelului
        - În testarea modelului
      - Robusteţea
        - Tratarea zgomotelor şi a valorilor lipsă
      - Scalabilitatea
        - Eficienţa gestionării seturilor mari de date
      - Interpretabilitatea
        - Modelului de clasificare
      - Proprietatea modelului de a fi compact
      - Scoruri

Sisteme inteligente – SIS – Învățare automată

Învăţare supervizată

- Calitatea învăţării
  - Definire
    - o măsură de performanţă a algoritmului
      - ex. acurateţea (Acc = nr de exemple corect clasificate / nr total de exemple)
    - calculată în
      - faza de antrenare
      - faza de testare
  - Metode de evaluare
    - Seturi disjuncte de antrenare şi testare
      - setul de antrenare poate fi împărţit în date de învăţare şi date de validare
      - setul de antrenare este folosit pentru estimarea parametrilor modelului (cei mai buni parametri obţinuţi pe validare vor fi folosiţi pentru construcţia modelului final)
      - pentru date numeroase
    - Validare încrucişată cu mai multe (h) sub-seturi egale ale datelor (de antrenament)
      - separararea datelor de h ori în (*h-1* sub-seturi pentru învăţare şi 1 sub-set pt validare)
      - dimensiunea unui sub-set = dimensiunea setului / h
      - performanţa este dată de media pe cele h rulări (ex. *h = 5* sau *h = 10)*
      - pentru date puţine
    - Leave-one-out cross-validation
      - similar validării încrucişate, dar h = nr de date ➔ un sub-set conţine un singur exemplu
      - pentru date foarte puţine
  - Dificultăţi
    - Învăţare pe derost (overfitting) → performanţă bună pe datele de antrenament, dar foarte slabă pe datele de test

Sisteme inteligente – SIS – Învățare automată

- Învăţare ne-supervizată
  - Definire
  - Exemple
  - Proces
  - Metode de evaluare şi măsuri de performanţă
  - Tipologie

Sisteme inteligente – SIS – Învățare automată

Învăţare ne-supervizată

- Scop
  - Găsirea unui model sau a unei structuri utile a datelor
  - Împărţirea unor exemple neetichetate în submulţimi disjuncte (clusteri) astfel încât:
    - exemplele din acelaşi cluster sunt foarte similare
    - exemplele din clusteri diferiţi sunt foarte diferite
- Definire
  - Se dă un set de date (exemple, instanţe, cazuri)
    - Date de antrenament sub forma ***atribute_data***$_i$, unde
      - $i = 1,N$ (N = nr datelor de antrenament)
      - ***atribute_data***$_i$= (atr_{i1}, atr_{i2}, ..., atr_{im}), m – nr atributelor (caracteristicilor, proprietăţilor) unei date
    - Date de test sub forma (***atribute_data***$_i$), $i = 1,n$ (n = nr datelor de test)
  - Se determină
    - o funcţie (necunoscută) care realizează gruparea datelor de antrenament în mai multe clase
      - Nr de clase poate fi pre-definit (k) sau necunoscut
      - Datele dintr-o clasă sunt asemănătoare
    - clasa asociată unei date (noi) de test folosind gruparea învăţată pe datele de antrenament
  - Învăţare supervizată vs. învăţare ne-supervizată

  - Distanţe între 2 elemente p şi q ϵ Rm
    - Euclideana → d(p,q)=sqrt(∑j=1,2,...,m(pj-qj)2)
    - Manhattan → d(p,q)=∑j=1,2,...,m|pj-qj|
    - Mahalanobis → d(p,q)=sqrt(p-q)S-1(p-q)), unde S este matricea de variaţie şi covariaţie (S= E[(p-E[p])(q-E[q])])
    - Produsul intern → d(p,q)=∑j=1,2,...,mpjqj
    - Cosine → d(p,q)=∑j=1,2,...,mpjqj / (sqrt(∑j=1,2,...,mpj2) * sqrt(∑j=1,2,...,mqj2))
    - Hamming → numărul de diferenţe între p şi q
    - Levenshtein → numărul minim de operaţii necesare pentru a-l transforma pe p în q
  - Distanţă vs. Similaritate
    - Distanţa → min
    - Similaritatea → max

Sisteme inteligente – SIS – Învățare automată

Învăţare ne-supervizată

- Alte denumiri
  - Clustering
- Procesul → 2 paşi
  - Antrenarea → Învăţarea (determinarea), cu ajutorul unui algoritm, a clusterilor existenţi
  - Testarea → Plasarea unei noi date într-unul din clusterii identificaţi în etapa de antrenament
- Caracteristic
  - Datele nu sunt adnotate (etichetate)
- Tip de probleme
  - Identificara unor grupuri (clusteri)
    - Analiza genelor
    - Procesarea imaginilor
    - Analiza reţelelor sociale
    - Segmentarea pieţei
    - Analiza datelor astronomice
    - Clusteri de calculatoare
  - Reducerea dimensiunii
  - Identificarea unor cauze (explicaţii) ale datelor
  - Modelarea densităţii datelor
- Exemple de probleme
  - Gruparea genelor
  - Studii de piaţă pentru gruparea clienţilor (segmentarea pieţei)
  - news.google.com

Învăţare ne-supervizată

- Calitatea învăţării (validarea clusterizări):
  - Criterii interne → Similaritate ridicată în interiorul unui cluster şi similaritate redusă între clusteri
    - Distanţa în interiorul clusterului
    - Distanţa între clusteri
    - Indexul Davies-Bouldin
    - Indexul Dunn
  - Criteri externe → Folosirea unor benchmark-uri formate din date pre-grupate
    - Compararea cu date cunoscute – în practică este imposibil
    - Precizia
    - Rapelul
    - F-measure

Învăţare ne-supervizată

- Calitatea învăţării → Criterii interne
  - Distanţa în interiorul clusterului c_j care conţine n_j instanţe
    - Distanţa medie între instanţe (average distance) $D_a(c_j) = \Sigma_{xi1,xi2\epsilon cj} ||x_{i1} - x_{i2}|| / (n_j(n_j-1))$
    - Distanţa între cei mai apropiaţi vecini $D_{nn}(c_j) = \Sigma_{xi1\epsilon cj} \min_{xi2\epsilon cj} ||x_{i1} - x_{i2}|| / n_j$
    - Distanţa între centroizi $D_c(c_j) = \Sigma_{xi,\epsilon cj} ||x_i - \mu_j|| / n_j$, unde $\mu_j = 1/ n_j \Sigma_{xi\epsilon cj} x_i$
  - Distanţa între 2 clusteri c_{j1} şi c_{j2}
    - Legătură simplă $d_s(c_{j1}, c_{j2}) = \min_{xi1\epsilon cj1,\ xi2\epsilon cj2} \{||x_{i1} - x_{i2}||\}$
    - Legătură completă $d_{co}(c_{j1}, c_{j2}) = \max_{xi1\epsilon cj1,\ xi2\epsilon cj2} \{||x_{i1} - x_{i2}||\}$
    - Legătură medie $d_a(c_{j1}, c_{j2}) = \Sigma_{xi1\epsilon cj1,\ xi2\epsilon cj2} \{||x_{i1} - x_{i2}||\} / (n_{j1} * n_{j2})$
    - Legătură între centroizi $d_{ce}(c_{j1}, c_{j2}) = ||\mu_{j1} - \mu_{j2}||$
  - Indexul Davies-Bouldin → min ➔ clusteri compacţi
    - $DB = 1/nc * \Sigma_{i=1,2,\ldots,nc} \max_{j=1,2,\ldots,nc,\ j \neq i}((\sigma_i + \sigma_j)/d(\mu_i, \mu_j))$, unde:
      - nc – numărul de clusteri
      - μ_i – centroidul clusterului i
      - σ_i – media distanţelor între elementele din clusterul i şi centroidul μ_i
      - $d(\mu_i, \mu_j)$ – distanţa între centroidul μ_i şi centroidul μ_j
  - Indexul Dunn
    - Identifică clusterii denşi şi bine separaţi
    - $D = d_{min}/d_{max}$, unde:
      - d_{min} – distanţa minimă între 2 obiecte din clusteri diferiţi – distanţa intra-cluster
      - d_{max} – distanţa maximă între 2 obiecte din acelaşi cluster – distanţa inter-cluster

Sisteme inteligente – SIS – Învățare automată

Învăţare ne-supervizată

- Tipologie
  - După modul de formare al clusterilor
    - Ierarhic
      - se crează un arbore taxonomic (dendogramă)
        - crearea clusterilor → recursiv
        - nu se cunoaşte *k* (nr de clusteri)
      - aglomerativ (de jos în sus) → clusteri mici spre clusteri mari
      - diviziv (de sus în jos) → clusteri mari spre clusteri mici
      - Ex. Clustering ierarhic aglomrativ
    - Ne-ierarhic
      - Partiţional → se determină o împărţire a datelor → toţi clusterii deodată
      - Optimizează o funcţie obiectiv definită local (doar pe anumite atribute) sau global (pe toate atributele) care poate fi:
        - Pătratul erorii – suma patratelor distanţelor între date şi centroizii clusterilor → min (ex. K-means)
        - Bazată pe grafuri (ex. Clusterizare bazată pe arborele minim de acoperire)
        - Pe modele probabilistice (ex. Identificarea distribuţiei datelor → Maximizarea aşteptărilor)
        - Pe cel mai apropiat vecin
      - Necesită fixarea apriori a lui k → fixarea clusterilor iniţiali
        - Algoritmii se rulează de mai multe ori cu diferiţi parametri şi se alege versiunea cea mai eficientă
      - Ex. K-means, ACO
    - bazat pe densitatea datelor
      - Densitatea şi conectivitatea datelor
        - Formarea clusterilor de bazează pe densitatea datelor într-o anumită regiune
        - Formarea clusterilor de bazează pe conectivitatea datelor dintr-o anumită regiune
      - Funcţia de densitate a datelor
        - Se încearcă modelarea legii de distribuţie a datelor
      - Avantaj:
        - Modelarea unor clusteri de orice formă
    - Bazat pe un grid
      - Nu e chiar o metodă nouă de lucru
        - Poate fi ierarhic, partiţional sau bazat pe densitate
      - Pp segmentarea spaţiului de date în zone regulate
      - Obiectele se plasează pe un grid multi-dimensional
      - Ex. ACO

Sisteme inteligente – SIS – Învățare automată

Învăţare ne-supervizată

- Tipologie
  - După modul de lucru al algoritmului
    - Aglomerativ
      1. Fiecare instanţă formează iniţial un cluster
      2. Se calculează distanţele între oricare 2 clusteri
      3. Se reunesc cei mai apropiaţi 2 clusteri
      4. Se repetă paşii 2 şi 3 până se ajunge la un singur cluster sau la un alt criteriu de stop
    - Diviziv
      1. Se stabileşte numărul de clusteri (k)
      2. Se iniţializează centrii fiecărui cluster
      3. Se determină o împărţire a datelor
      4. Se recalculează centrii clusterilor
      5. Se reptă pasul 3 şi 4 până partiţionarea nu se mai schimbă (algoritmul a convers)
  - După atributele considerate
    - Monotetic – atributele se consideră pe rând
    - Politetic – atributele se consideră simultan
  - După tipul de apartenenţă al datelor la clusteri
    - Clustering exact (*hard clustering*)
      - Asociază fiecarei intrări $\boldsymbol{x_i}$ o etichetă (clasă) c_j
    - Clustering fuzzy
      - Asociază fiecarei intrări $\boldsymbol{x_i}$ un grad (probabilitate) de apartenenţă f_{ij} la o anumită clasă c_j ➔ o instanţă $\boldsymbol{x_i}$ poate aparţine mai multor clusteri

Sisteme inteligente – SIS – Învățare automată

- Tipologie
  - În funcţie de experienţa acumulată în timpul învăţării
    - SI cu învăţare supervizată
    - SI cu învăţare nesupervizată
    - **SI cu învăţare activă**
    - SI cu învăţare cu întărire
  - În funcţie de modelul învăţat (algoritmul de învăţare)
    - Arbori de decizie
    - Reţele neuronale artificiale
    - Algoritmi evolutivi
    - Maşini cu suport vectorial
    - Modele Markov ascunse

Sisteme inteligente – SIS – Învățare automată

- Învăţare activă
  - Algoritmul de învăţare poate primi informaţii suplimentare în timpul învăţării pentru a-şi îmbunătăţi performanţa
    - Ex. pe care din datele de antrenament este mai uşor să se înveţe modelul de decizie

Figure 1.1: General schema for a passive learner.

Figure 1.2: General schema for an active learner.

Sisteme inteligente – SIS – Învățare automată

- Tipologie
  - În funcţie de experienţa acumulată în timpul învăţării
    - SI cu învăţare supervizată
    - SI cu învăţare nesupervizată
    - SI cu învăţare activă
    - SI cu învăţare cu întărire
  - În funcţie de modelul învăţat (algoritmul de învăţare)
    - **Metoda celor mai mici pătrate**
    - Metoda gradient descent
    - Algoritmi evolutivi
    - Logisitc regression
    - kNN
    - Arbori de decizie
    - Maşini cu suport vectorial
    - Reţele neuronale artificiale
    - Programare genetică
    - Modele Markov ascunse

Recapitulare

- Sisteme care învaţă singure (SIS)
  - Instruire (învăţare) automata (Machine Learning - ML)
    - Învăţare supervizată → datele de antrenament sunt deja etichetate cu elemente din E, iar datele de test trebuie etichetate cu una dintre etichetele din E pe baza unui model (învăţat pe datele de antrenament) care face corespondenţa date-etichete
    - Învăţare nesupervizată → datele de antrenament NU sunt etichetate, trebuie învăţat un model de etichetare, iar apoi datele de test trebuie etichetate cu una dintre etichetele identificate de model
  - Sisteme

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Metoda celor mai mici patrate, Gradient Descent, Logistic regression
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 şi 13 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Sisteme inteligente

Sisteme care învaţă singure

Laura Dioşan & Dragoş Dobrean

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învaţă singure
  - Metoda celor mai mici patrate, Gradient Descent, Logistic regression
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit şi legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 şi 11 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Conținut

- Sisteme inteligente
  - Sisteme care învaţă singure (SIS)
    - Instruire (învăţare) automata (Machine Learning - ML)
      - Problematică
      - Proiectarea unui sistem de învăţare automată
      - Tipologie
        - Învăţare supervizată
        - Învăţare nesupervizată
        - Învăţare cu întărire
        - Teoria învăţării
    - Exemple de sisteme

Conținut

- Sisteme care învaţă singure (SIS)

  - "Field of study that gives computers the ability to learn without being explicitly programmed." -- Arthur Samuels (1959)

- Invățare
  - Supervizată
  - Nesupervizată
  - Reinforcement

Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
  - Presupunem cazul unei probleme de regresie
    - Date de intrare $x \in R^d$
    - Date de ieșire $y \in R$

    - Se cere un model **liniar** f care transformă x în y
    - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_d x_d$
    - Invățare supervizată

Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
  - Presupunem cazul unei probleme de regresie
    - Date de intrare $x^i \in R^d$, i=1,n
    - Date de ieșire $y^i \in R$
    - Se cere un model **liniar** f care transformă orice x^i în y^i , i=1,n
    - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_d x_d$

    - Se poate defini o funcție de cost
    - Loss = $\sum_{i=1,n} (y^i - f(x^i))^2$ -- minimizată ➔ valorile optime ale lui β

$\mathbf{x} = (1, x) = (1, x_1, x_2, \ldots, x_d)^T \in R^{d+1}$
$\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \ldots, \beta_d)^T \in R^{d+1}$
$f(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}$
Loss $(\boldsymbol{\beta}) = \| y - \mathbf{X} \beta \|^2$

$$X = \begin{matrix} 1 & x_{1,1} & x_{1,2} & x_{1,3} & \ldots & x_{1,d} \\ . \\ . \\ . \\ 1 & x_{n,1} & x_{n,2} & x_{n,3} & \ldots & x_{n,d} \end{matrix}$$

- Metoda celor mai mici pătrate
  - Presupunem cazul unei probleme de regresie
    - Date de intrare $x^i \in R^d$, i=1,n
    - Date de ieșire $y^i \in R$
    - Se cere un model **liniar** f care transformă orice x^i în y^i , i=1,n
    - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_d x_d$
    - Se poate defini o funcție de cost
    - Loss = $\sum_{i=1,n} (y^i - f(x^i))^2$ -- minimizată ➔ valorile optime ale lui β
    - Derivarea loss-ului după β : $\beta = (X^T X)^{-1} X^T y$
    - Daca d = 1, $\beta_1 = cov(x,y)/var(x)$, $\beta_0 = y - \beta_1 x$

Sisteme inteligente – SIS – Învățare automată

- Metoda *celor mai mici pătrate*
  - Anscombe Quartet

Ref. imagine

- Metoda *celor mai mici pătrate*
  - Residual plot

Ref. imagine

- Metoda *celor mai mici pătrate*
  - Residual plot

Ref. imagine

Sisteme inteligente – SIS – Învățare automată

- Metoda *gradient descent*
  - Presupunem cazul unei probleme de regresie
    - Date de intrare $x \in R^d$
    - Date de ieșire $y \in R$

    - Se cere un model **liniar** f care transformă x în y
    - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_d x_d$
    - Invățare supervizată

Ref. imagine

Sisteme inteligente – SIS – Învățare automată

- Metoda *gradient descent*
  - Modelarea coeficienților β:
    - la iterația 0: valori random (sau 0)
    - la iterația t + 1 (t = 0, 1, 2, …)

    $\beta_k(t+1) = \beta_k(t)$ - learning_rate * error(t) * x_k, k=1,2,...,d

    $\beta_0(t+1) = \beta_0(t)$ - learning_rate * error(t)
      - Unde
        - error(t) = computed - realOutput
        - error(t) = $\beta_0(t) + \beta_1(t)*x_1 + \beta_2(t)*x_2 + \ldots + \beta_d(t)*x_d - y$

Sisteme inteligente – SIS – Învățare automată

- Metoda *gradient descent* – versiuni:
  - Stochastic GD
    - Eroarea se calculează pentru fiecare exemplu de antrenament
    - Modelul se updatează pentru fiecare exemplu de antrenament (*online learning*)
  - Batch GD
    - Eroarea se calculează pentru fiecare exemplu de antrenament
    - Modelul se updatează dupa ce toate exemplele de antrenament au fost evaluate (la finalul unei epoci)
  - Mini-batch GD
    - Combinare a precedentelor două
    - Setul de date se împarte în mai multe părți (mini-batch-uri)
    - Eroarea se calculează pentru fiecare exemplu de antrenament dintr-un mini-batch
    - Modelul se updatează pentru fiecare exemplu de antrenament dintr-un mini-batch

Sisteme inteligente – SIS – Învățare automată

- Regresie Logistică (clasificare)
  - Presupunem cazul unei probleme de clasificare
    - Date de intrare $x^i \in R^d$, i=1,n
    - Date de ieșire $y^i \in \{0,1\}$ sau {label1, label2}

    - Se cere un model **liniar** f care separa orice x^i în 2 clase (etichetate cu 0 și 1)
    - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_d x_d$
    - Invățare supervizată

Sisteme inteligente – SIS – Învățare automată

- Regresie Logistică (clasificare)
  - Mapează datele intr-un set discret de clase (label-uri)
  - Tipuri:
    - Binar (Pass/Fail, True/False)
    - Multi (Cat,Dog,Panda)
    - Ordinal (Low,Medium,High)
  - Folosește funcția sigmoid pentru a decide clasa de apartenteță
  - Putem folosi gradient descent pentru minimzarea erorii

Sisteme inteligente – SIS – Învățare automată

- Regresie Logistică – sigmoid:
  - Mapează orice numar real in intervalul (0,1)

$$S(z) = \frac{1}{1 + e^{-z}}$$

Ref. imagine

Sisteme inteligente – SIS – Învățare automată

- Regresie Logistica
  - Modelarea coeficienților β:
    - la iterația 0: valori random (sau 0)
    - la iterația t + 1 (t = 0, 1, 2, …)

    $\beta_k(t+1) = \beta_k(t)$ - learning_rate * error(t) * x_k, k=1,2,...,d

    $\beta_0(t+1) = \beta_0(t)$ - learning_rate * error(t)
      - Unde
        - error(t) = Sigmoid(computed) - realOutput
        - error(t) = Sigmoid($\beta_0(t) + \beta_1(t)*x_1 + \beta_2(t)*x_2 + \ldots + \beta_d(t)*x_d$) – y
  - Clasificarea rezultatelor
    - (0,1) -> [$label_0$, $label_1$, .. $label_n$]

Sisteme inteligente – SIS – Învățare automată

- Metoda bazată pe algoritmi evolutivi
  - Modelarea coeficienților β cu ajutorul cromozomilor
  - Fitness-ul calitatea coeficienților β

Recapitulare

- Sisteme care învaţă singure (SIS)
  - Instruire (învăţare) automata (Machine Learning - ML)
    - Învăţare supervizată → datele de antrenament sunt deja etichetate cu elemente din E, iar datele de test trebuie etichetate cu una dintre etichetele din E pe baza unui model (învăţat pe datele de antrenament) care face corespondenţa date-etichete
    - Învăţare nesupervizată → datele de antrenament NU sunt etichetate, trebuie învăţat un model de etichetare, iar apoi datele de test trebuie etichetate cu una dintre etichetele identificate de model

Recapitulare

- Sisteme care învaţă singure (SIS)
  - Metoda celor mai mici patrate
    - Supervizată
    - Output continu (vânzări, preț, etc.)
    - Panta constantă
  - Gradient descent
    - Supervizată
    - Output continu (vânzări, preț, etc.)
    - Optimizare
  - Regresie Logistică
    - CLASIFICARE
    - Supervizată
    - Output – label-uri (clase) – set discret
    - Folosește Gradient descent

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 şi 13 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Sisteme inteligente

Sisteme care învaţă singure

– reţele neuronale artificiale –

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit şi legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 şi 13 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente

Sisteme bazate pe cunoştinţe

Inteligenţă computaţională

Sisteme inteligente – SIS – Învățare automată

Tipologie

- În funcţie de experienţa acumulată în timpul învăţării:
  - SI cu învăţare supervizată
  - SI cu învăţare nesupervizată
  - SI cu învăţare activă
  - SI cu învăţare cu întărire

- În funcţie de modelul învăţat (algoritmul de învăţare):
  - Arbori de decizie
  - **Reţele neuronale artificiale**
  - Algoritmi evolutivi
  - Maşini cu suport vectorial
  - Modele Markov ascunse

Sisteme inteligente – SIS – RNA

- Reţele neuronale artificiale (RNA)
  - Scop
  - Definire
  - Tipuri de probleme rezolvabile
  - Caracteristici
  - Exemplu
  - Proiectare
  - Evaluare
  - Tipologie

Sisteme inteligente – SIS – RNA

- Scop
  - Clasificare binară pentru orice fel de date de intrare (discrete sau continue)
    - Datele pot fi separate de:
      - o dreaptă → $ax + by + c = 0$ (dacă $m = 2$)
      - un plan → $ax + by + cz + d = 0$ (dacă $m = 3$)
      - un hiperplan $\sum a_i x_i + b = 0$ (dacă $m > 3$)
    - Cum găsim valorile optime pt. a, b, c, d, a_i?
      - Reţele neuronale artificiale (RNA)
      - Maşini cu suport vectorial (MSV)
  - De ce RNA?
  - Cum învaţă creierul?

Sisteme inteligente – SIS – RNA

- Scop → De ce RNA?
  - Unele sarcini pot fi efectuate foarte uşor de către oameni, însă sunt greu de codificat sub forma unor algoritmi
    - Recunoaşterea formelor
      - vechi prieteni
      - caractere scrise de mână
      - vocea
    - Diferite raţionamente
      - conducerea autovehiculelor
      - cântatul la pian
      - jucarea baschetului
      - înnotul
  - Astfel de sarcini sunt dificil de definit formal şi este dificilă aplicarea unui proces de raţionare pentru efectuarea lor

Sisteme inteligente – SIS – RNA

- Scop → Cum învaţă creierul?
  - Creierul uman - componenţă
    - Aproximativ 10.000.000.000 de neuroni conectaţi prin sinapse
    - Fiecare neuron
      - are un corp (soma), un axon şi multe dendrite
      - poate fi într-una din 2 stări:
        - activ – dacă informaţia care intră în neuron depăşeşte un anumit prag de stimulare –
        - pasiv – altfel
    - Sinapsă
      - Legătura între axon-ul unui neuron şi dendritele altui neuron
      - Are rol în schimbul de informaţie dintre neuroni
      - 5.000 de conexiuni / neuron (în medie)
    - În timpul vieţii pot să apară noi conexiuni între neuroni

Sisteme inteligente – SIS – RNA

- Scop → Cum învaţă creierul?
  - Cum "învaţă" (procesează informaţii)?
    - Conexiunile care de-a lungul trăirii unor experienţe s-au dovedit utile devin permanente (restul sunt eliminate)
    - Creierul este interesat de noutăţi
    - Modelul de procesare a informaţiei
      - Învăţare
      - Depozitare
      - Amintire
    - Memoria
      - Tipologie
        - De scurtă durată
        - Imediată → 30 sec.
        - De lucru
        - De lungă durată
      - Capacitate
        - Creşte odată cu vârsta
        - Limitată → învăţarea unei poezii pe strofe
      - Influenţată şi de stările emoţionale
    - Creierul
      - reţea de neuroni
      - sistem foarte complex, ne-liniar şi paralel de procesare a informaţiei
    - Informaţia este depozitată şi procesată de întreaga reţea, nu doar de o anumită parte a reţelei → informaţii şi procesare globală
    - Caracteristica fundamentală a unei reţele de neuroni → învăţarea ➔ reţele neuronale artificiale (RNA)

Sisteme inteligente – SIS – RNA

- Definire
  - Ce este o RNA?
  - RN biologice vs. RN artificiale
  - Cum învaţă reţeaua?

Sisteme inteligente – SIS – RNA

- Definire → Ce este o RNA?
  - O structură similară unei reţele neuronale bilogice
  - O mulţime de noduri (unităţi, neuroni, elemente de procesare) dispuse ca într-un graf pe mai multe straturi (*layere*)
    - Nodurile
      - au intrări şi ieşiri
      - efectuează un calcul simplu prin intermediul unei funcţii asociate → funcţie de activare
      - sunt conectate prin legături ponderate
        - Conexiunile între noduri conturează structura (arhitectura) reţelei
        - Conexiunile influenţează calculele care se pot efectua
    - Straturile
      - Strat de intrare
        - Conţine *m* (nr de atribute al unei date) noduri
      - Strat de ieşire
        - Conţine *r* (nr de ieşiri) noduri
      - Straturi intermediare (ascunse) – rol în "complicarea" reţelei
        - Diferite structuri
        - Diferite mărimi

Sisteme inteligente – SIS – RNA

- Definire → RN biologice vs. RN artificiale

RNB	RNA
Soma	Nod
Dendrite	Intrare
Axon	Ieşire
Activare	Procesare
Synapsă	Conexiune ponderată

- Definire → Cum învaţă reţeaua?
  - Plecând de la un set de *n* date de antrenament de forma

$$((x_{p1}, x_{p2}, \ldots, x_{pm}, y_{p1}, y_{p2}, \ldots, y_{pr}))$$

cu $p = 1, 2, \ldots, n$, m – nr atributelor, r – nr ieşirilor

  - se formează o RNA cu *m* noduri de intrare, *r* noduri de ieşire şi o anumită structură internă
    - un anumit nr de nivele ascunse, fiecare nivel cu un anumit nr de neuroni
    - cu legături ponderate între oricare 2 noduri
  - se caută valorile optime ale ponderilor între oricare 2 noduri ale reţelei prin minimizarea erorii
    - diferenţa între rezultatul real ***y*** şi cel calculat de către reţea

Sisteme inteligente – SIS – RNA

- Tipuri de probleme rezolvabile cu RNA
  - Datele problemei se pot reprezenta prin numeroase perechi atribut-valoare
  - Funcţia obiectiv poate fi:
    - Unicriterială sau multicriterială
    - Discretă sau cu valori reale
  - Datele de antrenament pot conţine erori (zgomot)
  - Timp de rezolvare (antrenare) prelungit

Sisteme inteligente – SIS – RNA

- Proiectare
  - Construirea RNA pentru rezolvarea problemei P
  - Iniţializarea parametrilor RNA
  - Antrenarea RNA
  - Testarea RNA

Sisteme inteligente – SIS – RNA

- Proiectare
  - Construirea RNA pentru rezolvarea unei probleme P
    - pp. o problemă de clasificare în care avem un set de date de forma:
      - (x^d, t^d), cu:
      - $x^d \in \mathbf{R}^m \rightarrow x^d=(x^d_1, x^d_2,..., x^d_m)$
      - $t^d \in \mathbf{R}^R \rightarrow t^d=(t^d_1, t^d_2,..., t^d_R)$,
      - cu d = 1,2,...,n,n+1,n+2,...,N
    - primele n date vor fi folosite drept bază de antrenament a RNA
    - ultimele N-n date vor fi folosite drept bază de testare a RNA

    - se construieşte o RNA astfel:
      - stratul de intrare conţine exact m noduri (fiecare nod va citi una dintre proprietăţile de intrare ale unei instanţe a problemei – $x^d_1, x^d_2,..., x^d_m$)
      - stratul de ieşire poate conţine R noduri (fiecare nod va furniza una dintre proprietăţile de ieşire ale unei instanţe a problemei $t^d_1, t^d_2,..., t^d_R$)
      - unul sau mai multe straturi ascunse cu unul sau mai mulţi neuroni pe fiecare strat

Sisteme inteligente – SIS – RNA

- Proiectare
  - Construirea RNA pentru rezolvarea problemei P
  - **Iniţializarea parametrilor RNA**
  - **Antrenarea RNA**
  - Testarea RNA

Sisteme inteligente – SIS – RNA

Proiectare

- Iniţializarea parametrilor RNA
  - Iniţializarea ponderile între oricare 2 noduri de pe straturi diferite
  - Stabilirea funcţiei de activare corespunzătoare fiecărui neuron (de pe straturile ascunse)

- Antrenarea (învăţarea) RNA
  - Scop:
    - stabilirea valorii optime a ponderilor dintre 2 noduri
  - Algoritm
    - Se caută valorile optime ale ponderilor între oricare 2 noduri ale reţelei prin minimizarea erorii (diferenţa între rezultatul real y şi cel calculat de către reţea)
  - Cum învaţă reţeaua?
    - Reţeaua = mulţime de unităţi primitive de calcul interconectate între ele ➔
      - Învăţarea reţelei = ∪ învăţarea unităţilor primitive
    - Unităţi primitive de calcul
      - Perceptron
      - Unitate liniară
      - Unitate sigmoidală

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
      - Fiecare nod are intrări şi ieşiri
      - Fiecare nod efectuează un calcul simplu
    - Procesarea neuronului
      - Se transmite informaţia neuronului
      - Neuronul procesează informaţia
      - Se citeşte răspunsul neuronului
    - Învăţarea neuronului – algoritmul de învăţare a ponderilor care procesează corect informaţiile
      - Se porneşte cu un set iniţial de ponderi oarecare
      - Cât timp nu este îndeplinită o condiţie de oprire
        - Se procesează informaţia şi se stabileşte calitatea ponderilor curente
        - Se modifică ponderile astfel încât să se obţină rezultate mai bune

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
      - Fiecare nod are intrări şi ieşiri
      - Fiecare nod efectuează un calcul simplu prin intermediul unei funcţii asociate
    - Procesarea neuronului
      - Se transmite informaţia neuronului → se calculează suma ponderată a intrărilor

$$net = \sum_{i=1}^{n} x_i w_i$$

      - Neuronul procesează informaţia → se foloseşte o funcţie de activare:
        - Funcţia constantă
        - Funcţia prag
        - Funcţia rampă
        - Funcţia liniară
        - Funcţia sigmoidală
        - Funcţia Gaussiană
        - Funcția Relu

