Inteligență artificială

5. Elemente de teoria jocurilor (I)

Florin Leon

Universitatea Tehnică „Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare

Universitatea „Al. I. Cuza” din Iași
Facultatea de Informatică

Elemente de teoria jocurilor (I)

Elemente de teoria jocurilor (I)

1. Jocuri secvențiale
    1.1. Introducere și formalizare
    1.2. Algorimul minimax
    1.3. Retezarea alfa-beta
    1.4. Căutarea pe arbori Monte Carlo
2. Jocuri strategice
    2.1. Dominare
    2.2. Echilibrul Nash pur
3. Concluzii

Teoria jocurilor

- Studiază interacțiunile strategice între jucători/agenți raționali care aleg diferite acțiuni pentru a-și maximiza câștigul
- Mai formal, reprezintă studiul modelelor matematice de conflict și cooperare între decidenți inteligenți și raționali
- Teoria jocurilor este o abordare interdisciplinară menită să studieze comportamentul uman
- John von Neumann, Oskar Morgenstern: *Theory of Games and Economic Behavior* (1944)

Maximizarea câștigurilor

- Fiecare jucător trebuie să-și maximizeze câștigurile într-un mediu influențat de strategiile celorlalți jucători
  - Alegerile unui jucător depind de alegerile tuturor celorlalți jucători
  - Fiecare jucător încearcă să-și maximizeze câștigul indiferent de acțiunile celorlalți jucători
  - Conflicte, cooperare
  - Decizii sociale: cu cine și cum să coopereze
- Alegerea rațională a strategiilor poate presupune și maximizarea câștigurilor grupului de jucători

Jocuri secvențiale

- De-a lungul timpului, jocurile care necesită explorarea unor alternative au fost considerate provocări pentru inteligența umană
  - Dame (Mesopotamia, cc. 3000 î.Hr.)
  - Go (China, cc. 600 î.Hr.)
  - Șah (India, cc. 600 d.Hr.)

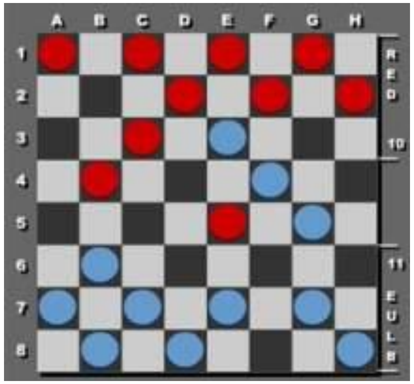
- Jocurile sunt folosite deseori pentru a proiecta și testa algoritmi de inteligență artificială

Jocuri secvențiale

- Unul din cele mai vechi subdomenii ale IA-ului (Zuse, Shannon, Wiener, Turing, anii 1950)
- Forme abstracte și pure de competiție care necesită inteligență
- Probleme dificile cu o structură inițială minimă de cunoștințe
  - Stări și acțiuni ușor de reprezentat
  - Puține cunoștințe necesare despre mediu
- Jocurile sunt un caz particular al problemelor de căutare
- Deseori au spații de căutare foarte mari
- Sunt distractive ☺

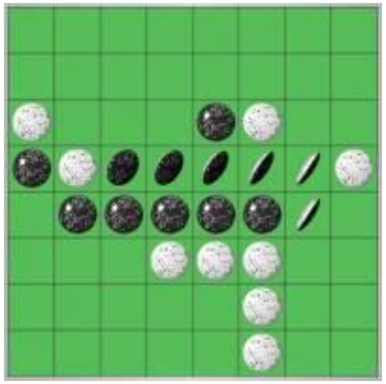
Jocuri deterministe cu informații perfecte

Șah

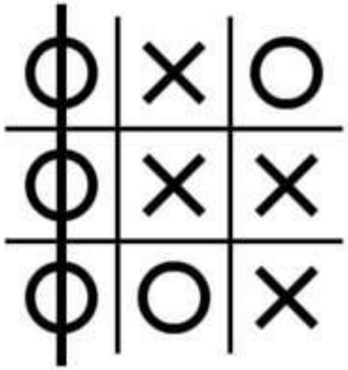


Dame

Go



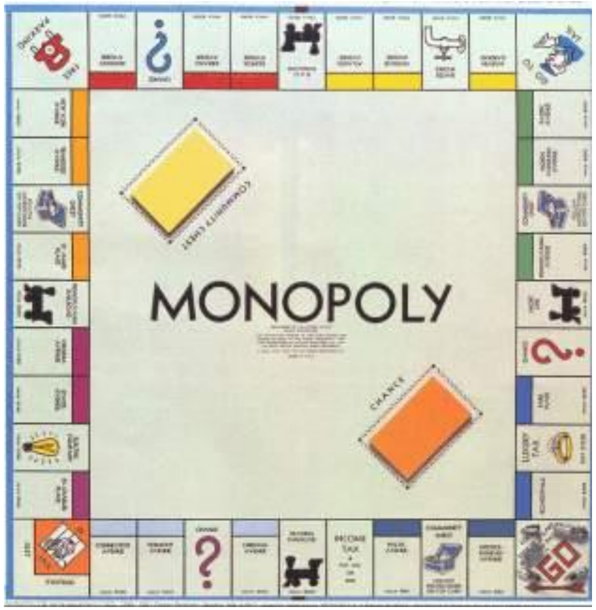
Othelo



X și 0

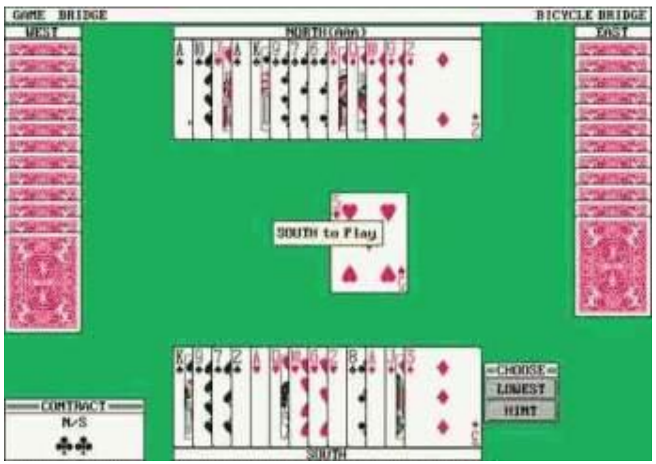
Jocuri nedeterministe cu informații perfecte

Table



Monopoly

Jocuri nedeterministe cu informații imperfecte



Bridge

Poker

Scrabble

Dimensiunea spațiului de căutare

- Șah („drosofila IA-ului”)
  - Factor de ramificare ≈ 35
  - ≈ 50 de mutări pe jucător
  - ≈ 35^{100} (10^{154}) noduri
  - 10^{40} stări distincte (dimensiunea grafului de căutare)
- Go
  - Factorul de ramificare începe de la 361 (tablă 19 x 19)
  - ≈ 200 de mutări pe stare, 300 de niveluri ⇒ 200^{300} (10^{690}) noduri în arbore

Jocurile și căutarea

- Căutarea clasică: un singur agent care încearcă fără piedici să își atingă obiectivul
- Jocurile: căutare în prezența unui adversar
- Reprezentarea jocurilor ca probleme de căutare
  - Stări: configurațiile tablei de joc
  - Operatori/acțiuni: mutările permise
  - Starea inițială: configurația *curentă* a tablei
  - Starea scop: configurația câștigătoare

Metode de rezolvare pentru jocuri

- Modalitatea de joc:
  - Se consideră toate mutările legale care se pot face
  - Fiecare mutare conduce către o nouă configurație (poziție)
  - Se evaluează fiecare poziție rezultată și se determină care este cea mai bună
  - Se face mutarea respectivă
  - Se așteaptă mutarea adversarului și se repetă algoritmul
- Probleme cheie:
  - Reprezentarea tablei
  - Generarea tuturor configurațiilor următoare valide
  - Evaluarea unei poziții
  - Considerarea mai multor mutări în avans

Funcția de evaluare

- Evaluează valoarea unei poziții
  - Considerând funcțiile *g* și *h* de la căutarea clasică, aici contează numai *h*; *g* este irelevant
- Este o funcție euristică și cuprinde cunoștințele expert despre domeniul problemei
- Inteligența calculatorului depinde în mare măsură de funcția de evaluare

Funcția de evaluare

- Presupunem că jocul este de sumă nulă
- Putem folosi o singură funcție de evaluare pentru ambii jucători
- $f(n) > 0$: poziția *n* este bună pentru calculator și rea pentru adversar (om)
- $f(n) < 0$: poziția *n* este rea pentru calculator și bună pentru adversar

Exemple: funcții de evaluare

- X și 0:
  - $f(n)$ = [numărul de direcții de 3 pătrățele deschise pentru calculator] – [numărul de direcții deschise pentru adversar]
  - Direcțiile sunt linii, coloane sau diagonale complete
- Șah (funcția lui Turing):
  - $f(n) = w(n) / b(n)$
  - $w(n)$ = suma punctelor pieselor albe
  - $b(n)$ = suma punctelor pieselor negre
  - Pion = 1 punct, cal/nebun = 3 puncte, turn = 5 puncte, regină = 9 puncte

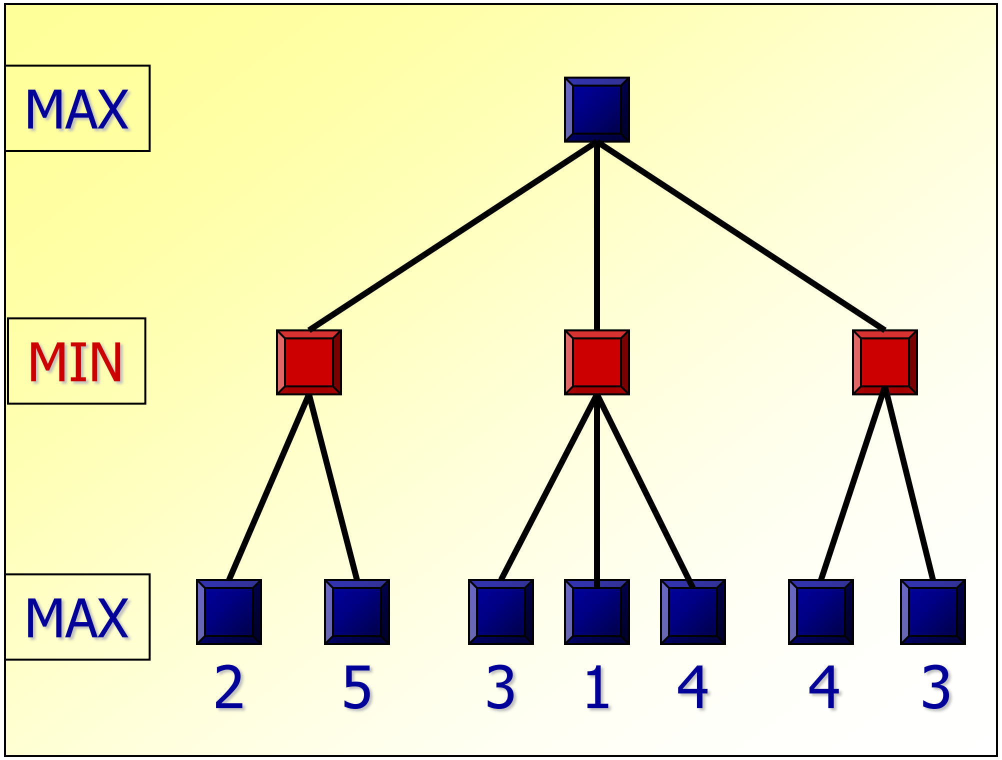
Exemple: funcții de evaluare

- Majoritatea funcțiilor de evaluare sunt sume ponderate ale trăsăturilor unei poziții:
  - $f(n) = w_1 \cdot t_1(n) + w_2 \cdot t_2(n) + \ldots + w_k \cdot t_k(n)$
  - Trăsături pentru șah sunt numărul de piese, plasarea pe tablă a pieselor, pătrățele controlate etc.
- Deep Blue avea în jur de 8000 de trăsături în funcția de evaluare
- Pot exista combinații neliniare de trăsături
  - Nebunul valorează mai mult spre sfârșitul jocului decât la început
  - 2 nebuni valorează mai mult decât dublul valorii unuia singur

Elemente de teoria jocurilor (I)

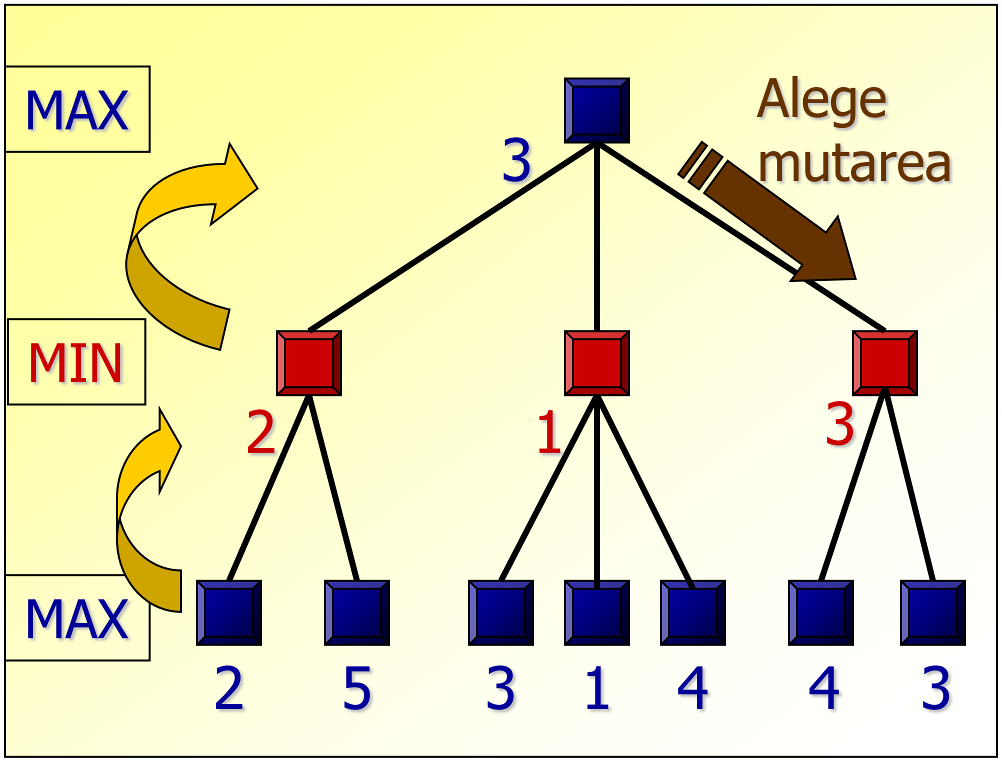
1. Jocuri secvențiale
    - 1.1. Introducere și formalizare
    - 1.2. Algorimul minimax
    - 1.3. Retezarea alfa-beta
    - 1.4. Căutarea pe arbori Monte Carlo
2. Jocuri strategice
    - 2.1. Dominare
    - 2.2. Echilibrul Nash pur
3. Concluzii

Minimax



- Restricții:
  - Doi jucători:
    - MAX (calculator)
    - MIN (adversar)
  - Joc determinist cu informații perfecte
- Se selectează o limită de adâncime (de ex. 2) și o funcție de evaluare

Modul de funcționare



- Se construiește arborele până la limita de adâncime

- Se calculează funcția de evaluare pentru frunze

- Se propagă evaluarea în sus
  - selectând minimele în MIN
  - selectând maximele în MAX