Sisteme inteligente – SIS – RNA

- Proiectare→ Antrenarea RNA → Cum învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia constantă f(net) = const
    - Funcţia prag (c - pragul)

$$f(net) = \begin{cases} a, & \text{dacă } net < c \\ b, & \text{dacă } net > c \end{cases}$$

      - Pentru a=+1, b =-1 şi c = 0 ➔ funcţia semn
      - Funcţie discontinuă

    - Funcţia rampă

$$f(net) = \begin{cases} a, & \text{dacă } net \le c \\ b, & \text{dacă } net \ge d \\ a + \dfrac{(net - c)(b - a)}{d - c}, & \text{altfel} \end{cases}$$

    - Funcţia liniară

$$f(net) = a * net + b$$

      - Pentru a = 1 şi b = 0 ➔ funcţia identitate f(net)=net
      - Funcţie continuă

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia sigmoidală
      - În formă de S
      - Continuă şi diferenţiabilă în orice punct
      - Simetrică rotaţional faţă de un anumit punct (net = c)
      - Atinge asimptotic puncte de saturaţie

$$\lim_{net \to -\infty} f(net) = a \qquad \lim_{net \to \infty} f(net) = b$$

      - Exemple de funcţii sigmoidale:

$$f(net) = z + \frac{1}{1 + \exp(-x \cdot net + y)}$$

$$f(net) = \tanh(x \cdot net - y) + z \qquad \text{unde} \quad \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

      - Pentru y=0 şi z = 0 ➔ a=0, b = 1, c=0
      - Pentru y=0 şi z = -0.5 ➔ a=-0.5, b = 0.5, c=0
      - Cu cât x este mai mare, cu atât curba este mai abruptă

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia Gaussiană
      - În formă de clopot
      - Continuă
      - Atinge asimptotic un punct de saturaţie

$$\lim_{net \to \infty} f(net) = a$$

      - Are un singur punct de optim (maxim) – atins când net = μ
      - Exemplu

$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2} \left(\frac{net - \mu}{\sigma} \right)^2 \right]$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia ReLU
      - În formă de rampă
      - Continuă, monotonă
      - Derivata ei este monotonă
      - Codomeniu pozitiv [0, ∞)

$$f(net) = \max(0, net)$$

$$f(net) = \begin{cases} 0, & \text{dacă } net < 0 \\ net, & \text{dacă } net \geq 0 \end{cases}$$

      - Variantă: Leaky ReLU
        - Compensează problemele cu argumentele negative dint ReLU

$$f(net) = \begin{cases} a \cdot net, & \text{dacă } net < 0 \\ net, & \text{dacă } net \geq 0 \end{cases}$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
    - Procesarea neuronului
      - Se transmite informaţia neuronului → se calculează suma ponderată a intrărilor

$$net = \sum_{i=1}^{n} x_i w_i$$

      - Neuronul procesează informaţia → se foloseşte o funcţie de activare:
        - Funcţia constantă
        - Funcţia prag
        - Funcţia rampă
        - Funcţia liniară
        - Funcţia sigmoidală
        - Funcţia Gaussiană
      - Se citeşte răspunsul neuronului → se stabileşte dacă rezultatul furnizat de neuron coincide sau nu cu cel dorit (real)

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
    - Procesarea neuronului
    - Învăţarea neuronului
      - Algoritm

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - 2 reguli de bază
      - Regula perceptronului → algoritmul perceptronului
        1. Se porneste cu un set de ponderi oarecare
        2. Se stabileşte calitatea modelului creat pe baza acestor ponderi pentru **UNA** dintre datele de intrare
        3. Se ajustează ponderile în funcţie de calitatea modelului
        4. Se reia algoritmul de la pasul 2 până când se ajunge la calitate maximă
      - Regula Delta → algoritmul scăderii după gradient
        1. Se porneste cu un set de ponderi oarecare
        2. Se stabileşte calitatea modelului creat pe baza acestor ponderi pentru **TOATE** dintre datele de intrare
        3. Se ajustează ponderile în funcţie de calitatea modelului
        4. Se reia algoritmul de la pasul 2 până când se ajunge la calitate maximă

        - Similar regulii perceptronului, dar calitatea unui model se stabileşte în funcţie de toate datele de intrare (tot setul de antrenament)

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Pp că avem un set de date de antrenament de forma:
      - (x^d, t^d), cu:
        - $x^d \in \mathbf{R}^m$ ➔ $x^d=(x^d_1, x^d_2,..., x^d_m)$
        - $t^d \in \mathbf{R}^R$ ➔ $t^d=(t^d_1, t^d_2,..., t^d_R)$, şi R = 1 (adică $t^d=(t^d_1)$)
        - cu d = 1,2,...,n
      - RNA = unitate primitivă de calcul (un neuron) ➔ o reţea cu:
        - m noduri de intrare
        - legate de neuronul de calcul prin ponderile w_i, i =1,2,...,m şi
        - cu un nod de ieşire

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul perceptronului
      - Se bazează pe minimizarea erorii asociată unei instanţe din setul de date de antrenament
      - Modificarea ponderilor pe baza erorii asociate unei instanţe din setul de antrenament

Iniţializare ponderi din reţea

w_i = random(a,b), unde i=1,2,...,m

d = 1

Cât timp mai există exemple de antrenament clasificate incorect

Se activează neuronul şi se calculează ieşirea

Perceptron ➔ funcţia de activare este funcţia semn (funcţie prag de tip discret, nediferenţiabil)

$$o^d = sign(\mathbf{wx}) = sign(\sum_{i=1}^{m} w_i x_i)$$

Se stabileşte ajustarea ponderilor $\Delta w_i = \eta(t^d - o^d)x_i^d$, unde $i = 1,2,...,m$

unde η - rată de învăţare

Se ajustează ponderile $w_i' = w_i + \Delta w_i$

Dacă d < n atunci d++

Altfel d = 1

SfCâtTimp

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul scădere după gradient
      - Se bazează pe eroarea asociată întregului set de date de antrenament
      - Modificarea ponderilor în direcţia dată de cea mai abruptă pantă a reducerii erorii E(**w**) pentru tot setul de antrenament

$$E(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{n}(t^d - o^d)^2$$

      - Cum se determină cea mai abruptă pantă? ➔ se derivează E în funcţie de w (se stabileşte gradientul erorii E)

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_m}\right)$$

      - Gradientul erorii E se calculează în funcţie de funcţia de activare a neuronului (care trebuie să fie diferenţiabilă, deci continuă)
        - Funcţia liniară $f(net) = \sum_{i=1}^{m} w_i x_i^d$
        - Funcţia sigmoidală $f(net) = \frac{1}{1+e^{-\mathbf{wx}}} = \frac{1}{1+e^{-\sum_{i=1}^{m} w_i x_i^d}}$
      - Cum se ajustează ponderile?

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \text{ unde } i = 1,2,\ldots,m$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul scădere după gradient → calcularea gradientului erorii
      - Funcţia liniară

$$f(net) = \sum_{i=1}^{m} w_i x_i^d$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial \frac{1}{2}\sum_{d=1}^{n}\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n}\frac{\partial\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n} 2\left(t^d - o^d\right)\frac{\partial\left(t^d - \mathbf{w}\mathbf{x}^d\right)}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \sum_{d=1}^{n}\left(t^d - o^d\right)\frac{\partial\left(t^d - w_1 x_1^d - w_2 x_2^d - \ldots - w_m x_m^d\right)}{\partial w_i} = \sum_{d=1}^{n}\left(t^d - o^d\right)\left(-x_i^d\right)$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = \eta\sum_{d=1}^{n}\left(t^d - o^d\right)x_i^d$$

      - Funcţie sigmoidală

$$f(net) = \frac{1}{1+e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1+e^{-\sum_{i=1}^{m} w_i x_i^d}} \qquad y = s(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{\partial s(z)}{\partial z} = s(z)\left(1 - s(z)\right)$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial \frac{1}{2}\sum_{d=1}^{n}\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n}\frac{\partial\left(t^d - o^d\right)^2}{\partial w_i} = \frac{1}{2}\sum_{d=1}^{n} 2\left(t^d - o^d\right)\frac{\partial\left(t^d - sig(\mathbf{w}\mathbf{x}^d)\right)}{\partial w_i} = \sum_{d=1}^{n}\left(t^d - o^d\right)\left(1 - o^d\right)o^d\left(-x_i^d\right)$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = \eta\sum_{d=1}^{n}\left(t^d - o^d\right)\left(1 - o^d\right)o^d x_i^d$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul scădere după gradient (ASG)

ASG simplu	ASG stocastic
Iniţializare ponderi din reţea w_i = random(a,b), unde i=1,2,...,m d = 1 Cât timp nu este îndeplinită condiţia de oprire Δw_i=0, unde i=1,2,...,m Pentru fiecare exemplu de antrenament (x^d,t^d), unde d=1,2,...,n Se activează neuronul si se calculează ieşirea o^d funcţia de activare = funcţia liniară ➔ $o^d=\mathbf{w}\mathbf{x}^d$ funcţia de activare = funcţia sigmoid ➔ $o^d=sig(\mathbf{w}\mathbf{x}^d)$ Pentru fiecare pondere w_i, unde i =1,2,...,m Se stabileşte ajustarea ponderii $\Delta w_i = \Delta w_i - \eta \frac{\partial E}{\partial w_i}$ Pentru fiecare pondere w_i, unde i =1,2,...,m Se ajustează fiecare pondere w_i $w_i = w_i + \Delta w_i$	Iniţializare ponderi din reţea w_i = random(a,b), unde i=1,2,...,m d = 1 Cât timp nu este îndeplinită condiţia de oprire Δw_i=0, unde i=1,2,...,m Pentru fiecare exemplu de antrenament (x^d,t^d), unde d=1,2,...,n Se activează neuronul si se calculează ieşirea o^d funcţia de activare = funcţia liniară ➔ $o^d=\mathbf{w}\mathbf{x}^d$ funcţia de activare = funcţia sigmoid ➔ $o^d=sig(\mathbf{w}\mathbf{x}^d)$ Pentru fiecare pondere w_i, unde i =1,2,...,m Se stabileşte ajustarea ponderilor $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ Se ajustează ponderea w_i $w_i = w_i + \Delta w_i$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului

Diferenţe	Algoritmul perceptronului	Algoritmul scădere după gradient (regula Delta)
Ce reprezintă o^d	o^d=sign($\mathbf{wx}^d$)	o^d=$\mathbf{wx}^d$ sau o^d=sig($\mathbf{wx}^d$)
Cum converge	Într-un nr finit de paşi (până la separarea perfectă)	Asimtotic (spre eroarea minimă)
Ce fel de probleme se pot rezolva	Cu date liniar separabile	Cu orice fel de date (separabile liniar sau ne-liniar)
Ce tip de ieşire are neuronul	Discretă şi cu prag	Continuă şi fără prag

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA
  - Cum învaţă reţeaua?
    - Reţeaua = mulţime de unităţi primitive de calcul interconectate între ele ➔
      - Învăţarea reţelei = ∪ învăţarea unităţilor primitive
    - Reţea cu mai mulţi neuroni aşezaţi pe unul sau mai multe straturi ➔ RNA este capabilă să înveţe un model mai complicat (nu doar liniar) de separare a datelor
    - Algoritmul de învăţare a ponderilor ➔ backpropagation
      - Bazat pe algoritmul scădere după gradient
      - Îmbogăţit cu:
        - Informaţia se propagă în RNA înainte (dinspre stratul de intrare spre cel de ieşire)
        - Eroarea se propagă în RNA înapoi (dinspre stratul de ieşire spre cel de intrare)

Se iniţializează ponderile
Cât timp nu este îndeplinită condiţia de oprire
Pentru fiecare exemplu (x^d,t^d)
Se activează fiecare neuron al reţelei
Se propagă informaţia înainte şi se calculează ieşirea corespunzătoare fiecărui neuron al reţelei
Se ajustează ponderile
Se stabileşte şi se propagă eroarea înapoi
Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire
Se propagă aceste erori înapoi în toată reţeaua → se distribuie erorile pe toate conexiunile existente în reţea proporţional cu valorile ponderilor asociate acestor conexiuni
Se modifică ponderile

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA
  - Cum învaţă o întreagă RNA?
    - Pp că avem un set de date de antrenament de forma:
      - (x^d, t^d), cu:
        - $x^d \in \mathbf{R}^m \rightarrow x^d=(x^d_1, x^d_2,..., x^d_m)$
        - $t^d \in \mathbf{R}^R \rightarrow t^d=(t^d_1, t^d_2,..., t^d_R)$
        - cu d = 1,2,...,n
    - Presupunem 2 cazuri de RNA
      - O RNA cu un singur strat ascuns cu H neuroni → RNA_1
        - m neuroni pe stratul de intrare,
        - R neuroni pe stratul de ieşire,
        - H neuroni pe stratul ascuns
        - Ponderile între stratul de intrare şi cel ascuns w^1_{ih} cu i=1,2,...,m, h = 1,2,...,H
        - Ponderile între stratul ascuns şi cel de işire w^2_{hr} cu h = 1,2,...,H şi r = 1,2,...,R

      - O RNA cu p straturi ascunse, fiecare strat cu Hi (i =1,2,...,p) neuroni → RNA_p
        - m neuroni pe stratul de intrare,
        - R neuroni pe stratul de ieşire,
        - P straturi ascunse
        - Hp neuroni pe stratul ascuns p, p =1,2,...,P
        - Ponderile între stratul de intrare şi primul strat ascuns $w^1_{ih_1}$ cu i=1,2,...,m, h1 = 1,2,...,H1
        - Ponderile între primul strat ascuns şi cel de-al doilea strat ascuns $w^2_{h_1h_2}$ cu h1 = 1,2,...,H1 şi h2 = 1,2,...,H2
        - Ponderile între cel de-al doilea strat ascuns şi cel de-al treilea strat ascuns $w^3_{h_2h_3}$ cu h2 = 1,2,...,H2 şi h3 = 1,2,...,H3
        - ...
        - Ponderile între cel de-al p-1 strat ascuns şi ultimul strat ascuns $w^p_{h_{p-1}h_p}$ cu hp-1 = 1,2,...,Hp-1 şi hp = 1,2,...,Hp
        - Ponderile între ultimul strat ascuns şi cel de ieşire $w^{p+1}_{h_pr}$ cu hp = 1,2,...,Hp şi r = 1,2,...,R

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă o întreagă RNA?
  - Algoritmul backpropagation pentru RNA_1

Se iniţializează ponderile w_{ih}^1 şi w_{hr}^2 cu i=1,2,...,m, h = 1,2,...,H şi r = 1,2,...,R

Cât timp nu este îndeplinită condiţia de oprire

Pentru fiecare exemplu (x^d,t^d)

Se activează fiecare neuron al reţelei

Se propagă informaţia înainte şi se calculează ieşirea corespunzătoare fiecărui neuron al reţelei

$$o_h^d = \sum_{i=1}^{m} w_{ih}^1 x_i^d \text{ sau } o_h^d = sig\left(\sum_{i=1}^{m} w_{ih}^1 x_i^d\right), \text{cu } h = 1,2,\ldots,H$$

$$o_r^d = \sum_{h=1}^{H} w_{hr}^2 o_h^d \text{ sau } o_r^d = sig\left(\sum_{h=1}^{H} w_{hr}^2 o_h^d\right), \text{cu } r = 1,2,\ldots,R$$

Se ajustează ponderile

Se stabileşte şi se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire

$$\delta_r^d = t_r^d - o_r^d \text{ sau } \delta_r^d = o_r^d(1-o_r^d)(t_r^d - o_r^d), \text{cu } r = 1,2,\ldots,R$$

Se modifică ponderile între nodurile de pe stratul ascuns şi stratul de ieşire

$$w_{hr}^2 = w_{hr}^2 + \eta\delta_r^d o_h^d, \text{ unde h} = 1,2,\ldots,H \text{ şi r} = 1,2,\ldots,R$$

Se propagă erorile nodurilor de pe stratul de ieşire înapoi în toată reţeaua → se distribuie erorile pe toate conexiunile existente în reţea proporţional cu valorile ponderilor asociate acestor conexiuni

$$\delta_h^d = \sum_{r=1}^{R} w_{hr}^2 \delta_r^d \text{ sau } \delta_h^d = o_h^d\left(1-o_h^d\right)\sum_{r=1}^{R} w_{hr}^2 \delta_r^d$$

Se modifică ponderile între nodurile de pe stratul de intrare şi stratul ascuns

$$w_{ih}^1 = w_{ih}^1 + \eta\delta_h^d x_i^d, \text{ unde } i = 1,2,\ldots,m \text{ şi } h = 1,2,\ldots,H$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă o întreagă RNA?
  - Algoritmul backpropagation pentru RNA_p

Se iniţializează ponderile $w^1_{ih_1}, w^2_{h_1h_2}, \ldots, w^p_{h_{p-1}h_p}, w^{p+1}_{h_pr}$

Cât timp nu este îndeplinită condiţia de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al reţelei

Se propagă informaţia înainte şi se calculează ieşirea corespunzătoare fiecărui neuron al reţelei

$$o^d_{h_1} = \sum_{i=1}^{m} w^1_{ih_1} x^d_i \text{ sau } o^d_{h_1} = sig\left(\sum_{i=1}^{m} w^1_{ih_1} x^d_i\right), \text{cu } h_1 = 1,2,\ldots,H_1$$

$$o^d_{h_2} = \sum_{h_1=1}^{H_1} w^2_{h_1h_2} o^d_{h_1} \text{ sau } o^d_{h_2} = sig\left(\sum_{h_1=1}^{H_1} w^2_{h_1h_2} o^d_{h_1}\right), \text{cu } h_2 = 1,2,\ldots,H_2$$

...

$$o^d_{h_p} = \sum_{h_{p-1}=1}^{H_{p-1}} w^p_{h_{p-1}h_p} o^d_{h_{p-1}} \text{ sau } o^d_{h_p} = sig\left(\sum_{h_{p-1}=1}^{H_{p-1}} w^p_{h_{p-1}h_p} o^d_{h_{p-1}}\right), \text{cu } h_p = 1,2,\ldots,H_p$$

$$o^d_r = \sum_{h_p=1}^{H_p} w^{p+1}_{h_pr} o^d_{h_p} \text{ sau } o^d_r = sig\left(\sum_{h_p=1}^{H_p} w^{p+1}_{h_pr} o^d_{h_p}\right), \text{cu } r = 1,2,\ldots,R$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă o întreagă RNA?
  - Algoritmul backpropagation pentru RNA_p

Se iniţializează ponderile $w^1_{ih_1}, w^2_{h_1h_2}, \ldots, w^p_{h_{p-1}h_p}, w^{p+1}_{h_pr}$

Cât timp nu este îndeplinită condiţia de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al reţelei

Se ajustează ponderile

Se stabileşte şi se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire

$$\delta^d_r = t^d_r - o^d_r \text{ sau } \delta^d_r = o^d_r(1-o^d_r)(t^d_r - o^d_r), \text{ cu } r = 1,2,\ldots,R$$

Se modifică ponderile între nodurile de pe ultimul strat ascuns şi stratul de ieşire

$$w^{p+1}_{h_pr} = w^{p+1}_{h_pr} + \eta\delta^d_r o^d_{h_p}, \text{ unde } h_p = 1,2,\ldots,H_p \text{ şi } r = 1,2,\ldots,R$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă o întreagă RNA?
  - Algoritmul backpropagation pentru RNA_p

Se iniţializează ponderile $w^1_{ih_1}, w^2_{h_1h_2}, \ldots, w^p_{h_{p-1}h_p}, w^{p+1}_{h_pr}$

Cât timp nu este îndeplinită condiţia de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al reţelei

Se ajustează ponderile

Se stabileşte şi se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire

Se modifică ponderile între nodurile de pe ultimul strat ascuns şi stratul de ieşire

Se propagă (pe starturi) aceste erori înapoi în toată reţeaua → se distribuie erorile pe toate conexiunile existente în reţea proporţional cu valorile ponderilor asociate acestor conexiuni şi se modifică ponderile corespunzătoare

$$\delta^d_{h_p} = \sum_{r=1}^{R} w^{p+1}_{h_pr}\delta^d_r \text{ sau } \delta^d_{h_p} = o^d_{h_p}\left(1-o^d_{h_p}\right)\sum_{r=1}^{R} w^{p+1}_{h_pr}\delta^d_r$$

$$w^{p+1}_{h_pr} = w^{p+1}_{h_pr} + \eta\delta^d_r o^d_{h_p}, \text{ unde } h_p = 1,2,\ldots,H_p \text{ şi } r = 1,2,\ldots,R$$

$$\delta^d_{h_{p-1}} = \sum_{h_p=1}^{H_p} w^p_{h_{p-1}h_p}\delta^d_{h_p} \text{ sau } \delta^d_{h_{p-1}} = o^d_{h_{p-1}}\left(1-o^d_{h_{p-1}}\right)\sum_{h_p=1}^{H_p} w^p_{h_{p-1}h_p}\delta^d_{h_p}$$

$$w^p_{h_{p-1}h_p} = w^p_{h_{p-1}h_p} + \eta\delta^d_{h_p} o^d_{h_{p-1}}, \text{ unde } h_{p-1} = 1,2,\ldots,H_{p-1} \text{ şi } h_p = 1,2,\ldots,H_p$$

...

$$\delta^d_{h_1} = \sum_{h_2=1}^{H_2} w^2_{h_1h_2}\delta^d_{h_2} \text{ sau } \delta^d_{h_1} = o^d_{h_1}\left(1-o^d_{h_1}\right)\sum_{h_2=1}^{H_2} w^2_{h_1h_2}\delta^d_{h_2}$$

$$w^1_{ih_1} = w^1_{ih_1} + \eta\delta^d_{h_1} x^d_i, \text{ unde } i = 1,2,\ldots,m \text{ şi } h_1 = 1,2,\ldots,H_1$$

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare?
  - Probleme de regresie
    - Stratul de iesire are un nr de neuroni egal cu nr de variabile care trebuie prezise
    - Identificarea erorii: pentru fiecare neuron de output
      - Suma diferentelor intre valorile prezise si cele reale
        - Diferentele in valoare absoluta (L1, MAE)
          - Robustete la valori extreme (outliers)
          - Dar, gradienti mari ←→ eroare mica
            - Solutii: rata de invatare dinamica sau MSE
        - Patratul diferentelor (L2, MSE)
          - Stabilitatea solutiilor
          - Dar sensibilitate la valori extreme
        - Huber

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

      - Pentru
        - Un exemplu din date
        - Pentru mai multe exemple din date (batch)
    - Functia de cost (loss) = eroarea

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare?
  - Probleme de clasificare – clasificarea fructelor in imagini
    - Binara

    - Multi-clasa

    - Multi-label

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare? → Problem clasificare – clasificarea fructelor in imagini → Binara
  - Stratul de iesire are un singur neuron cu functie de activare sigmoid
    - Outputul neuronului este transformat prin sigmoid ➔ o valoare intre 0 si 1 ➔ probabilitatea apartenentei la una din cele 2 clase (➔ acuratetea, precizia, rapelul)

  - Functia de cost != acuratete (precizie, recall)
    - Functia de cost = functia de loss = Cross-entropy loss (Logistic Loss or Multinomial Logistic Loss)
      - Cross Entropy pentru 2 distributii p si q
        - CE(p,q)=-sum(p_i $\log_2(q_i)$), I = 1, 2, …
      - Cross-entropy pt clasificare binara
        - Pentru fiecare exemplu din date
          - Se transforma fiecare eticheta reala intr-o lista cu 2 probabilitati ➔ ***p***
            [1-eticheta_clasa, eticheta_clasa]
          - se transforma outputul sigmoidat al neuronului de iesire intr-o lista de 2 probabilitati ➔ ***q***
            [1 – out_sigm, out_sigm]
          - se calculeaza entropia conditionata intre ***p*** si ***q*** : CE(exemplul crt)
        - Se calculeaza media entropiilor conditionate pt fiecare exemplu

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificarea fructelor in imagini → multi-clasa
  - Abordarea one-vs-all
    - O problema cu C-clase ➔ C probleme de clasificare binara
      - Apple vs. others, Pear vs. others, Grape vs others
    - Se antreneaza C modele de clasificare binara (neuronul de output are activare sigmoidala)
    - La testare, se aplica toate cele C modele antrenate si se alege clasa indicata de cel cu outputul cel mai mare
  - Abordarea soft-max
    - O retea cu C neuroni in stratul de iesire si activare softmax

$$out'(neuron_i) = \frac{e^{out(neuron_i)}}{\sum_{i=1}^{C} e^{out(neuron_i)}}$$

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificare fructelor in imagini → multi-clasa → Abordarea soft-max
  - Stratul de iesire are C neuroni (C – nr de clase)
    - Outputrile neuronilor (s) se transforma prin softmax ➔ o valoare intre 0 si 1 ➔ probabilitatea apartenentei la una din cele C clase (➔ acuratetea, precizia, rapelul)

$$q = \left[\frac{e^{s_1}}{\sum_{j=1}^{C} e^{s_j}}, \frac{e^{s_2}}{\sum_{j=1}^{C} e^{s_j}}, \ldots, \frac{e^{s_C}}{\sum_{j=1}^{C} e^{s_j}} \right]$$

  - Functia de cost != acuratete (precizie, recall)
    - Functia de cost = functia de loss = categorical cross-entropy loss (Softmax Loss)
      - Cross Entropy pentru 2 distributii p si q
        - CE(p,q)=-sum(p_i $\log_2(q_i)$), I = 1, 2, …
      - Cross-entropy pt clasificare multi-clasa
        - Pentru fiecare exemplu din date
          - Se transforma fiecare eticheta reala intr-o lista cu C probabilitati, probabilitatea clasei corecte fiind 1, restul 0 (one-hot encoding) ➔ ***p***

            p = [1, 0, 0], **p** = [0,1,0], **p** = [0,0,1]. **p** = [0,1,0]
          - Pentru fiecare exemplu din date, se transforma outputurile neuronilor cu softmax ➔ ***q***
          - Se calculeaza entropia conditionata intre ***p*** si ***q*** : CE(exemplul crt)

$$CE = -\sum_{i=1}^{C} p_i \log(q_i) = -\log(q_{correctClass})$$

        - Se calculeaza media entropiilor conditionate pt fiecare exemplu

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificarea fructelor in imagini → multi-label
  - Stratul de iesire are C neuroni (C – nr de clase)
    - Outputrile neuronilor (s) se transforma prin sigmoid ➔ o valoare intre 0 si 1 ➔ probabilitatea apartenentei la una din cele C clase (➔ acuratetea, precizia, rapelul)

$$\left[\frac{1}{1+e^{-s_1}}, \frac{1}{1+e^{-s_2}}, \ldots, \frac{1}{1+e^{-s_C}}\right]$$

  - Functia de cost != acuratete (precizie, recall)
    - Functia de cost = functia de loss = Binary cross entropy (sigmoid cross entropy) pt fiecare eticheta posibila
      - Cross Entropy pentru 2 distributii p si q
        - CE(p,q)=-sum(p_i $\log_2(q_i)$), i= 1, 2, …
      - Cross-entropy pt clasificare multi-label
        - Pentru fiecare exemplu din date
          - Se transforma fiecare eticheta reala intr-o lista cu C probabilitati, probabilitatea clasei corecte fiind 1, restul 0 (one-hot encoding) ➔ ***p***
            p = [1, 0, 0], **p** = [0,1,0], **p** = [0,0,1]. **p** = [0,1,0]
          - Pentru fiecare exemplu din date, se transforma outputurile neuronilor cu sigmoid ➔ ***q***
            qi **=** sigm(si), i = 1, 2, .., C, ***q*** = [qi, 1- qi]
          - Se calculeaza entropia conditionata intre ***p*** si ***q*** : CE(exemplul crt)

$$CE_i = -p_i \log(q_i) - (1-p_i)\log(1-q_i)$$

$$CE = \sum_{i=1}^{C} CE_i$$

        - Se calculeaza media entropiilor conditionate pt fiecare exemplu

Sisteme inteligente – SIS – RNA

- Cum se masoara calitatea algoritmului de invatare? → Probleme de clasificare – clasificarea fructelor in imagini → multi-class & multi-label
  - Cross-entropy
    - Classic loss

$$CE = -\sum_{i=1}^{C} p_i \log(q_i)$$

    - Focal loss

$$CE = -\sum_{i=1}^{C} (1-q_i)^{\gamma} p_i \log(q_i)$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă o întreagă RNA?
  - Algoritmul backpropagation
    - Condiţii de oprire
      - S-a ajuns la eroare 0
      - S-au efectuat un anumit număr de iteraţii
        - La o iteraţie se procesează un singur exemplu
        - n iteraţii = o epocă

Sisteme inteligente – SIS – RNA

- Proiectare
  - Construirea RNA pentru rezolvarea problemei P
  - Iniţializarea parametrilor RNA
  - Antrenarea RNA
  - **Testarea RNA**

Sisteme inteligente – SIS – RNA

- Proiectare
  - Testarea RNA
    - Se decodifică modelul învăţat de RNA
      - prin combinarea ponderilor cu intrările
      - ţinând cont de funcţiile de activare a neuronilor şi de structura reţelei

Sisteme inteligente – SIS – RNA

- Exemplu

Clasificare binară cu m=2 intrări

Clasificare binară cu m=3 intrări

Sisteme inteligente – SIS – RNA

- Exemplu
  - Perceptron pentru rezolvarea problemei *ŞI logic*

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Sisteme inteligente – SIS – RNA

- Exemplu
  - Perceptron - limitări
    - Un perceptron poate învăţa operaţiile AND şi OR, dar nu poate învăţa operaţia XOR (nu e liniar separabilă)

(a) AND ($x_1 \cap x_2$) (b) OR ($x_1 \cup x_2$) (c) Exclusive-OR ($x_1 \oplus x_2$)

    - Nu poate clasifica date non-liniar separabile
      - soluţii
        - Neuron cu un prag continu
        - Mai mulţi neuroni

Sisteme inteligente – SIS – RNA

- Tipologie
  - RNA feed-forward
    - Informaţia se procesează şi circulă de pe un strat pe altul
    - Conexiunile între noduri nu formează cicluri
    - Se folosesc în special pentru învăţarea supervizată
    - Funcţiile de activare a nodurilor → liniare, sigmoidale, gaussiene
  - RNA recurente (cu feedback)
    - Pot conţine conexiuni între noduri de pe acelaşi strat
    - Conexiunile între noduri pot forma cicluri
    - RNA de tip Jordan
    - RNA de tip Elman
    - RNA de tip Hopfield

      } pentru învăţarea supervizată
    - RNA auto-organizate → pentru învăţarea nesupervizată
      - De tip Hebbian
      - De tip Kohonen (*Self organised maps*)

Sisteme inteligente – SIS – RNA

- Avantaje
  - Pot rezolva atât probleme de învăţare super-vizată, cât şi nesupervizată
  - Pot identifica relaţii dinamice şi neliniare între date
  - Pot rezolva probleme de clasificare cu oricâte clase (multi-clasă)
  - Se pot efectua calcule foarte rapid (în paralel şi distribuit)

- Dificultăţi şi limite
  - RNA se confruntă cu problema overfitting-ului chiar şi când modelul se învaţă prin validare încrucişată
  - RNA pot găsi (uneori) doar optimele locale (fără să identifice optimul global)

Recapitulare

- Sisteme care învaţă singure (SIS)
  - Reţele neuronale artificiale
    - Modele computaţionale inspirate de reţelele neuronale artificiale
    - Grafe speciale cu noduri aşezate pe straturi
      - Strat de intrare → citeşte datele de intrare ale problemei de rezolvat
      - Strat de ieşire → furnizează rezultate problemei date
      - Strat(uri) ascunse → efectuează calcule
    - Nodurile (neuronii)
      - Au intrări ponderate
      - Au funcţii de activare (liniare, sigmoidale, etc)
      - necesită antrenare → prin algoritmi precum:
        - Perceptron
        - Scădere după gradient
    - Algoritm de antrenare a întregii RNA → Backpropagation
      - Informaţia utilă se propagă înainte
      - Eroarea se propagă înapoi