Structura de date

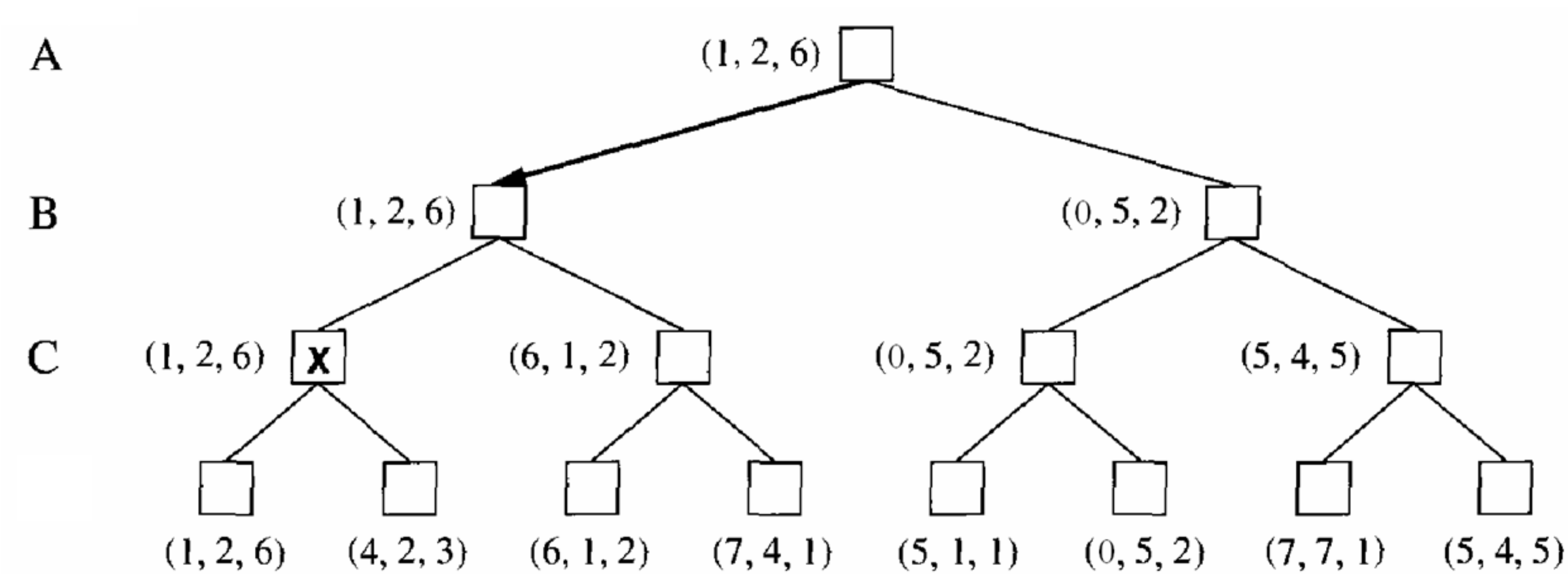
```
Joc =
{
    Stare-curentă : STARE
        » starea în care trebuie să mute jucătorul care utilizează algoritmul
    Acţiuni(s : STARE) : ACŢIUNE*
        » o funcţie care întoarce lista de acţiuni (mutări) care se pot efectua în starea s
    Stare-succesoare(s : STARE, a : ACŢIUNE) : STARE
        » o funcţie care întoarce starea în care se ajunge aplicând acţiunea a în starea s
    Este-terminală(s : STARE) : BOOLEAN
        » o funcţie care verifică dacă starea s este o stare terminală (câştig, pierdere sau
        » remiză)
    Evaluare(s : STARE) : NUMĂR-REAL
        » funcţia de evaluare statică, care calculează valoarea poziţiei s
        » joc de sumă nulă cu doi jucători: cu cât valoarea funcţiei este mai mare,
        » cu atât poziţia este mai bună pentru jucătorul care utilizează algoritmul
        » şi mai rea pentru adversar
}
```

Pseudocod

```
MINIMAX
   intrări: joc : JOC
            stare : STARE » în apelul iniţial: joc.Stare-curentă
            este-nivel-maximizant : BOOLEAN » în apelul iniţial: Adevărat
            adâncime-curentă : NUMĂR-NATURAL » în apelul iniţial: 0
            adâncime-maximă : NUMĂR-NATURAL
   ieşiri:  acţiune : ACŢIUNE
            valoare : NUMĂR-REAL
-------------------------------------------------------------------------------------------
dacă adâncime-curentă ≥ adâncime-maximă sau joc.Este-terminală(stare) atunci
   întoarce (Nul, joc.Evaluare(stare))

pentru-fiecare acţiune din joc.Acţiuni(stare) execută
   (_, valori[acţiune]) ← MINIMAX(joc, joc.Stare-succesoare(stare, acţiune),
      nu este-nivel-maximizant, adâncime-curentă + 1, adâncime-maximă))
      » acţiunea contează doar în rădăcină
      » pentru celelalte niveluri contează doar valorile
      » nivelul următor este de tip opus celui curent (maximizant, minimizant)

dacă este-nivel-maximizant atunci
   întoarce (joc.Acţiuni(stare)[valori.Argmax()], valori.Max()) » acțiunea cu valoare maximă
altfel
   întoarce (joc.Acţiuni(stare)[valori.Argmin()], valori.Min()) » acțiunea cu valoare minimă
```

Jocuri cu mai mulți jucători

- Funcția de evaluare este vectorială și redă utilitățile tuturor jucătorilor



Alianțele

- Alianțele pot rezulta din aplicarea strategiilor optime individuale
- Cooperarea poate rezulta din urmărirea comportamentului rațional individual (egoist)

Elemente de teoria jocurilor (I)

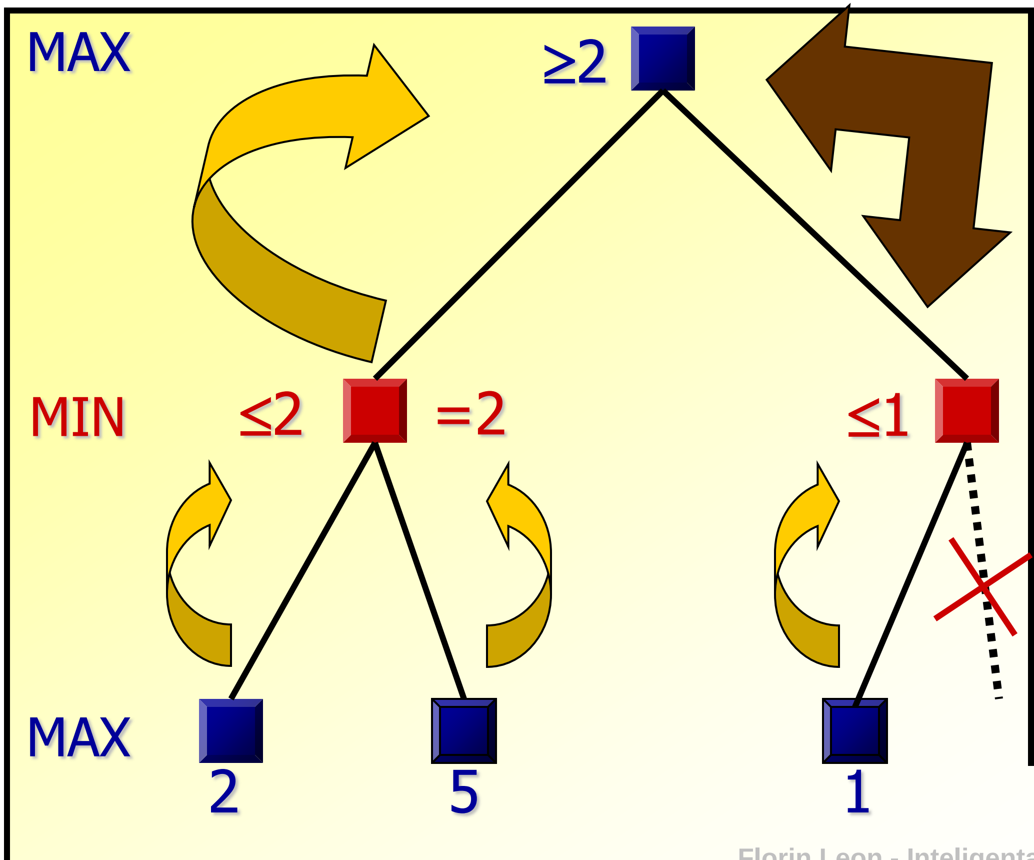
1. Jocuri secvențiale
    1.1. Introducere și formalizare
    1.2. Algorimul minimax
    1.3. Retezarea alfa-beta
    1.4. Căutarea pe arbori Monte Carlo
2. Jocuri strategice
    2.1. Dominare
    2.2. Echilibrul Nash pur
3. Concluzii

Retezarea alfa-beta

- engl. “alpha-beta pruning”, elagajul alfa-beta
- Optimizare aplicată algoritmului minimax
- Minimax
  - Creează mai întâi întregul arbore, până la limita de adâncime
  - Apoi realizează propagarea
- Retezarea alfa-beta
  - Întrețese generarea arborelui cu propagarea valorilor
  - Unele valori obținute în arbore furnizează informații privind redundanța altor părți, care nu mai trebuie generate

Ideea alfa-beta

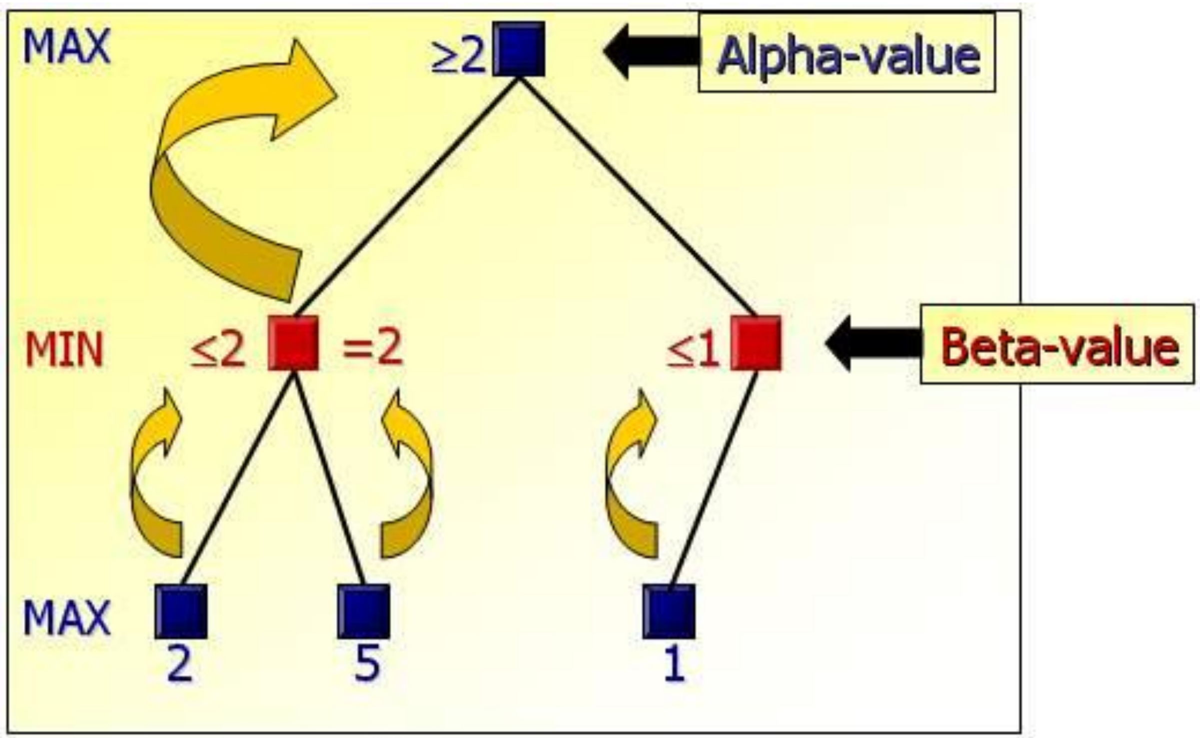
- Se generează arborele în adâncime, de la stânga la dreapta
- Se propagă valorile finale ale nodurilor ca estimări inițiale pentru părinții lor



- Valoarea MIN (1) este deja mai mică decât valoarea MAX a părintelui (2)
- Valoarea MIN poate doar să descrească în continuare
- Valoarea MAX poate doar să crească
- Nu are sens să mergem mai jos sub acest nod

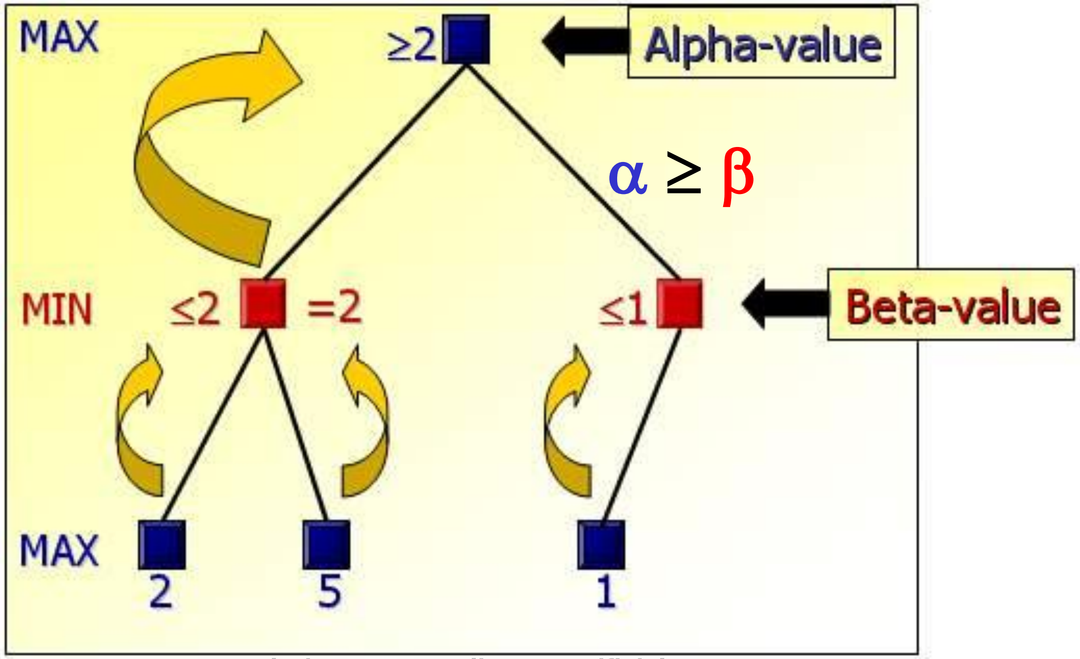
Terminologie

- Valorile (temporare) în nodurile MAX sunt valori alfa
- Valorile (temporare) în nodurile MIN sunt valori beta



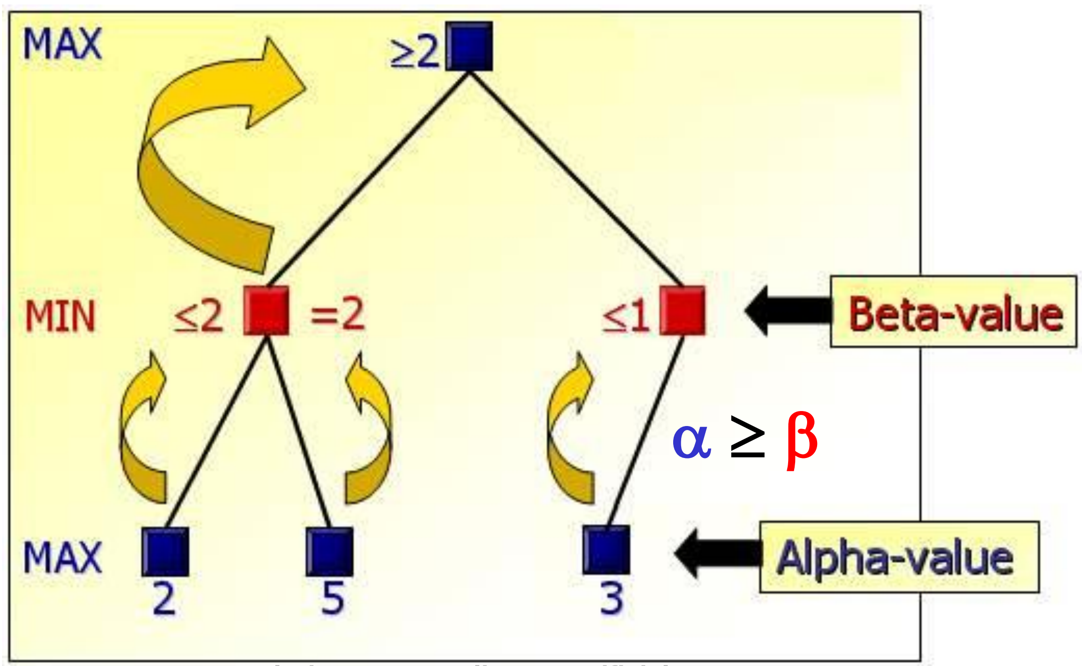
Regulile retezării alfa-beta

- **Dacă** valoarea alfa este mai mare sau egală decât valoarea beta a unui nod descendent, **atunci** se oprește generarea fiilor nodului descendent

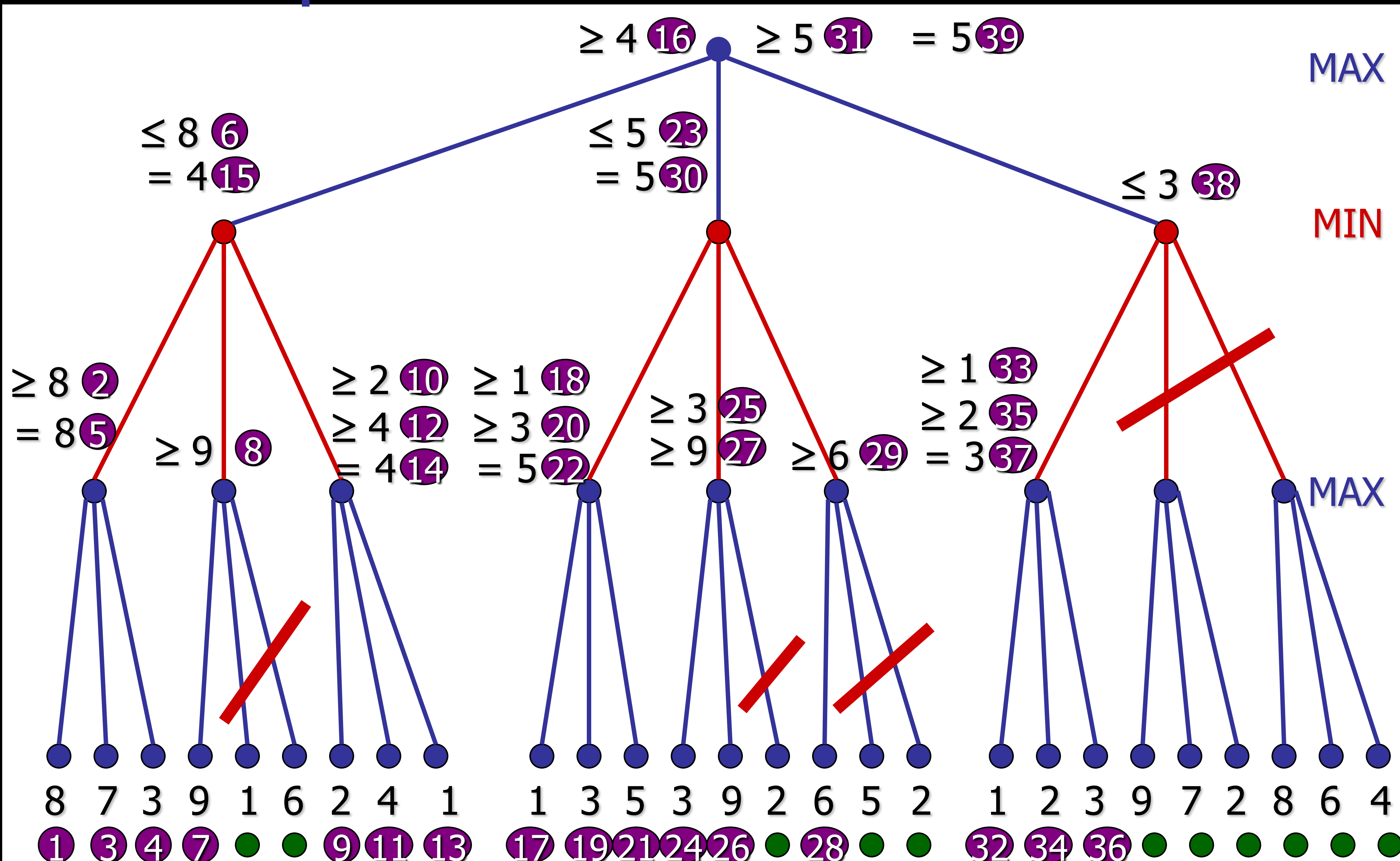


Regulile retezării alfa-beta

- **Dacă** valoarea beta este mai mică sau egală decât valoarea alfa a unui nod descendent, **atunci** se oprește generarea fiilor nodului descendent



Exemplu: minimax cu alfa-beta



11 evaluări evitate

Pseudocod

MINIMAX-CU-ALFA-BETA
    intrări: *joc* : JOC
        adâncime-maximă : NUMĂR-NATURAL
    ieşiri: *acţiune* : ACŢIUNE

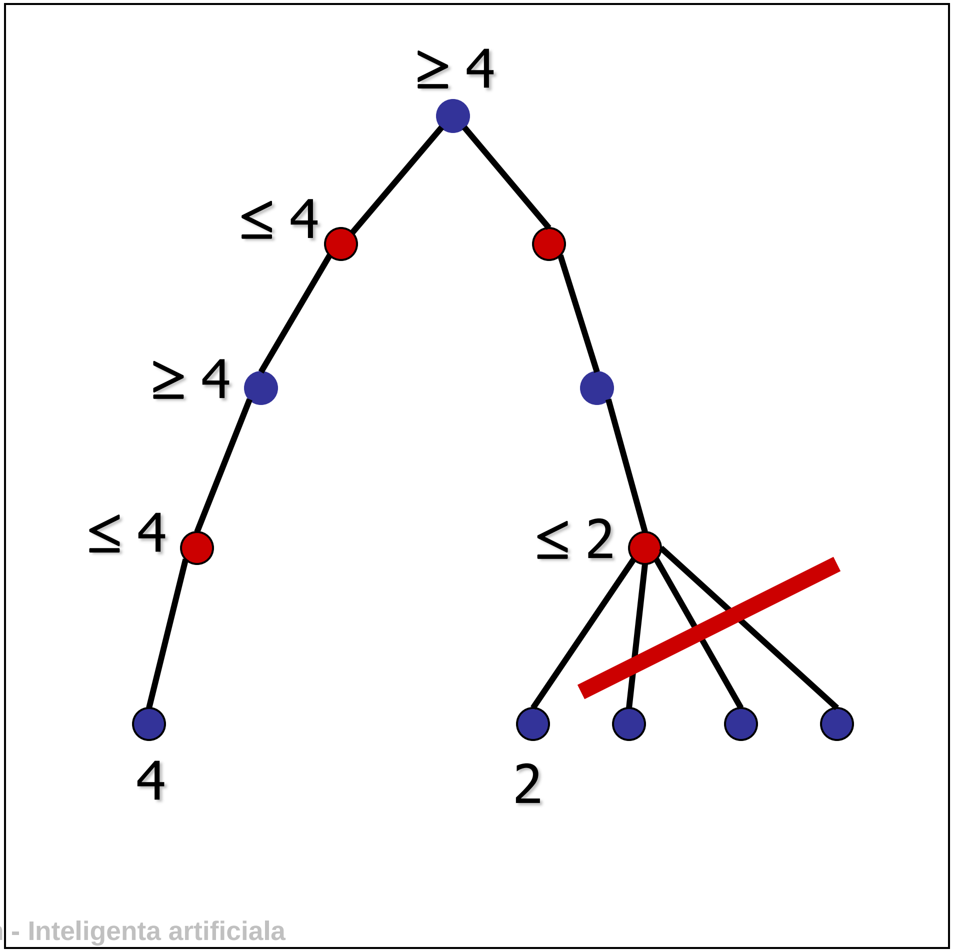
(*acţiune*, _) ← VALOARE-MAXIMĂ(*joc*, *joc*. Stare-curentă, *alfa* ← −∞, *beta* ← ∞, *adâncime-curentă* ← 0, ...
    adâncime-maximă)
întoarce *acţiune*

```
VALOARE-MAXIMĂ
    intrări: joc : JOC
             stare : STARE
             alfa : NUMĂR-REAL
             beta : NUMĂR-REAL
             adâncime-curentă : NUMĂR-NATURAL
             adâncime-maximă : NUMĂR-NATURAL
    ieşiri:  acţiune : ACŢIUNE
             valoare : NUMĂR-REAL
------------------------------------------------------------------------------------------
dacă adâncime-curentă ≥ adâncime-maximă sau joc.Este-terminală(stare) atunci
    întoarce joc.Evaluare(stare)

val-max ← −∞, acţiune-optimă ← Nul
pentru-fiecare acţiune din joc.Acţiuni(stare) execută
    (act, val) ← VALOARE-MINIMĂ(joc, joc.Stare-succesoare(stare, acţiune), alfa, beta, ...
        adâncime-curentă + 1, adâncime-maximă)
    dacă val-max < val atunci
        val-max ← val, acţiune-optimă ← act
    dacă val-max ≥ beta atunci » alfa ≥ beta, nu se mai încearcă celelalte acţiuni
        întoarce (acţiune-optimă, val-max)
    alfa ← Max(val-max, alfa)
întoarce (acţiune-optimă, val-max)
```

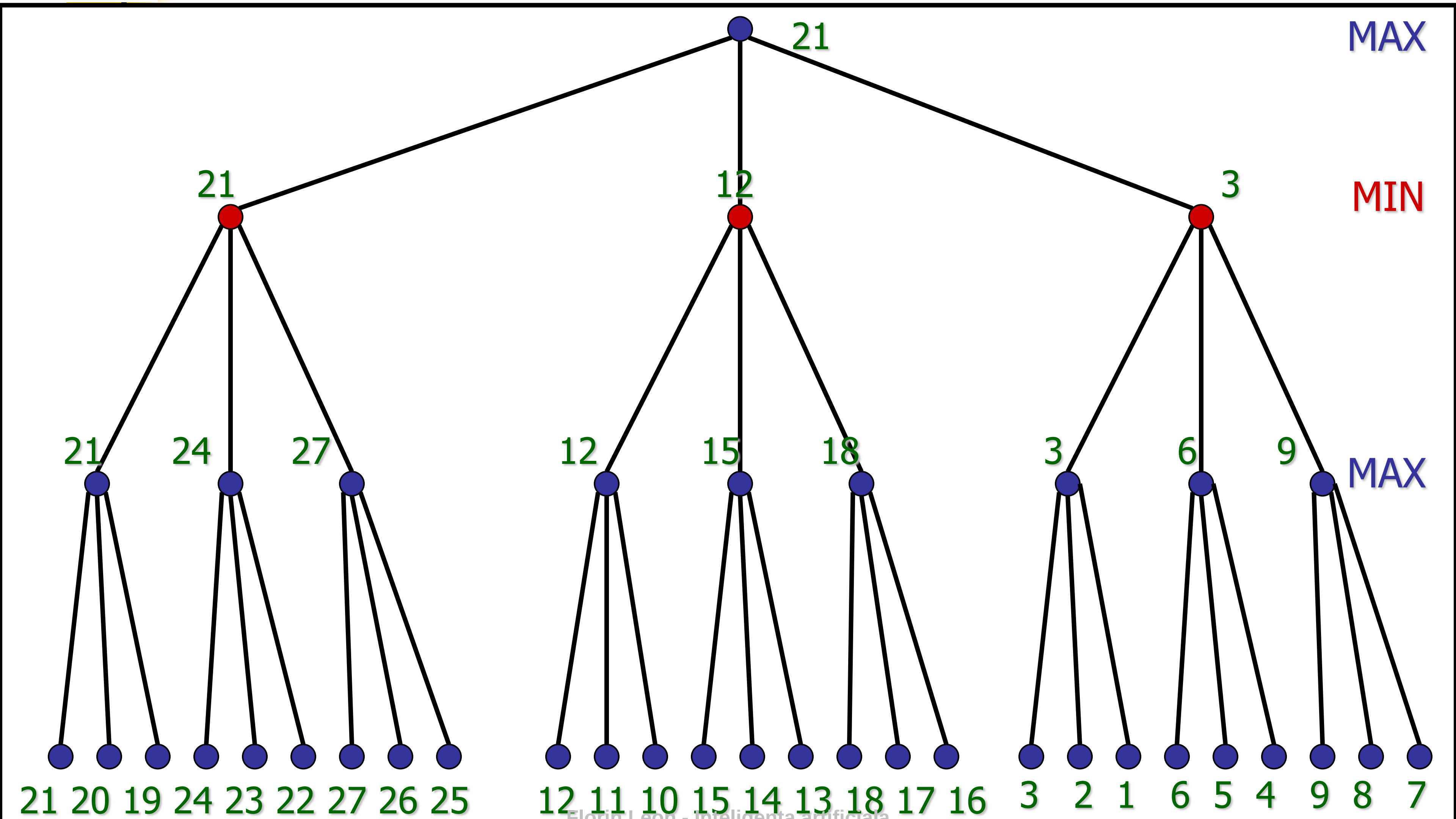
```
VALOARE-MINIMĂ
    intrări: joc : JOC
             stare : STARE
             alfa : NUMĂR-REAL
             beta : NUMĂR-REAL
             adâncime-curentă : NUMĂR-NATURAL
             adâncime-maximă : NUMĂR-NATURAL
    ieşiri:  acţiune : ACŢIUNE
             valoare : NUMĂR-REAL
-----------------------------------------------------------------------------------------------
dacă adâncime-curentă ≥ adâncime-maximă sau joc.Este-terminală(stare) atunci
    întoarce joc.Evaluare(stare)

val-min ← ∞, acţiune-optimă ← Nul
pentru-fiecare acţiune din joc.Acţiuni(stare) execută
    (act, val) ← VALOARE-MAXIMĂ (joc, joc.Stare-succesoare(stare, acţiune), alfa, beta, ...
        adâncime-curentă + 1, adâncime-maximă)
    dacă val-min > val atunci
        val-min ← val, acţiune-optimă ← act
    dacă alfa ≥ val-min atunci » alfa ≥ beta, nu se mai încearcă celelalte acţiuni
        întoarce (acţiune-optimă, val-min)
    beta ← Min(val-min, beta)
întoarce (acţiune-optimă, val-min)
```

Retezare în adâncime

- Pentru arbori cu cel puțin 4 niveluri min/max, retezarea alfa-beta se aplică și pentru niveluri mai adânci