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- capitolul 15din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documentele din directorul *12_svm* şi *13_GP*

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

INTELIGENȚĂ ARTIFICIALĂ

Sisteme inteligente

Sisteme care învaţă singure

– kNN si programare genetică –

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - kNN
  - Algoritmi evolutivi
  - Maşini cu suport vectorial
- Sisteme bazate pe reguli
- Sisteme hibride

Materiale de citit şi legături utile

- capitolul 15 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 şi 11 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente

Sisteme inteligente – SIS – Învățare automată

- Tipologie
  - În funcţie de experienţa acumulată în timpul învăţării:
    - SI cu învăţare supervizată
    - SI cu învăţare nesupervizată
    - SI cu învăţare activă
    - SI cu învăţare cu întărire
  - În funcţie de modelul învăţat (algoritmul de învăţare):
    - Reţele neuronale artificiale
    - **Maşini cu suport vectorial (MSV)**
    - **Algoritmi evolutivi**
    - **kNN**
    - Arbori de decizie
    - Modele Markov ascunse

Materiale de citit şi legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 şi 11 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente – SIS – Învățare automată

Tipologie

- În funcţie de experienţa acumulată în timpul învăţării:
  - SI cu învăţare supervizată
  - SI cu învăţare nesupervizată
  - SI cu învăţare activă
  - SI cu învăţare cu întărire

- În funcţie de modelul învăţat (algoritmul de învăţare):
  - Arbori de decizie
  - Reţele neuronale artificiale
  - **Maşini cu suport vectorial (MSV)**
  - Algoritmi evolutivi
  - Modele Markov ascunse

Sisteme inteligente – SIS – MSV

- Maşini cu suport vectorial (MSV)
  - Definire
  - Tipuri de probleme rezolvabile
  - Avantaje
  - Dificultăţi
  - Tool-uri

Sisteme inteligente – SIS – MSV

- Definire
  - Dezvoltate de Vapnik în 1970
  - Popularizate după 1992
  - Clasificatori liniari care identifică un hiperplan de separare a clasei pozitive de clasa negativă
  - Au o fundamentare teoretică foarte riguroasă
  - Funcţionează foarte bine pentru date de volum mare (analiza textelor, analiza imaginilor)

  - **Reamintim**
    - Problemă de învăţare supervizată în care avem un set de date de forma:
      - (x^d, t^d), cu:
      - $x^d \in \mathbf{R}^m \rightarrow x^d=(x^d_1, x^d_2,..., x^d_m)$
      - $t^d \in \mathbf{R} \rightarrow t^d \in \{1, -1\}$, 1 → clasă pozitivă, -1 → clasă negativă
      - cu d = 1,2,...,n,n+1,n+2,...,N
    - Primele n date (se cunosc x^d şi t^d) vor fi folosite drept bază de antrenament a MSV
    - Ultimele N-n date (se cunosc doar x^d, fără t^d) vor fi folosite drept bază de testare a MSV

Sisteme inteligente – SIS – MSV

- Definire
  - MSV găseşte o funcţie liniară de forma $f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$, ($\mathbf{w}$ -vector pondere) a.î.

$$y_i = \begin{cases} 1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & if \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

  - $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ → hiperplanul de decizie care separă cele 2 clase

Sisteme inteligente – SIS – MSV

- Definire
  - Pot exista mai multe hiperplane
    - Care este cel mai bun hiperplan?
  - MSV caută hiperplanul cu cea mai largă margine (cel care micşorează eroarea de generalizare)
    - Algoritmul SMO (*Sequential minimal optimization*)

Sisteme inteligente – SIS – MSV

- Tipuri de probleme rezolvabile
  - Probleme de clasificare → Cazuri de date
    - Liniar separabile
      - Separabile
        - Eroarea = 0

      - Ne-separabile
        - Se relaxează constrângerile → se permit unele erori
        - C – coeficient de penalizare

Sisteme inteligente – SIS – MSV

Cazuri de date

- Non-liniar separabile
  - Spaţiul de intrare se transformă într-un spaţiu cu mai multe dimensiuni (*feature space*), cu ajutorul unei funcţii kernel, unde datele devin liniar separabile

  - Kernele posibile
    - Clasice
      - Polynomial kernel: $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + 1)^d$
      - RBF kernel: $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = exp(- \sigma\ |\boldsymbol{x}_1 - \boldsymbol{x}_2|^2)$
    - Kernele multiple
      - Liniare: $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum w_i K_i$
      - Ne-liniare
        - Fără coeficienţi: $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = K_1 + K_2 * exp(K_3)$
        - Cu coeficienţi: $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = K_1 + c_1 * K_2 * exp(c_2 + K_3)$

Sisteme inteligente – SIS – MSV

- Configurarea MSV
  - Parametrii unei MSV
    - Coeficientul de penalizare C
      - C – mic → convergenţă lentă
      - C – mare → convergenţă rapidă
    - Parametrii funcţiei kernel (care kernel şi cu ce parametri)
      - Dacă m (nr de atribute) este mult mai mare decât n (nr de instanţe)
        - MSV cu kernel liniar (MSV fără kernel) → $K(\boldsymbol{x}^{d1}, \boldsymbol{x}^{d2}) = \boldsymbol{x}^{d1} \cdot \boldsymbol{x}^{d2}$
      - Dacă m (nr de atribute) este mare, iar n (nr de instanţe) este mediu
        - MSV cu kernel Gaussian $K(\boldsymbol{x}^{d1}, \boldsymbol{x}^{d2}) = \exp(-||\boldsymbol{x}^{d1} - \boldsymbol{x}^{d2}||^2/2\sigma^2)$
        - σ – dispersia datelor de antrenament
        - Atributele instanţelor trebuie normalizate (scalate la (0,1))
      - m (nr de atribute) este mic, iar n (nr de instanţe) este mare
        - Se adaugă noi atribute, iar apoi
        - MSV cu kernel liniar

Sisteme inteligente – SIS – MSV

- MSV pentru probleme de clasificare supervizate cu mai mult de 2 clase
  - Una vs. restul (one *vs.* all)

Sisteme inteligente – SIS – MSV

- MSV structurate
  - Învăţare automată
    - Normală f: $\mathcal{X} \rightarrow \mathbf{R}$
      - Intrări de orice fel
      - Ieşiri numerice (naturale, întregi, reale)
    - Structurată: $\mathcal{X} \rightarrow \mathcal{Y}$
      - Intrări de orice fel
      - Ieşiri de orice fel (simple sau structurate)
  - Informaţii structurate
    - Texte şi hiper-texte
    - Molecule şi structuri moleculare
    - Imagini

Sisteme inteligente – SIS – MSV

- MSV structurate
  - Aplicaţii
    - Procesarea limbajului natural
      - Traduceri automate (ieşiri → propoziţii)
      - Analiza sintactică şi/sau morfologică a propoziţiilor (ieşiri → arborele sintactic şi/sau morfologic)
    - Bioinformatică
      - Predicţia unor structuri secundare (ieşirile → grafe bi-partite)
      - Predicţia funcţionării unor enzime (ieşirile → *path*-uri în arbori)
    - Procesarea vorbirii
      - Transcrieri automate (ieşiri → propoziţii)
      - Transformarea textelor în voce (ieşiri → semnale audio)
    - Robotică
      - Planificare (ieşirile → secvenţe de acţiuni)

Sisteme inteligente – SIS – MSV

- Avantaje
  - Pot lucra cu orice fel de date (liniar separabile sau nu, distribuit uniform sau nu, cu distribuţie cunoscută sau nu)
    - Funcţiile kernel care crează noi atribute (features) → straturile ascunse dintr-o RNA
  - Dacă problema e convexă oferă o soluţie unică → optimul global
    - RNA pot asocia mai multe soluţii → optime locale
  - Selectează automat mărimea modelului învăţat (prin vectorii suport)
    - În RNA straturile ascunse trebuie configurate de către utilizator *apriori*
  - Nu învaţă pe derost datele (overfitting)
    - RNA se confruntă cu problema overfitting-ului chiar şi cand modelul se învaţă prin validare încrucişată

- Dificultăţi
  - Doar atribute reale
  - Doar clasificare binară
  - Background matematic dificil

- Tool-uri
  - LibSVM → http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - Weka → SMO
  - SVMLight → http://svmlight.joachims.org/
  - SVMTorch → http://www.torch.ch/
  - http://www.support-vector-machines.org/

Sisteme inteligente – SIS – Învățare automată

- Programare genetică
  - Definire
  - Proiectare
  - Avantaje
  - Limite
  - Versiuni

Sisteme inteligente – SIS – PG

Reamintim

- Învăţare supervizată → problemă de regresie (Studiul legăturii între variabile)
  - Se dă un set de *n* date (exemple, instanţe, cazuri)
    - date de antrenament – sub forma unor perechi (atribute_data$_i$, ieşire$_i$), unde
      - $i = 1,n$ (n = nr datelor de antrenament)
      - ***atribute_data$_i$*** $= (atr_{i1}, atr_{i2}, \ldots, atr_{im})$, m – nr atributelor (caracteristicilor, proprietăţilor) unei date
      - *ieşire$_i$* – un număr real
    - date de test
      - sub forma (***atribute_data$_i$***), $i = n+1,N$ (N-n = nr datelor de test)
  - Să se determine
    - o funcţie (necunoscută) care realizează corespondenţa atribute – ieşire pe datele de antrenament
    - Ieşirea (valoarea) asociată unei date (noi) de test folosind funcţia învăţată pe datele de antrenament

- Cum găsim forma (expresia) funcţiei?
  - Algoritmi evolutivi → Programare genetică

Sisteme inteligente – SIS – PG

Reamintim

- Algoritmi evolutivi
  - Inspiraţi din natură (biologie)
  - Iterativi
  - Bazaţi pe
    - populaţii de potenţiale soluţii
    - căutare aleatoare ghidată de
      - Operaţii de selecţie naturală
      - Operaţii de încrucişare şi mutaţie
  - Care procesează în paralel mai multe soluţii
- Metafora evolutivă

Evoluţie naturală	Rezolvarea problemelor
Individ	Soluţie potenţială (candidat)
Populaţie	Mulţime de soluţii
Cromozom	Codarea (reprezentarea) unei soluţii
Genă	Parte a reprezentării
Fitness (măsură de adaptare)	Calitate
Încrucişare şi mutaţie	Operatori de căutare
Mediu	Spaţiul de căutare al problemei

Sisteme inteligente – SIS – PG

Reamintim

- Algoritmi evolutivi

```
Initializare populaţie P(0)
Evaluare P(0)
g := 0; //generaţia
CâtTimp (not condiţie_stop) execută
    Repetă
      Selectează 2 părinţi p1 şi p2 din P(g)
      Încrucişare(p1,p2) =>o1 şi o2
      Mutaţie(o1) => o1*
      Mutaţie(o2) => o2*
      Evaluare(o1*)
      Evaluare(o2*)
      adăugare o1* şi o* în P(g+1)
    Până când P(g+1) este completă
    g := g + 1
Sf CâtTimp
```

Sisteme inteligente – SIS – PG

- Definire
  - Propusă de john Koza în 1988
  - http://www.genetic-programming.org/
  - Un tip particular de algoritmi evolutivi
  - Cromozomi
    - sub formă de arbore care codează mici programe
  - Fitness-ul unui cromozom
    - Performanţa programului codat în el
  - Scopul PG
    - Evoluarea de programe de calculator
    - AG evoluează doar soluţii pentru probleme particulare

Sisteme inteligente – SIS – PG

- Proiectare
  - Reprezentarea cromozomilor
    - Foarte importantă, dar este o sarcină dificilă
    - Cromozomul = un arbore cu noduri de tip
      - Funcţie → operatori matematici (+,-,*,/,*sin*,*log*, *if*,...)
      - Terminal → atribute ale datelor problemei sau constante (*x,y,z,a,b,c,...*)
    - care codează expresia matematică a unui program (problema regresiei → a unei funcţii)

$x(y+2)$

$x+y^2+3$

Sisteme inteligente – SIS – PG

- Proiectare
  - Fitness
    - Eroarea de predicţie – diferenţa între ceea ce dorim să obţinem şi ceea ce obţinem de fapt
    - pp o problemă de regresie cu următoarele date de intrare (2 atribute şi o ieşire) şi 2 cromozomi:
      - $c_1 = 3x_1 - x_2 + 5 = (((3, x1, *), x2, -), 5, +)$
      - $c_2 = 3x_1 + 2x_2 + 2$ $f^*(x1,x2) = 3x_1 + 2x_2 + 1$ – necunoscută

x_1	x_2	$f^*(x_1,x_2)$	$f_1(x_1,x_2)$	$f_2(x_1,x_2)$	$\lvert f^*-f_1 \rvert$	$\lvert f^*-f_2 \rvert$
1	1	6	7	7	1	1
0	1	3	4	4	1	1
1	0	4	8	5	4	1
-1	1	0	1	1	1	1
					$\sum=7$	$\sum= 4$

→ c_2 e mai bun ca c_1

Sisteme inteligente – SIS – PG

- Proiectare
  - Fitness
    - Eroarea de predicţie – diferenţa între ceea ce dorim să obţinem şi ceea ce obţinem de fapt
    - pp o problemă de clasificare cu următoarele date de intrare (2 atribute şi o ieşire) şi 2 cromozomi:
      - $c_1 = 3x_1 - x_2 + 5$ *f1(1,1) = 7 -: sigm(7) – (0,1) -: Theta*
      - $c_2 = 3x_1 + 2x_2 + 2$

x_1	x_2	$f^*(x_1,x_2)$	$f_1(x_1,x_2)$	$f_2(x_1,x_2)$	$\lvert f^*-f_1 \rvert$	$\lvert f^*-f_2 \rvert$
1	1	Yes	Yes	Yes	0	0
0	1	No	Yes	No	1	0
1	0	Yes	No	No	1	1
-1	1	Yes	No	yes	1	0
					$\Sigma=3$	$\Sigma= 1$

➔ c_2 e mai bun ca c_1

Sisteme inteligente – SIS – PG

- Proiectare
  - Iniţializarea cromozomilor
    - Generare aleatoare de arbori corecţi → programe valide (expresii matematice valide)
    - Se stabileşte o adâncime maximă a arborilor D_{max}
    - 3 metode de iniţializare
      - *Full* → fiecare ramură a rădăcinii are adâncimea D_{max}
        - Nodurile aflate la o adâncime d < Dmax se iniţializează cu una dintre funcţiile din F
        - Nodurile aflate la o adâncime d = Dmax se iniţializează cu unul dintre terminalele din T
      - *Grow* → fiecare ramură a rîdăcinii are o adâncime < D_{max}
        - Nodurile aflate la o adâncime d < Dmax se iniţializează cu un element din $F \cup T$
        - Nodurile aflate la o adâncime d = Dmax se iniţializează cu unul dintre terminalele din T
      - *Ramped half and half* → ½ din populaţia de cromozomi se iniţializează folosind metoda *full*, ½ din populaţia de cromozomi se iniţializează folosind metoda *grow*

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Selecţia pentru recombinare
    - similar oricărui algoritm evolutiv
    - recomandare → selecţie proporţională
    - over-selection → pentru populaţii f mari
      - Se ordonează populaţia pe baza fitness-ului şi se împarte în 2 grupuri:
        - Grupul 1: cei mai buni x% cromozomi din populaţie
        - Grupul 2: restul de (100-x)% cromozomi din populaţie
        - Pentru populaţii cu 1000, 2000, 4000, 8000 de cromozomi, x este stabilit la 32%, 16%, 8%, respectiv 4%
      - 80% din operaţiile de selecţie vor alege cromozomi din grupul 1, 20% din grupul 2

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Selecţia de supravieţuire
    - Scheme
      - Generaţională
      - steady-state
    - Probleme
      - *Bloat* → supravieţuirea celui mai "gras" individ (dimensiunea cromozomilor creşte de-a lungul evoluţiei)
      - Soluţii
        - Interzicerea operatorilor de variaţie care produc descendenţi prea mari
        - Presiunea economiei (zgârceniei) – penalizarea cromozomilor prea mari

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Încrucişare şi mutaţie
    - Parametri
      - O probabilitate *p* de alegere între încrucişare şi mutaţie
        - p = 0 (cf. Koza) sau p = 0.05 (cf. Banzhaf)
      - O probabilitate p_c şi respectiv p_m de stabilire a nodului care urmează a fi supus modificării
    - Dimensiunea descendenţilor diferă de dimensiune părinţilor

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Încrucişare
    - Cu punct de tăietură – se interchimbă doi sub-arbori
    - Punctul de tăietură se generează aleator

$p_1=(x+y)*(z-sin(x))$

$p_2=xyz+x^2$

$f_1=(x+y)yz$

$f_2=(z-sin(x))x+x^2$

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Mutaţie
    - Mutaţie de tip *grow* → Înlocuirea unei frunze cu un nou sub-arbore

p=(x+y)(z-sin(x))*

f=(x+y)(z-sin(x+4))*

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Mutaţie
    - Mutaţie de tip *shrink* → Înlocuirea unui sub-arbore cu o frunză

p=(x+y)(z-sin(x))*

*f=(x+y)*4*

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Mutaţie
    - Mutaţie de tip *Koza* → Înlocuirea unui nod (intern sau frunză) cu un nou sub-arbore

p=(x+y)(z-sin(x))*

*f=(x+y)*sin(x+4)*

Sisteme inteligente – SIS – PG

- Proiectare
  - Operatori genetici → Mutaţie
    - Mutaţie de tip *switch*
      - selectarea unui nod intern şi re-ordonarea sub-arborilor săi
    - Mutaţie de tip cycle
      - selectarea unui nod şi înlocuirea lui cu unul nou de acelaşi tip (intern – cu o funcţie – sau frunză – cu un terminal)

Sisteme inteligente – SIS – PG

- Comparaţie AG şi PG
  - Forma cromozomilor
    - AG – cromozomi liniari
    - PG – cromozomi ne-liniari
  - Dimensiunea cromozomilor
    - AG – fixă
    - PG – variabilă (în adâncime sau lăţime)
  - Schema de creare a descendenţilor
    - AG – încrucişare şi mutaţie
    - PG – încrucişare sau mutaţie

Sisteme inteligente – SIS – PG

- Avantaje
  - PG găseşte soluţii problemelor care nu au o soluţie optimă
    - Un program pentru conducerea maşinii → nu există o singură soluţie
    - Unele soluţii implică un condus sigur, dar lent
    - Alte soluţii implică o viteză mare, dar un risc ridicat de accidente
    - Coducerea maşinii ←→ compromis între viteză mare şi siguranţă
  - PG este utilă în problemele a căror variabile se modifică frecvent
    - Conducerea maşinii pe autostradă
    - Conducerea maşinii pe un drum forestier
- Limite
  - Timpul mare necesar evoluţiei pentru identificarea soluţiei

Sisteme inteligente – SIS – PG

- Versiuni ale PG
  - PG liniară (Cramer, Nordin)
  - Gene Expression Programming (Ferreira)
  - Multi Expression Programing (Oltean)
  - Gramatical Evolution (Ryan, O'Neill)
  - Cartesian Genetic Programming (Miller)

Sisteme inteligente – SIS – PG

- PG liniară
  - Evoluarea de programe scrise într-un limaj imperativ (calculul fitness-ului nu necesită interpretare) → viteză mare de lucru
  - Reprezentare
    - Vector de instrucţiuni, fiecare instrucţiune fiind de forma (pp ca aritatea maximală a unei funcţii din F este n):
      - Index_op, registru_out, registru_in$_1$, registru_in$_2$,...,registru_in$_n$

- $v_i = v_j * v_k$ // instruction operating on two registers
- $v_i = v_j * c$ // instruction operating on one register and one constant
- $v_i = sin(v_j)$ // instruction operating on one register

```
void LGP_program (double v[11])
  {
      ...
      v[8] = v[0] - 10;
      v[6] = v[2] * v[0];
      v[5] = v[8] * 7;
      v[4] = v[2] - v[0];
      v[10] = v[1]/v[4];
      v[3] = sin(v[1]);
      v[1] = v[8] - v[6];
      v[7] = v[10] * v[3];
      v[9] = v[0] + v[7];
      v[2] = v[7] + 3;
      ...
  }
```

```
void LGP_effective_program (double v[11])
  {
      ...
      v[4] = v[2] - v[0];
      v[10] = v[1]/v[4];
      v[3] = sin(v[1]);
      v[7] = v[10] * v[3];
      v[9] = v[0] + v[7];
      ...
  }
```

Sisteme inteligente – SIS – PG

- PG liniară
  - Iniţializare
    - Aleatoare
    - Restricţii
      - Lungimea iniţială a cromozomului (nr de instrucţiuni)
  - Operatori genetici de variaţie
    - Încrucişare – cu 2 puncte de tăietură
    - Mutaţie
      - Micro mutaţie → schimbarea unui operand sau operator (nu se modifică dimensiunea cromozomului)
      - Macro mutaţie → inserarea sau eliminarea unei instrucţiuni (se modifică dimensiunea cromozomului)

Sisteme inteligente – SIS – PG

- PG liniară
  - Avantaje
    - Evoluare într-un limbaj de nivel redus (low-level)
  - Dezavantaje
    - Numărul de regiştri necesari (numărul de atribute ale problemei)
  - Resurse
    - Register Machine Learning Technologies http://www.aimlearning.com
    - *Peter Nordin's home page http://fy.chalmers.se/~pnordin*
    - *Wolfgang Banzhaf's home page http://www.cs.mun.ca/~banzhaf*
    - *Markus Brameier's home page http://www.daimi.au.dk/~brameier*

Sisteme inteligente – SIS – PG

- Gene Expression Programming (GEP)
  - Ideea de bază
    - Reprezentarea liniară a expresiilor codabile în arbori (prin parcuregerea în lăţime a acestora – breadth-first)

$$C = +a * /Sb - bcacabbc$$

  - Reprezentare
    - Un cromozom este format din mai multe gene
      - Legate între ele prin + sau *
    - Fiecare genă este formată din:
      - Cap
        - conţine h elemente → funcţii şi terminale
      - Coadă
        - conţine doar terminale, în număr de t = (n-1)*h+1, unde n – aritatea maximă a unei funcţii din F

Sisteme inteligente – SIS – PG

- GEP
  - Iniţializare
    - Aleatoare, cu elemente din F şi T conform regulilor precizate anterior
  - Operatori de variaţie
    - Încrucişare
      - La nivel de alelă
        - Cu un punct de tăietură
        - Cu două puncte de tăietură
      - Încrucişare la nivel de genă
        - Cromozomii schimbă între ei anumite gene (plasate pe aceeaşi poziţie)
    - Mutaţie
      - La nivel de alelă
        - se modifică un element din cap sau coadă, respectând regulile de la iniţializare
    - Transpoziţii

Sisteme inteligente – SIS – PG

- GEP
  - Avantaje
    - Codarea în cromozomi a unor programe corecte datorită separării unei gene în cap şi coadă
  - Dezavantaje
    - Cromozomii multi-genă
      - Câte gene?
      - Cum se leagă genele între ele?
  - Resurse
    - Gene Expression Programming website, http://www.gepsoft.com
    - *Heitor Lopes's home page* http://www.cpgei.cefetpr.br/~hslopes/index-english.html
    - *Xin Li's home page* *http://www.cs.uic.edu/~xli1*
    - *GEP in C#* http://www.c-sharpcorner.com/Code/2002/Nov/GEPAlgorithm.asp

Sisteme inteligente – SIS – PG

- Multi Expression Programming (MEP)
  - Ideea de bază
    - Cromozomul este format din mai multe gene, fiecare genă → cod cu 3 adrese
      - Similar PG liniară, dar mai rapid
  - Reprezentare
    - Liniară
    - O genă conţine o funcţie (unară sau binară) şi pointeri spre argumentele sale
    - Cromozomul codează mai multe potenţiale soluţii → fiecare soluţie corespunde unei gene
      - Calitatea unei soluţii (gene) = suma (peste datele de antrenament) între ceea ce trebuia obţinut şi ceea ce se obţine
      - Calitatea unui cromozom = fitness-ul celei mai bune gene

Sisteme inteligente – SIS – PG

- MEP
  - Iniţializare
    - Prima genă trebuie să fie un terminal
    - Restul genelor pot conţine
      - un terminal sau
      - o funcţie (unară sau binară) şi pointeri spre argumentele sale
        - Argumentele unei funcţii poziţionată în a i-a genă trebuie să fie poziţionate în cromozom la indici mai mici decât i
  - Operatori de variaţie
    - Încrucişare → schimbarea unor gene între părinţi
      - Cu un punct de tăietură
      - Cu două puncte de tăietură
      - Uniformă
    - Mutaţie → modificarea unei gene
      - Prima genă → generarea unui nou terminal
      - Restul genelor → generarea unui terminal sau a unei funcţii (simbolul funcţiei şi argumentele funcţiei)
        - Generarea are loc la fel ca la iniţializare

Sisteme inteligente – SIS – PG

- MEP
  - Avantaje
    - Ieşire dinamică corespunzătoare unui cromozom
      - Complexitatea programului (expresiei) căutat(e)
      - Programe (expresii) de lungime variabilă obţinute fără operatori speciali
      - Programe de lungime exponenţială codate în cromozomi de lungime polinomială
  - Dezavantaje
    - Complexitatea decodării pt date de antrenament necunoscute → evoluarea strategiilor de joc
  - Resurse
    - Mihai Oltean's home page http://www.cs.ubbcluj.ro/~
    - *Crina Gro»san's home page http://www.cs.ubbcluj.ro/~cgrosan*
    - MEP web page http://www.mep.cs.ubbcluj.ro
    - *MEP in C# http://www.c-sharpcorner.com*

Sisteme inteligente – SIS – PG

- Grammatical Evolution (GE)
  - Ideea de bază
    - Evoluarea de programe în forma Backus-Naur (program exprimat sub forma unei gramatici cu simboluri terminale şi non-terminale, simbol de start, reguli/producţii)
  - Reprezentare
    - String binar de codons (grupuri de 8 biţi) → care regulă a gramaticii tb aplicată
    - Exemplu
      - G={N,T,S,P}, N={+, -,*, /, sin, (,)}, T= {expr, op2, op1}, S=⟨expr⟩, iar P este:
        - ⟨expr⟩ ::=a|b|c| ⟨expr⟩ ⟨op2⟩ ⟨expr⟩|(⟨expr⟩ ⟨op2⟩ ⟨expr⟩)| ⟨op1⟩ ⟨expr⟩
        - ⟨op2⟩::=+|-|*|/,
        - ⟨op1⟩::=sin
      - C^*_{GE}=(9 12 12 3 15 7 11 4 2 5 0 6 11 0 1 7 12)
      - S= ⟨expr⟩ ➔ ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ➔ a⟨op2⟩ ⟨expr⟩ ➔a + ⟨expr⟩ ➔ a + ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ➔ a + ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ⟨op2⟩ ⟨expr⟩ ➔ a + b ⟨op2⟩ ⟨expr⟩ ⟨op2⟩ ⟨expr⟩

$$C_{GE} = (00001001\ \ 00001100\ \ 00001100\ \ 00000011\ \ 00001111\ \ 00000111\ \ 00001011\ \ 00000100\ \ 00000010\ \ 00000101\ \ 00000000\ \ 00000110\ \ 00001011\ \ 00000000\ \ 00000001\ \ 00000111\ \ 00001100)$$

$$\begin{aligned}
&a + b/\langle expr\rangle\langle op_2\rangle\langle expr\rangle\\
&a + b/(\langle expr\rangle\langle op_2\rangle\langle expr\rangle)\langle op_2\rangle\langle expr\rangle\\
&a + b/(c\langle op_2\rangle\langle expr\rangle)\langle op_2\rangle\langle expr\rangle\\
&a + b/(c - \langle expr\rangle)\langle op_2\rangle\langle expr\rangle\\
&a + b/(c - a)\langle op_2\rangle\langle expr\rangle\\
&a + b/(c - a) * \langle expr\rangle\\
&a + b/(c - a) * \langle op_1\rangle\langle expr\rangle\\
&a + b/(c - a) * \sin\langle expr\rangle
\end{aligned}$$

$$E = a + b/(c - a) * \sin(b)$$

Sisteme inteligente – SIS – PG

- GE
  - Iniţializare
    - Stringul binar este iniţializat aleator cu 0 şi 1 fără restricţii ➔ programe valide
    - Decodarea se termină când s-a obţinut un program complet
      - Dacă s-au terminat codons şi încă nu s-a format tot programul, se reiau codons de la primul element → wrapping
  - Operatori de variaţie
    - Încrucişare
      - Cu un punct de tăietură
    - Mutaţie
      - Schimbarea probabilistică a unui bit în opusul său
    - Duplicare
      - O secvenţă de gene este copiată la sfârşitul cromozomului
    - Pruning
      - Eliminarea genelor ne-folosite în procesul de transformare (decodare) a cromozomului

Sisteme inteligente – SIS – PG

- GE
  - Avantaje
    - Evoluarea de programe scrise în limbaje a căror instrucţiuni pot fi exprimate ca reguli de tip BNF
    - Reprezentarea poate fi schimbată prin modificarea gramaticii
  - Dezavantaje
    - Wrapping-ul la infinit → limitarea repetărilor şi penalizarea cromozomilor care depăşesc un anumit prag de repetări
  - Resurse
    - Grammatical Evolution web page, http://www.grammatical-evolution.org
    - *Conor Ryan's home page, http://www.csis.ul.ie/staff/conorryan*
    - *Michael O'Neill's home page, http://ncra.ucd.ie/members/oneillm.html*
    - *John James Collins's home page, http://www.csis.ul.ie/staff/jjcollins*
    - *Maarten Keijzer's home page, http://www.cs.vu.nl/~mkeijzer*
    - *Anthony Brabazon's home page http://ncra.ucd.ie/members/brabazont.*html

Sisteme inteligente – SIS – PG

- Cartesian Genetic Programming (CGP)
  - Ideea de bază
    - Cromozomi sub formă de graf (matrice) ➔ programe mai complexe decât cele din arbori
  - Reprezentare
    - În sistem cartezian (matrice de noduri)
    - Un nod are asociate
      - O funcţie
      - Intrări
      - Ieşiri
    - Ouputul cromozomului
      - Outputul oricărui nod

$$C = (1,2,3,\ \mathbf{2,0,1},\ 0,1,2,\ \mathbf{1,4,3},\ 4,3,0,\ \mathbf{1,5,4},\ 1,8,4,\ 0,3,1,\ \mathbf{6,8,2},\ 10,7,2,\ \mathbf{0,11,0},\ 4,6,3,\ 13)$$

Sisteme inteligente – SIS – PG

- CGP
  - Iniţializare
    - Aleatoare
    - Intrările oricărui nod trebuie să fie noduri de pe coloanele anterioare
      - Nodurile de pe prima coloană au ca intrări caracteristicile datelor de antrenament
  - Operatori de variaţie
    - Încrucişare
      - Nu se aplică
    - Mutaţie
      - Modificarea elementelor unui nod

Sisteme inteligente – SIS – PG

- CGP
  - Avantaje
    - Evoluarea indicelui nodului care furnizează ieşirea programului codat în cromozom
    - Programul evoluat poate avea una sau mai multe ieşiri
  - Dezavantaje
    - Stabilirea numărului de coloane influenţează rezultatele obţinute
  - Resurse
    - Julian. F. Miller's home page http://www.elec.york.ac.uk/intsys/users/jfm7
    - *Lukás Sekanina's home page http://www.fit.vutbr.cz/~sekanina/*

Sisteme inteligente – SIS – PG

- Aplicaţii
  - Probleme în care există o relaţie între intrări şi ieşiri
  - Probleme de regresie