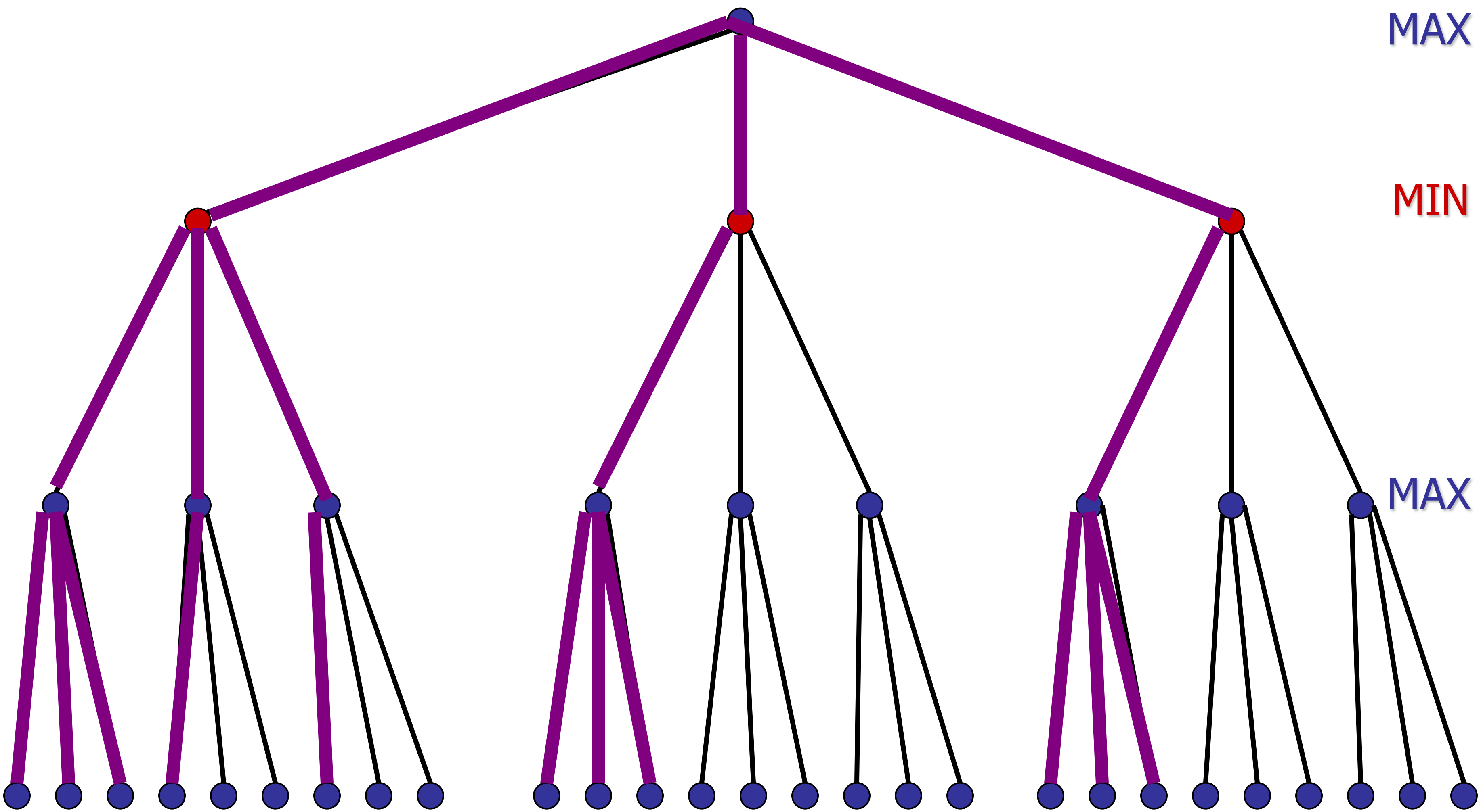


Cazul cel mai favorabil: arbore perfect ordonat



Cazul cel mai favorabil

- Când pe fiecare nivel cel mai bun nod este primul din stânga



Doar **ramurile îngroșate** sunt explorate

Reordonarea

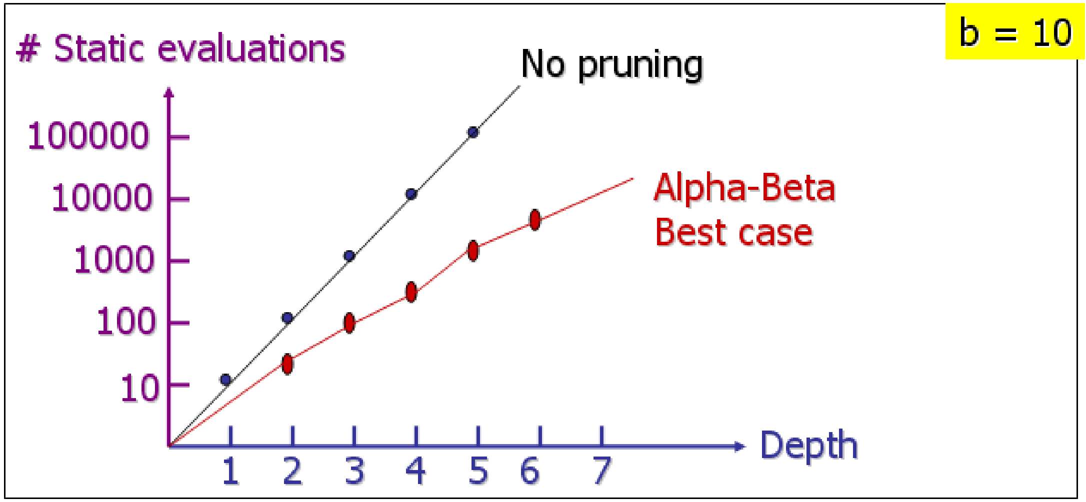
- De exemplu, pentru șah:
  - Mutările cele mai bune descoperite în căutarea anterioară
  - Capturile
  - Atacurile
  - Mutările înainte
  - Mutările înapoi

Cazul cel mai favorabil

- Numărul de evaluări statice:
  - $n_{es} = 2 \cdot b^{d/2} - 1$, dacă d este par
  - $n_{es} = b^{(d+1)/2} + b^{(d-1)/2} - 1$, dacă d este impar
- În exemplul anterior:
  - $d = 3$, $b = 3 \Rightarrow n_{es} = 9 + 3 - 1 = 11$

Comparație între minimax și alfa-beta

- Cazul cel mai favorabil

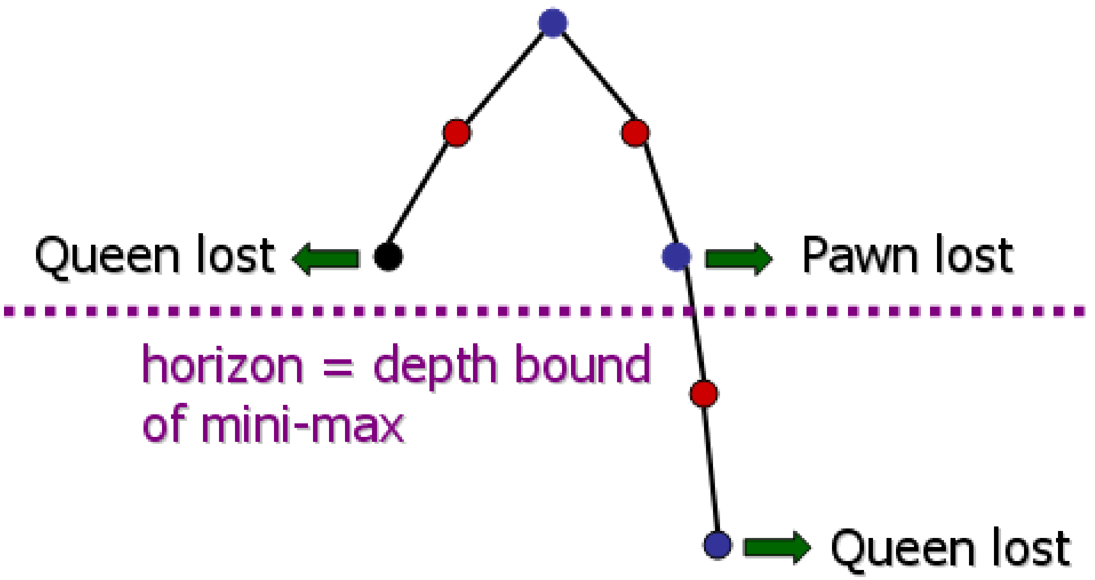


- Graficul are scară logaritmică
  - Retezarea alfa-beta are tot complexitate exponențială
- Cazul cel mai defavorabil
  - Pentru unii arbori, retezarea alfa-beta nu are niciun efect
  - Pentru unii arbori, este imposibilă reordonarea pentru a permite retezarea

Performanțele retezării alfa-beta

- Alfa-beta garantează calcularea aceleiași valori pentru rădăcină ca și minimax, cu o complexitate mai mică sau egală
- Cazul cel mai defavorabil: nu se face nicio retezare, se examinează $O(b^d)$ noduri
- Cazul mediu: $O\left(\left(\frac{b}{\log b}\right)^d\right)$
- Cazul cel mai favorabil: $O(b^{d/2})$
  - Poate căuta pe o adâncime de două ori mai mare decât minimax
  - Când cea mai bună mutare este și prima alternativă generată
- În cazul Deep Blue, s-a descoperit empiric că retezarea alfa-beta a redus factorul mediu de ramificare de la 35 la 6

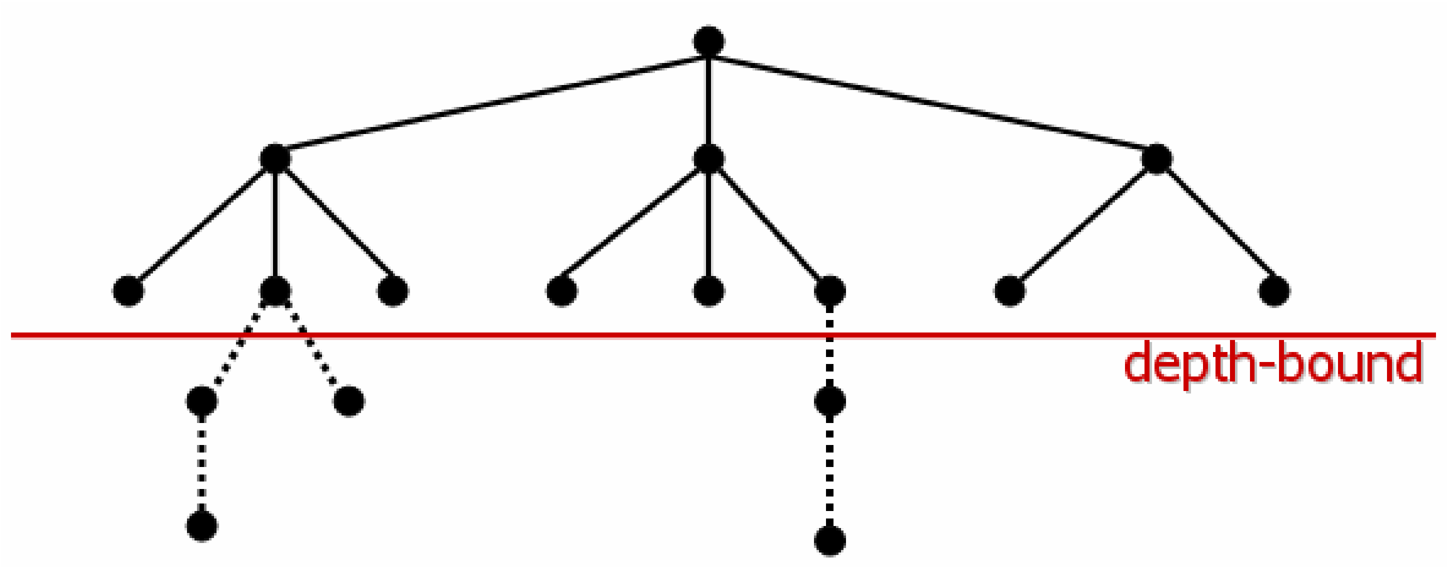
Efectul orizontului



- Decizia din dreapta pare mai bună decât cea din stânga, dar nu este. Alte mutări ar putea fi mai bune (de exemplu, pierderea unui cal)
- Soluția: continuarea euristică

Continuarea euristică

- În situații strategice cruciale (regele în pericol, pierdere iminentă de piese, pion transformat în regină etc.), se extinde căutarea dincolo de limita de adâncime



Căutarea secundară

- Uneori, este utilă verificarea unei mutări
- De exemplu, dacă se face căutarea pe 6 niveluri (*plies*) și s-a găsit cea mai bună mutare, se poate expanda numai acea poziție pentru încă 2 niveluri pentru a verifica dacă rămâne bună în continuare

Retezarea înainte

- engl. “forward pruning”
- Un jucător uman nu ia în calcul toate mutările posibile, ci doar pe cele care i se par utile
- Se ignoră un sub-arbore
  - Când există mai multe mutări simetrice sau echivalente
  - Pentru mutări care par iraționale (care conduc în situații aparent defavorbile)
  - Numai la adâncimi mari în arbore
  - Nerecomandate în apropierea rădăcinii

Limite de timp

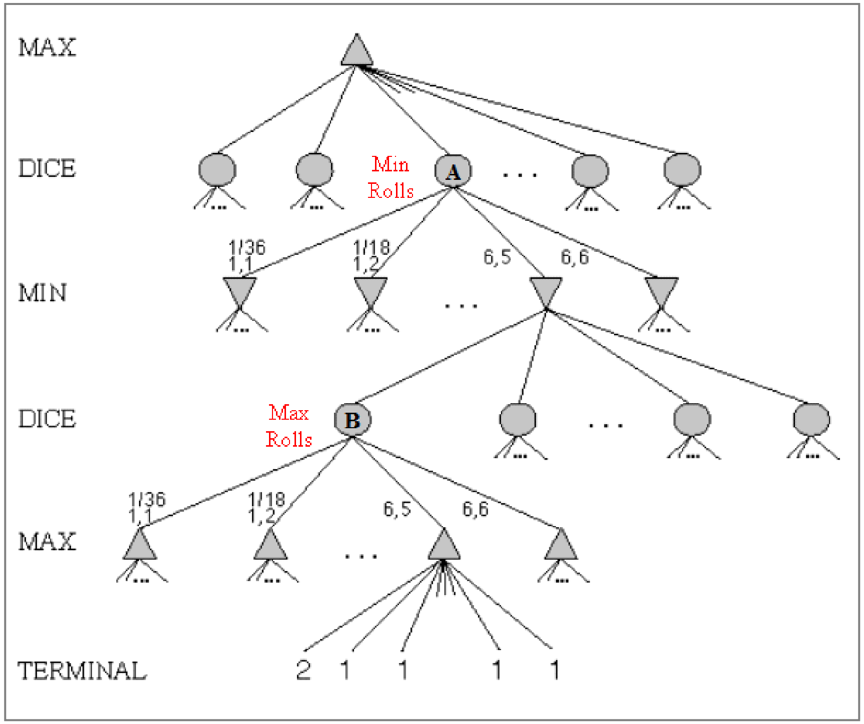
- Chiar și când există limite de adâncime, timpii pot varia mult
- Soluție: căutarea iterativă în adâncime
  - În orice moment, este disponibilă o mutare
  - Calitatea mutării crește în timp

Efectul euristicilor: șahul

- Cu minimax, putem căuta pe aproximativ 5 niveluri
- Un jucător mediu analizează 6-8 niveluri
- Cu reducerea alfa-beta, putem căuta pe aproximativ 10 niveluri (reducerea alfa-beta face diferența)
- Deep Blue
  - Căutare în medie pe 14 niveluri, maxim 40
  - Evaluarea a 30 de miliarde de poziții pe mutare
  - Bază de date cu 700 000 de jocuri
  - 4000 de strategii de deschidere
  - Toate rezolvările pentru pozițiile cu 5 piese și multe pentru pozițiile cu 6 piese
- Metodele euristice recente pot scădea factorul de ramificare de la 35 la aproximativ 3
  - De exemplu, mutarea nulă: oponentul mută de două ori la început și apoi se face o căutare alfa-beta pe un număr redus de niveluri

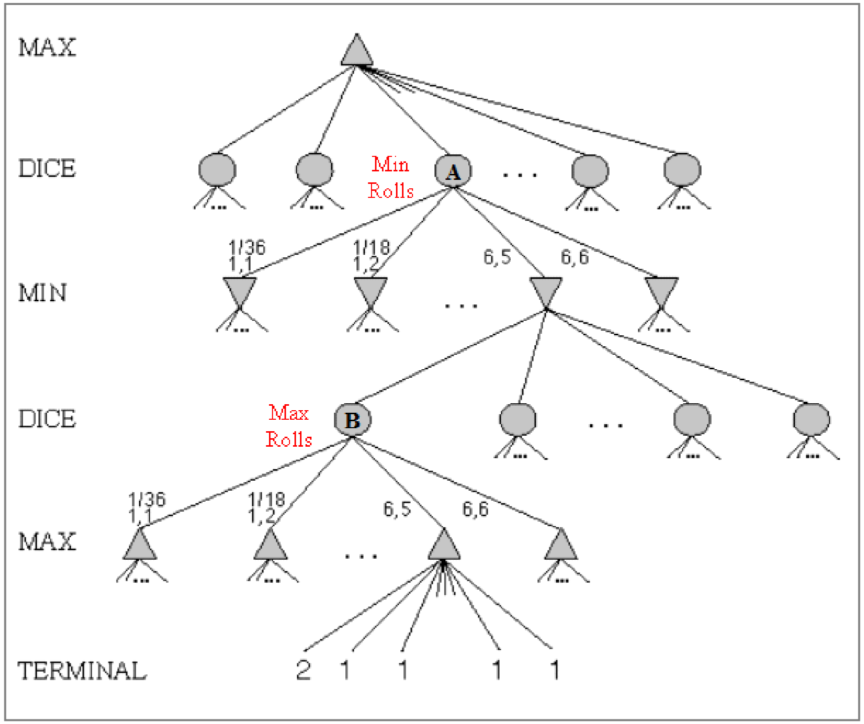
Jocuri nedeterministe

- Arborii includ noduri-șansă (cercurile din figură), care reprezintă evenimente aleatorii
- Pentru un eveniment aleatoriu cu *n* rezultate posibile, fiecare nod-șansă are *n* fii distincți, iar fiecare fiu are asociată o probabilitate
- De exemplu, pentru 2 zaruri sunt posibile 21 de rezultate

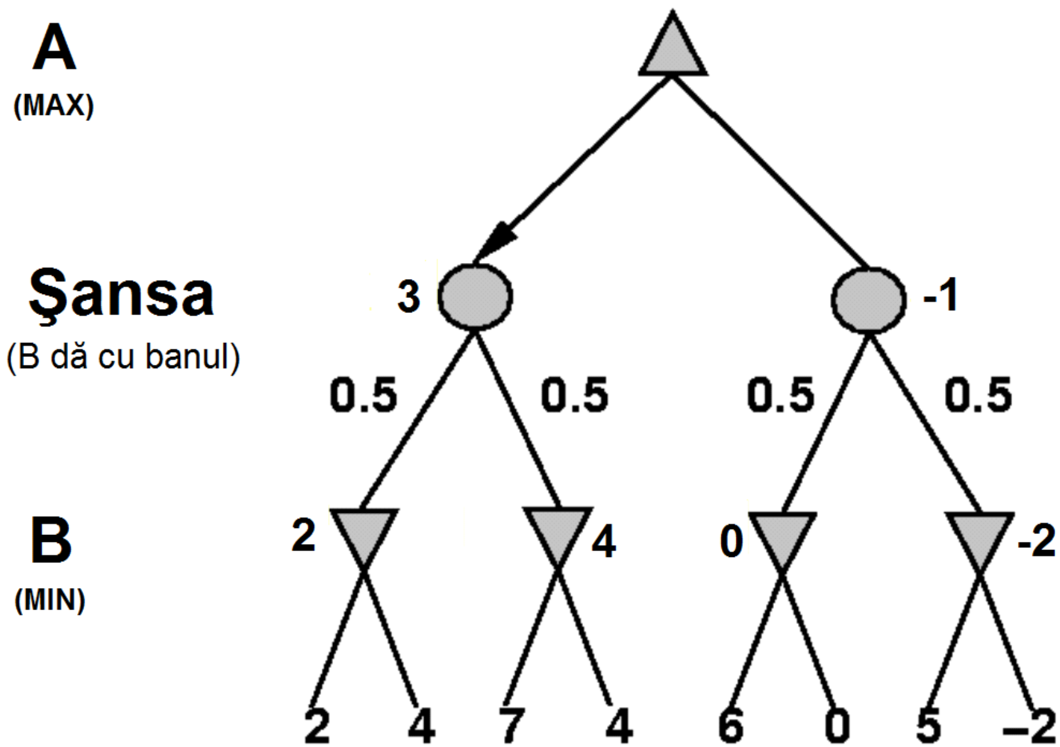


Arbori de joc cu noduri-șansă

- Se folosește minimax pentru a calcula valorile nodurilor MAX și MIN
- Se folosesc valorile așteptate pentru nodurile-șansă
- Pentru nodurile-șansă la un nod MIN
  - $expectimin(A) = \Sigma_i(P(d_i) \cdot minvalue(i))$
- Pentru nodurile-șansă la un nod MAX
  - $expectimax(B) = \Sigma_i(P(d_i) \cdot maxvalue(i))$



Exemplu



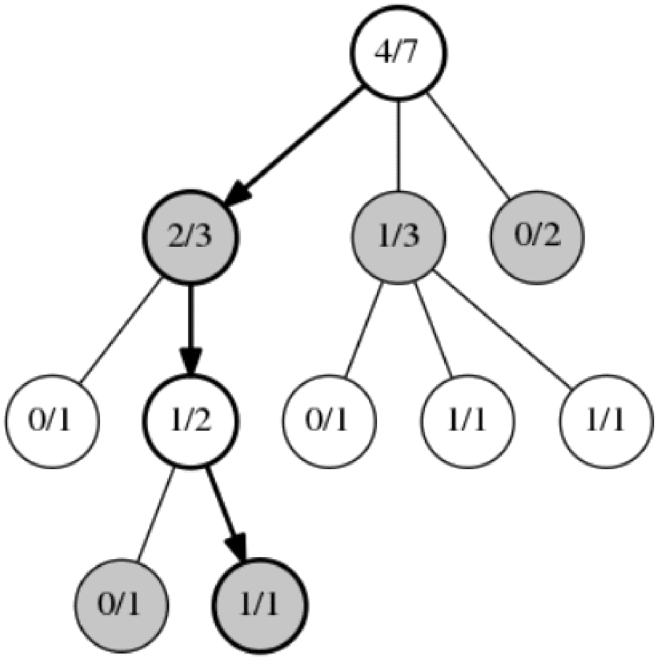
Elemente de teoria jocurilor (I)

Căutarea pe arbori Monte Carlo

- Căutarea pe arbori Monte Carlo (*Monte Carlo Tree Search*, MCTS) este o metodă stohastică de căutare a soluțiilor în jocuri cu factori mari de ramificare
- A fost folosită cu succes de Google DeepMind pentru AlphaGo, în combinație cu rețele neuronale profunde și metode de învățare cu întărire

Faza 1. Selecția

- Culorile nodurilor din figură reprezintă cei doi jucători
- Se presupune că algoritmul s-a aplicat deja de câteva ori
- Fiecare nod conține numărul de victorii / numărul de jocuri în care s-a trecut prin el
- Mutările selectate de algoritmul UCB1 (vezi slide-ul următor) sunt marcate cu linii îngroșate
- Selecția se aplică până în nodurile în care nu există statistici pentru toți copiii



Selecția UCB1

- engl. "Upper Confidence Bound"
- Din nodul curent, se selectează acțiunea (nodul fiu) *i* care maximizează următoarea expresie:

$$\frac{w_i}{n_i} + \sqrt{\frac{2 \ln n}{n_i}}$$

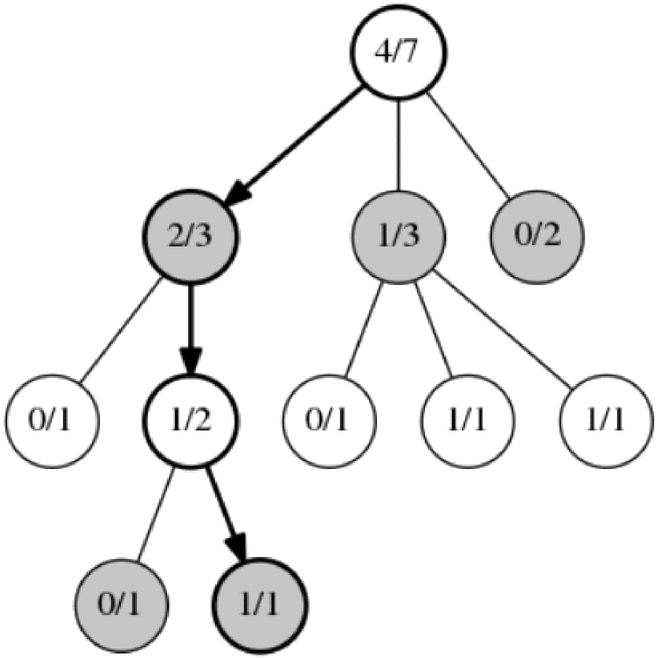
- unde: w_i este numărul de victorii în fiul *i*, n_i este numărul de simulări în fiul *i*, iar *n* este numărul de simulări în nodul curent
- Primul termen reprezintă exploatarea, al doilea explorarea
- MCTS cu selecția acțiunii UCB1 descoperă secvența optimă de mutări
- În loc de 2, poate fi altă valoare. Astfel, se ponderează explorarea și exploatarea

Exemplu: selecția UCB1

- Nodul 2/3: $$\frac{2}{3} + \sqrt{\frac{2 \cdot \ln 7}{3}} = 1.806$$

- Nodul 1/3: $$\frac{1}{3} + \sqrt{\frac{2 \cdot \ln 7}{3}} = 1.472$$

- Nodul 0/2: $$\frac{0}{2} + \sqrt{\frac{2 \cdot \ln 7}{2}} = 1.395$$

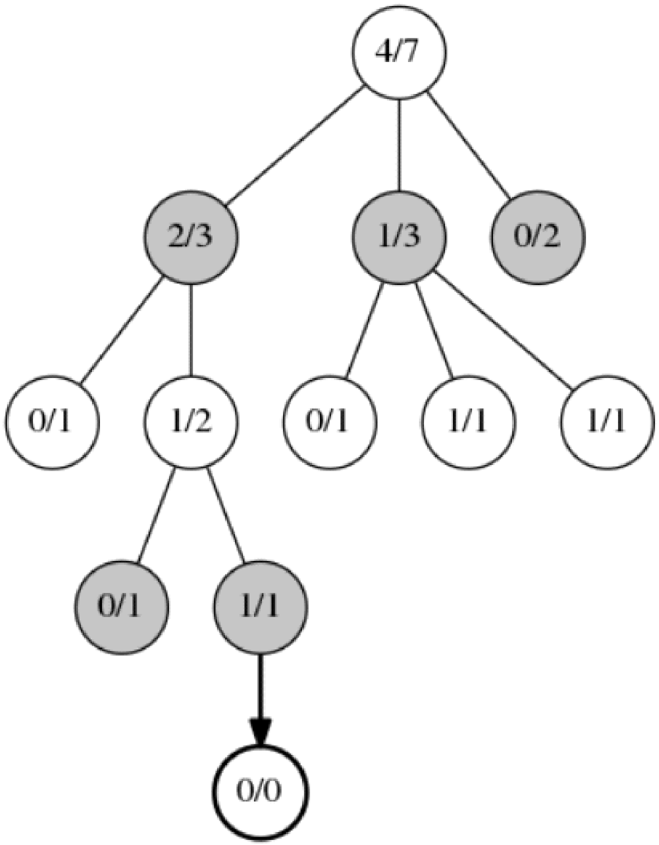


Noduri terminale

- Selecția poate alege un nod care reprezintă deja o stare terminală
- Dacă starea reprezintă o înfrângere, nodului i se poate da o valoare foarte mică (sau -∞), pentru a nu mai fi ales data viitoare
- Dacă starea reprezintă o victorie, nodului i se poate da o valoare foarte mare (sau +∞), iar părintelui său imediat o valoare foarte mică (sau -∞)

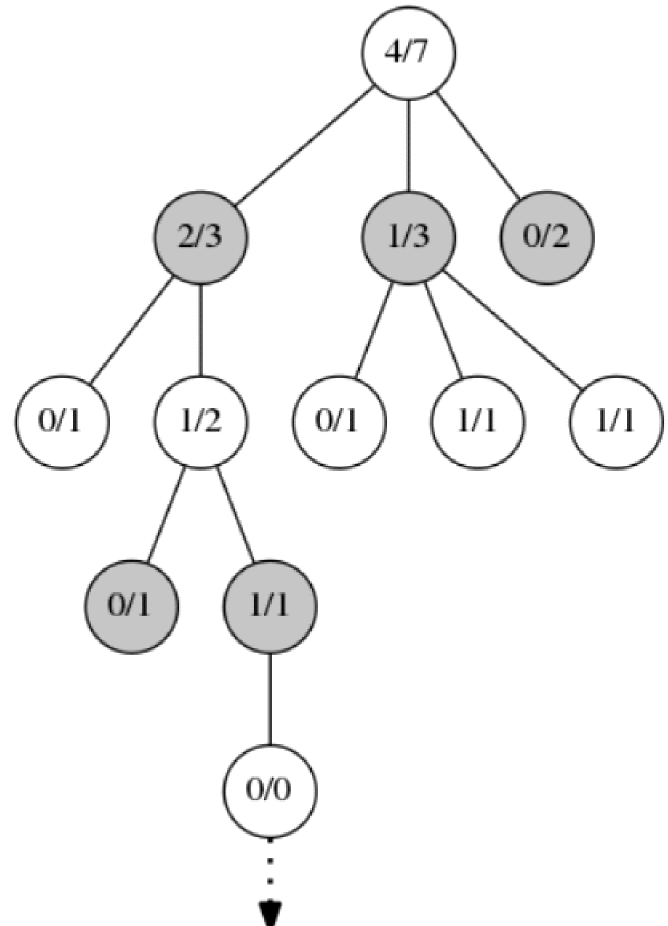
Faza 2. Expandarea

- Expandarea se aplică atunci când nu se mai poate aplica selecția
- Este selectat în mod aleatoriu un nod frunză încă nevizitat și se adaugă câte o nouă înregistrare de statistici (0/0) pentru fiecare fiu
- Nodurile cu 0/0 vor fi selectate când se va ajunge la ele în viitor, deoarece se consideră că au valoarea UCB1 infinită



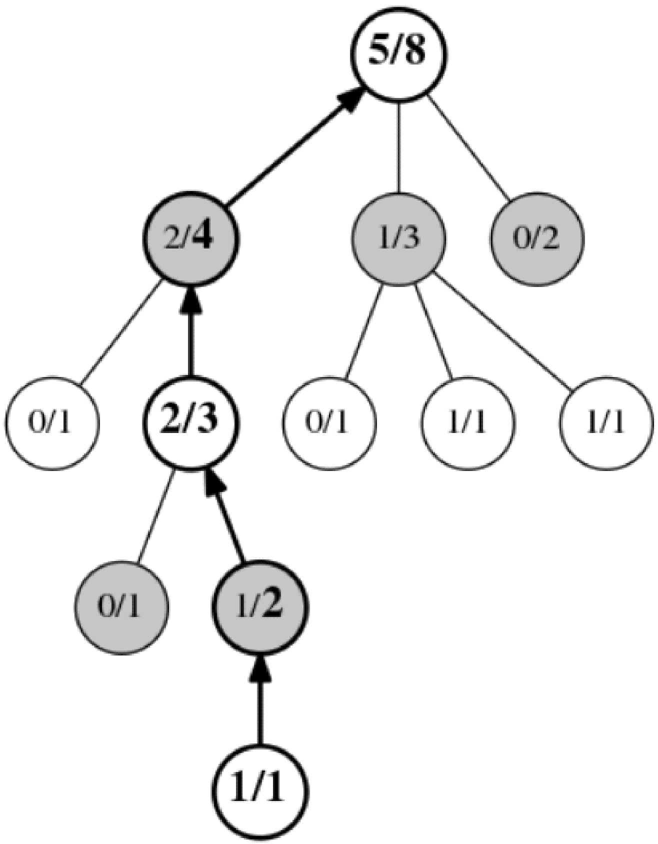
Faza 3. Simularea

- După expandare, începe simularea Monte Carlo, în care se aleg mutări în mod aleatoriu până se ajunge într-o stare terminală (de exemplu, victorie sau înfrângere)
- O astfel de simulare se mai numește *rollout* sau *playout*
- Uneori, în locul căutării pur aleatorii, se pot folosi euristici de ponderare care să aleagă mutări considerate mai bune