  - Probleme de clasificare

Sisteme inteligente – SIS – PG

- Aplicaţii
  - Probleme de design
    - Evoluarea de circuite digitale
    - Evoluarea de antene
      - http://idesign.ucsc.edu/projects/evo_antenna.html
    - Evoluarea de programe (scrise într-un anumit limbaj)
    - Evoluarea de picturi şi muzică
      - http://www.cs.vu.nl/~gusz/

    - Altele
      - http://www.genetic-programming.com/humancompetitive.html

Sisteme inteligente – SIS - kNN

Cei mai apropiați k vecini (*k-nearest neighbours - kNN*)

- Unul dintre cei mai simpli algoritmi de clasificare
- În etapa de antrenament, algoritmul doar citeşte datele de intrare (atributele şi clasa fiecărei instanţe)
- În etapa de testare, pentru o nouă instanţă (fără clasă):
  - se caută (printre instanţele de antrenament) **cei mai apropiaţi *k* vecini**
    - distanţa Minkowski (Manhattan, Euclidiană) – atribute continue
    - distanţa Hamming, Levensthein – analiza textelor
    - alte distanţe (funcţii kernel)

  - se preia clasa majoritară a acestor *k* vecini

Sisteme inteligente – SIS - kNN

- Unul dintre cei mai simpli algoritmi de clasificare
- În etapa de antrenament, algoritmul doar citeşte datele de intrare (atributele şi clasa fiecărei instanţe)
- În etapa de testare, pentru o nouă instanţă (fără clasă)
  - se caută (printre instanţele de antrenament) cei mai apropiaţi *k* vecini şi
  - **se preia clasa majoritară a acestor *k* vecini**

Sisteme inteligente – SIS - kNN

- Tool-uri
  - Sklearn (python)
    - https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
  - Weka (java)
    - https://weka.sourceforge.io/doc.dev/weka/classifiers/lazy/IBk.html
- Biblio
  - https://github.com/rasbt/stat479-machine-learning-fs19/tree/master/02_knn

Sisteme inteligente – SIS – Arbori de decizie

- Scop
  - Divizarea unei colecţii de articole în seturi mai mici prin aplicarea succesivă a unor reguli de decizie → adresarea mai multor întrebări
    - Fiecare întrebare este formulată în funcţie de răspunsul primit la întrebarea precedentă
  - Elementele se caracterizează prin informaţii non-metrice
- Definire
  - Fiecare nod intern corespunde unui atribut
  - Fiecare ramură de sub un nod (atribut) corespunde unei valori a atributului
  - Fiecare frunză corespunde unei clase (ieşire de tip discret)

Sisteme inteligente – SIS – Arbori de decizie

- Exemplu

rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31…40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31…40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31…40	Medium	No	Excellent	Yes
r13	31…40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

Sisteme inteligente – SIS – Arbori de decizie

- Definire
  - Arborele de decizie
    - Un graf special → arbore orientat bicolor
    - Conţine noduri de 3 tipuri:
      - Noduri de decizie → posibilităţile decidentului (ex. Diversele examinări sau tratamente la care este supus pacientul) şi indică un test pe un atribut al articolului care trebuie clasificat
      - Noduri ale hazardului – evenimente aleatoare în afara controlului decidentului (rezultatul examinărilor, efectul terapiilor)
      - Noduri rezultat – situaţiile finale cărora li se asociază o utilitate (apreciată aprioric de către un pacient generic) sau o etichetă
    - Nodurile de decizie şi cele ale hazardului alternează pe nivelele arborelui
    - Nodurile rezultat – noduri terminale (frunze)
    - Muchiile arborelui (arce orientate) → consecinţele în timp (rezultate) ale decizilor, respectiv ale realizării evenimentelor aleatoare (pot fi însoţite de probabilităţi)
  - Fiecare nod intern corespunde unui atribut
  - Fiecare ramură de sub un nod (atribut) corespunde unei valori a atributului
  - Fiecare frunză corespunde unei clase

Sisteme inteligente – SIS – Arbori de decizie

- Tipuri de probleme
  - Exemplele (instanţele) sunt reprezentate printr-un număr fix de atribute, fiecare atribut putând avea un număr limitat de valori
  - Funcţia obiectiv ia valori de tip discret
  - AD reprezintă o disjuncţie de mai multe conjuncţii, fiecare conjuncţie fiind de forma atributul a_i are valoarea v_j
  - Datele de antrenament pot conţine erori
  - Datele de antrenament pot fi incomplete
    - Anumitor exemple le pot lipsi valorile pentru unele atribute
  - **Probleme de clasificare**
    - Binară
      - exemple date sub forma [($atribut_{ij}$, $valoare_{ij}$), $clasă_i$, i=1,2,...,n, j=1,2,...,m, $clasă_i$ putând lua doar 2 valori]
    - Multi-clasă
      - exemple date sub forma [($atribut_{ij}$, $valoare_{ij}$), $clasă_i$, i=1,2,...,n, j=1,2,...,m, $clasă_i$ putând lua k valori]
  - **Probleme de regresie**
    - AD se construiesc similar cazului problemei de clasificare, dar în locul etichetării fiecărui nod cu eticheta unei clase se asociază nodului o valoare reală sau o funcţie dependentă de intrările nodului respectiv
    - Spaţiul de intrare se împarte în regiuni de decizie prin tăieturi paralele cu axele Ox şi Oy
    - Are loc o transformare a ieşirilor discrete în funcţii continue
    - Calitatea rezolvării problemei
      - Eroare (pătratică sau absolută) de predicţie

Sisteme inteligente – SIS – Arbori de decizie

- Proces
  - Construirea (creşterea, inducţia) arborelui
    - Se bazează pe un set de date de antrenament
    - Lucrează de jos în sus sau de sus în jos (prin divizare – *splitting*)
  - Utilizarea arborelui ca model de rezolvare a problemelor
    - Ansamblul decizilor efectuate de-a lungul unui drum de la rădăcină la o frunză formează o regulă
    - Regulile formate în AD sunt folosite pentru etichetarea unor noi date
  - Tăierea (curăţirea) arborelui (pruning)
    - Se identifică şi se mută/elimină ramurile care reflectă zgomote sau excepţii

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD
  - Divizarea datelor de antrenament în subseturi pe baza caracteristicilor datelor
    - Un nod → întrebare legată de o anumită proprietate a unui obiect dat
    - Ramurile ce pleacă din nod → etichetate cu posibilele răspunsuri la întrebarea din nodul curent
    - La început toate exemplele sunt plasate în rădăcină
      - La pornire, un atribut va fi rădăcina arborelui, iar valorile atributului vor deveni ramuri ale rădăcinii
    - Pe următoarele nivele exemplele sunt partiţionate în funcţie de atribute ➔ ordinea considerării atributelor
      - Pentru fiecare nod se alege în mod recursiv câte un atribut (cu valorile lui pe ramurile descendente din nodul curent)
    - Divizarea → greedy în luarea decizilor
  - Proces iterativ
    - Reguli de oprire
      - toate exemplele aferente unui nod fac parte din aceeaşi clasă → nodul devine frunză şi este etichetat cu Ci
      - Nu mai sunt exemple → nodul devine frunză şi este etichetat cu clasa majoritară în setul de date de antrenament
      - nu mai pot fi considerate noi atribute

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD
  - Exemplu

rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31…40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31…40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31…40	Medium	No	Excellent	Yes
r13	31…40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD
  - Exemplu
    - Pentru rădăcină se alege atributul *age*

income	student	credit	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	student	credit	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

income	student	credit	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD
  - Exemplu
    - Pentru rădăcină se alege atributul *age*
    - Pe ramura <=30 se alege atributul student

- Proces → Construirea AD
  - Exemplu
    - Pentru rădăcină se alege atributul *age*
    - Pe ramura <=30 se alege atributul student
    - Pe ramura > 40 se alege atributul credit

in	st	cl
l	y	n
m	n	n

in	st	cl
m	n	y
l	y	y
m	y	y

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5
  - Greedy, recursiv, top-down, divide-and-conquer

```
generare(D, A){      //D – partiţionare a exemplelor de antrenament, A – lista de atribute
    Crearea unui nod nou N
    Dacă exemplele din D fac parte dintr-o singură clasă C atunci
        nodul N devine frunză şi este etichetat cu C
        returnează nodul N
    Altfel
        Dacă A=∅ atunci
            nodul N devine frunză şi este etichetat cu clasa majoritară în D
            returnează nodul N
        Altfel
            atribut_separare = Selectează_atribut(D, A)
            Etichetează nodul N cu atribut_separare
            Pentru fiecare valoare posibilă vj a lui atribut_separare
                Fie Dj mulţimea exemplelor din D pentru care atribut_separare = vj
                Dacă Dj =∅ atunci
                    Ataşează  nodului N o frunză etichetată cu clasa majoritară în D
                Altfel
                    Ataşează nodului N un nod returnat de generare(Dj, A –atribut_separare)
            Returnează nodul N
}
```

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5
  - Selectează_atribut(D, A) → Alegerea atributului aferent unui nod (rădăcină sau intern)
    - Aleatoare
    - Atributul cu cele mai puţine/multe valori
    - Pe baza unei ordini prestabilite a atributelor
      - Câştigul de informaţie
      - Rata câştigului
      - Indicele Gini
      - Distanţa între partiţiile create de un atribut

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut
  - Câştigul de informaţie
    - O măsură de impuritate
      - 0 (minimă) dacă toate exemplele fac parte din aceeaşi clasă
      - 1 (maximă) dacă avem număr egal de exemple din fiecare clasă
    - Se bazează pe entropia datelor
      - măsoară impuritatea datelor
      - numărul sperat (aşteptat) de biţi necesari pentru a coda clasa unui element oarecare din setul de date
      - clasificare binară (cu 2 clase): $E(S) = -p_+ log_2 p_+ - p_- log_2 p_-$ *unde*
        - p_+ - proporţia exemplelor pozitive în setul de date S
        - p_- - proporţia exemplelor negative în setul de date S
      - clasificare cu mai multe clase: $E(S) = \sum_{i=1, 2, \ldots, k} - p_i log_2 p_i$ – entropia datelor relativ la atributul ţintă (atributul de ieşire), unde
        - p_i – proporţia exemplelor din clasa i în setul de date S
    - câştigul de informaţie (*information gain*) al unei caracterisitici a (al unui atribut al) datelor
      - Reducerea entropiei setului de date ca urmare a eliminării atributului *a*
      - $Gain(S, a) = E(S) - \sum_{v \in valori(a)} |S_v| / |S| \, E(S_v)$
      - $\sum_{v \in valori(a)} |S_v| / |S| \, E(S_v)$ – *informaţia scontată*

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut
  - Câştigul de informaţie
    - exemplu

	a1	a2	a3	Clasa
d1	mare	roşu	cerc	clasa 1
d2	mic	roşu	pătrat	clasa 2
d3	mic	roşu	cerc	clasa 1
d4	mare	albastru	cerc	clasa 2

$S = \{d1, d2, d3, d4\} \rightarrow p_+ = 2 / 4,\ p_- = 2 / 4 \rightarrow E(S) = - p_+\log_2 p_+ - p_-\log_2 p_- = 1$

$S_{v=mare} = \{d1, d4\} \rightarrow p_+ = ½,\ p_- = ½ \rightarrow E(S_{v=mare}) = 1$

$S_{v=mic} = \{d2, d3\} \rightarrow p_+ = ½,\ p_- = ½ \rightarrow E(S_{v=mic}) = 1$

$S_{v=rosu} = \{d1, d2, d3\} \rightarrow p+ = 2/3,\ p- = 1/3 \rightarrow E(S_{v=rosu}) = 0.923$

$S_{v=albastru} = \{d4\} \rightarrow p+ = 0,\ p- = 1 \rightarrow E(S_{v=albastru}) = 0$

$S_{v=cerc} = \{d1, d3, d4\} \rightarrow p+ = 2/3,\ p- = 1/3 \rightarrow E(S_{v=cerc}) = 0.923$

$S_{v=patrat} = \{d2\} \rightarrow p+ = 0,\ p- = 1 \rightarrow E(S_{v=patrat}) = 0$

$Gain(S, a) = E(S) - \sum_{v \in valori(a)} |S_v| / |S|\ E(S_v)$

$Gain(S, a_1) = 1 – (|S_{v=mare}| / |S|\ E(S_{v=mare}) + |S_{v=mic}| / |S|\ E(S_{v=mic})) = 1 – (2/4 * 1 + 2/4 * 1) = 0$

$Gain(S, a_2) = 1 – (|S_{v=rosu}| / |S|\ E(S_{v=rosu}) + |S_{v=albastru}| / |S|\ E(S_{v=albastru})) = 1 – (3/4 * 0.923 + 1/4 * 0) = 0.307$

$Gain(S, a_3) = 1 – (|S_{v=cerc}| / |S|\ E(S_{v=cerc}) + |S_{v=patrat}| / |S|\ E(S_{v=patrat})) = 1 – (3/4 * 0.923 + 1/4 * 0) = 0.307$

Sisteme inteligente – SIS – Arbori de decizie

- Proces → Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut
  - Rata câştigului
    - Penalizează un atribut prin încorporarea unui termen – *split information* – sensibil la gradul de împrăştiere şi uniformitate în care atributul separă datele
      - *Split information* – entropia relativ la valorile posibile ale atributului *a*
      - *Sv* – proporţia exemplelor din setul de date *S* care au atributul *a* evaluat cu valoarea *v*
      - *splitInformation(S,a)=* $-\sum_{v=value(a)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$

Sisteme inteligente – SIS – Arbori de decizie

- Studiu de caz - Problema vampirilor

Items	Shadow	Complexion	Garlic	Accent	Vampire
i1	?	Pale	Yes	None	No
i2	Yes	Ruddy	Yes	None	No
i3	?	Ruddy	No	None	Yes
i4	No	Average	No	Heavy	Yes
i5	?	Average	No	Odd	Yes
i6	Yes	Pale	No	Heavy	No
i7	Yes	Average	No	Heavy	No
i8	?	Ruddy	Yes	Odd	No

Sisteme inteligente – SIS – Arbori de decizie

- Proces
  - Construirea arborelui
  - Utilizarea arborelui ca model de rezolvare a problemelor
    - Ideea de bază
      - Se extrag regulile formate în arborele anterior construit → Reguli extrase din arborele dat în exemplul anterior:
        - IF *age* = *"<=30"* AND *student* = *"no"* THEN *buys_computer* = *"no"*
        - IF *age* = *"<=30"* AND *student* = *"yes"* THEN *buys_computer* = *"yes"*
        - IF *age* = *"31…40"* THEN *buys_computer* = *"yes"*
        - IF *age* = *">40"* AND *credit_rating* = *"excellent"* THEN *buys_computer* = *"no"*
        - IF *age* = *">40"* AND *credit_rating* = *"fair"* THEN *buys_computer* = *"yes"*
      - Regulile sunt folosite pentru a clasifica datele de test (date noi). Fie *x* o dată pentru care nu se ştie clasa de apartenenţă → Regulile se pot scrie sub forma unor predicate astfel:
        - IF *age(x, <=30)* AND *student(x, no)* THEN *buys_computer (x, no)*
        - IF *age(x, <=30)* AND *student (x, yes)* THEN *buys_computer (x, yes)*

Sisteme inteligente – SIS – Arbori de decizie

- Proces
  - Construirea arborelui
  - Utilizarea arborelui ca model de rezolvare a problemelor
    - Dificultăţi
      - *Underfitting* (sub-potrivire) → AD indus pe baza datelor de antrenament este prea simplu ➔ eroare de clasificare mare atât în etapa de antrenare, cât şi în cea de testare
      - *Overfitting* (supra-potrivire, învăţare pe derost) → AD indus pe baza datelor de antrenament se potriveşte prea accentuat cu datele de antrenament, nefiind capabil să generalizeze pentru date noi
      - Soluţii:
        - fasonarea arborelui (pruning) → Îndepărtarea ramurilor nesemnificative, redundante ➔ arbore mai puţin stufos
        - validare cu încrucişare

Sisteme inteligente – SIS – Arbori de decizie

- Proces
  - Construirea arborelui
  - Utilizarea arborelui ca model de rezolvare a problemelor
  - Tăierea (fasonarea) arborelui
    - Necesitate
      - Odată construit AD, se pot extrage reguli (de clasificare) din AD pentru a putea reprezenta cunoştinţele sub forma regulilor *if-then* atât de uşor de înţeles de către oameni
      - O regulă este creată (extrasă) prin parcurgerea AD de la rădăcină până la o frunză
      - Fiecare pereche (atribut,valoare), adică (nod, muchie), formează o conjuncţie în ipoteza regulii (partea dacă), mai puţin ultimul nod din drumul parcurs care este o frunză şi reprezintă consecinţa (ieşirea, partea atunci) regulii
    - Tipologie
      - Prealabilă (*pre-pruning*)
        - Se opreşte creşterea arborelui în timpul inducţiei prin sistarea divizării unor noduri care devin astfel frunze etichetate cu clasa majoritară a exemplelor aferente nodului respectiv
      - Ulterioară (*post-pruning*)
        - După ce AD a fost creat (a crescut) se elimină ramurile unor noduri care devin astfel frunze ➔ se reduce eroarea de clasificare (pe datele de test)

Sisteme inteligente – SIS – Arbori de decizie

- Tool-uri
  - http://webdocs.cs.ualberta.ca/~aixplore/learning/DecisionTrees/Applet/DecisionTreeApplet.html
  - WEKA → J48
  - http://id3alg.altervista.org/
  - http://www.rulequest.com/Personal/c4.5r8.tar.gz

- Biblio
  - http://www.public.asu.edu/~kirkwood/DAStuff/decisiontrees/index.html

Sisteme inteligente – SIS – Arbori de decizie

- Avantaje
  - Uşor de înţeles şi interpretat
  - Permit utilizarea datelor nominale şi categoriale
  - Logica deciziei poate fi urmărită uşor, regulile fiind vizibile
  - Lucrează bine cu seturi mari de date

- Dezavantaje
  - Instabilitate → modificarea datelor de antrenament
  - Complexitate → reprezentare
  - Greu de manevrat
  - Costuri mari pt inducerea AD
  - Inducerea AD necesită multă informaţie

Sisteme inteligente – SIS – Arbori de decizie

- Dificultăţi
  - Existenţa mai multor arbori
    - Cât mai mici
    - Cu o acurateţe cât mai mare (uşor de "citit" şi cu performanţe bune)
    - Găsirea celui mai bun arbore → problemă NP-dificilă
  - Alegerea celui mai bun arbore
    - Algoritmi euristici
    - ID3 → cel mai mic arbore acceptabil
      - → teorema lui Occam: "always choose the simplest explanation"
  - Atribute continue
    - Separarea în intervale
      - Câte intervale?
      - Cât de mari sunt intervalele?
  - Arbori prea adânci sau prea stufoşi
    - Fasonarea prealabilă (pre-pruning) → oprirea construirii arborelui mai devreme
    - Fasonarea ulterioară (post-pruning) → înlăturarea anumitor ramuri

Învățare supervizată – algoritmi
Arbori de decizie

- Tool-uri
  - http://webdocs.cs.ualberta.ca/~aixplore/learning/DecisionTrees/Applet/DecisionTreeApplet.html
  - WEKA → J48
  - http://id3alg.altervista.org/
  - http://www.rulequest.com/Personal/c4.5r8.tar.gz
  - https://scikit-learn.org/stable/modules/tree.html

- Biblio
  - http://www.public.asu.edu/~kirkwood/DAStuff/decisiontrees/index.html
  - https://github.com/rasbt/stat479-machine-learning-fs19/tree/master/06_trees

Recapitulare

- Sisteme care învaţă singure (SIS)
  - Algoritmi de programare genetică (PG)
    - Algoritmi evolutivi cu cromozomi sub formă de arbore
    - Cromozomii
      - Arborescenţi
      - Matriciali
      - Liniari
    - codează potenţiale soluţii de tipul
      - Expresiilor matematice → probleme de regresie/clasificare
      - Expressilor de tip Boolean → probleme de tip EvenParity / proiectare de circuite digitale
      - Programelor → evoluarea de cod sursă pentru rezolvarea unor probleme

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- **Sisteme care învaţă singure**
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- capitolul 15 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

ÎNVĂȚAREA NESUPERVIZATĂ

LIMBOI SERGIU

AGENDA

- Învățarea automată-recapitulare
- Învățarea nesupervizată-generalități
- Gruparea datelor (Clustering)
  - Generalități
  - Concepte de bază
  - Măsuri de similaritate
  - Algoritmi
  - Evaluarea rezultatelor
- Aplicații ale clustering-ului
- Concluzii

Învățarea automată

- învățarea poate fi explicată ca fiind activitatea de a obține cunoștințe sau de a le înțelege, precum și abilitatea de a-ți însuși noțiuni prin studiu, instruire sau experiență [1].

- Învățarea automată (*Machine Learning*) [1] se referă la modificări din sisteme care realizează sarcini asociate cu diverse teme, concepte din Inteligența Artificială, sarcini precum diagnoză, predicție, planificare, control sau detectare.

ÎNVĂȚAREA AUTOMATĂ

- Scop -> proiectarea și dezvoltarea unor algoritmi și metode utilizate pentru ca un sistem computațional să „învețe”.
- De ce să învețe sistemele informatice?
  - unele sarcini se pot defini doar prin exemple; de aceea trebuie să putem furniza perechi de intrări-ieșiri, în absența unei legături concrete între datele de intrare și rezultate;
  - este posibil ca anumite informații ascunse, nedescifrate, să reflecte corelații sau legături;
  - cantitatea mare de informații poate fi dificil de codificat de către mintea umană.

Învățarea automată

- Pentru a defini conceptele utilizate în problematica învățării se folosește o functie *f*, iar sarcina celui care învață este de a "ghici" această funcție.
- Tipuri de învățare
  - **Supervizată**- știm (uneori doar aproximativ) valorile funcției *f* pentru *m* cazuri ale unui set de antrenare; sistemul poate intui, după etapa de antrenare, care ar fi funcția și pentru un alt set de date
  - **Nesupervizată-** avem doar o mulțime de vectori (un set de date) pentru care nu știm funcția. Presupune divizarea (partiționarea) setului de date în submulțimi sau grupuri. În acest caz valoarea funcției va fi numele subgrupului (clasei) din care vectorul de intrare face parte
  - **Prin întărire (reinforcement)-** sistemul interacționează cu mediul și poate primi recompense sau penalizări

ÎNVĂȚAREA NESUPERVIZATĂ GENERALITĂȚI

- Ideea de bază- sistemul primește informații (secvențe de forma $x_1, x_{2,\ldots}$), dar nu i se furnizează rezultate prestabilite, obținute anterior și nici nu primește răsplată din partea mediului

- Stilul nesupervizat este mai comun creierului decât cel supervizat.

- Oamenii și animalele învață să-și analizeze mediul și să identifice obiectele și evenimentele din jurul lor.

Învățarea nesupervizată generalități

- Putem obține grupuri de fructe pe baza anumitor caracteristici?

Învățarea nesupervizată-exemple [2]

- Dar dacă avem fructe și legume?

Învățarea nesupervizată-exemple [2]

Învățarea nesupervizată-exemple [2]

Învățarea nesupervizată-exemple [2]

Învățarea nesupervizată-exemple

Învățarea nesupervizată generalități

- “Ce dorim să învețe sistemul, dacă nu îi livrăm informație externă?”
  - descoperirea unor grupuri (clase) din setul de date inițial;
  - extragerea unor atribute ce caracterizează datele de intrare într-un mod mai compact;
  - identificarea unor coincidențe (similitudini) naturale în cadrul datelor primite.

Învățarea nesupervizată generalități

- Utilitate în clasificarea de obiecte neetichetate.
- Plecând de la o mulțime de instanțe, se dorește găsirea unor mecanisme de a grupa obiectele pe baza unor similitudini, astfel încât obiectele din același grup să aibă o similitudine maximă.

Învățarea nesupervizată generalități

- Abordări
  - gruparea datelor (*clustering*);
  - rețele cu auto-organizare (*self-organizing maps*);
  - reguli de asociere;
  - algoritmi de maximizare a așteptărilor;
  - analiza componentelor independente;
  - analiza componentelor principale;
  - descompunerea în valori singulare.
  - metoda momentelor

GRUPAREA DATELOR CLUSTERING

- *Clustering-ul* (gruparea datelor) [3] este procesul de grupare a obiectelor (instanțe, observații) pe baza caracteristicilor lor.
- Scopul acestei activități este ca în cadrul aceluiași grup, obiectele să fie similare unele cu altele și diferite de obiectele din alte grupuri.
- Omogenitatea (similaritatea) din cadrul unui grup, denumit și *cluster*, precum și diferența cât mai mare între grupuri caracterizează acest proces.

GRUPAREA DATELOR CLUSTERING

- În clasificarea supervizată se dă o colecție de date etichetate (preclasificate), problema fiind de a putea eticheta un nou set de date.

- În cazul *clustering-ului*, problema este de a grupa o colecție de date neetichetate în grupuri cu o anumită semnificație.

- Într-o anumită măsură, etichetele sunt asociate grupurilor, dar aceste etichete sunt date pe seama informațiilor inițiale (nu se oferă etichete din sursă externă), așa numitele etichete *data driven*.

GRUPAREA DATELOR CLUTERING

- Componentele (pașii) procesului de grupare a datelor [4] :
  - stabilirea datelor de intrare (opțional selectarea atributelor–*feature selection*);
  - definirea unei măsuri de proximitate (vecinătate) adecvată domeniul datelor;
  - gruparea propriu-zisă (*clustering*);
  - abstractizare (dacă este cazul);
  - evaluarea rezultatelor.

Clustering concepte

- Stabilirea datelor se referă la identificarea numărului de clase (*clusteri*), numărului și tipurilor atributelor din cadrul setului de date.
- În cadrul acestei etape se poate aplica și selectarea atributelor (*feature selection*), proces de identificare a submulțimii, din mulțimea de atribute care va fi efectiv folosită pentru grupare.
- Tipuri de date
  - După nr de atribute- binare, discrete, continue
  - După tipul valorilor- calitative și cantivative

CLUSTERING CONCEPTE

- O măsură de proximitate sau de similaritate reprezintă, de fapt, o funcție distanță, pentru a determina cât de similari/diferiți sunt doi clusteri
- Abstractizarea este procesul de extragere a unei reprezentări simple și compacte a setului de date. Se dorește simplicitate (să fie ușor de înțeles), reliefată printr-o descriere concisă a fiecărui grup de obiecte, cu ajutorul unor termeni corespunzători acestui domeniu de *clustering* (ex: medoizi, centroizi).

CLUSTERING CONCEPTE

- Evaluarea rezultatelor
  - Se poate folosi o examinare externă care presupune compararea structurii obținute cu cea *a priori* sau o examinare internă care evaluează structura intrinsecă potrivită pentru setul de date inițial.

CE ESTE UN CLUSTER?

How many clusters?

Six Clusters

Two Clusters

Four Clusters

CE ESTE UN CLUSTER?

- Cormack (1971) și Gordon (1999) [5] au stabilit că un *cluster* poate fi definit în funcție de două proprietăți interne: omogenitate și izolare externă (separare).
- Vipin Kumar [3] prezintă câteva definiții ale unui *cluster*:
  - definiție bazată pe delimitări/separări: *cluster* = o mulțime de puncte în care orice punct este mai apropiat de un altul din mulțime, decât de un punct din afara ei

CE ESTE UN CLUSTER?

- definiție bazată pe centru: *cluster* = o mulțime de obiecte în care un obiect este mai apropiat de centrul mulțimii decât de centrul unei alte mulțimi

- definiție bazată pe similaritate: *cluster*= mulțime de obiecte care sunt "similare", dar diferite de obiectele din alte grupuri

MĂSURI DE SIMILARITATE

- O funcție de similaritate "măsoară" cât de bun este un anumit grup. În general termenul utilizat pentru astfel de măsuri este proximitate sau vecinătate. Două instanțe sunt "apropiate", atunci când disimilaritatea (distanța) este mică sau similaritatea este mare.

- Distanța-minimă, similaritatea-maximă
- Exemple:
- Distanța euclideană d(i,j)=$\sqrt{\sum_{k=1}^{n} |x_{ik} - x_{jk}|^2}$
- Distanța *Manhattan*: d(i,j)=$\sum_{k=1}^{n} |x_{ik} - x_{jk}|$.

MĂSURI DE SIMILARITATE

- Similaritatea cosinus
  - cos(A,B)=(A·B) / ||A|| ||B||,
    - unde · este produsul vectorial, iar ||A|| lungimea vectorului A.

- Similaritate Jaccard – unde A, B mulțimi
  - între 0 și 1

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

TEHNICI DE CLUSTERING

- metode ierarhice
  - algoritmi aglomerativi;
  - algoritmi divizivi.
- metode partiționale
  - *k-medoids*;
  - *k-means*;
  - algoritmi bazați pe densitate.
- metode bazate pe grilă /rețea
- algoritmi bazați pe scalabilitate
- algoritmi pentru date de dimensiuni mari
  - *clustering* de subspațiu;
  - tehnici de proiecție;

TEHNICI DE CLUSTERING

- Jain [4] consideră că metodele de grupare a datelor pot fi organizate în funcție de anumite criterii:
  - **aglomerativ** vs. **diviziv**- abordare ce se referă la modul de formare a grupurilor;
  - ***hard*** (strictă) vs ***fuzzy*** (maleabilă/permisivă) –se referă la modul de atribuire a instanțelor la un anumit grup.
    - Atribuirea *hard* indică faptul că un obiect poate aparține unui singur grup, de-a lungul procesului. Atribuirea *fuzzy* (*soft*) definește grade de apartenență la diferite grupuri, pentru fiecare instanță;

CLUSTERING IERARHIC

- În clasificarea ierarhică datele nu sunt partiționate într-un anumit număr de grupuri dintr-un singur pas.

- Tehnicile ierarhice se împart în metode aglomerative și divizive.

- Cele aglomerative procesează o serie de fuziuni succesive, a n instanțe, în grupuri. Cele divizive separă, succesiv, n indivizi în grupuri fine, mai precise.

CLUSTERING IERARHIC [5]

CLUSTERING IERARHIC

- Aglomerativ: Se pornește cu obiecte individuale. Se unifică, gradual, obiecte care au similaritate maximă (distanță minimă). Se continuă până când toate obiectele vor fi conținute într-un singur grup sau până când se ajunge la număr dorit de *clusteri*.
- Diviziv: Se pornește cu un grup ce conține toate obiectele. Se separă, gradual, grupul în două, atribuind obiectele celor doi noi *clusteri* astfel încât să se maximizeze similaritatea din cadrul fiecărui grup. Se continuă divizarea până când se obțin *clusteri* ce conțin un singur obiect sau până când se obține numărul dorit de *clusteri*.

CLUSTERINGUL PARTIȚIONAL

- Un algoritm de *clustering* partițional obține o partiție a setului de date, comparativ cu *clustering-ul* ierarhic care produce o structură
- O problemă a algoritmilor partiționali este alegerea numărului de *clusteri* doriți a fi obținuți în urma procesului.
- Metodele partiționale au ca scop construirea sau găsirea unei partiții formate din k *clusteri*, plecând de la un set de date cu n instanțe.
- Aceste tehnici cuprind și câteva metode euristice: *k-means* și *k-medoids* [5]. Algoritmul *k-means* presupune că fiecare grup este reprezentat de centrul lui, iar algoritmul *k-medoids* implică faptul că fiecare grup este reprezentat de unul dintre obiectele lui.

K-MEANS

- Denumirea provine de la reprezentarea a k *clusteri* pe baza mediei obiectelor din cadrul lor.
- Un astfel de obiect, definit ca fiind media instanțelor unui *cluster*, poartă denumirea de centroid [5]

K-means

- Algoritmul
  - selectează k instanțe ca fiind centroizii inițiali;
  - atribuie obiectele la cel mai apropiat centroid;
  - recalculează centroidul fiecărui grup;
  - revenire la pasul 2, oprirea realizându-se atunci când nu mai au loc modificări.

K-MEANS

- Alegerea centroizilor inițiali
  - etapa de bază a procedurii *k-means*. Este foarte ușor să alegi aleator k obiecte care să fie centroizi, dar rezultatele, de regulă, nu sunt mulțumitoare.
  - Este posibilă și abordarea ce implică mai multe rulări, fiecare cu un număr diferit de obiecte alese aleator ca reprezentând centroizi

K-MEANS

- Alegerea naturală a centroizilor [5]

- centroizii inițiali sunt aleși, adesea, din regiuni dense, în așa fel încât obiectele să fie foarte bine separate, pentru ca doi centroizi să nu fie aleși din cadrul aceluiași grup.

K-MEANS EXEMPLU

- Se dau punctele P1(2,3), P2(3,1), P3(4,2), P4(11,5), P5(12,4), P6(12,6), P7(7,5), P8(8,4), P9(8,6)
- Aplicați k-means pornind de la centroizii K1=P2 și K2=P8
- Se folosește distanța euclideană

REZOLVARE

- Calculăm distanța dintre fiecare obiect neselectat și centroizi
- d(P1,K1)=2,23 și d(P1,K2)=6,08 -> P1 se atribuie lui K1
- d(P3,K1)=1,41 d(P3,K2)=4,47 deci P3 în K1
- d(P4,K1)=8,94 d(P4,K2)=3,16 , P4 în K2
- .....

REZOLVARE

- Calculăm noii centroizi
  - K1=(3,2) K1x=9/3 K1y=6/3
  - K2=....

K-MEDOIDS

1. se selectează k obiecte reprezentative numite medoizi
2. se înlocuiește unul dintre obiectele selectate (medoizi) cu unul dintre obiectele neselectate. Se calculează distanța dintre fiecare obiect neselectat și cel mai apropiat medoid candidat, iar apoi distanța este însumată pentru toate punctele/ instanțele. Această distanță reprezintă costul configurației curente
3. se selectează configurația cu costul cel mai mic. Dacă există o nouă configurație, se repetă pasul 2;
4. dacă nu există o configurație nouă, se atribuie fiecare obiect neselectat celui mai apropiat medoid și algoritmul se oprește.

Evaluarea rezultatelor

- De ce să evaluăm procesul de *clustering*?
  - pentru a compara diferiți algoritmi de *clustering*;
  - pentru a compara două grupuri;
  - pentru a compara mulțimi de *clusteri*.

- Aspecte legate de validarea *clustering-ului*:
  - determinarea numărului ideal de *clusteri*;
  - găsirea unor legături între structurile identificate în urma procesului, din cadrul setului de date, și informații externe legate de datele de intrare;
  - evaluează cât de bine se potrivesc rezultatele produse de analiza de *clustering* cu setul de date, fără a referi informații externe (utilizează doar datele).

Evaluarea rezultatelor

- Există două abordări referitoare la activitatea de evaluare a *clustering-ului*:
  - externă-bazată pe informații anterioare despre date (posibil să cunoaștem etichetele datelor);
  - internă-bazată doar pe informații intrinseci (doar setul de date), fără să utilizăm surse externe.

EVALUAREA INTERNA

- Indexul Dunn [6] -acest index determină raportul dintre cea mai mică distanță *intercluster* și cea mai mare distanță *intracluster*, în cadrul partiției.

$$D = \frac{\min_{1 \le i < j \le n} d(i,j)}{\max_{1 \le k \le n} d'(k)},$$

- unde d(i,j) reprezintă distanța *intercluster* (distanța între *clusterul* i și *clusterul* j), iar d'(k) reprezintă *distanța intracluster* (distanța în clusterul k).
- Ca distanță între doi *clusteri* se poate folosi orice distanță, de exemplu distanța între centroizii grupurilor. Distanța *intracluster* (d') poate fi cea mai mare distanță între orice pereche de elemente din *clusterul* k. Valorile mari pentru indexul Dunn, indică soluții bune.

EVALUAREA INTERNA

- <u>Coeficientul Silhouette [6]</u>

$$s(i) = \frac{\left(b(i) - a(i)\right)}{Max\{a(i), b(i)\}}$$

- unde a(i) este distanța medie între obiectul i și celelalte obiecte din X_j, iar b(i) reprezintă cea mai mică distanță medie între obiectul i și obiectele din ceilalți *clusteri*, din care i nu face parte.

- Valorile pentru s(i) sunt din [-1,1], soluții bune când indexul este cât mai aproape de 1

EVALUAREA INTERNA

- <u>Indexul Davies-Bouldin [6]</u>

$$BD = \frac{1}{c}\sum_{i=1}^{c} Max_{i \neq j} \left\{ \frac{d(X_i) + d(X_j)}{d(c_i, c_j)} \right\}$$

- unde c reprezintă numărul de *clusteri*, $d(X_i)$ și $d(X_j)$ sunt distanțele între toate obiectele din *clusterul* X_i, respectiv X_j, și centroizii acelor *clusteri*. $D(c_i,c_j)$ este distanța între centroizii celor doi *clusteri*, c_i și c_j. Cu cât acest index produce valori mai mici, cu atât se obțin soluții mai bune.

EVALUARE EXTERNĂ

- Acuratețea
- Precizia
- Specificitatea
- Senzitivitatea (recall)

APLICAȚII ALE CLUSTERINGULUI

- Marketing și comerț
  - segmentare=modalitatea de organizare a clienților în grupuri, în funcție de preferințe pentru anumite produse, trăsături sau așteptări

APLICAȚII ALE CLUSTERINGULUI

- Geico [7] - planificări pentru personalizarea ofertelor de asigurări auto și voia să înțeleagă exact ce dorințe și așteptări au clienții de la furnizorul de asigurări
- Respondenții implicați în chestionare au fost grupați în clustere/segmente
- acțiunile luate de clienții din același segment ilustrează răspunsuri similare, dar diferite de cele date de clienții din restul segmentelor. Așadar, aspectele privind asigurarea auto pe care un client le consideră importante vor fi importante și pentru restul clienților din același *cluster*/segment

APLICAȚII ALE CLUSTERINGULUI

- Motoare de căutare [9]

APLICAȚII ALE CLUSTERINGULUI

- Analiza crimelor [8]
- Pentru aplicarea *clustering-ului* în analiza crimelor s-au utilizat seturi de date înregistrare de poliția din Anglia și Țara Galilor în perioada 1990-2011. Seturile de date se refereau la omucideri (uciderea unei alte persoane).

APLICAȚII ALE CLUSTERINGULUI

- *k-means* a avut drept obiectiv urmărirea ratei criminalității
- s-au luat în considerare omuciderile și s-au format *clusteri* ce ofereau informații legate de numărul de omucideri din diferiți ani.
- *Clusterul 0* cuprindea între 0 și 15 crime, *clusterul* 1 între 0 și 60, *clusterul* 2 între 0 și 600, etc.
- De pe graficele corespunzătoare fiecărui *cluster* se observa anul cu număr de omucideri minim și anul cu număr maxim. La finalul analizei s-a observat că omuciderile au scăzut din 1990 până în 2011.
- tendințele legate de crime, infracționalități pentru următorii ani și se pot proiecta tehnici de prevenție și de reducere a acestor acțiuni.

APLICAȚII ALE CLUSTERINGULUI

- K-Means în predicția performanței studenților [10]
  - monitorizarea performanțelor studenților
  - rezultatele studenților de la un institut privat din Nigeria
- Economie
- Finanțe – identificarea unor categorii de clienți
- Bio-arheologie

CONCLUZII

- *Clustering-ul* reprezintă o ramură importantă a învățării nesupervizate, aplicarea lui având rezultate semnificative în multe domenii

- *clustering-ul* simplifică mult o muncă manuală care, uneori, se dovedește a fi anevoioasă.

Bibliografie

- [1] Nils J.Nilsson, *Introduction to Machine Learning*, Stanford University, Stanford, 1996
- [2] Gabor Veress, Clustering training, 2013, https://www.slideshare.net/gveress/cluster-training-2013
- [3] Vipin Kumar, *An introduction to cluster analysis for data mining*, course, 2000
- [4] A. K. Jain, M. N. Murty and P. J. Flynn, *Data clustering: A review*., ACM Comput.Surv., 31(3):264-323, 1999
- [5] Brian S.Everitt, Sabine Landau, Morven Leese, Daniel Stahl, *Cluster Analysis*, 5th edition, Wiley, Londra, 2011

BIBLIOGRAFIE

- [6] E.Rendón, I.Abundez, A.Arizmendi and E. M. Quiroz, *Internal versus External cluster validation indexes*, International Journal of Computers and Communications, Vol.5: 27-34, 2011
- [7] R. Venkatesan, *Cluster analysis for segmentation,* Darden Business Publishing,University of Virginia, 2007
- [8] Jyoti Agarwal, Renuka Nagpal, Rajni Sehgal, *Crime Analysis using K-Means Clustering*, International Journal of Computer Applications, 83(4):1-4, 2013

BIBLIOGRAFIE

- [9] L.V. Bijuraj, *Clustering and its Applications*, National Conference on New Horizons in IT, 169-172, 2013
- [10] O.J. Oyelade, O.O. Oladipupo, I.C. Obagbuwa, *Application of k-means clustering algorithm for prediction of students' academic performance*, International Journal of Computer Science and Information Security,1:292-295, 2010

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Sisteme inteligente

Sisteme bazate pe reguli în medii certe

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme hibride
- Sisteme bazate pe reguli în medii certe
- Sisteme bazate pe reguli în medii incerte (Bayes, factori de certitudine, Fuzzy)

Materiale de citit şi legături utile

- capitolul III din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 4 şi 5 din *H.F. Pop, G. Şerban, Inteligenţă artificială, Cluj Napoca, 2004*
- capitolul 2 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 6 şi 7 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

Conținut

- Sisteme inteligente
  - Sisteme bazate pe cunoştinţe
    - Sisteme bazate pe logică
    - Sisteme bazate pe reguli în medii certe

Sisteme inteligente

Sisteme bazate pe cunoştinţe (SBC) Inteligenţă computaţională

Sisteme inteligente – sisteme bazate pe cunoştinţe (SBC)

- sistemele computaţionale – alcătuite din 2 module principale (roluri):
  - Domeniul de cunoştinţe (baza de cunoştinţe – BC – *knowledge base*)
    - Informaţiile specifice despre un domeniu
  - Modulul de control (MC – *inference engine*)
    - Regulile prin care se pot obţine informaţii noi
    - Algoritmi independenţi de domeniu

Sisteme inteligente – SBC

Baza de cunoştinţe (BC)

- Conţinut
- Tipologie
- Modalităţi de reprezentare a cunoştinţelor
- Modalităţi de stocare a cunoştinţelor

Sisteme inteligente – SBC

Baza de cunoştinţe (BC)

- Conţinut
  - Informaţii (exprimate într-o anumită reprezentare – ex. propoziţii) despre mediu
  - informaţii necesare pentru înţelegerea, formularea şi rezolvarea problemelor
  - mulţime de propoziţii (exprimate/reprezentate într-un limbaj formal) care descriu mediul
    - reprezentare uşor interpretabilă de către calculator ➔ limbaj de reprezentare a cunoştinţelor
    - mecanismul de obţinere a unor propoziţii noi pe baza celor vechi ➔ inferenţă/raţionare

- Tipologie
  - cunoştinţe exacte (perfecte)
  - cunoştinţe imperfecte (nesigure, incerte)
    - Inexacte
    - Incomplete
    - Incomensurabile

- Modalităţi de reprezentare a cunoştinţelor
  - Logica formală (limbaje formale)
  - Reguli
  - Reţele semantice

Sisteme inteligente – SBC

Baza de cunoştinţe (BC)

- Modalităţi de reprezentare a cunoştinţelor
  - Logica formală (limbaje formale)
    - Definiţie
      - Ştiinţa principiilor formale de raţionament
    - Componente
      - Sintaxă – simbolurile atomice folosite de către limbaj şi regulile de construcţie a expresiilor (structurilor/propoziţiilor) limbajului
      - Semantică – asociază un înţeles simbolurilor şi o valoare de adevăr (Adevărat sau Fals) regulilor (propoziţiilor) limbajului
      - Metodă de inferenţă sintactică – regulile necesare determinării unei submulţimi de expresii logice → teoreme (folosite pentru obţinerea de noi expresii)
    - Tipologie
      - În funcţie de numărul valorilor de adevăr:
        - logică duală
        - logică polivalentă
      - În funcţie de tipul elementelor de bază:
        - clasică → primitivele = propoziţii (predicate)
        - probabilistică → primitivele = variabile aleatoare
      - În funcţie de obiectul de lucru:
        - logica propoziţională → se lucrează doar cu propoziţii declarative, iar obiectele descrise sunt fixe sau unice (Ionică este student)
        - logica predicatelor de ordin I → se lucrează cu propziţii declarative, cu predicate şi cuantificări , iar obiectele descrise pot fi unice sau variabile asociate unui obiect unic (Toţi studenţii sunt prezenţi)
  - Reguli
  - Reţele semantice

Sisteme inteligente – SBC

Baza de cunoştinţe (BC)

- Modalităţi de reprezentare a cunoştinţelor
  - Logica formală (limbaje formale)
  - Reguli
    - Euristici speciale care generează informaţii (cunoştinţe)
    - O modalitate de exprimare (reprezentare) a cunoştinţelor
    - Ex. *dacă Ionică lucrează la Facebook, atunci el câştigă mulţi bani şi are puţin timp liber*
    - Interdependenţele între reguli ➔ reţea de inferenţă
    - Fac legătura între cauză şi efect - memorate în calculator sub forma unor structuri de control *IF cauză THEN efect*
  - Reţele semantice
    - Grafuri orientate cu noduri care conţin concepte şi arce care reprezintă relaţii semantice între concepte precum:
      - *Meronymy* (A este meronym al lui B dacă A este o parte a lui B)
        - Ex. Degetul este un meronym al mâinii, roata este un meronym al maşinii
      - *Holonymy* (A este holonym al lui B dacă B este o parte a lui A)
        - Ex. Copacul este un holonym al scoarţei
      - *Hyponymy* (A este hyponym al lui B dacă A este un fel de B)
        - Ex. Tractorul este un hyponym al autovehiculului
      - *Hypernymy* (A este hypernym al lui B dacă A este o generalizare al lui B)
        - Ex. Fructul este un hypernym al portocalei
      - *Synonymy* (A este sinonim al lui B dacă A denotă acelaşi lucru ca B)
        - Ex. A alerga este sinonim cu a fugi
      - *Antonymy* (A este antonim al lui B dacă A denotă lucruri opuse ca B)
        - Ex. Uscat este antonim cu ud

Sisteme inteligente – SBC

Baza de cunoştinţe (BC)

- Modalităţi de stocare a cunoştinţelor
  - Relaţii
    - Simple → baze de date
    - Ierarhice → ierarhii de concepte (reţele semantice)
  - Logică formală
    - Reguli
  - Logică procedurală
    - Algoritmi

Sisteme inteligente – SBC

Modulul de control (MC)

- Conţinut
  - Responsabil cu inferenţa
  - A ajunge la o concluzie plecând de la anumite premise (cunoştinţe) şi aplicând anumite reguli de inferenţă
  - MC depinde de complexitate şi tipul cunoştinţelor cu care are de-a face

- Tipologie
  - În funcţie de direcţia inferenţei:
    - MC cu legătură înainte (*forward chaining*)
      - Pornesc de la informaţia disponibilă (fapte date, condiţii) şi încearcă să ajungă la o concluzie (fapte derivate)
      - Se bazează pe date (*data driven*)
    - MC cu legătură înapoi (*backward chaining*)
      - Pornesc de la o concluzie potenţială (ipoteză) şi caută evidenţe care să o suporte-contrazică (explicaţii)
      - Se bazează pe scop *(goal driven*)

- Tehnici de raţionare (tehnici de inferenţă)
  - În medii certe
    - bazate pe logică
    - bazate pe reguli
  - În medii incerte
    - bazate pe teoria probabilităţilor
    - bazate pe teoria posibilităţii

Sisteme inteligente – SBC

Tipologia SBC

- **Sisteme bazate pe logică (SBL)**
- **Sisteme bazate pe reguli (SBR)**
- *Case-based reasoning*
- *Hypertext manipulating systems*
- *Data bases and intelligent UI*
- *Intelligent tutoring systems*

Sisteme inteligente – SBC

Sisteme bazate pe logică (SBL)

- Conţinut şi obiective
- Arhitectură
- Tipologie
- Tool-uri
- Avantaje şi limite

Sisteme inteligente – SBC – SBL

Conţinut şi obiective

- Conţinut
  - explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode specifice logicii formale
  - Un sistem logic este alcătuit din:
    - limbaj (sintaxă + semantică)
    - metodă de deducţie (inferenţă)
- Scopul SBL
  - Rezolvarea de probleme cu ajutorul programării declarative
    - descriind ceea ce este adevărat sau nu în rezolvarea problemelor
    - permiţând tehnici de raţionare automată
  - Exemple de probleme rezolvate de SBRL
    - demonstrarea automată a teoremelor
- De ce se studiază SBL?
  - Logica formală este precisă şi definită

Sisteme inteligente – SBC – SBL

Arhitectură

- Baza de cunoştinţe (BC)
  - Sintaxă
    - simbolurile atomice folosite de către limbaj şi regulile de construcţie a expresiilor (structurilor/propoziţiilor) limbajului
  - Semantică
    - asociază un înţeles simbolurilor şi o valoare de adevăr (Adevărat sau Fals) regulilor (propoziţiilor) limbajului

- Modulul ce control (MC)
  - Metodă de inferenţă sintactică – regulile necesare determinării unei submulţimi de expresii logice → teoreme (folosite pentru obţinerea de noi expresii)

Sisteme inteligente – SBC – SBL

Tipologie

- **Sisteme bazate pe logica propoziţională**
  - se lucrează doar cu propoziţii declarative
  - obiectele descrise sunt unice şi fixe (*Ionică este student)*

- Sisteme bazate pe logica predicatelor de ordin I
  - se lucrează cu propoziţii declarative, cu predicate (*Student(a)*) şi cuantificări (variabile cuantificabile → *pentru orice a, Student(a)* → *AccesWiFi(a)*)
  - obiectele descrise pot fi unice sau dinamice (variabile asociate unui obiect unic – *Toţi studenţii sunt prezenţi*)
  - predicatele au argumente simple (*Student(a)*)

- Sisteme bazate pe logica predicatelor de ordin superior (≥ 2)
  - se lucrează cu propoziţii declarative, cu predicate şi cuantificări (variabile cuantificabile)
  - permit variabilelor să reprezinte mai multe relaţii între obiecte
  - predicatele pot avea argumente simple, argumente de tip predicat (*StudentSenator*(*Student*(*a*))) sau argumente de tip funcţie (*Bursier(a are media peste 9.50*))

- Sisteme temporale
  - Reprezintă valoarea de adevăr a faptelor de-a lungul timpului (*Ionică este uneori grăbit*)

- Sisteme modale
  - Reprezintă şi fapte îndoielnice (*Ionică poate să promoveze examenul*)

Sisteme inteligente – SBC – SBL

Sisteme bazate pe logica propoziţională

- Baza de cunoştinţe
  - Poate fi alcătuită din:
    - Simboluri (A, B, P, Q, ...)
    - Propoziţii (formule)
      - definite astfel:
        1. Un simbol
        2. Dacă P este o propziţie, atunci şi $\neg P$ este tot propoziţie
        3. Dacă P şi Q sunt propoziţii, atunci $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$ sunt tot propoziţii
        4. Un număr finit de aplicări ale regulilor (1) – (3)
      - Interpretare a unei propoziţii → stabilirea valorii de adevăr
      - Model → interpretare a unei mulţimi de propoziţii astfel încât toate propoziţiile să fie adevărate

- Modulul de control
  - realizează inferenţa
    - stabilirea valorii de adevăr a unei propoziţii obiectiv pe baza informaţiilor din BC
  - în mai multe moduri
    - verificarea modelului
      - enumerarea tuturor combinaţiilor posibile pentru valorile de adevăr ale simbolurilor şi propoziţiilor implicate în SBL
    - deducţia modelului cu ajutorul regulilor de inferenţă

Sisteme inteligente – SBC – SBL

Sisteme bazate pe logica propoziţională – modulul de control

- Problemă
  - Se dă o BC = {P1, P2, ..., Pm} formată din simbolurile {X1, X2, ..., Xn} şi o propoziţie obiectiv O.
  - Se poate deduce O din BC?

- Verificarea modelului
  - Etape
    - Se construieşte tabelul corespunzător tuturor combinaţiilor posibile pentru valorile de adevăr ale simbolurilor
    - Se determină dacă toate modelele BC sunt şi modele ale lui O
      - Modelele BC – acele linii în care toate propziţiile din BC sunt *adevărate*
  - Exemplu
    - P = *Afară este foarte cald*
    - Q = *Afară este umezeală*
    - R = *Afară plouă*
    - BC = {P∧Q⇒R, Q⇒P, Q}
    - R = *Va ploua?*

P	Q	R	P∧Q⇒R	Q⇒P	Q	BC	R	BC⇒R
T	T	T	T	T	T	T	T	T
T	T	F	F	T	T	F	F	T
T	F	T	T	T	F	F	T	T
T	F	F	T	T	F	F	F	T
F	T	T	T	F	T	F	T	T
F	T	F	T	F	T	F	F	T
F	F	T	T	T	F	F	T	T
F	F	F	T	T	F	F	F	T

  - Dificultăţi
    - Nr tuturor combinaţiilor creşte exponenţial cu n ➔ timp mare de calcul
    - Soluţia: deducerea prin folosirea regulilor de inferenţă

Sisteme inteligente – SBC – SBL

Sisteme bazate pe logica propoziţională – modulul de control

- Problemă
  - Se dă o BC = {P1, P2, ..., Pm} formată din simbolurile {X1, X2, ..., Xn} şi o propoziţie obiectiv O.
  - Se poate deduce O din BC?

- Deducţia modelului cu ajutorul regulilor de inferenţă
  - Etape
    - Construirea unei demonstraţii a valorii de adevăr a propoziţiei obiectiv pe baza:
      - propoziţiilor
        - originale din BC
        - derivate
      - regulilor de inferenţă

Regulă de inferenţă	Premisă	Propoziţia derivată
Modus ponens	$A, A \Rightarrow B$	B
Şi introductiv	A, B	$A \wedge B$
Şi eliminativ	$A \wedge B$	A
Negaţie dublă	$\neg\neg A$	A
Rezoluţie unitară	$A \vee B, \neg B$	A
Rezoluţie	$A \vee B, \neg B \vee C$	$A \vee C$

  - Exemplu
    - Problemă
      - P = Afară este foarte cald
      - Q = Afară este umezeală
      - R = Afară plouă
      - BC = $\{P \wedge Q \Rightarrow R, Q \Rightarrow P, Q\}$
      - R = Va ploua?

Soluţie

1.	Q	Premisă
2.	$Q \Rightarrow P$	Premisă
3.	P	Modus Ponens (1,2)
4.	$(P \wedge Q) \Rightarrow R$	Premisă
5.	$P \wedge Q$	Şi introductiv (1,3)
6.	R	Modus Ponens (4,5)

Sisteme inteligente – SBC

Tipologia SBC

- **Sisteme bazate pe logică (SBL)**
- **Sisteme bazate pe reguli (SBR)**
- *Case-based reasoning*
- *Hypertext manipulating systems*
- *Data bases and intelligent UI*
- *Intelligent tutoring systems*

Sisteme inteligente – SBC

- Sisteme bazate pe reguli (SBR)
  - Conţinut şi obiective
  - Proiectare
  - Arhitectură
  - Tool-uri şi exemple
  - Avantaje şi limite

Sisteme inteligente – SBC – SBR

Conţinut şi obiective

- Conţinut
  - explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode asemănătoare cu experții umani
  - pot avea succes la problemele fără soluție algoritmică deterministă
  - încearcă să imite un expert uman (într-un anumit domeniu)
  - SBR nu înlocuiesc experienţa umană, dar îi lărgesc sfera disponibilităţii permiţând ne-experţilor să lucreze mai bine ➔ **Sisteme expert (SE)**

- Scopul SBR
  - Rezolvarea acelor tipuri de probleme care, de obicei, necesită experţi umani prin
    - Transferul expertizei de la un expert la un sistem computaţional şi
    - Apoi la alţi oameni (ne-experţi)
  - Exemple de probleme rezolvate de SBR → Probleme de recomandare/consultare
    - Consultant medical – aplicaţie care înlocuieşte medicul (dându-se simptomele, SE sugerează un diagnostic şi un tratament)
    - Detector al problemelor de funcţionare ale unei maşini
    - Detector de probleme în sistemele de operare - Microsoft Windows troubleshooting
    - Consultant financiar

- De ce se studiază SBR?
  - Pentru a înţelege metodele umane de raţionare
  - Experţii umani au nevoie de vacanţe, pot pleca la alte companii, se pot îmbolnăvi, cer măriri de salar, etc.
  - Au foarte mare succes comercial

Sisteme inteligente – SBC – SBR

Proiectare

- Achiziţionarea informaţiilor (cunoştinţelor)
- Reprezentarea cunoştinţelor
- Inferenţa cunoştinţelor
- Transmiterea către utilizator a cunoştinţelor

Sisteme inteligente – SBC – SBR

Arhitectură

- Baza de cunoştinţe (BC)
  - Informaţiile specifice despre un domeniu
- Modulul de control (MC)
  - Regulile prin care se pot obţine informaţii noi
- Interfața cu utilizatorul
  - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din BC pentru adăugare sau actualizare
- Modulul de îmbogățire a cunoașterii
  - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- Modulul explicativ
  - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul

Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoştinţe

- Conţine
  - Informaţiile specifice despre un domeniu sub forma unor
    - fapte – afirmaţii corecte
    - reguli - euristici speciale care generează informaţii (cunoştinţe)

- Rol
  - stocarea tuturor elementelor cunoașterii (fapte, reguli, metode de rezolvare, euristici) specifice domeniului de aplicație, preluate de la experții umani sau din alte surse

Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoştinţe

- Fapte
  - Definiţie
    - Afirmaţii necondiţionate corecte (propoziţii)
    - Memorate în calculator sub forma unor structuri de date
  - Exemplu
    - *Ionică lucrează la Facebook*
  - Tipologie
    - În funcţie de persistenţă (ritmul de modificare)
      - Fapte statice – aprox. permanente (*Ionică lucrează la Facebook*)
      - Fapte tranzitive – specifice unei instanţe/rulări (*Ionică este în pauza de masă*)
    - În funcţie de modul de generare
      - Fapte date (*Ionică participă la şedinţă*)
      - Fapte derivate – rezultate prin aplicarea unor reguli (*Dacă Ionică este PM, atunci el trebuie să conducă şedinţa*)

Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoştinţe

- Reguli
  - Definiţie
    - Euristici speciale care generează informaţii (cunoştinţe)
    - O modalitate de exprimare (reprezentare) a cunoştinţelor
    - Interdependenţele între reguli ➔ reţea de inferenţă
    - Fac legătura între cauză şi efect - memorate în calculator sub forma unor structuri de control *IF cauză THEN efect*
      - Deducţie – cauză + regulă ➔ efect
      - Abducţie – efect + regulă ➔ cauză
      - Inducţie – cauză + efect ➔ regulă
  - Exemplu
    - O cauză şi mai multe consecinţe (combinate cu ŞI)
      - *DACĂ Ionică lucrează la Facebook, ATUNCI el câştigă mulţi bani ŞI are puţin timp liber*
    - O cauză şi mai multe consecinţe (combinate cu SAU)
      - *DACĂ anotimpul este iarnă ATUNCI vremea este rece SAU este zăpadă*
    - Mai multe cauze/antecedente (combinate cu ŞI) şi un efect/o consecinţă
      - *DACĂ anotimpul este iarnă ŞI temperatura este sub 0 grade ŞI bate vântul ATUNCI nu mergem la plimbare*
    - Mai multe cauze/antecedente (combinate cu SAU) şi un efect/o consecinţă
      - *DACĂ anotimpul este iarnă SAU temperatura este sub 0 grade SAU bate vântul ATUNCI vremea este rece*
    - Mai multe cauze/antecedente (combinate cu ŞI şi SAU) şi un efect/o consecinţă
      - *DACĂ anotimpul este iarnă ŞI temperatura este sub 0 grade SAU bate vântul ATUNCI avioanele nu pot ateriza*

Sisteme inteligente – SBC – SBR

Arhitectură → baza de cunoştinţe

- Reguli
  - Tipologie
    - În funcţie de gradul de incertitudine
      - Reguli sigure – *Dacă eşti angajat, atunci primeşti salar*
      - Reguli nesigure – *Dacă este iarnă, temperatura este sub 0 grade*
    - În funcţie de ceea ce exprimă
      - Relaţii – ex. *Dacă studentul are media peste 9.50, atunci el primeşte bursă*
      - Recomandări – ex. *Dacă plouă, atunci să luăm umbrela*
      - Directive – ex. *Dacă bateria telefonului este gata, atunci trebuie pusă la încărcat*
      - Euristici – ex. *Dacă lumina telefonului este stinsă, atunci bateria este plină*
  - Avantajul lucrului cu reguli
    - Uşor de înţeles (o formă naturală a cunoştinţelor)
    - Simplu de explicat
    - Simplu de modificat şi întreţinut
  - Limitări ale regulilor
    - Cunoştinţele complexe necesită exprimarea prin foarte multe reguli
    - Căutarea în sistemele cu numeroase reguli devine greoaie

Sisteme inteligente – SBC – SBR

Arhitectură

- Baza de cunoştinţe (BC)
  - Informaţiile specifice despre un domeniu
- Modulul de control (MC)
  - Regulile prin care se pot obţine informaţii noi
- Interfața cu utilizatorul
  - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din BC pentru adăugare sau actualizare
- Modulul de îmbogățire a cunoașterii
  - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- Modulul explicativ
  - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control

- Conţinut
  - Regulile prin care se pot obţine informaţii noi
  - Algoritmi independenţi de domeniu
  - Creierul SBR – un algoritm de deducere bazat pe BC şi specific metodei de raţionare
    - un program în care s-a implementat cunoașterea de control, procedurală sau operatorie, cu ajutorul căruia se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii.
  - depinde de complexitate şi tipul cunoştinţelor cu care are de-a face

- Rol
  - cu ajutorul lui se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii

- Tipologie – în funcţie de direcţia inferenţei:
  - MC cu legătură înainte (*forward chaining*)
  - MC cu legătură înapoi (*backward chaining*)

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înainte (*forward chaining*)

- Ideea de bază
  - Se porneşte de la informaţia disponibilă (fapte date, condiţii) şi se încearcă ajungerea la o concluzie (fapte derivate) folosind regulile disponibile
  - Regulile sunt de forma:
    - partea stângă (PS) =>partea dreaptă (PD)
    - partea de condiţii =>partea de consecinţe (efecte)
  - Se bazează pe date (*data driven*)

- Exemplu
  - **Întrebare (problemă)**: *Angajatul Popescu are telefon?*

  - **Regulă**: *Dacă Popescu este angajat, atunci el are telefon.*
  - **Fapt curent**: *Popescu este angajat.*
  - **Concluzie**: *Popescu are telefon.*

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înainte (*forward chaining*)

- Algoritm
  - Ciclul de execuţie
    - Repetă
      - Se selectează o regulă a cărei condiţii din PS sunt satisfăcute de starea curentă a faptelor stocată în memoria curentă
      - Se execută PD a regulii anterior selectate (schimbând starea curentă)
    - Până când nu se mai poate aplica nici o regulă
  - Observaţii
    - Faptele sunt reprezentate în memoria curentă (de lucru) care este continuu actualizată
    - Regulile reprezintă acţiuni care pot fi executate atunci când condiţiile specificate sunt satisfăcute de elementele stocate în memoria curentă
    - Condiţiile sunt, de obicei, şabloane care se potrivesc cu elementele din memoria curentă
    - Acţiunile implică, de obicei, adăugarea sau eliminarea unor elemente în memoria curentă

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înapoi (*backward chaining*)

- Ideea de bază
  - Se porneşte de la o concluzie potenţială (ipoteză) şi se caută evidenţe care să o suporte/contrazică (explicaţii)
  - Regulile sunt de forma:
    - partea stângă (PS) ==>partea dreaptă (PD)
  - Se bazează pe scop (goal driven)

- Exemplu
  - **Întrebare (problemă)**: *Angajatul Popescu are calculator?*

  - **Afirmaţie**: *Popescu are calculator.*
  - **Fapt curent**: *Popescu este programator*
  - **Regulă**: *Dacă Popescu este programator, atunci el are calculator*

  - Se verifică setul de reguli şi se caută ce trebuie să fie Adevărat (în PS) pentru ca Popescu să aibă calculator: un programator. Popescu este programator este un fapt, deci atunci el are calculator.