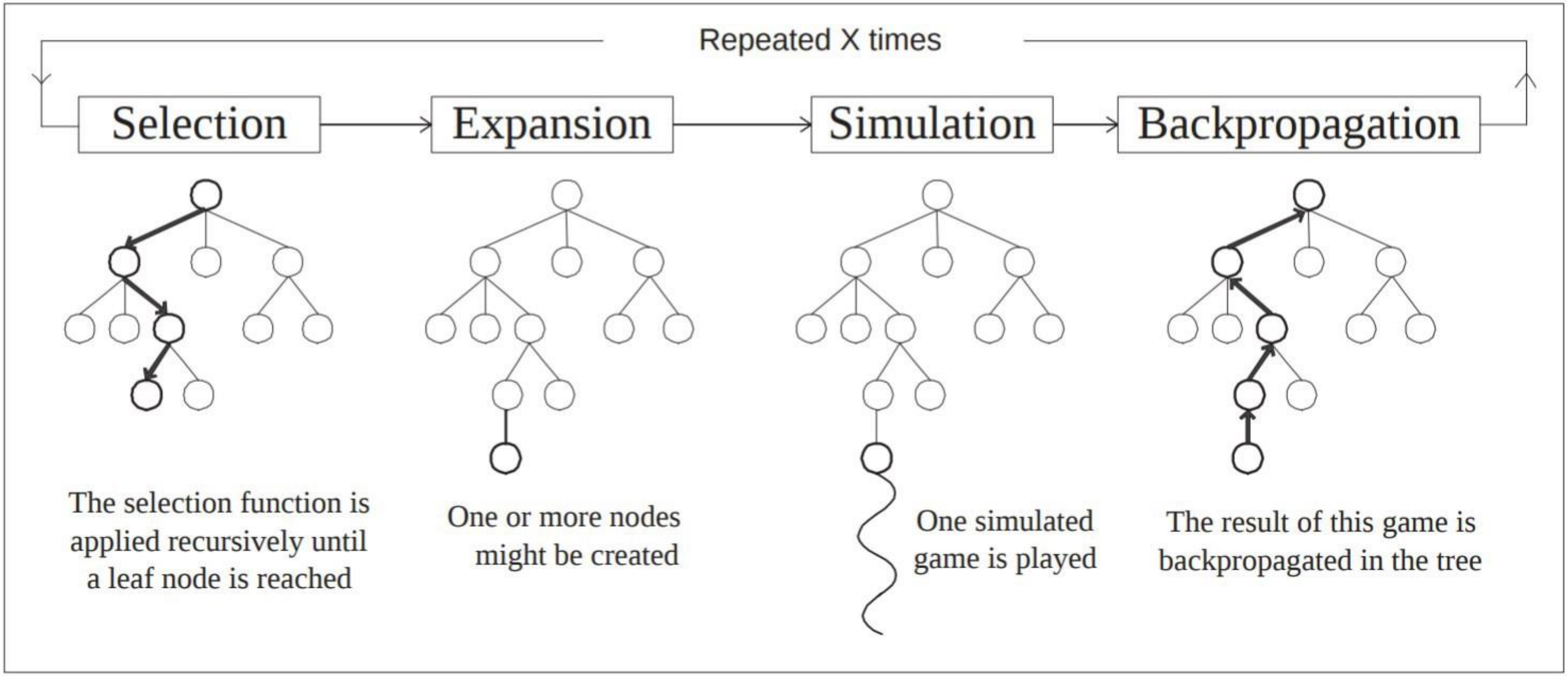


Faza 4. Actualizarea (sau retro-propagarea)

- După terminarea simulării, pentru toate pozițiile vizitate se incrementează numărul de jocuri și, dacă e cazul, numărul de victorii
- Victoria se incrementează numai la nodurile de pe nivelurile jucătorului câștigător
- Nu se actualizează nodurile parcurse în jos în simulare, ci doar cele pornind în sus de la nodul selectat (în exemplul considerat, noul nod expandat și toți predecesorii lui)



Algoritmul MCTS



Alegerea mutării

- După aplicarea algoritmului (în mod repetat), se alege mutarea cu cel mai mare număr de vizite (n_i în w_i / n_i), deoarece valoarea sa este cel mai bine estimată
- Întrucât a fost explorată cel mai mult, și valoarea sa propriu-zisă (w_i / n_i) ar trebui să fie mare

- După ce calculatorul și adversarul (omul) fac câte o mutare, la căutarea următoare se pot refolosi valorile din subarborele corespunzător ca valori inițiale

MCTS vs. minimax

- MCTS nu are nevoie de euristici
- Căutarea MCTS este asimetrică: explorarea arborelui converge către mutările mai bune
- MCTS este un algoritm *anytime*: la orice moment de timp, poate produce o estimare a mutării optime

Stadiul actual al programelor de jocuri

- Table (TD-Gammon): învățare cu întărire și rețele neuronale, top 3 mondial (1992)
- Dame (Chinook, 1995), Othello (Logistello, 1997): programele sunt mai bune decât oamenii
- Bridge (GIB): campion mondial (1998)
- Go (AlphaGo): l-a învins pe Ke Jie, cel mai bun jucător de go din lume (2017), variante: AlphaGo Zero, Alpha Zero
- Poker (Libratus): a învins patru dintre cei mai buni jucători din lume (2017)
- StarCraft (AlphaStar): a învins unul din cei mai buni jucători profesioniști (2019)
- Șah (Stockfish 12): 3665 puncte Elo (2020), Magnus Carlsen, cel mai mare punctaj uman: 2882

Elemente de teoria jocurilor (I)

1. Jocuri secvențiale
   1.1. Introducere și formalizare
   1.2. Algorimul minimax
   1.3. Retezarea alfa-beta
   1.4. Căutarea pe arbori Monte Carlo
2. Jocuri strategice
   2.1. Dominare
   2.2. Echilibrul Nash pur
3. Concluzii

Jocuri strategice

- Sunt diferite de jocurile secvențiale rezolvabile prin metodele descrise anterior
- Aceste „jocuri” reprezintă interacțiuni strategice între agenți/jucători raționali, care aleg *simultan* acțiuni diferite pentru a-și maximiza câștigul

Elementele unui joc

- Un joc strategic este o situație în care:
  - Există cel puțin 2 jucători/agenți
  - Fiecare agent are la dispoziție un număr de strategii posibile
  - Strategiile alese de fiecare agent determină rezultatul (*outcome*) jocului
  - Pentru fiecare rezultat, există un câștig (*payoff*) numeric pentru fiecare jucător

Dilema deținutului

- engl. “prisoner’s dilemma”
- Jucători
  - 2 deținuți
- Acțiuni
  - Deținutul 1: mărturisește sau neagă
  - Deținutul 2: mărturisește sau neagă
- Strategii
  - Deținuții își aleg acțiunile simultan, fără a cunoaște acțiunea celuilalt
- Rezultate
  - Numărul de ani de închisoare
- Câștigul
  - Mai puțini ani ⇒ câștig mai mare

Reprezentarea în forma normală (strategică)

- O matrice care conține agenții, strategiile și câștigurile
- Se presupune că jucătorii/agenții acționează simultan
- Pentru dilema deținutului:

		Agent 2	
		Denies	*Confesses*
Agent 1	*Denies*	-1, -1	-5, 0
	Confesses	0, -5	-3, -3

		Agent 2	
		Denies	*Confesses*
Agent 1	*Denies*	win-win	lose much-win much
	Confesses	win much-lose much	lose-lose

Aplicații ale jocurilor strategice

- Oriunde există interacțiuni strategice între agenți raționali
  - Economie
  - Strategii geo-politice
  - Psihologie
  - Sociologie
  - Rețele de calculatoare: rutarea, partajarea în rețele *peer-to-peer* etc.

Jocuri de sumă nulă

- Numite și „jocuri de sumă zero”
- Pentru orice rezultat al jocului, câștigurile jucătorilor au suma 0

$$\begin{bmatrix} 2 & -1 \\ 1 & -2 \end{bmatrix} \qquad \begin{bmatrix} 2 & -3 \\ 0 & 2 \\ -5 & 10 \end{bmatrix}$$

În limba română ar putea fi Laura și Cristi

Câștigul lui Rose (“rows”) = – câștigul lui Colin (“columns”)

Pentru un joc de sumă generală, sunt necesare perechi

Pentru un joc de sumă nulă, (2) este echivalent cu (2, –2)

Elemente de teoria jocurilor (I)

Dominare. Definiții

- O strategie *S* domină o strategie *T* (*T* este dominată de *S*) dacă orice rezultat al lui *S* este cel puțin la fel de bun ca rezultatul corespunzător al lui *T*
- Un agent rațional nu trebuie să joace niciodată o strategie dominată
- Dacă fiecare jucător are o strategie dominantă și o joacă, atunci combinația acestora și câștigurile corespunzătoare constituie echilibrul strategiilor dominante ale jocului

Exemple

Example 1. Consider the game with payoff table:

	C	D
A	(1,3)	(4,2)
B	(2,4)	(7,1)

Clearly B dominates A for Player 1 and C dominates D for Player 2. Thus (B, C) will be the dominant strategy equilibrium.

Exemple

Example 2. Consider the game with payoff table:

	C	D	E
~~A~~	~~(1,1)~~	~~(2,0)~~	(3,−1)
B	(2,1)	(4,3)	(2,0)

At first sight, it seems that the row player's strategies A and B do not dominate each other, so there will be no dominant strategy equilibrium. However, the game still has it. What we need is *the Principle of Higher Order Dominance*: we first cross out any dominated strategies for the players. In the resulting smaller game, some strategies may become dominated, even though they weren't in the original game. For this example, we first cross out strategy E because it is dominated by D. Once we did that, we find out that B will dominate A, so we cross out A. Then finally, we see that D dominates C, and we arrive at our dominant strategy equilibrium (B, D).

Exemple

Example 3. Consider the following 2-person zero-sum game:

	E	F	G	H
A	1	1	1	2
B	2	1	1	2
C	2	2	1	1
D	2	2	2	3

Can you find its dominant strategy equilibrium? The process of elimination should be like E, F, A, B, C, and H. Then finally we get the solution (D, G).

Echilibrul strategiilor dominante

$$\begin{bmatrix} 2 & -1 \\ 1 & -2 \end{bmatrix}$$

The entry -1 is the payoff to Rose if Rose plays his first strategy R_1 and Colin plays his second strategy C_2. In this case, the payoff to Colin is $-(-1) = 1$. More precisely, in this case, Rose will have to pay \$1 to Colin.

Notice that the game in Example 1 has a dominant strategy equilibrium. In fact, R_1 dominates R_2 while C_2 dominates C_1. Thus the dominant strategy equilibrium is (R_1, C_2). However, not all 2-by-2 zero-sum games have dominant strategy equilibrium

Elemente de teoria jocurilor (I)

1. Jocuri secvențiale
    1.1. Introducere și formalizare
    1.2. Algorimul minimax
    1.3. Retezarea alfa-beta
    1.4. Căutarea pe arbori Monte Carlo
2. Jocuri strategice
    2.1. Dominare
    2.2. Echilibrul Nash pur
3. Concluzii

Echilibrul Nash

- O combinație de strategii este un echilibru Nash dacă fiecare jucător își maximizează câștigul, date fiind strategiile folosite de ceilalți jucători
- Echilibrul Nash identifică acele combinații de strategii care sunt stabile, în sensul că fiecare jucător este mulțumit cu acțiunea aleasă, date fiind acțiunile celorlalți
- Comportamentul generat de un echilibru Nash este de așteptat să persiste în timp

Echilibrul Nash

- Echilibru Nash pentru o strategie pură

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$$

Câștigul (utilitatea) agentului *i*

Strategia din echilibrul Nash

Strategia agentului *i*

Strategiile celorlalți agenți cu excepția lui *i*

Deterministă, care nu implică probabilități

Echilibrul Nash

- Echilibrul Nash pentru o strategie pură

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$$

- Echilibrul Nash este strict dacă:

$$u_i(s_i^*, s_{-i}^*) > u_i(s_i, s_{-i}^*)$$

- Stările din care niciun jucător nu-și poate mări câștigul prin schimbarea unilaterală a strategiei

Calculul echilibrelor Nash pure

- Se evidențiază maximele pe linii pentru primul jucător cu {
- Se evidențiază maximele pe coloane pentru al doilea jucător cu }
- Stările încadrate de { } sunt echilibre Nash pure

		Deținutul 2	
		Neagă	*Mărturisește*
Deținutul 1	*Neagă*	-1, -1	-5, 0 }
	Mărturisește	{ 0, -5	{ -3, -3 }

Exemple

Dilema deținutului

		Agent 2	
		Denies	*Confesses*
Agent 1	*Denies*	-1, -1	-5, 0
	Confesses	0, -5	-3, -3

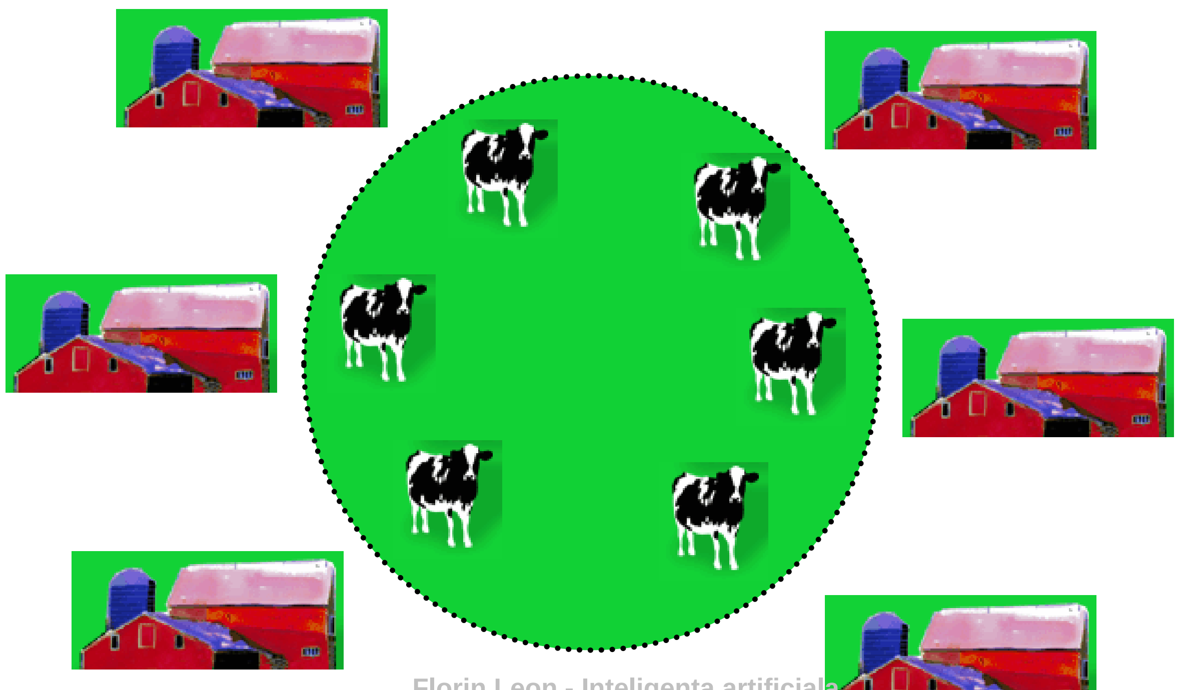
NE

Bătălia sexelor

		Mary	
		Football (A)	*Opera (B)*
John	*Football (A)*	2, 1	0, 0
	Opera (B)	0, 0	1, 2