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control cu legătură înapoi (*backward chaining*)

- Algoritm
  - Ciclul de execuţie
    - Se începe cu starea obiectiv
    - Se verifică dacă obiectivul nu se potriveşte cu unul din faptele iniţiale. Dacă da, atunci STOP. Altfel, se caută acele reguli a căror concluzie se potriveşte cu starea obiectiv.
    - Se alege una dintre reguli şi se încearcă demonstrarea faptelor din precondiţie (folosind acelaşi mecanism), care devin noi obiective.
  - Observaţii
    - este necesară memorarea obiectivelor urmărite

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Baza de cunoştinţe
  - Fapte
    - A – secreţii nazale
    - B – sinuzită
    - C – dureri de cap
    - D - ameţeli
    - E – febră
    - F – probleme cu tensiunea
    - X – infecţie
    - Y – antibiotic
    - Z – repaus la pat

  - Reguli
    - R1: *dacă A este adevărat şi C este adevărat atunci B este adevărat*
    - R2: *dacă C este adevărat şi D este adevărat atunci F este adevărat*
    - R3: *dacă C este adevărat şi D este adevărat şi E este adevărat atunci X este adevărat*
    - R4: *dacă A este adevărat şi B este adevărat şi X este adevărat atunci Y este adevărat*
    - R5: *dacă D este adevărat şi Y este adevărat atunci Z este adevărat*
  - Scop
    - faptul Z

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Algoritm *forward chaining*
  - Se repetă
    - Se selectează regulile aplicabile pentru faptele existente în BC
      - Regulile care conţin în PS a lor doar fapte deja existente în BC
    - Dacă pentru un fapt se pot aplica mai multe reguli, se alege doar una dintre ele (care nu a mai fost folosită)
    - Se aplică regulile selectate, iar faptele noi obţinute se adaugă în BC
  - Până când se ajunge la concluzie sau la o regulă care indică oprirea procesului

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 1

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 2

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 3

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Algoritm *backward chaining*
  - Se repetă
    - Se selectează regulile care se potrivesc cu scopul
      - Regulile care conţin în PD a lor scopul urmărit
      - Dacă pentru un scop se pot aplica mai multe reguli, se alege doar una dintre ele
    - Se verifică regulile selectate
      - Se înlocuieşte scopul cu premisele (cauzele) regulii selectate, acestea devenind sub-scopuri
  - Până când toate sub-scopurile sunt adevărate
    - Sunt fapte cunoscute (existente iniţial în BC)
    - sunt informaţii oferite de utilizator
  - Se repetă
    - Se aplică regulile anterior verificate în ordine inversă
  - Până la aplicarea tuturor regulilor şi obţinerea scopului urmărit (ca fapt in BC)

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 1.1

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 1.1

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 1.2

R1: if A & C then B
R2: if C & D then F
R3: if C & D & E then X
R4: if A & B & X then Y
R5: if D & Y then Z

$X \notin BC \rightarrow X$
= sub-scop

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 1.2

R1: if A & C then B
R2: if C & D then F
R3: if C & D & E then X
R4: if A & B & X then Y
R5: if D & Y then Z

$X \notin BC \rightarrow X$
= sub-scop

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 2.1

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 2.1

Sisteme inteligente – SBC – SBR

Arhitectură → modulul de control – exemplu

- Iteraţia 2.1

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control

- Dificultăţi
  - *Forward Chaining* (FC) sau *Backward chaining* (BC)?
  - Rezolvarea conflictelor

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăţi

- *Forward Chaining* (FC) sau *Backward chaining* (BC)?
  - FC se recomandă a fi folosit atunci când:
    - Toate (sau aproape toate) informaţiile se dau de la început în problemă
    - Există un număr mare de scopuri potenţiale, dar doar o parte din ele sunt realizabile pentru o instanţă dată a problemei
    - Este dificilă formularea unui scop sau a unor ipoteze
  - BC se recomandă a fi folosit atunci când:
    - Scopul sau ipotezele se dau în problemă sau sunt uşor de formulat
    - Există numeroase reguli care se potrivesc cu faptele din BC, producând numeroase concluzii
    - Datele problemei nu se dau (sau nu sunt uşor accesibile), dar trebuie achiziţionate (în anumite sisteme)

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăţi

- Rezolvarea conflictelor
  - Dacă se pot aplica mai multe reguli, care regulă este aleasă?
    - De ex.
      - R1: *dacă culoarea este galben atunci fructul este măr*
      - R2: *dacă culoarea este galben şi forma este lunguiaţă atunci fructul este banană*
      - R3: *dacă forma este rotundă atunci fructul este măr*
  - Strategii de alegere a regulii care se aplică
    - **prima** regulă
    - o regulă **aleatoare**
    - regula **cea mai specifică**
    - **cea mai veche** regulă
    - **cea mai bună** regulă

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăţi

- Rezolvarea conflictelor
  - Alegerea **primei** reguli care se potriveşte (***First in first serve***)
    - Exemplu
      - R1: dacă culoarea este galben atunci fructul este măr
      - R2: dacă culoarea este galben şi forma este lunguiaţă atunci fructul este banană
      - R3: dacă forma este rotundă atunci fructul este măr
    - Observaţii
      - Regulile sunt ordonate doar în sistemele mici

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăţi

- Rezolvarea conflictelor
  - Alegerea **aleatoare** a unei reguli care se potriveşte
    - Exemplu
      - R1: dacă culoarea este galben atunci fructul este măr
      - R2: dacă culoarea este galben şi forma este lunguiaţă atunci fructul este banană
      - R3: dacă forma este rotundă atunci fructul este măr
    - Observaţii
      - Alegerea poate fi bună sau mai puţin bună

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăţi

- Rezolvarea conflictelor
  - Alegerea **celei mai specifice** reguli (***Specificity***)
    - Cea cu cele mai multe condiţii, fiind cea mai relevantă pentru datele existente
    - Exemplu
      - R1: dacă culoarea este galben atunci fructul este măr
      - R2: dacă culoarea este galben şi forma este lunguiaţă atunci fructul este banană
      - R3: dacă forma este rotundă atunci fructul este măr
    - Observaţii
      - O regulă specifică procesează mai multă informaţie decât o regulă generală ➔*longest matching strategy*

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăţi

- Rezolvarea conflictelor
  - Alegerea **celei mai vechi** reguli utilizate (***Recency***):
    - Fiecare regulă are asociată o marcă temporală – ultima dată când a fost folosită
    - Exemplu
      - R1: dacă culoarea este galben atunci fructul este măr [12.01.2012 – 13.45]
      - R2: dacă culoarea este galben şi forma este lunguiaţă atunci fructul este banană [7.02.2012 – 21.10]
      - R3: dacă forma este rotundă atunci fructul este măr [10.01.2012 – 10.25]
    - Observaţii
      - Noile reguli au fost adăugate de experţi mai puţin pregătiţi decât vechile reguli (adăugate de experţi mai bine pregătiţi – cu mai multe cunoştinţe în domeniu)

Sisteme inteligente – SBC – SBR

Arhitectură – modulul de control – dificultăţi

- Rezolvarea conflictelor
  - Alegerea **celei mai bune** reguli (***Prioritization***)
    - Fiecare regulă are asociată o pondere care specifică importanţa ei (relativ la alte reguli)
    - Exemplu
      - R1: dacă culoarea este galben atunci fructul este măr [30%]
      - R2: dacă culoarea este galben şi forma este lunguiaţă atunci fructul este banană [30%]
      - R3: dacă forma este rotundă atunci fructul este măr [40%]
    - Observaţii
      - Necesită expertiză umană pentru stabilirea importanţei

Sisteme inteligente – SBC – SBR

Arhitectură

- Baza de cunoştinţe (BC)
  - Informaţiile specifice despre un domeniu
- Modulul de control (MC)
  - Regulile prin care se pot obţine informaţii noi
- Interfața cu utilizatorul
  - permite dialogul cu utilizatorii în timpul sesiunilor d consultare, precum și accesul acestora la faptele și cun din bază pentru adăugarea sau actualizarea bazei.

- Modulul de îmbogățire a cunoașterii
  - ajută utilizatorul expert să introducă în bază noi cunoștințe

  într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- Modulul explicativ
  - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul

Sisteme inteligente – SBC – SBR

Arhitectură

- Interfaţa cu utilizatorul
  - Procesarea limbajului de dialog
    - Tehnici de procesare a limbajului
    - Meniuri
    - Elemente grafice, etc
- Modulul de îmbogățire a cunoașterii
  - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe

Sisteme inteligente – SBC – SBR

Arhitectură

- Modulul explicativ
  - are rolul de a explica utilizatorilor
    - cunoștințele de care dispune sistemul,
    - raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare.
  - explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul
  - Exemplu
    - Un expert medical care prescrie un tratament unui pacient trebuie să explice
      - motivele pentru care a ajuns la acea recomandare
      - riscurile unui astfel de tratament
      - alternative la acest tratament

Sisteme inteligente – SBC – SBR

- Conţinut şi obiective
- Proiectare
- Arhitectură
- Tool-uri şi exemple
- Avantaje şi limite

Sisteme inteligente – SBC – SBR

Tool-uri si limbaje existente

- PROLOG
  - Limbaj de programare care utilizează *backward chaining*
- ART (Inference Corporation)
  - în 1984, Inference Corporation a dezvoltat Automated Reasoning Tool (ART), un sistem expert bazat pe *forward chaining*
- CLIPS
  - NASA preia abilităţile de *forward chaining* ale sintaxei ART şi dezvoltă *C Language Integrated Production System*" (CLIPS)
- ART-IM (Inference Corporation)
  - Inference Corporation implementează o versiunea *forward-chaining a* ART/CLIPS, numită ART-IM.
- OPS5 (Carnegie Mellon University)
  - Primul libaj de IA utilizat pentru sisteme de producţie (XCON)
- Eclipse (The Haley Enterprise, Inc.)
  - Eclipse este singurul modul de control pentru C/C++ care suportă atât *forward chaining*, cât şi *backward chaining*

Sisteme inteligente – SBC – SBR

Exemple

- DENDRAL (1965-1983)
  - Analizează structura moleculelor şi propune structuri plauzibile pentru compuşi chimici noi sau necunoscuţi
- MYCIN (1972-1980)
  - Program interactiv pentru
    - diagnosticarea unor boli infecţioase sangvine
    - Recomandări terapeutice antimicrobiene
- EMYCIN, HEADMED, CASNET și INTERNIST
  - pentru domeniul medical
- PROSPECTOR (1974-1983)
  - Oferă recomandări pentru explorările mineralelor
- TEIRESIAS
  - pentru achiziția inteligentă a cunoașterii
- XCON (1978-1999)
  - Oferă recomandări pentru configurarea calculatoarelor
- SBR financiare
  - ExpertTAX, Risk Advisor (Coopers & Lybrand), Loan Probe, Peat/1040 (KPMG), VATIA, Flow Eval (Ernst & Young), Planet, Compas, Comet (Price Waterhouse), Rice (Arthur Andersen), Audit Planning Advisor, World Tax Planner (Deloitte Touche)

Sisteme inteligente – SBC – SBR

Avantaje şi limite

- Avantaje
  - Oderă recomandări celor mai puţin experţi în anumite domenii
  - Permit companiilor replicarea celor mai buni angajaţi
    - Preiua informaţia şi cunoştinţele intelectuale ale experţilor şi le pun la dispoziţia celorlalţi oameni
  - Se reduc erorile datoarate proceselor de automatizare a sarcinilor monotone, repetitive sau critice
  - Se reduce necesarul de forţă umană şi de timp pentru testarea şi analizarea datelor
  - Se reduc costurile prin accelerarea procesului de observare a greşelilor
  - Se elimină munca pe care oamenii nu ar trebui să o facă (dificilă, consumatoare de timp, susceptibilă de erori, care necesită antrenare lungă şi costisitoare)
  - Se elimină munca pe care oamenii nu şi-o doresc să o facă (luarea unor decizii care nu-i pot mulţumi pe toţi – sistemele expert nu pot fi acuzate de favoritsime)

- Dezavantaje
  - Domeniu îngust de aplicare a unui SBR
  - Focus limitat la anumite obiective
  - Lipsa capacităţii de învăţare şi adaptare
  - Probleme de întreţinere
  - Costuri de dezvoltare mari

Recapitulare

- SBC
  - Sisteme computaţionale în care
  - baza de cunoştinţe şi modulul de control se suprapun
- SBC pot fi
  - SBL
    - explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode specifice logicii formale
    - Componenţă
      - limbaj (sintaxă + semantică) şi
      - metoda de deducţie (inferenţă)
  - SBR
    - explorează o multitudine de cunoștințe date pentru a obține concluzii noi despre activități dificil de examinat, folosind metode asemănătoare cu experții umani
    - pot avea succes la problemele fără soluție algoritmică deterministică
    - încearcă să imite un expert uman (într-un anumit domeniu)
    - Componenţă
      - Baza de cunoştinţe → fapte şi reguli
      - Modulul de control → inferenţă înainte sau înapoi

Cursul următor

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare
- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente
- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme hibride
- Sisteme bazate pe reguli în medii certe
- Sisteme bazate pe reguli în medii incerte (Bayes, factori de certitudine, Fuzzy)

Cursul următor – Materiale de citit şi legături utile

- Capitolul V din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 3 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 8 şi 9 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop

UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENŢĂ ARTIFICIALĂ

Sisteme inteligente

Sisteme bazate pe reguli în medii incerte

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme hibride
- Sisteme bazate pe reguli în medii certe
- Sisteme bazate pe reguli în medii incerte (Bayes, factori de certitudine, Fuzzy)

Materiale de citit şi legături utile

- capitolul V din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 3 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 8 şi 9 din *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

Conținut

- Sisteme inteligente
  - Sisteme bazate pe cunoştinţe
    - Sisteme bazate pe reguli în medii incerte

Sisteme inteligente

Sisteme bazate pe cunoştinţe

Inteligenţă computaţională

Sisteme inteligente–sisteme bazate pe cunoştinţe (SBC)

- sistemele computaţionale – alcătuite din 2 module principale (roluri):
  - Domeniul de cunoştinţe (baza de cunoştinţe – BC – *knowledge base*)
    - Informaţiile specifice despre un domeniu
  - Modulul de control (MC – *inference engine*)
    - Regulile prin care se pot obţine informaţii noi
    - Algoritmi independenţi de domeniu

Sisteme inteligente – SBC

Baza de cunoştinţe (BC)

- Conţinut
  - Informaţii (exprimate într-o anumită reprezentare – ex. propoziţii) despre mediu
  - informaţii necesare pentru înţelegerea, formularea şi rezolvarea problemelor
  - mulţime de propoziţii (exprimate/reprezentate într-un limbaj formal) care descriu mediul
    - reprezentare uşor interpretabilă de către calculator ➔ limbaj de reprezentare a cunoştinţelor
    - mecanismul de obţinere a unor propoziţii noi pe baza celor vechi ➔ inferenţă/raţionare

- Tipologie
  - cunoştinţe exacte (perfecte)
    - Raţionarea exactă se bazează pe logica clasică
      - *IF A is true THEN A is ┐false*
      - *IF B is false THEN B is ┐true*
  - cunoştinţe imperfecte (nesigure, incerte)
    - Inexacte
    - Incomplete
    - Incomensurabile

Sisteme inteligente – SBC

- Surse ale incertitudinii
  - Imperfecţiunea regulii
  - Nesiguranţa evidenţei (dovezii)
    - Încrederea în concluzie trebuie scalată
  - Utilizarea unui limbaj vag, imprecis

- Moduri de exprimare a incertitudinii
  - Probabilităţi
  - Logica fuzzy
  - Teorema lui Bayes
  - Teoria Dempster-Shafer

- Moduri de reprezentare a incertitudinii
  - Cu ajutorul unui singur număr → factor de certitudine, confidenţă (încredere), valoare de adevăr
    - Cât de siguri suntem că respectivele date sunt valide
  - Cu ajutorul a 2 numere → logica intervalelor
    - Min →limita inferioară a certitudinii (încredere, necesitate)
    - Max → limita superioară a certitudinii (plauzabilitate, posibilitate)

Sisteme inteligente – SBC

- Tehnici de raţionare în medii nesigure
  - Teoria Bayesiana – metodă probabilistică
  - Teoria certitudinii
  - Teoria posibilităţii (logica fuzzy)

Metode euristice

Sisteme inteligente – SBC – factori de certitudine

- sisteme de tip Bayes
  - SBR (Sisteme expert) în care faptele şi regulile sunt probabilistice
- sisteme cu factori de certitudine
  - SBR (sisteme expert) în care faptele şi regulile au asociate câte un factor de certitudine (FC)/coeficient de încredere
  - Un fel de sisteme de tip Bayes în care probabilităţile sunt înlocuite cu factori de certitudine

- Dacă A şi B atunci C
- Dacă C şi D atunci F

SBR classic

- Dacă A şi B atunci C [cu prob p_1]
- Dacă C şi D atunci F [cu prob p_2]

SBR de tip Bayes

- Dacă A şi B atunci C [FC_1]
- Dacă C şi $D_{[FC2]}$ atunci F [FC_3]

SBR cu FC

Sisteme inteligente – SBC – factori de certitudine

- SBR de tip Bayes *vs.* SBR cu FC

Bayes	FC
Teorie probabilităţilor este veche şi fundamentată matematic	Teoria FC este nouă şi fără demostrţii matematice
Necesită existenţa unor informaţii statistice	Nu necesită existenţa unor date statistice
Propagarea încrederii creşte în timp exponenţial	Informaţia circulă repede şi eficient în SBR
Este necesar calculul apriori a probabilităţilor	Nu este necesar calculul apriori a probabilităţilor
Ipotezele susţinute de probe pot să fie independente sau nu	Ipotezele susţinute de probe sunt independente

Sisteme inteligente – SBC – sisteme de tip Bayes

- Elemente de teoria probabilităţilor
- Conţinut şi arhitectură
- Tipologie
- Tool-uri
- Avantaje şi limite

Sisteme inteligente – SBC – sisteme de tip Bayes

Amintim componenţa unui SBC

- Baza de cunoştinţe (BC) → Modalităţi de reprezentare a cunoştinţelor
  - Logica formală (limbaje formale)
    - Definiţie
      - Ştiinţa principiilor formale de raţionament
    - Componente
      - Sintaxă
      - Semantică
      - Metodă de inferenţă sintactică
    - Tipologie
      - În funcţie de numărul valorilor de adevăr:
        - logică duală
        - logică polivalentă
      - În funcţie de tipul elementelor de bază:
        - clasică → primitivele = propoziţii (predicate)
        - probabilistică → primitivele = variabile aleatoare
      - În funcţie de obiectul de lucru:
        - logica propoziţională → se lucrează doar cu propoziţii declarative, iar obiectele descrise sunt fixe sau unice (Ionică este student)
        - logica predicatelor de ordin I → se lucrează cu propziţii declarative, cu predicate şi cuantificări , iar obiectele descrise pot fi unice sau variabile asociate unui obiect unic (Toţi studenţii sunt prezenţi)
  - Reguli
  - Reţele semantice
- Modulul de control (MC – pentru inferenţă)

Sisteme inteligente – SBC – sisteme de tip Bayes

Elemente de teoria probabilităţilor

- Teorii ale probabilităţilor
- Concepte de bază
  - Teoria clasică şi teoria modernă
  - Eveniment
  - Probabilitate simplă
  - Probabilitate condiţionată
  - Axiome

Elemente de teoria probabilităţilor

- Teorii ale probabilităţilor
  - Teoria clasică (*a priori*)
    - Propusă de Pascal şi Fermat în 1654
    - Lucrează cu sisteme ideale:
      - toate posibilele evenimente sunt cunoscute
      - toate evenimentele se pot produce cu aceeaşi probabilitate (sunt uniform distribuite)
    - evenimente discrete
    - metode combinatoriale
    - spaţiul rezultatelor posibile este continuu
  - Teoria modernă
    - evenimente continue
    - metode combinatoriale
    - spaţiul rezultatelor posibile este cuantificabil

Elemente de teoria probabilităţilor

- Concepte de bază
  - Considerăm un experiment care poate produce mai multe ieşiri (rezultate)
    - Ex. *Ev1: Aruncarea unui zar poate produce apariţia uneia din cele 6 feţe ale zarului (deci 6 rezultate)*
  - Eveniment
    - Definiţie
      - producerea unui anumit rezultat
      - Ex. *Ev2: Apariţia feţei cu nr 3*
      - Ex. *Ev3: Apariţia unei feţe cu un nr par (2,4,6)*
    - Tipologie
      - Evenimente independente şi mutual exclusive
        - Nu se pot produce simultan
        - Ex. *Ev4: Apariţia feţei 1 la aruncarea unui zar* şi *Ev5: Apariţia feţei 3 la aruncarea unui zar*
      - Dependente
        - Producerea unor evenimente afectează producerea altor evenimente
        - Ex. *Ev6: Apariţia feţei 6 la prima aruncare a unui zar* şi Ev7: *Apariţia unor feţe a căror numere însumate să dea 8 la 2 aruncări succesive ale unui zar*
  - Mulţimea tuturor rezultatelor = *sample space* al experimentului
    - Ex. pentru *Ev1*: (1,2,3,4,5,6)
  - Mulţimea tuturor rezultatelor tuturor evenimentelor posibile = *power set* (mulţimea părţilor)

Elemente de teoria probabilităţilor

- Concepte de bază
  - Probabilitate simplă *p(A)*
    - probabilitatea producerii unui eveniment A independent de alte evenimente (B)
    - şansa ca acel eveniment să se producă
    - proporţia cazurilor de producere a evenimnetului în mulţimea tuturor cazurilor posibile
    - nr cazurileor favorabile / nr cazurilor posibile
    - un număr real în [0,1]
      - 0 – imposibilitate absolută
      - 1 – posibilitate absolută
    - Ex. P(*Ev1*) =1/6, P(*Ev3*)=3/6
  - Probabilitate condiţionată *p(A|B)*
    - probabilitatea producerii unui eveniment A dependentă de producerea altor evenimente (B)
    - proporţia cazurilor de producere a evenimnetului A şi a evenimentului B în mulţimea tuturor cazurilor producerii evenimentului B
    - probabilitatea comună /proabilitatea lui B

$$p(A \mid B) = \frac{p(A \cap B)}{p(B)}$$

Elemente de teoria probabilităţilor

- Concepte de bază
  - Axiome
    - $0 \leq p(E) \leq 1$ pentru orice eveniment E
    - $p(Adevărat) = 1$, $p(Fals) = 0$
    - $\sum_i p(E_i) = 1$
    - Dacă A şi B sunt independente
      - $p(A \cup B) = p(A) + p(B)$
      - $p(A \cap B) = p(A) * p(B)$
    - Dacă A şi B nu sunt independente
      - Dacă A depinde de B
        - $p(A \cup B) = p(A) + p(B) - p(A \cap B)$
        - $p(A \cap B) = p(A|B) * p(B)$
        - $p(B \cap A) = p(A \cap B)$
        - $p(A|B) = \frac{p(B|A)p(A)}{p(B)}$ (b)
      - Dacă A depinde de $B_1, B_2, \ldots, B_n$ (evenimente mutual exclusive)
        - $p(A) = \sum_{i=1}^{n} p(A|B_i)\,p(B_i)$ (a)

Elemente de teoria probabilităţilor

- Concepte de bază
  - Exemplu
    - Dacă A depinde de 2 evenimente mutual exclusive (B şi ┐B), FC ec.

$$p(A) = \sum_{i=1}^{n} p(A \mid B_i) p(B_i)$$ avem:

      - *p(B) = p(B|A)p(A) + p(B|┐A)p(┐A)*
    - Înlocuind pe *p(B)* în ec. $p(A \mid B) = \dfrac{p(B \mid A) p(A)}{p(B)}$ se obţine ec.:

$$p(A \mid B) = \frac{p(B \mid A) p(A)}{p(B \mid A) p(A) + p(B \mid \neg A) p(\neg A)} \qquad \text{(c)}$$

    - Ecuaţia (c) se foloseşte pentru controlul incertitudinii în sistemele expert

Sisteme inteligente – SBC – sisteme de tip Bayes

Reamintim ca un SBR are următoarea

Arhitectură

- Baza de cunoştinţe (BC)
  - Informaţiile specifice despre un domeniu
- Modulul de control (MC)
  - Regulile prin care se pot obţine informaţii noi
- Interfața cu utilizatorul
  - permite dialogul cu utilizatorii în timpul sesiunilor de consultare, precum și accesul acestora la faptele și cunoștințele din BC pentru adăugare sau actualizare
- Modulul de îmbogățire a cunoașterii
  - ajută utilizatorul expert să introducă în bază noi cunoștințe într-o formă acceptată de sistem sau să actualizeze baza de cunoștințe.
- Modulul explicativ
  - are rolul de a explica utilizatorilor atât cunoștințele de care dispune sistemul, cât și raționamentele sale pentru obținerea soluțiilor în cadrul sesiunilor de consultare. Explicațiile într-un astfel de sistem, atunci când sunt proiectate corespunzător, îmbunătățesc la rândul lor modul în care utilizatorul percepe și acceptă sistemul

Sisteme inteligente – SBC – sisteme de tip Bayes

Reamintim: SBR – arhitectură

- baza de cunoştinţe
  - Conţinut
    - Informaţiile specifice despre un domeniu sub forma unor
      - fapte – afirmaţii corecte
      - reguli - euristici speciale care generează informaţii (cunoştinţe)
  - Rol
    - stocarea tuturor elementelor cunoașterii (fapte, reguli, metode de rezolvare, euristici) specifice domeniului de aplicație, preluate de la experții umani sau din alte surse

- modulul de control
  - Conţinut
    - regulile prin care se pot obţine informaţii noi
    - algoritmi independenţi de domeniu
    - creierul SBR – un algoritm de deducere bazat pe BC şi specific metodei de raţionare
      - un program în care s-a implementat cunoașterea de control, procedurală sau operatorie, cu ajutorul căruia se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii.
    - depinde de complexitate şi tipul cunoştinţelor cu care are de-a face
  - Rol
    - cu ajutorul lui se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Ideea de bază
  - SBR (Sisteme expert) în care faptele şi regulile sunt probabilistice

- Dacă A şi B atunci C
- Dacă C şi D atunci F

SBR classic

- Dacă A şi B atunci C [cu probabilitatea p_1]
- Dacă C şi D atunci F [cu probabilitatea p_2]

SBR de tip Bayes

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Regulile din BC sunt (în general) de forma:
  - Dacă evenimentul (faptul) I este adevărat, atunci evenimentul (faptul) D este adevărat [cu probabilitatea p]
  - Dacă evenimentul I s-a produs, atunci evenimentul D se va produce cu probabilitatea p
    - I – ipoteza (aserţiune, concluzie)
    - D – dovada (premisa) care susţine ipoteza

$$p(I \mid D) = \frac{p(D \mid I)\,p(I)}{p(D \mid I)\,p(I) + p(D \mid \neg I)\,p(\neg I)} \qquad \text{(d)}$$

  - unde:
    - p(I) – probabilitatea apriori ca ipoteza I să fie adevărată
    - p(D|I) – probabilitatea ca ipoteza I fiind adevărată să implice dovada D
    - p(¬ I) – probabilitatea apriori ca ipoteza I să fie falsă
    - p(D|¬ I) - probabilitatea găsirii dovezii D chiar dacă ipoteza I este falsă
- Cum şi cine calculează aceste probabilităţi? → modulul de control

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Cum calculează MC aceste probabilităţi într-un SBR?

$$p(I \mid D) = \frac{p(D \mid I)p(I)}{p(D \mid I)p(I) + p(D \mid \neg I)p(\neg I)} \qquad \text{(d)}$$

  - utilizatorul furnizează informaţii privind dovezile observate
  - experţii determină probabilităţile necesare rezolvării problemei
    - Probabilităţi apriori pentru posibile ipoteze (adevărate sau false) – p(I) şi p(¬ I)
    - Probabilităţile condiţionate pentru observarea dovezii D dacă ipoteza I este adevărată p(D|I), respectiv falsă p(D|¬ I)
  - SBR calculează probabilitatea posteriori p(I|D) pentru ipoteza I în condiţiile dovezilor D furnizate de utilizator

- Actualizare de tip Bayes
  - O tehnică de actualizare a probabilităţii *p* asociate unei reguli care susţine o ipoteză pe baza dovezilor (pro sau contra)
  - Inferenţă (raţionament) de tip Bayes

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Actualizare de tip Bayes
  - O tehnică de actualizare a probabilităţii *p* asociate unei reguli care susţine o ipoteză pe baza dovezilor (pro sau contra)
  - Actualizarea poate ţine cont de:
    - una sau mai multe (m) ipoteze (exclusive şi exhaustive)
    - una sau mai multe (n) dovezi (exclusive şi exhaustive)
  - Cazuri:
    - Mai multe ipoteze şi o singură dovadă

$$p(I_i \mid D) = \frac{p(D \mid I_i)p(I_i)}{\sum_{k=1}^{m} p(D \mid I_k)p(I_k)}$$

    - Mai multe ipoteze şi mai multe dovezi

$$p(I_i \mid D_1 D_2 \ldots D_n) = \frac{p(D_1 D_2 \ldots D_n \mid I_i)p(I_i)}{\sum_{k=1}^{m} p(D_1 D_2 \ldots D_n \mid I_k)p(I_k)} = \frac{p(D_1 \mid I_i)p(D_2 \mid I_i)\ldots p(D_n \mid I_i)p(I_i)}{\sum_{k=1}^{m} p(D_1 D_2 \ldots D_n \mid I_k)p(I_k)}$$

Conţinut şi arhitectură

- Exemplu numeric
  - Pp. un SBR în care:
    - utilizatorul
      - furnizează 3 dovezi condiţionate independente D_1, D_2 şi D_3
    - expertul
      - crează 3 ipoteze mutual exclusive şi exhaustive I_1, I_2 şi I_3 şi stabileşte probabilităţile asociate lor – $p(I_1)$, $p(I_2)$ şi $p(I_3)$
      - determină probabilităţile condiţionate pentru observarea fiecărei dovezi pentru toate ipotezele posibile

probabilitatea	Ipotezele		
	i = 1	i = 2	i = 3
$p(I_i)$	0.40	0.35	0.25
$p(D_1\|I_i)$	0.30	0.80	0.50
$p(D_2\|I_i)$	0.90	0.00	0.70
$p(D_3\|I_i)$	0.60	0.70	0.90

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Exemplu numeric
  - Presupunem că prima dovadă observată este D_3

probabilitatea	Ipotezele		
	i = 1	i = 2	i = 3
$p(I_i)$	0.40	0.35	0.25
$p(D_1\|I_i)$	0.30	0.80	0.50
$p(D_2\|I_i)$	0.90	0.00	0.70
$p(D_3\|I_i)$	0.60	0.70	0.90