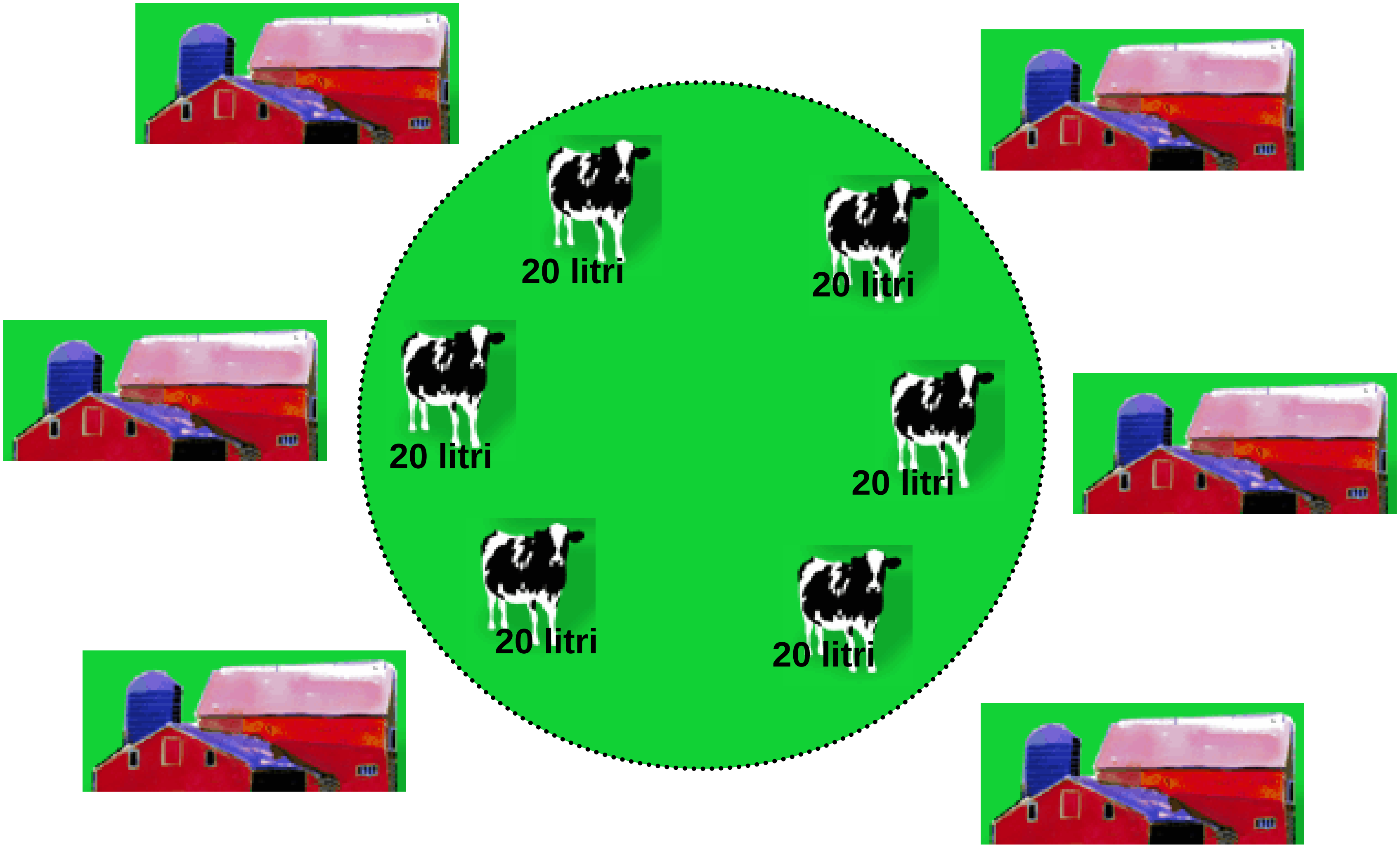
Tragedia pășunii comunale

- engl. "the tragedy of the commons"
- Pășunea este folosită în comun de 6 țărani, fiecare cu câte o vacă



Tragedia pășunii comunale

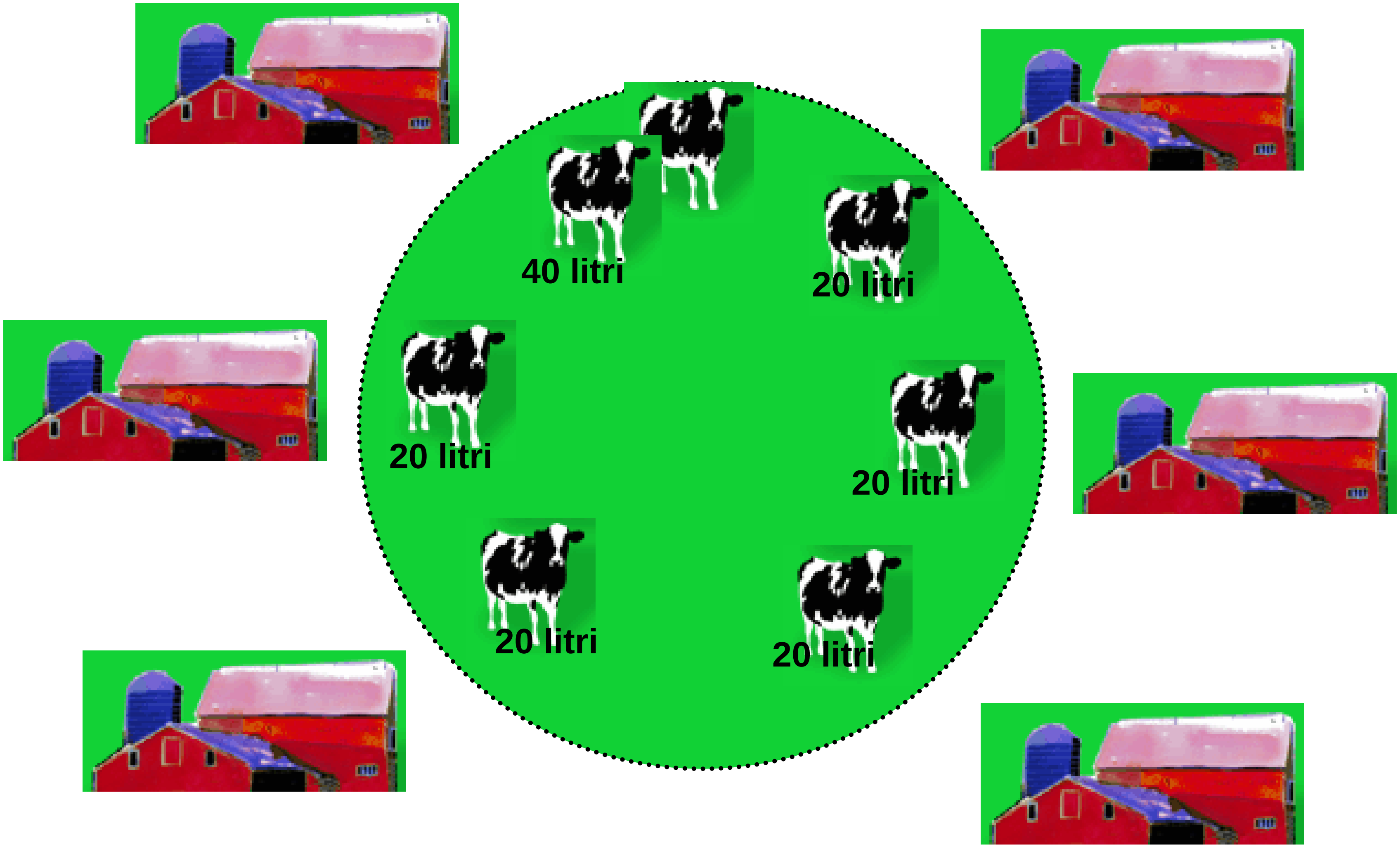
- Fiecare vacă dă 20 de litri de lapte pe zi
- Capacitatea pășunii este de 8 vaci
- Pentru fiecare vacă peste 8, producția de lapte scade cu 2 litri
  - Există mai puțină iarbă de păscut pentru fiecare vacă, deci și mai puțin lapte



Producția zilnică totală de lapte: 120 litri

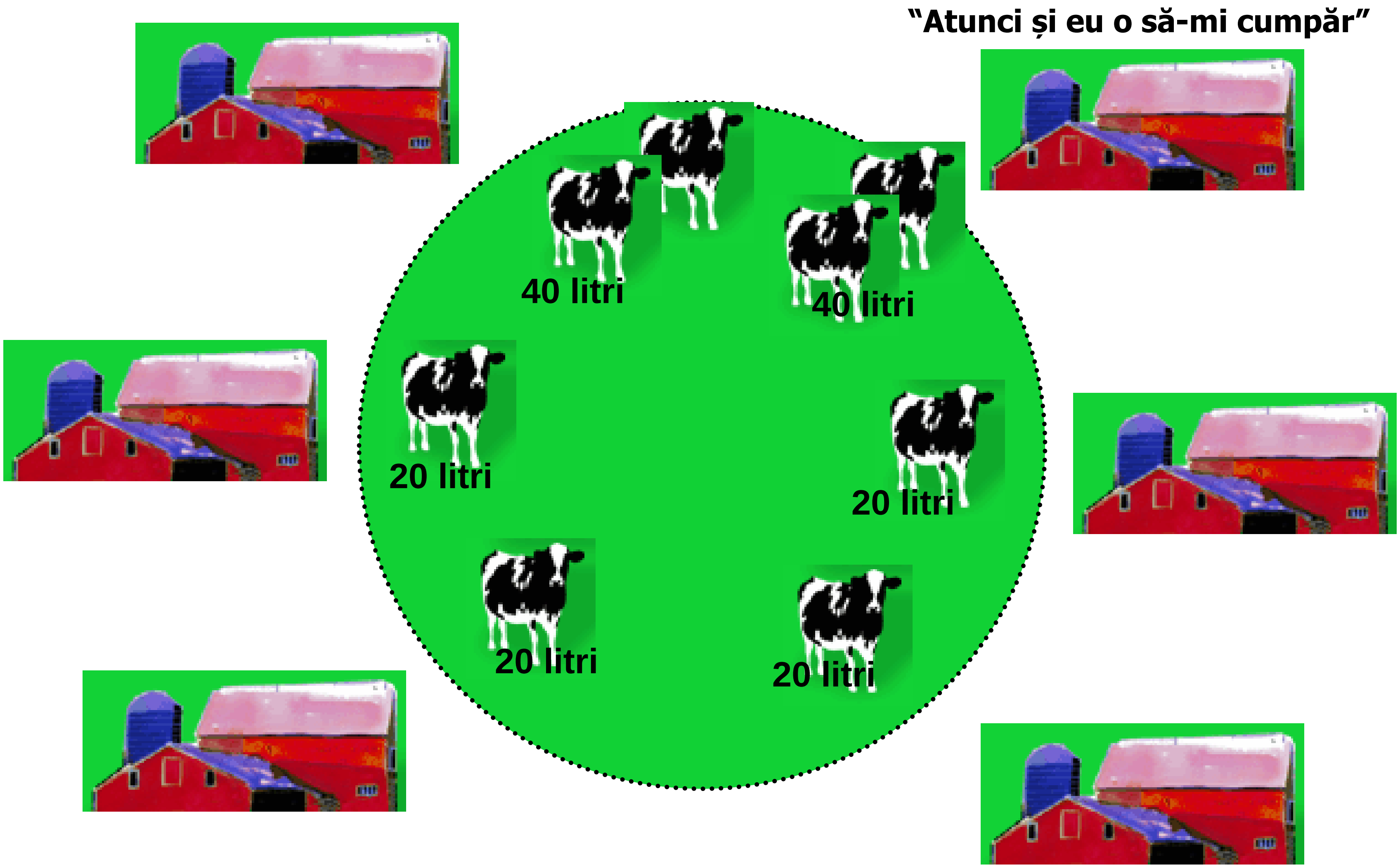
Țăranii vor să-și maximizeze producția de lapte

"O să cumpăr încă o vacă"

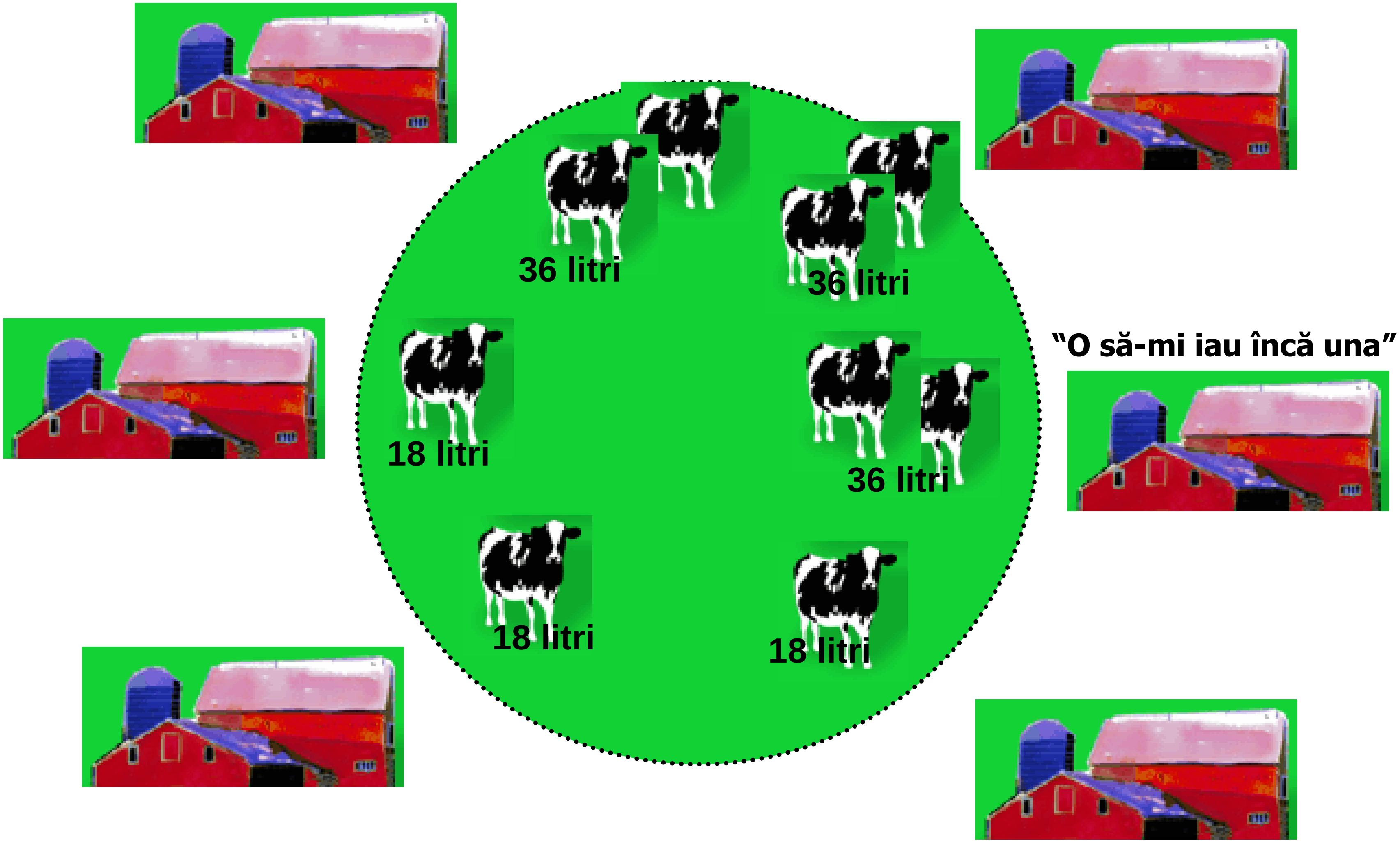


Producția zilnică totală de lapte: 140 litri (7 vaci)

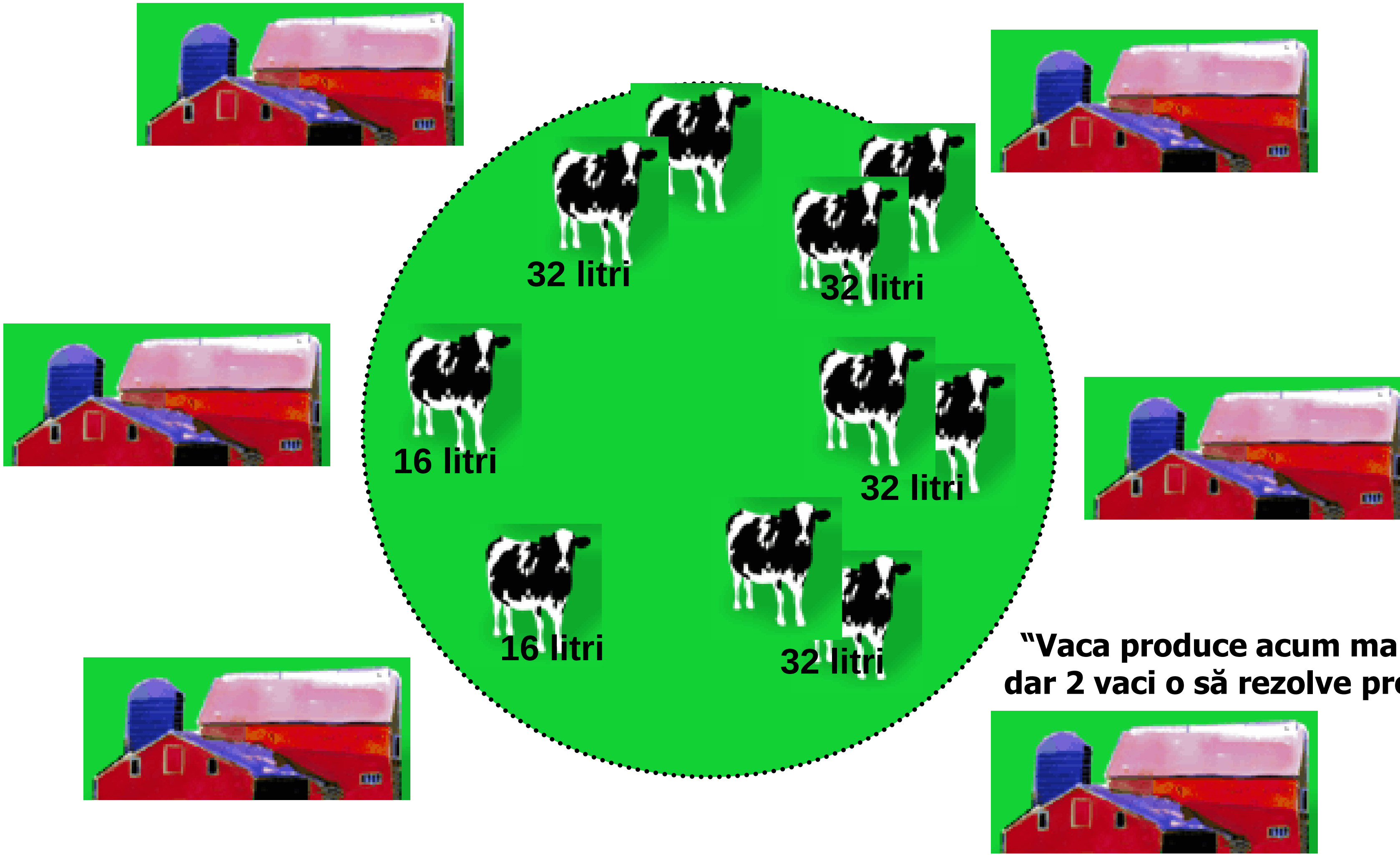
Acum s-a atins capacitatea maximă a pășunii. Dar țăranii nu se opresc.



Producția zilnică totală de lapte: 160 litri (8 vaci)

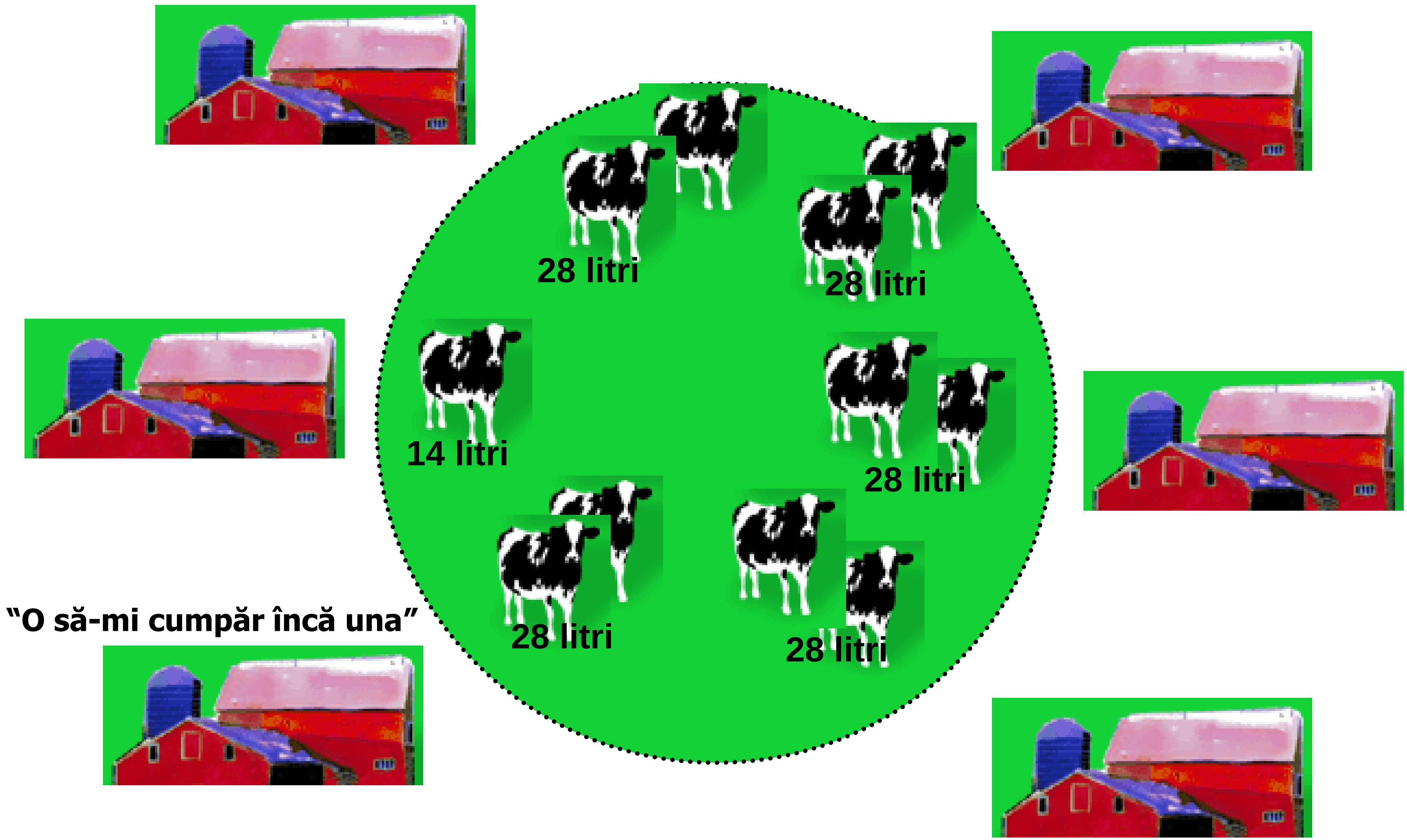


Producția zilnică totală de lapte: 162 litri (9 vaci)

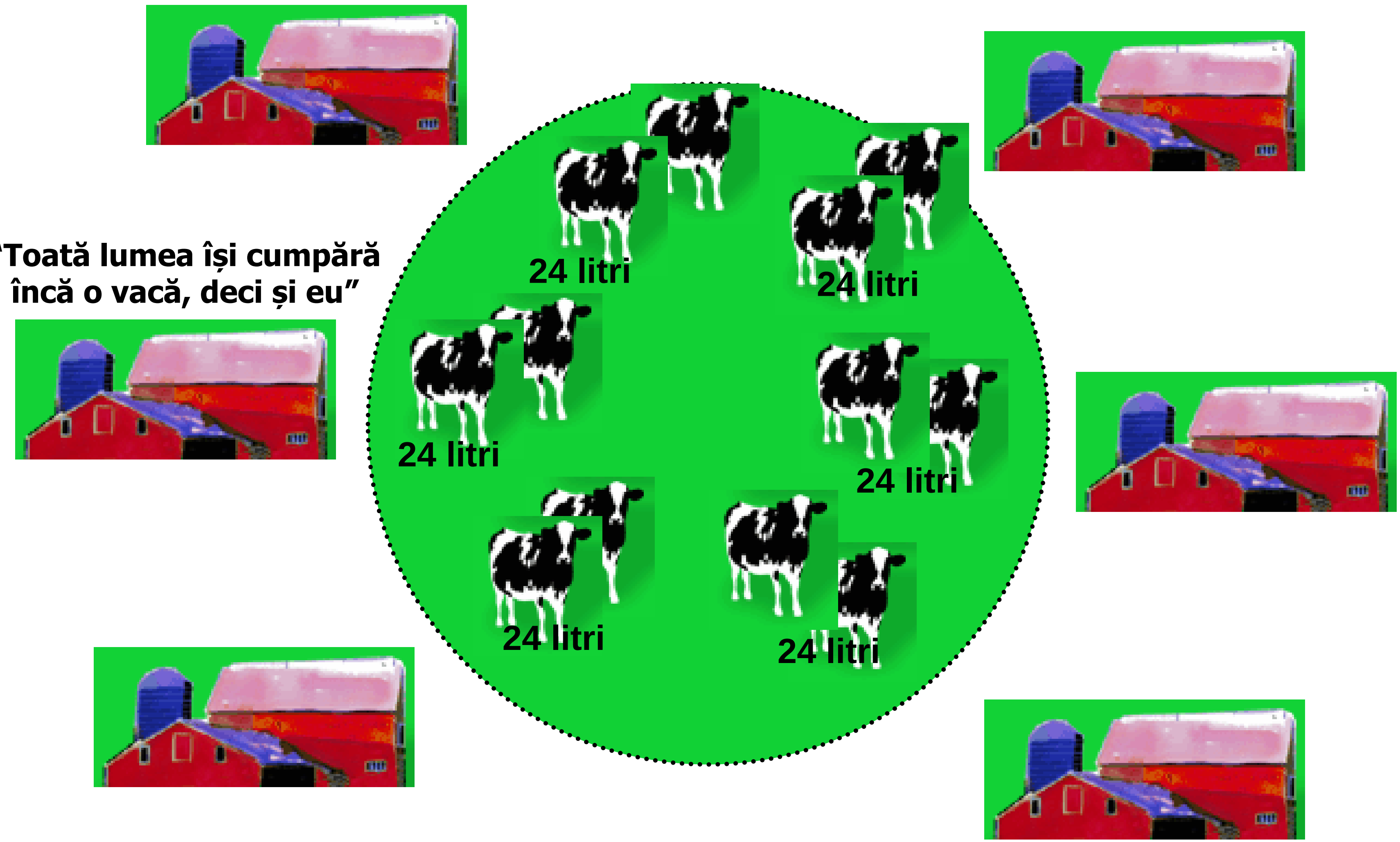


“Vaca produce acum mai puțin, dar 2 vaci o să rezolve problema”

Producția zilnică totală de lapte: 160 litri (10 vaci)

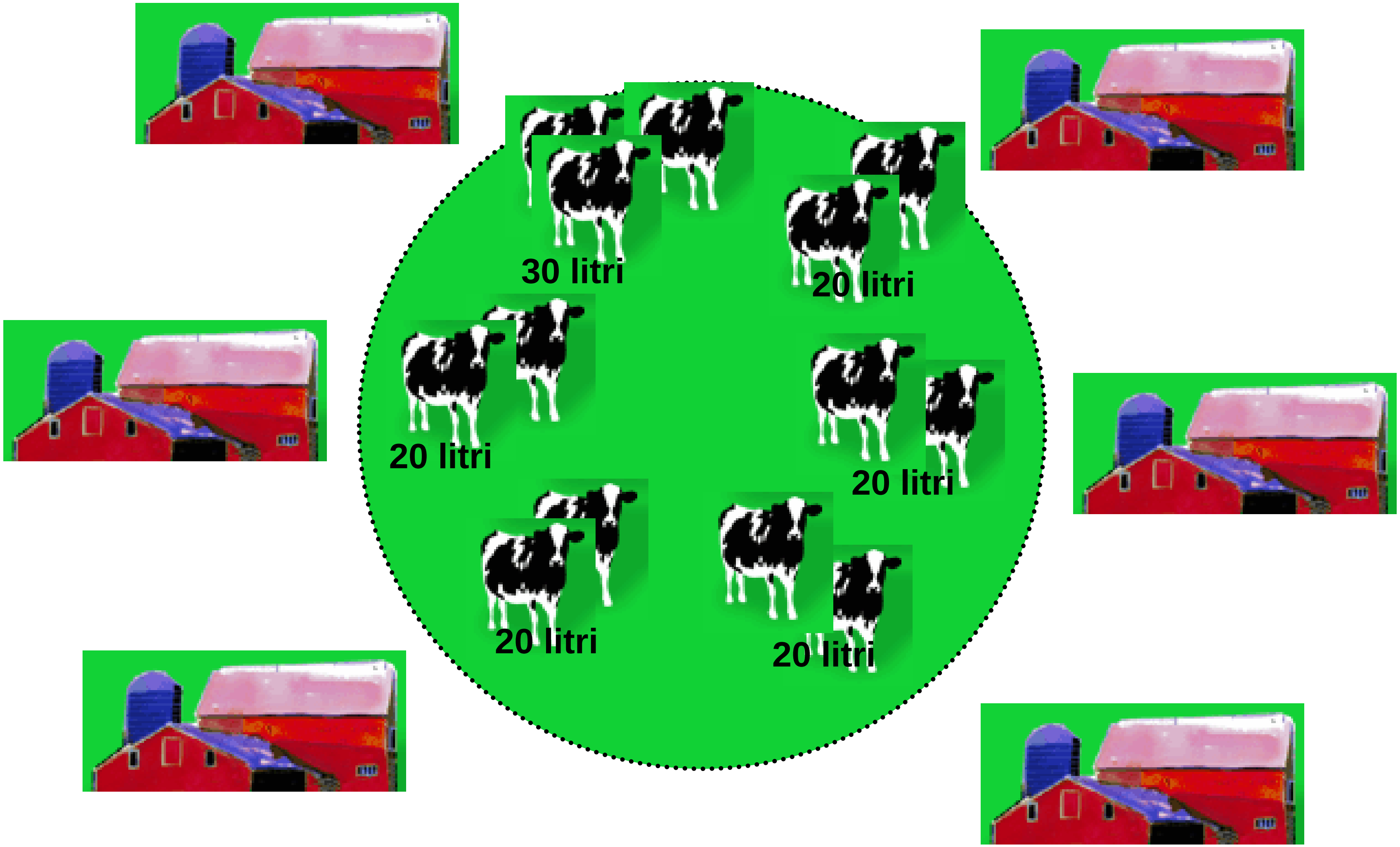


Producția zilnică totală de lapte: 154 litri (11 vaci)