  - SE calculează probabilităţile posteriori $p(I_i|D_3)$ pentru toate ipotezele:

$$p(I_1 \mid D_3) = \frac{0.60 \cdot 0.40}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.34$$

$$p(I_2 \mid D_3) = \frac{0.70 \cdot 0.35}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.34$$

$$p(I_3 \mid D_3) = \frac{0.90 \cdot 0.25}{0.60 \cdot 0.40 + 0.70 \cdot 0.35 + 0.90 \cdot 0.25} = 0.32$$

  - După observarea dovezii D_3
    - încrederea în ipoteza I_2 este aceeaşi cu încrederea în ipoteza I_1
    - încrederea în ipoteza I_3 creşte

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Exemplu numeric
  - Presupunem că a doua dovadă observată este D_1

probabilitatea	Ipotezele		
	i = 1	i = 2	i = 3
$p(I_i)$	0.40	0.35	0.25
$p(D_1\|I_i)$	0.30	0.80	0.50
$p(D_2\|I_i)$	0.90	0.00	0.70
$p(D_3\|I_i)$	0.60	0.70	0.90

  - SE calculează probabilităţile posteriori $p(I_i|D_1D_3)$ pentru toate ipotezele:

$$p(I_1 \mid D_1D_3) = \frac{0.30 \cdot 0.60 \cdot 0.40}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.19$$

$$p(I_2 \mid D_1D_3) = \frac{0.80 \cdot 0.70 \cdot 0.35}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.52$$

$$p(I_3 \mid D_1D_3) = \frac{0.50 \cdot 0.90 \cdot 0.25}{0.30 \cdot 0.60 \cdot 0.40 + 0.80 \cdot 0.70 \cdot 0.35 + 0.50 \cdot 0.90 \cdot 0.25} = 0.29$$

  - După observarea dovezii D_1
    - încrederea în ipoteza I_1 scade
    - încrederea în ipoteza I_2 creşte (fiind cea mai probabilă de a fi adevărată)
    - încrederea în ipoteza I_3 creşte

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Exemplu numeric
  - Presupunem că ultima dovadă observată este D_2
  - Se calculează probabilităţile posteriori $p(I_i|D_2D_1D_3)$ pentru toate ipotezele:

probabilitatea	Ipotezele		
	i = 1	i = 2	i = 3
$p(I_i)$	0.40	0.35	0.25
$p(D_1\|I_i)$	0.30	0.80	0.50
$p(D_2\|I_i)$	0.90	0.00	0.70
$p(D_3\|I_i)$	0.60	0.70	0.90

$$p(I_1 \mid D_2D_1D_3) = \frac{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.45$$

$$p(I_2 \mid D_2D_1D_3) = \frac{0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.00$$

$$p(I_3 \mid D_2D_1D_3) = \frac{0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25}{0.90 \cdot 0.30 \cdot 0.60 \cdot 0.40 + 0.00 \cdot 0.80 \cdot 0.70 \cdot 0.35 + 0.70 \cdot 0.50 \cdot 0.90 \cdot 0.25} = 0.55$$

  - După observarea dovezii D_2
    - încrederea în ipoteza I_1 creşte
    - încrederea în ipoteza I_2 e nulă (ipoteza e falsă)
    - Încrederea în ipoteza I_3 creşte

Sisteme inteligente – SBC – sisteme de tip Bayes

Conţinut şi arhitectură

- Exemplu practic
  - Presupunem cazul unei maşini care nu porneşte când este accelerată, dar scoate fum
    - Dacă scoate fum, atunci acceleraţia este defectă [cu probabilitatea 0.7]
    - $P(I_1|D_1) = 0.7$
  - Pe baza unor observări statistice, experţii au constatat:
    - următoarea regulă:
      - Dacă acceleraţia este defectă, atunci maşina scoate fum [cu probabilitatea 0.85]
    - probabilitatea ca maşina să pornească din cauză că acceleraţia este defectă = 0.05 (probabilitate *apriori*)
    - deci avem
      - 2 ipoteze:
        - I_1: acceleraţia este defectă
        - I_2: acceleraţia nu este defectă
      - o dovadă
        - D_1: maşina scoate fum
      - probabilitatea că acceleraţia este defectă dacă maşina scoate fum
        - $P(I_1|D_1)=p(D_1|I_1)*p(I_1)/(p(D_1|I_1)*p(I_1)+p(D_1|I_2)*p(I_2))$
        - $P(I_1|D_1)=0.23 < 0.7$

	I_1	I_2
$p(I_i)$	0.05	1-0.05=0.95
$P(D_1\|I_i)$	0.85	1-0.85=0.15

Sisteme inteligente – SBC – sisteme de tip Bayes

Tipologie

- Sisteme simple de tip Bayes
  - Consecinţele unei ipoteze nu sunt corelate
- Reţele de tip Bayes
  - Consecinţele unei ipoteze pot fi corelate

- De exemplu, reţinem informaţii despre vârsta (V), educaţia (E), câştigurile (C) şi preferinţa pentru teatru (T) ale unor persoane

Sistem Bayes simplu (naiv)

Reţea Bayes simplu

Sisteme inteligente – SBC – sisteme de tip Bayes

- Tool-uri
  - MSBNx – view
  - JavaBayes – view
  - BNJ – view

Sisteme inteligente – SBC – sisteme de tip Bayes

- Avantaje ale inferenţei de tip Bayes
  - Tehnică bazată pe teoreme statistice
  - Probabilitatea dovezilor (simptomelor) în ipotezele (cauzele) date sunt posibil de furnizat
  - Probabilitatea unei ipoteze se poate modifica datorită uneia sau mai multor dovezi
- Dezavantaje ale inferenţei de tip Bayes
  - Trebuie cunoscute (sau ghicite) probabilităţile apriori ale unor ipoteze

Sisteme inteligente – SBC

- Tehnici de raţionare în medii nesigure
  - Teoria Bayesiana – metodă probabilistică
  - **Teoria certitudinii**
  - Teoria posibilităţii (logica fuzzy)

Metode euristice

Sisteme inteligente – SBC – factori de certitudine

- Conţinut şi arhitectură
- Tipologie
- Tool-uri
- Avantaje şi dezavantaje

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- Ideea de bază
  - SBR (sisteme expert) în care faptele şi regulile au asociate câte un factor de certitudine (FC)/coeficienţ de încredere
  - Un fel de sisteme de tip Bayes în care probabilităţile sunt înlocuite cu factori de certitudine

- Dacă A şi B atunci C
- Dacă C şi D atunci F

SBR classic

- Dacă A şi B atunci C [cu prob p_1]
- Dacă C şi D atunci F [cu prob p_2]

SBR de tip Bayes

- Dacă A şi B atunci C [FC_1]
- Dacă C şi $D_{[FC2]}$ atunci F [FC_3]

SBR cu FC

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC măsoară încrederea acordată unor fapte sau reguli
- Utilizarea FC → alternativă la actualizarea de tip Bayes

- FC pot fi aplicaţi
  - faptelor
  - regulilor (concluziei/concluzilor unei reguli)
  - fapte + reguli

- Într-un SBR (sistem expert) cu factori de certitudine
  - regulile sunt de forma:
    - dacă dovada atunci ipoteza [FC]
    - dacă $\text{dovada}_{[FC]}$ atunci ipoteza
    - dacă $\text{dovada}_{[FC]}$ atunci ipoteza [FC]
  - ipotezele susţinute de probe sunt independente

Conţinut şi arhitectură

- FC – mod de calcul
  - Măsura încrederii (measure of belief – MB)
    - măsura creşterii încrederii în ipoteza I pe baza dovezii D

$$MB(I,D)=\begin{cases}1, & \text{dacă } p(I)=1\\ \dfrac{p(I\mid D)-p(I)}{1-p(I)} & \text{dacă } p(I)<1\end{cases}$$

  - Măsura neîncrederii (measure of disbelief – MD)
    - măsura creşterii neîncrederii în ipoteza I pe baza dovezii D

$$MD(I,D)=\begin{cases}1, & \text{dacă } p(I)=0\\ \dfrac{p(I)-p(I\mid D)}{p(I)} & \text{dacă } p(I)>0\end{cases}$$

  - Pentru evitarea valorilor negative ale MB şi MD:

$$MB(I,D)=\begin{cases}1, & \text{dacă } p(I)=1\\ \dfrac{\max\{p(I\mid D),p(I)\}-p(I)}{1-p(I)} & \text{dacă } p(I)<1\end{cases}$$

$$MD(I,D)=\begin{cases}1, & \text{dacă } p(I)=0\\ \dfrac{\min\{p(I\mid D),p(I)\}-p(I)}{0-p(I)} & \text{dacă } p(I)>0\end{cases}$$

  - FC – încrederea în ipoteza *I* dată fiind dovada *D*
    - Număr din [-1,1]
    - FC=-1 dacă se ştie că ipoteza *I* este falsă
    - FC=0 dacă nu se ştie nimic despre ipoteza *I*
    - FC=1 dacă se ştie că ipoteza *I* este adevărată

$$FC(I,D)=\frac{MB(I,D)-MD(I,D)}{1-\min\{MB(I,D),MD(I,D)\}}$$

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul
  - încrederea în ipoteza I dată fiind dovada D
    - FC=-1 dacă se ştie că ipoteza este falsă
    - FC=0 dacă nu se ştie nimic despre ipoteză
    - FC=1 dacă se ştie că ipoteza este adevărată

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul
  - încrederea în ipoteza *I* dată fiind dovada *D*
  - ipoteza *I* poate fi:
    - simplă (ex. *Dacă D atunci I*)
    - compusă (ex. *Dacă D atunci* I_1 *şi* I_2 *şi ...* I_n)
  - dovada *D* poate fi
    - dpdv al compoziţiei:
      - simplă (ex. Dacă D atunci I)
      - compusă (ex. Dacă D1 şi D2 şi ... Dn atunci I)
    - dpdv al incertitudinii (încrederii în dovadă):
      - sigură (ex. Dacă D atunci I)
      - nesigură (ex. Dacă D[FC] atunci I)

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul pentru combinarea încrederii
  - o dovadă incertă care susţine sigur o ipoteză
  - mai multe dovezi incerte care susţin sigur o singură ipoteză
  - o dovadă incertă care susţine incert o ipoteză
  - mai multe dovezi incerte care susţin incert o ipoteză

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul pentru combinarea încrederii
  - O dovadă incertă care susţine sigur o ipoteză

$$FC(I) = \begin{cases} FC(D), & \text{dacă } FC(D) > 0 \\ 0, & \text{altfel} \end{cases}$$

  - Exemplul 1
    - R_1: *Dacă* $A_{[FC=0.9]}$ *atunci B*
    - R_2: *Dacă B atunci C*

    - FC(B)=FC(A)=0.9
    - FC(C)=FC(B)=0.9

  - Exemplul 2
    - R_1: *Dacă* $E_{[FC=-0.2]}$ *atunci F*

    - FC(E este adevărat) = -0.2 ➔ dovadă negativă ➔ nu putem spune nimic despre faptul că E este adevărat ➔ nu se poate spune nimic despre F

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul pentru combinarea încrederii
  - Mai multe dovezi incerte care susţin sigur o singură ipoteză
    - Dovezi (probe) adunate incremental
    - Mai multe reguli care, pe baza unor dovezi diferite, furnizează aceeaşi concluzi
    - Aceeaşi ipoteză (valoare de atribut) I este obţinută pe două căi de deducţie distincte, cu două perechi diferite de valori pentru *FC, FC[I,D$_1$] si FC[I,D$_2$]*
    - Cele doua cai de deductie distincte, corespunzatoare dovezilor (probelor) D_1 şi D_2 pot fi:
      - ramuri diferite ale arborelui de cautare generat prin aplicarea regulilor
      - dovezi (probe) indicate explicit sistemului

$$FC(I, D_1 \wedge D_2) = \begin{cases} CF(D_1) + CF(D_2)(1 - CF(D_1), & \text{dacă } CF(D_1),\ CF(D_2) > 0 \\ CF(D_1) + CF(D_2)(1 + CF(D_1), & \text{dacă } CF(D_1),\ CF(D_2) < 0 \\ \dfrac{\mathrm{CF}(D_1) + \mathrm{CF}(D_2)}{1 - \min\{|CF(D_1|, |CF(D_2)|\}}, & \text{dacă } sign(CF(D_1)) \neq sign(CF(D_2)) \end{cases}$$

  - Exemplu
    - R_1: Dacă $D_{1\ [FC=0.6]}$ atunci I
    - R_2: Dacă $D_{2[FC=-0.3]}$ atunci I
    - $FC(I,D_1 \wedge D_2)=(0.6+(-0.3))/(1-0.3)=0.42$

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul pentru combinarea încrederii
  - O dovadă incertă care susţine incert o ipoteză

$$FC(I) = \begin{cases} FC(D) * FC(\textit{regulă}), & \text{dacă } FC(D) > 0 \\ 0, & \text{altfel} \end{cases}$$

  - Exemplul 1
    - R_1: *Dacă* $A_{[FC=0.9]}$ *atunci B [FC=0.4]*
    - R_2: *Dacă B atunci C [FC=0.3]*

    - FC(B)=FC(A)*FC(R_1)=0.9*0.4=0.36
    - FC(C)=FC(B)*FC(R_2)=0.36*0.3=0.108

  - Exemplul 2
    - R_1: *Dacă* $E_{[FC=-0.2]}$ *atunci F [FC=0.6]*

    - FC(E este adevărat) = -0.2 ➔ dovadă negativă ➔ nu putem spune nimic despre faptul că E este adevărat ➔ nu se poate spune nimic despre F

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul pentru combinarea încrederii
  - Mai multe dovezi incerte care susţin incert o ipoteză
    - Dovezile sunt legate prin ŞI logic

$$CF(I) = \begin{cases} \min\{CF(D_1), CF(D_2), ..., CF(D_n),\} * CF(regulă), & dacă\ \mathrm{CF(D_i)} > 0, \mathrm{i} = 1,2,...,\mathrm{n} \\ 0, & \text{altfel} \end{cases}$$

  - Una sau mai multe dintre dovezile incerte care susţin incert o ipoteză
    - Dovezile sunt legate prin SAU logic

$$CF(I) = \begin{cases} \max\{CF(D_1), CF(D_2), ..., CF(D_n),\} * CF(regulă), & dacă\ \mathrm{CF(D_i)} > 0, \mathrm{i} = 1,2,...,\mathrm{n} \\ 0, & \text{altfel} \end{cases}$$

  - Exemplul 1
    - R_1: *Dacă* $D_{1[FC = 0.8]}$ *şi* $D_{2[FC = 0.7]}$ *şi* $D_{3[FC = 0.5]}$ *şi* $D_{4[FC = 0.3]}$ *şi* $D_{5[FC = 0.9]}$ *atunci I [FC = 0.65]*
    - *FC(I) = 0.3 * 0.65 = 0.195*

Sisteme inteligente – SBC – factori de certitudine

Conţinut şi arhitectură

- FC – mod de calcul pentru combinarea încrederii
  - Mai multe dovezi incerte care susţin incert o ipoteză
    - Dovezile sunt legate prin ŞI logic

$$CF(I) = \begin{cases} \min\{CF(D_1), CF(D_2), ..., CF(D_n),\} * CF(regulă), & dacă\ \mathrm{CF}(\mathrm{D_i}) > 0, \mathrm{i} = 1,2,...,\mathrm{n} \\ 0, & \text{altfel} \end{cases}$$

  - Una sau mai multe dintre dovezile incerte susţin incert o ipoteză
    - Dovezile sunt legate prin SAU logic

$$CF(I) = \begin{cases} \max\{CF(D_1), CF(D_2), ..., CF(D_n),\} * CF(regulă), & dacă\ \mathrm{CF}(\mathrm{D_i}) > 0, \mathrm{i} = 1,2,...,\mathrm{n} \\ 0, & \text{altfel} \end{cases}$$

  - Exemplul 2
    - R_1: *Dacă* $D_{1[FC = 0.8]}$ *sau* $D_{2[FC = 0.7]}$ *sau* $D_{3[FC = 0.5]}$ *sau* $D_{4[FC = 0.3]}$ *sau* $D_{5[FC = 0.9]}$ *atunci I [FC = 0.65]*
    - *FC(I) = 0.9 * 0.65 = 0.585*

Sisteme inteligente – SBC – factori de certitudine

Exemplu

- Sistem expert pentru diagnosticarea unei răceli
  - Fapte în baza de date:
    - Febra pacientului 37.4
    - Pacientul tuşeşte de mai puţin de 24 ore
    - Pacientul nu are expectoraţii
    - Pacientul are o durere de cap cu FC = 0.4
    - Pacientul are nasul înfundat cu FC = 0.5
  - Reguli:
    - R_1: Dacă *A: febra < 37.5* atunci *B: simptomele de răceală sunt prezente* [FC=0.5]
    - R_2: Dacă *C: febra > 37.5* atunci *B: simptomele de răceală sunt prezente* [FC=0.9]
    - R_3: Dacă *D: tuşeşte > 24 ore* atunci *E: durerea de gât e prezentă* [FC=0.5]
    - R_4: Dacă *F: tuşeşte > 48 ore* atunci *E: durerea de gât e prezentă* [FC=1.0]
    - R_5: Dacă *B: are simptome de răceală* şi *G: nu expectorează* atunci *H: a răcit* [FC=-0.2]
    - R_6: Dacă *E: îl doare gâtul* atunci *H: a răcit* [FC=0.5]
    - R_7: Dacă *I: îl doare capul* şi *J: are nasul înfundat* atunci *H: a răcit* [FC=0.7]
  - Concluzia:
    - Pacientul este sau nu răcit?

Sisteme inteligente – SBC – factori de certitudine

Exemplu

- Sistem expert pentru diagnosticarea unei răceli
  - Fapte în baza de date:
    - Febra pacientului 37.4
    - Pacientul tuşeşte de mai puţin de 24 ore
    - Pacientul nu are expectoraţii
    - Pacientul are o durere de cap cu FC = 0.4
    - Pacientul are nasul înfundat cu FC = 0.5
  - Reguli:
    - R_1: Dacă *A: febra < 37.5* atunci *B: simptomele de răceală sunt prezente* [FC=0.5]
    - R_2: Dacă *C: febra > 37.5* atunci *B: simptomele de răceală sunt prezente* [FC=0.9]
    - R_3: Dacă *D: tuşeşte > 24 ore* atunci *E: durerea de gât e prezentă* [FC=0.5]
    - R_4: Dacă *F: tuşeşte > 48 ore* atunci *E: durerea de gât e prezentă* [FC=1.0]
    - R_5: Dacă *B: are simptome de răceală* şi *G: nu expectorează* atunci *H: a răcit* [FC=-0.2]
    - R_6: Dacă *E: îl doare gâtul* atunci *H: a răcit* [FC=0.5]
    - R_7: Dacă *I: îl doare capul* şi *J: are nasul înfundat* atunci *H: a răcit* [FC=0.7]
  - Concluzia:
    - Pacientul este sau nu răcit?

Sisteme inteligente – SBC – factori de certitudine

- Avantaje
  - Nu este necesar calculul apriori a probabilităţilor
- Limite
  - ipotezele sustinute de probe sunt independente.
  - exemplu:
    - Fie următoarele fapte:
      - A: Aspersorul a funcţionat noaptea trecută
      - U: Iarba este udă dimineaţă
      - P: Noaptea trecută a plouat.
    - şi următoarele două reguli care leagă între ele aceste fapte:
      - R_1: dacă aspersorul a funcţionat noaptea trecută atunci există o încredere puternică (0.9) că iarba este udă dimineaţa
      - R_2: dacă iarba este udă dimineaţa atunci există o încredere puternică (0.8) că noaptea trecută a plouat
    - Deci:
      - FC[U,A] = 0.9 - deci proba aspersor sustine iarba uda cu 0.9
      - FC[P,U] = 0.8 - deci iarba uda sustine ploaie cu 0.8
      - FC[P,A] = 0.8 * 0.9 = 0.72 - deci aspersorul sustine ploaia cu 0.72

Sisteme inteligente – SBC – factori de certitudine

- SBR de tip Bayes *vs.* SBR cu FC

Bayes	FC
Teorie probabilităţilor este veche şi fundamentată matematic	Teoria FC este nouă şi fără demostrţii matematice
Necesită existenţa unor informaţii statistice	Nu necesită existenţa unor date statistice
Propagarea încrederii creşte în timp exponenţial	Informaţia circulă repede şi eficient în SBR

SBC

- Tehnici de raţionare în medii nesigure
  - Teoria Bayesiana – metodă probabilistică
  - Teoria certitudinii
  - **Teoria posibilităţii (logica fuzzy)**

Metode euristice

Sisteme inteligente – SBC – sisteme fuzzy

- Teoria posibilităţii
- Conţinut şi arhitectură
- Tipologie
- Tool-uri
- Avantaje şi limite

Sisteme inteligente – SBC – sisteme fuzzy

Teoria posibilităţii (logica fuzzy)

- De ce fuzzy?
  - Problemă: transpuneţi în cod (C++) următoarele propoziţii:
    - Georgel este înalt.
    - Afara este frig.
- Când este important fuzzy?
  - Interogări în limbaj natural
  - Reprezentarea cunoştinţelor în sisteme expert
  - Controlul fuzzy – când se lucrează cu fenomene imprecise (perturbate de zgomot)

Sisteme inteligente – SBC – sisteme fuzzy

Amintim componenţa unui SBC

- Baza de cunoştinţe (BC) → Modalităţi de reprezentare a cunoştinţelor
  - Logica formală (limbaje formale)
    - Definiţie
      - Ştiinţa principiilor formale de raţionament
    - Componente
      - Sintaxă
      - Semantică
      - Metodă de inferenţă sintactică
    - Tipologie
      - În funcţie de numărul valorilor de adevăr:
        - logică dual
        - **logică polivalentă**
      - În funcţie de tipul elementelor de bază:
        - clasică → primitivele = propoziţii (predicate)
        - probabilistică → primitivele = variabile aleatoare
      - În funcţie de obiectul de lucru:
        - logica propoziţională → se lucrează doar cu propoziţii declarative, iar obiectele descrise sunt fixe sau unice (Ionică este student)
        - logica predicatelor de ordin I → se lucrează cu propziţii declarative, cu predicate şi cuantificări , iar obiectele descrise pot fi unice sau variabile asociate unui obiect unic (Toţi studenţii sunt prezenţi)
  - Reguli
  - Reţele semantice
- Modulul de control (MC – pentru inferenţă)

Sisteme inteligente – SBC – sisteme fuzzy

Teoria posibilităţii - Un pic de istorie

- Parminedes (400 B.C.)
- Aristotle
  - "Law of the Excluded Middle" – fiecare propoziţie trebuie să fie Adevărată sau Falsă
- Plato
  - O a treia regiune între Adevărat şi Fals
  - Pune bazele logicii fuzzy
- Lukasiewicz (1900)
  - Propune o alternativă sistematică la logica bivalentă a lui Aristotle – logica trivalentă: Adevărat, Fals, Posibil
- Lotfi A. Zadeh (1965)
  - Descrie matematic teoria mulţimilor fuzzy şi logica fuzzy: funcţia de apartenenţă (valorile Adevărat şi Fals) operează pe intervalul [0,1]
    - A propus noi operaţii de calcul pt logica fuzzy
    - A considerat logica fuzzy o generalizare a logicii clasice
  - A publicat primul articol despre mulţimile fuzzy

Sisteme inteligente – SBC – sisteme fuzzy

Teoria posibilităţii

- Logica fuzzy
  - Generalizare a logicii Boolene
  - Manipulează conceptul de adevăr parţial
    - Logica clasică – totul este exprimat în termeni binari
      - 0 sau 1, alb sau negru, da sau nu
    - Logica fuzzy – exprimarea graduală a unui adevăr
      - Valori între 0 şi 1

- Analogia *logică vs. algebră*
  - Operatorii logici exprimaţi în termeni matematici (George Boole):
    - Conjunţie = minimum ➔ $a \wedge b = \min(a, b)$
    - Disjuncţie = maximum ➔ $a \vee b = \max(a, b)$
    - Negaţie = scădere ➔ $\neg a = 1 - a$

Sisteme inteligente – SBC – sisteme fuzzy

Reamintim: SBR – arhitectură

- baza de cunoştinţe
  - Conţine
    - Informaţiile specifice despre un domeniu sub forma unor
      - fapte – afirmaţii corecte
      - reguli - euristici speciale care generează informaţii (cunoştinţe)
  - Rol
    - stocarea tuturor elementelor cunoașterii (fapte, reguli, metode de rezolvare, euristici) specifice domeniului de aplicație, preluate de la experții umani sau din alte surse

- modulul de control
  - Conţinut
    - regulile prin care se pot obţine informaţii noi
    - algoritmi independenţi de domeniu
    - creierul SBR – un algoritm de deducere bazat pe BC şi specific metodei de raţionare
      - un program în care s-a implementat cunoașterea de control, procedurală sau operatorie, cu ajutorul căruia se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii.
    - depinde de complexitate şi tipul cunoştinţelor cu care are de-a face
  - Rol
    - cu ajutorul lui se exploatează baza de cunoștințe pentru efectuarea de raționamente în vederea obținerii de soluții, recomandări sau concluzii

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Ideea de bază
  - Cf. teoriei informaţiilor certe
    - *Popescu este tânăr*
  - Cf. teoriei informaţiilor incerte
    - Cf. teoriei probabilităţilor:
      - *Există 80% şanse ca Popescu să fie tânăr*
    - Cf. logicii fuzzy:
      - *Gradul de apartenenţă al lui Popescu la grupul de oameni tineri este 0.80*

- Necesitate
  - Abordarea fenomenelor reale implică utilizarea mulţimilor fuzzy
  - Exemplu
    - *Temperatura unei camere poate fi:*
      - *joasă,*
      - *medie sau*
      - *ridicată*

    - Aceste mulţimi de temperaturi posibile se pot suprapune
      - o temperatura poate aparţine uneia sau mai multor mulţimi în funcţie de cel care face evaluarea

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către expert
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
      - Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă
  - Construirea unei baze de reguli de către expert
    - Matricea de decizie a bazei de cunoştinţe
  - Evaluarea regulilor
    - Inferenţa – transformarea intrărilor fuzzy în ieşiri fuzzy prin aplicarea regulilor din baza de cunoştinţe
  - Agregarea rezultatelor
  - Defuzificarea
  - Interpretarea rezultatelor

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către expert
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
      - Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă
  - **Construirea unei baze de reguli de către expert**
    - **Matricea de decizie a bazei de cunoştinţe**
  - Evaluarea regulilor
    - Inferenţa – transformarea intrărilor fuzzy în ieşiri fuzzy prin aplicarea regulilor din baza de cunoştinţe
  - Agregarea rezultatelor
  - Defuzificarea
  - Interpretarea rezultatelor

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy)
  - Fapte (mulţimi) fuzzy
    - Definire
    - Reprezentare
    - Operaţii – complement, containment, intersecţie, reuniune, egalitate, produs algebric, sumă algebrică
    - Proprietăţi – asociativitate, distributivitate, comutativitate, tranzitivitate, idempotenţi, identitate, involuţie, legile lui De Morgan
    - Hedges
  - Variabile fuzzy
    - Definiţie
    - Proprietăţi
- Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy → **definire**
  - Definirea oricărei mulţimi – 2 moduri:
    - Prin enumerarea elementelor
      - Ex. *Mulţimea elevilor = {Ana, Maria, Ioana}*
    - Prin specificare unei proprietăţi a elementelor mulţimii
      - Ex. *Mulţimea numerelor pare ={x| x = 2n, unde n – număr natural}*
  - Funcţia caracteristică *μ* a unei mulţimi
    - Fie X o mulţime universală şi x un element al mulţimii (xϵX)
    - Logica clasică
      - Fie R o submulţime a lui X: R⊂X, R - mulţime regulară
      - Orice element x aparţine sau nu mulţimii R
      - μ_R : X → {0, 1}, unde $\mu_R(x) = \begin{cases} 1, & x \in R \\ 0, & x \notin R \end{cases}$
    - Logica fuzzy
      - Fie F o submulţime a lui X (univers de discurs): F⊂X, F - mulţime fuzzy
      - orice element x aparţine mulţimii F într-un anumit grad $\mu_F(x)$
      - μ_F : X → [0, 1], $\mu_F(x)=g$, unde gϵ [0,1] – gradul de aparteneţă al lui x la F
      - g = 0 ➔ ne-aparteneţă
      - g = 1 ➔ apartenenţă
      - O mulţime fuzzy = o pereche (F, μ_F), unde $\mu_F(x) = \begin{cases} 1, & \text{dacă } x \text{ este total în } F \\ 0, & \text{dacă } x \text{ nu este în } F \\ \in (0,1) & \text{dacă } x \text{ este parte din } F\,(x \text{ este numar fuzzy}) \end{cases}$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy → **definire**
  - Exemplul 1
    - X – mulţimea numerelor naturale mai mici decât 11
    - R – mulţimea numerelor mai mici decât 7
    - F – mulţimea numerelor aproapiate de 6

x	$\mu_R(x)$	$\mu_F(x)$
0	1	0
1	1	0.1
2	1	0.25
3	1	0.5
4	1	0.6
5	1	0.8
6	1	1
7	0	0.8
8	0	0.6
9	0	0.5
10	0	0.25

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy → **definire**
  - Exemplul 2
    - O anumită temperatură *t* poate avea 3 valori de adevăr:
      - Roşu (0): nu e fierbinte
      - Portocalui (0.3): puţin cald
      - Albastru (0.7): aproape rece

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy → **reprezentare**
  - Mulţimile regulare
    - Delimitări clare (exacte) → diagrame Venn
  - Mulţimile fuzzy
    - Delimitări graduale → reprezentări bazate pe funcţia de apartenenţă
      - Singulară
        - *μ(x)* = *s*, unde *s* este un scalar
      - Triunghiulară
        - $\mu(x) = \max\left\{0, \min\left\{\frac{x-a}{b-a}, 1, \frac{c-x}{c-b}\right\}\right\}$
      - Trapezoidală
        - $\mu(x) = S(x) = \max\left\{0, \min\left\{\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right\}\right\}$
      - Funcţia Z
        - *μ(x) = 1- S(x)*
      - Funcţia Π
        - $\mu(x) = \Pi(x) = \begin{cases} S(x), & \text{dacă } x \le c \\ Z(x), & \text{dacă } x > c \end{cases}$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy → **reprezentare**
  - Exemplu
    - *Vârsta unei persoane*

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy →**Operaţii**
  - complement
  - incluziune
  - intersecţie
  - reuniune
  - egalitate
  - produs algebric
  - sumă algebrică

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy →**Operaţii**
  - Complement
    - X - un univers
    - A – o mulţimea fuzzy (cu universul X)
    - B – o mulţimea fuzzy (cu universul X)
    - B este complementul lui A (B= ┐A) dacă:
      - $\mu_B(x)=\mu_{\neg A}(x)=1-\mu_A(x)$ pentru orice $x \in X$
    - Exemplu:
      - *persoane bătrâne (pe baza vârstei)*
        - A={(30,0), (40, 0.2), (50, 0.4), (60, 0.6), (70, 0.8), (80, 1)}
      - *persoane tinere (care nu sunt bătrâne) (pe baza vârstei)*
        - ┐A={(30,1), (40, 0.8), (50, 0.6), (60, 0.4), (70, 0.2), (80, 0)}

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy →**Operaţii**
  - Incluziune
    - X - un univers
    - A – o mulţimea fuzzy (cu universul X)
    - B – o mulţimea fuzzy (cu universul X)
    - B este submuţime a lui A ($B \subset A$) dacă:
      - $\mu_B(x) \leq \mu_A(x)$ pentru orice $x \in X$
    - Exemplu
      - *persoane bătrâne (pe baza vârstei)*
        - A={(60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)}
      - *persoane foarte bătrâne (pe baza vârstei)*
        - B={(60, 0.6), (65, 0.67) (70, 0.8), (75, 0.8), (80, 0.95)}

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy →**Operaţii**
  - Intersecţie
    - X - un univers
    - A – o mulţimea fuzzy (cu universul X)
    - B – o mulţimea fuzzy (cu universul X)
    - C – o mulţimea fuzzy (cu universul X)
    - C este intersecţia lui A cu B (C=A∩B) dacă:
      - $\mu_C(x)=\mu_{A\cap B}(x)=\min\{\mu_A(x), \mu_B(x)\}=\mu_A(x)\cap\mu_B(x)$ pentru orice $x\in X$

    - Exemplu
      - *persoane bătrâne (pe baza vârstei)*
        - A={(30,0) (40, 0.1) (50, 0.2) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)}
      - *persoane de vârstă medie*
        - B={(30,0.1) (40, 0.2) (50, 0.6) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)}
      - *persoane bătrâne şi de vârstă medie*
        - C={(30,0) (40, 0.1) (50, 0.2) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)}

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy →**Operaţii**
  - Reuniune
    - X - un univers
    - A – o mulţimea fuzzy (cu universul X)
    - B – o mulţimea fuzzy (cu universul X)
    - C – o mulţimea fuzzy (cu universul X)
    - C este reuniunea lui A cu B ($C=A\cup B$) dacă:
      - $\mu_C(x)=\mu_{A\cup B}(x)=\max\{\mu_A(x), \mu_B(x)\}=\mu_A(x) \cup \mu_B(x)$ pentru orice $x\in X$

    - Exemplu
      - *persoane bătrâne (pe baza vârstei)*
        - A={(30,0) (40, 0.1) (50, 0.2) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)}
      - *persoane de vârstă medie*
        - B={(30,0.1) (40, 0.2) (50, 0.6) (60, 0.5), (65, 0.2) (70, 0.1), (75, 0), (80, 0)}
      - *persoane bătrâne sau de vârstă medie*
        - C={(30,0.1) (40, 0.2) (50, 0.6) (60, 0.6), (65, 0.7) (70, 0.8), (75, 0.9), (80, 1)}

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy →**Operaţii**
  - Egalitate, produs şi sumă algebrică
    - X - un univers
    - A – o mulţimea fuzzy (cu universul X)
    - B – o mulţimea fuzzy (cu universul X)
    - C – o mulţimea fuzzy (cu universul X)

    - B este egal cu A (B=A) dacă:
      - $\mu_B(x)=\mu_A(x)$ pentru orice $x \in X$
    - C este produsul dintre A şi B (C=A*B) dacă:
      - $\mu C(x)=\mu A*B(x)=\mu A(x)*\mu B(x)$ pentru orice $x \in X$
    - C este suma lui A cu B (C=A+B) dacă:
      - $\mu C(x)=\mu A+B(x)=\mu A(x)+\mu B(x)$ pentru orice $x \in X$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy → **proprietăţi**
  - Asociativitate
  - Comutativitate
  - Distributivitate
  - Tranzitivitate
  - Idempotenţă
  - Identitate
  - Involuţie

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Fapte (mulţimi) fuzzy →**Hedges (nuanţatori)**
  - Ideea de bază
    - Modificatori, adjective sau adverbe care schimbă valorile de adevăr ale unor afirmaţii
      - Ex. *Foarte, mai puţin, mai mult, aproape*, etc.
    - Modifică forma mulţimilor fuzzy
    - Pot acţiona la nivel de
      - Numere fuzzy
      - Valori de adevăr
      - Funcţii de apartenenţă
    - Euristici
  - Utilitate
    - Apropierea de limbajul natural → subiectivism
    - Evaluarea variabilelor lingvistice

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) →

Fapte (mulţimi) fuzzy →***Hedges* (nuanţatori)**

- Tipologie
  - *Hedge*-uri care reduc valoarea de adevăr (producând o concentraţie)
    - *foarte* $\mu_{A_foarte}(x)=(\mu_A(x))^2$
    - *extrem* $\mu_{A_extrem}(x)=(\mu_A(x))^3$
    - *foarte foarte* $\mu_{A_foarte_foarte}(x)=(\mu_{A_foarte}(x))^2=(\mu_A(x))^4$
  - *Hedge*-uri care măresc valoarea de adevăr (producând o dilatare)
    - *oarecum* $\mu_{A_oarecum}(x)=(\mu A(x))^{1/2}$
    - *uşor* $\mu_{A_uşor}(x)=(\mu A(x))^{1/3}$
  - *Hedge*-uri care intensifică valoarea de adevă
    - *întradevăr*

$$\mu_{A_\text{întradevăr}}(x)=\begin{cases} 2(\mu_A(x))^2, & \text{dacă } 0\le\mu_A(x)\le 0.5 \\ 1-2(1-\mu_A(x))^2, & \text{dacă } 0.5\le\mu_A(x)\le 1 \end{cases}$$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy)
  - Fapte (mulţimi) fuzzy
    - Definire
    - Reprezentare
    - Operaţii – complement, containment, intersecţie, reuniune, egalitate, produs algebric, sumă algebrică
    - Proprietăţi – asociativitate, distributivitate, comutativitate, tranzitivitate, idempotenţi, identitate, involuţie, legile lui De Morgan
    - Hedges
  - **Variabile fuzzy**
    - **Definiţie**
    - **Proprietăţi**
- Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Variabile fuzzy → **Definiţie**
  - O variabilă fuzzy V este definită de quadruplul *V = {x, l, u, m}*, unde:
    - X – numele variabilei simbolice
    - L – mulţimea etichetelor posibile pentru variabila x
    - U – universul variabilei
    - M – regulile semantice care definesc înţelesul fiecărei etichete din L (funcţiile de apartenenţă)
  - Funcţia de apartenenţă
    - Evaluare subiectivă
      - forma funcţiilor este definită de experţi/specialişti
    - Evaluare ad-hoc
      - funcţii simple care să poată rezolva problema
    - Evaluare bazată pe distribuţii şi probabilităţi ale informaţilor extrase din măsurători
    - Evaluare adaptată
      - prin teste
    - Evaluare automată
      - algoritmi folosiţi pentru definirea funcţiilor pe baza unor date de antrenament
  - Exemplu
    - X = Temperatura
    - L = {joasă, medie, ridicată}
    - U = {x∈X | -70° ≤ x ≤+70°}
    - M = →

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Variabile fuzzy → **Proprietăţi**
  - Completitudine
    - O variabilă fuzzy V este completă dacă pentru orice $x \in X$ există o mulţime fuzzy A astfel încât $\mu_A(x)>0$

Completă

Incompletă

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Variabile fuzzy → **Proprietăţi**
  - Partiţionare a unităţii
    - O variabilă fuzzy V formează o partiţie a unităţii dacă pentru orice valoare de intrare x avem $\sum_{i=1}^{p} \mu_{A_i}(x) = 1$
    - Unde p este numărul mulţimilor cărora aparţine x
    - Nu există reguli pentru definirea suprapunerii a 2 mulţimi vecine
      - De obicei, suprapunerea trebuie să fie între 25% şi 50%

Partiţionare a unităţii

Nepartiţionare a unităţii

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy) → Variabile fuzzy → **Proprietăţi**
  - Partiţionare a unităţii
    - O variabilă fuzzy completă poate fi transformată într-o partiţionare a unităţii astfel:

$$\mu_{\hat{A}_i}(x) = \frac{\mu_{A_i}(x)}{\sum_{j=1}^{p} \mu_{A_j}(x)} \; for \; i = 1, \ldots, p$$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Elemente de teoria posibilităţii (logica fuzzy)
  - Fapte (mulţimi) fuzzy
    - Definire
    - Reprezentare
    - Operaţii – complement, containment, intersecţie, reuniune, egalitate, produs algebric, sumă algebrică
    - Proprietăţi – asociativitate, distributivitate, comutativitate, tranzitivitate, idempotenţi, identitate, involuţie, legile lui De Morgan
    - Hedges
  - Variabile fuzzy
    - Definiţie
    - Proprietăţi
- **Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă → mecanism**

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Mecanism
  - Se stabilesc datele (de intrare şi ieşire) brute ale sistemului
  - Pentru fiecare dată de intrare se definesc funcţiile de apartenenţă (2, 3 sau mai multe)
    - Fiecărei funcţii de apartenenţă i se asociază o etichetă calitativă – variabilă lingvistică
    - unei variabile brute îi pot corespunde una sau mai multe variabile lingvistice
    - Exemplu
      - Variabilă brută: temperatura T
      - Variabile lingvistice asociate: redusă → A1, medie→A2, înaltă→A3
  - Se transformă fiecare dată brută de intrare într-o variabilă lingvistică → fuzzificare
    - Se stabileşte mulţimea fuzzy din care face parte variabila brută
    - Cum?
      - Pentru o valoare brută dată a unei variabile (care poate aparţine uneia sau mai multor mulţimi fuzzy) se calculează valoarea asociată funcţiei caracteristice corespunzătoare pentru fiecare dintre mulţimile de apartenenţă (folosind funcţia de apartenenţă)
    - Exemplu
      - T ($=x_n$) =5°
      - A_1 → $\mu_{A1}(T) = 0.6$
      - A_2 → $\mu_{A2}(T) = 0.4$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → fuzzificarea datelor de intrare

- Mecanism
  - Exemplu pentru reglarea unui aparat de aer condiţionat
    - Intrări:
      - x (temperatura – rece, normal, fierbinte) şi
      - y (umiditatea – mică, mare)
    - Ieşiri:
      - z (puterea aparatului – redusă, medie, ridicată)
    - Date de intrare:
      - Temperatura x = 37
        - $\mu_{A1}(x)=0$, $\mu_{A2}(x)=0.6$, $\mu_{A3}(x)=0.3$
      - Umiditatea y = 0.8
        - $\mu_{B1}(x)=0.9$, $\mu_{B2}(x)=0$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către expert
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
      - Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă
  - **Construirea unei baze de reguli de către expert**
    - **Matricea de decizie a bazei de cunoştinţe**
  - Evaluarea regulilor
    - Inferenţa – transformarea intrărilor fuzzy în ieşiri fuzzy prin aplicarea regulilor din baza de cunoştinţe
  - Agregarea rezultatelor
  - Defuzificarea
  - Interpretarea rezultatelor

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Construirea unei baze de reguli de către expert

- Reguli
  - Definiţie
    - Construcţii lingvistice de tipul
      - Afirmaţiilor: A
      - Condiţionărilor: dacă A atunci B
    - Unde A şi B sunt (colecţii de) propoziţii care conţin variabile lingvistice
      - A – premisa regulii
      - B – consecinţa regulii
  - Tipologie
    - Necondiţionale
      - x este (în) A_i
      - ex. *Salvează energia*
    - Condiţionale
      - Dacă x este (în) A_i atunci y este (în) C_k
      - Dacă x este (în) A_i şi y este (în) B_j atunci z este (în) C_k
      - Dacă x este (în) A_i sau y este (în) B_j atunci z este (în) C_k

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Construirea unei baze de reguli de către expert

- Reguli
  - Exemplu

	Reguli în logica clasică	Reguli în logica fuzzy
R_1	*Dacă temperatura este -5 atunci vremea este rece*	*Dacă temperatura este joasă atunci vremea este rece*
R_2	*Dacă temperatura este 15 atunci vremea este călduţă*	*Dacă temperatura este medie atunci vremea este călduţă*
R_3	*Dacă temperatura este 35 atunci vremea este caniculară*	*Dacă temperatura este ridicată atunci vremea este caniculară*

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Construirea unei baze de reguli de către expert

- Reguli
  - Baza de reguli fuzzy
    - R_{11}: Dacă x este A_1 şi y este B_1 atunci z este C_u
    - R_{12}: Dacă x este A_1 şi y este B_2 atunci z este C_v
    - ...
    - R_{1n}: Dacă x este A_1 şi y este B_n atunci z este C_x

    - R_{21}: Dacă x este A_2 şi y este B_1 atunci z este C_y
    - R_{22}: Dacă x este A_2 şi y este B_2 atunci z este C_z
    - ...
    - R_{2n}: Dacă x este A_2 şi y este B_n atunci z este C_v

    - ...

    - R_{m1}: Dacă x este A_m şi y este B_1 atunci z este C_z
    - R_{m2}: Dacă x este A_m şi y este B_2 atunci z este C_v
    - ...
    - R_{mn}: Dacă x este A_m şi y este B_n atunci z este C_u

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Construirea unei baze de reguli de către expert

- Reguli
  - Proprietăţi ale bazei de reguli
    - Completitudine
      - O bază de reguli fuzzy este completă
        - dacă orice valoare de intrare are asociată o valaore între 0 şi 1
        - dacă orice variabilă fuzzy este completă
        - dacă mulţimile folosite in bază au suport ne-compact
    - Consistenţă
      - O mulţime de reguli fuzzy este incosistentă dacă 2 reguli care au aceleaşi premise conduc la consecinţe diferite
        - Dacă x este A_1 şi y este B_1 atunci z este C_r
        - Dacă x este A_1 şi y este B_1 atunci z este C_v
  - Probleme cu baza de reguli fuzzy
    - Explozia regulilor
      - Numărul regulilor creşte exponenţial odată cu numărul variabilelor de intrare
      - Numărul de combinaţii ale mulţimilor de intrare este
        - unde a *i*-a variabilă este formată din p_i mulţimi $P = \prod_{i=1}^{n} p_i$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Construirea unei baze de reguli de către expert

- Matricea de decizie a bazei de cunoştinţe
  - Exemplu pentru reglarea unui aparat de aer condiţionat
    - Intrări:
      - x (temperatura – rece, normal, fierbinte) şi
      - y (umiditatea – mică, mare)
    - Ieşiri:
      - z (puterea aparatului – redusă, menţinută constantă, mărită)
    - Reguli:
      - *Dacă temperatura este normală şi umiditatea mică atunci puterea aparatului de aer condiţionat trebuie să fie menţinută constantă*

		Data de intrare y	
		Mică	Mare
Data de intrare x	Rece	Redusă	Constantă
	Normal	Constantă	Mărită
	Fierbinte	Mărită	Mărită

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către expert
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
      - Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă
  - Construirea unei baze de reguli de către expert
    - Matricea de decizie a bazei de cunoştinţe
  - **Evaluarea regulilor**
    - **Inferenţa – transformarea intrărilor fuzzy în ieşiri fuzzy prin aplicarea regulilor din baza de cunoştinţe**
  - Agregarea rezultatelor
  - Defuzificarea
  - Interpretarea rezultatelor

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor (inferenţa fuzzy)

- Care reguli se evaluează mai întâi?
  - Inferenţă de tip Fuzzy
    - Regulile se evaluează în **paralel**, fiecare regulă contribuind la forma rezultatului final.
    - Mulţimile fuzzy rezultate sunt defuzzificate **abia după ce toate regulile** au fost evaluate

  Amintim
  - Inferenţă înainte
    - Pentru o stare a problemei se colectează informaţiile necesare şi se aplică regulile posibile pentru BC construită
  - Inferenţă înapoi
    - Se identifică regulile care determină apariţia faptului obiectiv şi (dacă este posibil) se aplică aceste reguli

- Cum se evaluează regulile?
  - Evaluarea antecedentelor
  - Evaluarea consecinţelor

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor (inferenţa fuzzy)

- Evaluarea antecedentelor
  - Pentru fiecare premisă a unei reguli (dacă *x ieste (în) A*) se ia în considerare gradul de apartenenţă al datelor brute la fiecare mulţime fuzzy din care face parte (anterior calculat în etapa de fuzzificare)
  - O regulă poate avea una sau mai multe premise (antecedente) legate prin operatori logici: *AND, OR* sau *NOT* → Aplicarea operatorilor fuzzy în evaluarea regulilor
    - Operator *AND* → intersecţia (minimul) a 2 mulţimi
      - $\mu_{A \cap B}(x) = min\{\mu_A(x),\mu_B(x)\}$
    - Operator *OR* → reuniunea (maximul) a 2 mulţimi
      - $\mu_{A \cup B}(x) = max\{\mu_A(x),\mu_B(x)\}$
    - Operator *NOT* → negaţia (complementul) unei mulţimi
      - $\mu_{\neg A}(x) = 1-\mu_A(x)$
  - Rezultatul evaluării premisei unei reguli
    - Gradul în care antecedentul unei reguli este satisfăcut
    - Poate fi rezultatul unei operaţii de tip *AND* sau *OR*
    - Alte denumiri:
      - Rule's firing strength
      - Degree of fulfillment

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor (inferenţa fuzzy)

- Evaluarea consecinţelor (aplicarea metodei de implicare)
  - Determinarea rezultatelor
    - stabilirea gradului de apartenenţă la mulţimile fuzzy corespunzătoare a variabilelor existente în consecinţa regulilor
  - Fiecare regiune de ieşire va trebui defuzzificată pentru obţinerea valorilor crisp
  - În funcţie de tipul consecinţelor unei reguli
    - Modelul Mamdani – consecinţa regulii este de forma "variabila de ieşire face parte dintr-o mulţime fuzzy"
    - Modelul Sugeno – consecinţa regulii este de forma "variabila de ieşire este o funcţie crisp care depinde de intrări"
    - Modelul Tsukamoto – consecinţa regulii este de forma "variabila de ieşire face parte dintr-o mulţime fuzzy cu o funcţie de aparteneţă monotonă"

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor (inferenţa fuzzy) → evaluarea consecinţelor

- Modelul Mamdani
  - Ideea de bază:
    - consecinţa regulii este de forma "variabila de ieşire face parte dintr-o mulţime fuzzy"
    - Rezultatul evaluării premiselor se aplică pentru funcţia de apartenenţă a consecinţei
    - Exemplu
      - ***Dacă x este în A şi y este în B atunci z este în C***
  - Tipologie (în funcţie de modul de aplicare a rezultatului asupra funcţiei de apartenenţă a consecinţei)
    - Mulţimi fuzzy rezultat de tip *clipped*
    - Mulţimi fuzzy rezultat de tip *scaled*

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor (inferenţa fuzzy) → evaluarea consecinţelor

- Modelul Mamdani
  - Tipologie (în funcţie de modul de aplicare a rezultatului asupra funcţiei de apartenenţă a consecinţei)
    - Mulţimi fuzzy rezultat de tip *clipped*
      - Funcţia de apartenenţă a consecinţei se taie la nivelul indicat de valoarea de adevăr a rezultatului
      - Avantaj → calcule uşoare
      - Dezavantaj → se pierd informaţii

    - Mulţimi fuzzy rezultat de tip *scaled*
      - Funcţia de apartenenţă a consecinţei se ajustează prin scalare (multiplicare) valoarea de adevăr a rezultatului
      - Avantaj → se pierde mai puţină informaţie
      - Dezavantaj → calcule mai complicate

Sisteme inteligente – SBC – sisteme fuzzy

- Conţinut şi arhitectură → Evaluarea regulilor → evaluarea consecinţelor → Modelul Mamdani
  - Exemplu pentru reglarea unui aparat de aer condiţionat
    - Intrări:
      - x (temperatura – rece (A_1), normal (A_2), fierbinte (A_3)) şi
      - y (umiditatea – mică (B_1), mare (B_2))
    - Ieşiri:
      - z (puterea aparatului – redusă (C_1), medie (C_2), ridicată (C_3))
    - Date de intrare:
      - Temperatura x = 37
        - $\mu_{A1}(x)=0$, $\mu_{A2}(x)=0.6$, $\mu_{A3}(x)=0.3$
      - Umiditatea y = 0.8
        - $\mu_{B1}(x)=0.9$, $\mu_{B2}(x)=0$

		Data de intrare y	
		Mică	Mare
Data de intrare x	Rece	Redusă	Constantă
	Normal	Constantă	Mărită
	Fierbinte	Mărită	Mărită

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor → evaluarea consecinţelor → Modelul Mamdani

R1: dacă x este în A_1 şi y este în B_1 atunci z este în C_1

$\mu_{A1}(x) = 0$ $\mu_{B1}(y)=0.9$ $\mu_{C1}(z)=0$

R2: dacă x este în A_2 sau y este în B_1 atunci z este în C_2

$\mu_{A2}(x) = 0.6$ $\mu_{B1}(y)=0.9$ $\mu_{C2}(z)=0,9$

R3: dacă x este în A_3 sau y este în B_2 atunci z este în C_3

$\mu_{A3}(x) = 0.3$ $\mu_{B2}(y)=0$ $\mu_{C3}(z)=0.3$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor (inferenţa fuzzy) → evaluarea consecinţelor

- Modelul Sugeno
  - Ideea de bază
    - consecinţa regulii este de forma "variabila de ieşire este o funcţie crisp care depinde de intrări"
    - Exemplu
      - ***Dacă x este în A şi y este în B atunci z este f(x,y)***
    - Tipologie (în funcţie de caracteristicile lui f(x,y))
      - Model Sugeno de grad 0 → dacă f(x,y) = k – constantă (funcţiile de apartenenţă ale consecinţelor sunt de forma *singleton* – mulţime fuzzy a cărei funcţie de apartenenţă ia valoarea 1 pentru un singur punct din universul de discurs şi 0 în toate celelalte puncte)
      - Model Sugeno de grad 1 → dacă f(x,y) = ax + by+c

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor → evaluarea consecinţelor → Modelul Sugeno

R1: dacă x este în A_1 şi y este în B_1 atunci z este în C_1

$\mu_{A1}(x) = 0$ $\mu_{B1}(y)=0.9$ $\mu_{C1}(z)=0$

R2: dacă x este în A_2 sau y este în B_1 atunci z este în C_2

$\mu_{A2}(x) = 0.6$ $\mu_{B1}(y)=0.9$ $\mu_{C2}(z)=0,9$

R3: dacă x este în A_3 sau y este în B_2 atunci z este în C_3

$\mu_{A3}(x) = 0.3$ $\mu_{B2}(y)=0$ $\mu_{C3}(z)=0.3$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor (inferenţa fuzzy) → evaluarea consecinţelor

- Modelul Tsukamoto
  - Ideea de bază
    - consecinţa regulii este de forma "variabila de ieşire face parte dintr-o mulţime fuzzy cu o funcţie de apartenetă monotonă"
      - Prin inferenţă se obţine o ieşire crisp indusă gradul de satisfacere a regulii (*rule's firing strength*)

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Evaluarea regulilor → evaluarea consecinţelor → Modelul Tsukamoto

R1: dacă x este în A_1 şi y este în B_1 atunci z este în C_1

$\mu_{A1}(x) = 0$ $\mu_{B1}(y)=0.9$ $\mu_{C1}(z)=0$

R2: dacă x este în A_2 sau y este în B_1 atunci z este în C_2

$\mu_{A2}(x) = 0.6$ $\mu_{B1}(y)=0.9$ $\mu_{C2}(z)=0,9$

R3: dacă x este în A_3 sau y este în B_2 atunci z este în C_3

$\mu_{A3}(x) = 0.3$ $\mu_{B2}(y)=0$ $\mu_{C3}(z)=0.3$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către expert
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
      - Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă
  - Construirea unei baze de reguli de către expert
    - Matricea de decizie a bazei de cunoştinţe
  - Evaluarea regulilor
    - Inferenţa – transformarea intrărilor fuzzy în ieşiri fuzzy prin aplicarea regulilor din baza de cunoştinţe
  - **Agregarea rezultatelor**
  - Defuzificarea
  - Interpretarea rezultatelor

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Agregarea rezultatelor

- Unificarea ieşirilor tuturor regulilor aplicate
- Se iau funcţiile de apartenenţă corespunzătoare tuturor consecinţelor şi se combină într-o singură mulţime fuzzy, respectiv într-un singur rezultat
- Procesul de agregare are ca
  - intrări → funcţiile de apartenenţă (tăiate sau scalate) ale consecinţelor
  - ieşire → o mulţime fuzzy pentru variabila de ieşire
- Exemplu
  - Mamdani

  - Sugeno

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către expert
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
      - Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă
  - Construirea unei baze de reguli de către expert
    - Matricea de decizie a bazei de cunoştinţe
  - Evaluarea regulilor
    - Inferenţa – transformarea intrărilor fuzzy în ieşiri fuzzy prin aplicarea regulilor din baza de cunoştinţe
  - Agregarea rezultatelor
  - **Defuzificarea**
  - Interpretarea rezultatelor

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Defuzificarea

- Ideea de bază
  - Transformarea rezultatului fuzzy al agregării într-o valoare crisp
  - Inferenţa → obţinerea unor regiuni fuzzy pentru fiecare variabilă de ieşire
  - Defuzzificarea → Fiecare regiune de ieşire trebuie defuzzificată pentru a produce valori crisp
    - Se poate pierde informaţie
- Metode
  - Bazate pe centrul de greutate
    - Centrul de greutate (COG sau COA – *Centroid Area*)
    - Bisectoarea (BOA – *Bisector of area*)
  - Bazate pe maximul funcţiei de apartenenţă
    - Media maximelor (MOM - *Mean of maximum*)
    - Cel mai mic maxim (SOM - *Smallest of maximum*)
    - Cel mai mare maxim (LOM - *Largest of maximum*)

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Defuzificarea → Metode

- Centrul de greutate (COG sau COA – Centroid Area)
  - Găseşte punctul z din mijlocul mulţimii agregate

$$COG = \frac{\sum_{i=0}^{n} x_i \mu_A(x_i)}{\sum_{i=0}^{n} \mu_A(x_i)} \text{ sau } COG = \frac{\int_{i=0}^{n} x_i \mu_A(x_i)}{\int_{i=0}^{n} \mu_A(x_i)}$$

  - Exemplu
    - Modelul Mamdani → estimarea centrului prin calcularea COG pe baza unui eşantion de n puncte (x_i, i =1,2,..., n) din mulţimea fuzzy rezultat

$$COG = \frac{5*0+6*0.3+7*0.5+8*0.7+9*0.9+10*0.9+11*0.7+12*0.5+13*0.3+14*0.3+15*0.3+16*0.3}{0+0.3+0.5+0.7+0.9+0.9+0.7+0.5+0.3+0.3+0.3+0.3}$$

$$COG \cong 13.7$$

    - Modelul Sugeno sau Tsukamoto, COG devine media ponderată a celor *m* valori crisp rezultate prin aplicarea tuturor regulilor (în număr de *m*)

$$COG = \frac{9*0.9+11*0.3}{0.9+0.3}$$

$$COG \cong 9.5$$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Defuzificarea → Metode

- Bisectoarea (BOA – Bisector of area)
  - Găseşte punctul z în care o linie verticală va separa mulţimea agregată în 2 părţi de arii egale

$$BOA = \int_{\alpha}^{z} \mu_A(x)dx = \int_{z}^{\beta} \mu_A(x)dx,$$

unde $\alpha = \min\{x \mid x \in A\}$ şi $\beta = \min\{x \mid x \in A\}$

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Defuzificarea → Metode

- Media maximelor (MOM - *Mean of maximum*)
  - Găseşte punctul z care reprezintă media punctelor din mulţimea agregată care au valorea funcţiei de apartenenţă (μ) maximă

$$MOM = \frac{\sum_{x_i \in \max \mu} x_i}{|max\mu|}, \text{unde } max\mu = \mu^* = \{x \mid x \in A, \mu(x) = maxim\}$$

- Cel mai mic maxim (SOM - *Smallest of maximum*)
  - Găseşte cel mai mic punct z din mulţimea agregată care are valorea funcţiei de apartenenţă (μ) maximă

- Cel mai mare maxim (LOM - *Largest of maximum*)
  - Găseşte cel mai mare punct z din mulţimea agregată care are valorea funcţiei de apartenenţă (μ) maximă

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură → Defuzificarea

- Ideea de bază
  - Transformarea rezultatului fuzzy al agregării într-o valaore crisp
- Metode
  - Bazate pe centrul de greutate
    - Centrul de greutate (COG sau COA – Centroid Area)
    - Bisectoarea (BOA – Bisector of area)
  - Bazate pe maximul funcţiei de apartenenţă
    - Media maximelor (MOM - Mean of maximum)
    - Cel mai mic maxim (SOM - Smallest of maximum)
    - Cel mai mare maxim (LOM - Largest of maximum)

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către expert
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
      - Stabilirea variabilelor fuzzy şi a mulţimilor fuzzy pe baza funcţilor de apartenenţă
  - Construirea unei baze de reguli de către expert
    - Matricea de decizie a bazei de cunoştinţe
  - Evaluarea regulilor
    - Inferenţa – transformarea intrărilor fuzzy în ieşiri fuzzy prin aplicarea regulilor din baza de cunoştinţe
  - Agregarea rezultatelor
  - Defuzificarea
  - **Interpretarea rezultatelor**

Sisteme inteligente – SBC – sisteme fuzzy

Conţinut şi arhitectură

- Paşi în construirea unui sistem fuzzy
  - Definirea intrărilor şi ieşirilor de către exeprt
    - Datele de intrare şi ieşire brute
    - Fuzzificarea datelor de intrare şi ieşire
  - Construirea unei baze de reguli de către expert
  - Evaluarea regulilor
  - Agregarea rezultatelor
  - Defuzificarea
  - Interpretarea rezultatelor

Domeniul expertului

Motorul de inferenţă

Sisteme inteligente – SBC – sisteme fuzzy

- Avantaje
  - Se pot folosi reguli pentru exprimarea impreciziei şi aproximărilor din lumea reală
  - Uşor de înţeles, testat şi întreţinut
  - Robuste → pot opera şi când regulile nu sunt foarte clare
  - Necesită mai puţine reguli decât alte SBC
  - Regulile se evaluează în paralel

- Dezavantaje
  - Necesită multe simulări şi testări
  - Nu învaţă singure
  - Este dificilă stabilirea celor mai corecte reguli
  - Lipsa unui model matematic exact

Sisteme inteligente – SBC – sisteme fuzzy

Aplicaţii

- Control în spaţiu
  - Altitudinea sateliţilor
  - Reglaje în avioane
- Control auto
  - Transmisie automată, controlul traficului, Sisteme împotriva blocării roţilor în timpul frânării
  - *Dacă frâna este caldă şi viteza nu este foarte mare atunci presiunea de frânare se reduce uşor*
- Business
  - Sisteme de decizie, evaluarea personalului, managementul fondurilor, predicţii de piaţă, sisteme de asigurarea a securităţii tranzacţiilor
- Industrie
  - Controlul schimbului de energie, controlul purificării apei,
  - Controlul pH-ului, distilarea chimică, producerea de polimeri, formarea metalelor
- Electronică
  - Controlul expunerii automate a camerelor foto/video, controlul umidităţii, sisteme de aer condiţionat, Sisteme pentru reglajul duşului
  - Sisteme pentru reglarea congelatoarelor
  - Sisteme pentru reglarea maşinilor de spălat (Încărcătura, concentraţia de detergent, etc.)

Sisteme inteligente – SBC – sisteme fuzzy

Aplicaţii

- Alimentaţie
  - Procese de producere a brânzei
- Militar
  - Recunoaşterea subacvatică, recunoaşterea imaginilor în infraroşu, sisteme de decizie în traficul naval
- Marină
  - Piloţi automaţi pentru nave, şi submarine selecţia automată a rutelor
- Medical
  - Sisteme de diagnosticare (diabet, cancer), controlul presiunii arteriale în timpul anesteziei, modelarea rezultatelor neuropatologice pentru pacienţii cu Alzheimer
- Robotică
  - Controlul manipulatorilor flexibili şi a braţelor robotice

Recapitulare

- SBC
  - Sisteme computaţionale în care baza de cunoştinţe şi modulul de control se suprapun
- SBC pot funcţiona
  - în medii certe
    - SBL
    - SBR
  - În medii incerte
    - Sisteme de tip Bayes
      - Regulile au asociate probabilităţi de realizare
    - Sisteme bazate pe factori de certitudine
      - Faptele şi regulile au asociaţi factori de certitudine
    - Sisteme fuzzy
      - Faptele au asociate grade de apartenenţă la anumite mulţimi

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop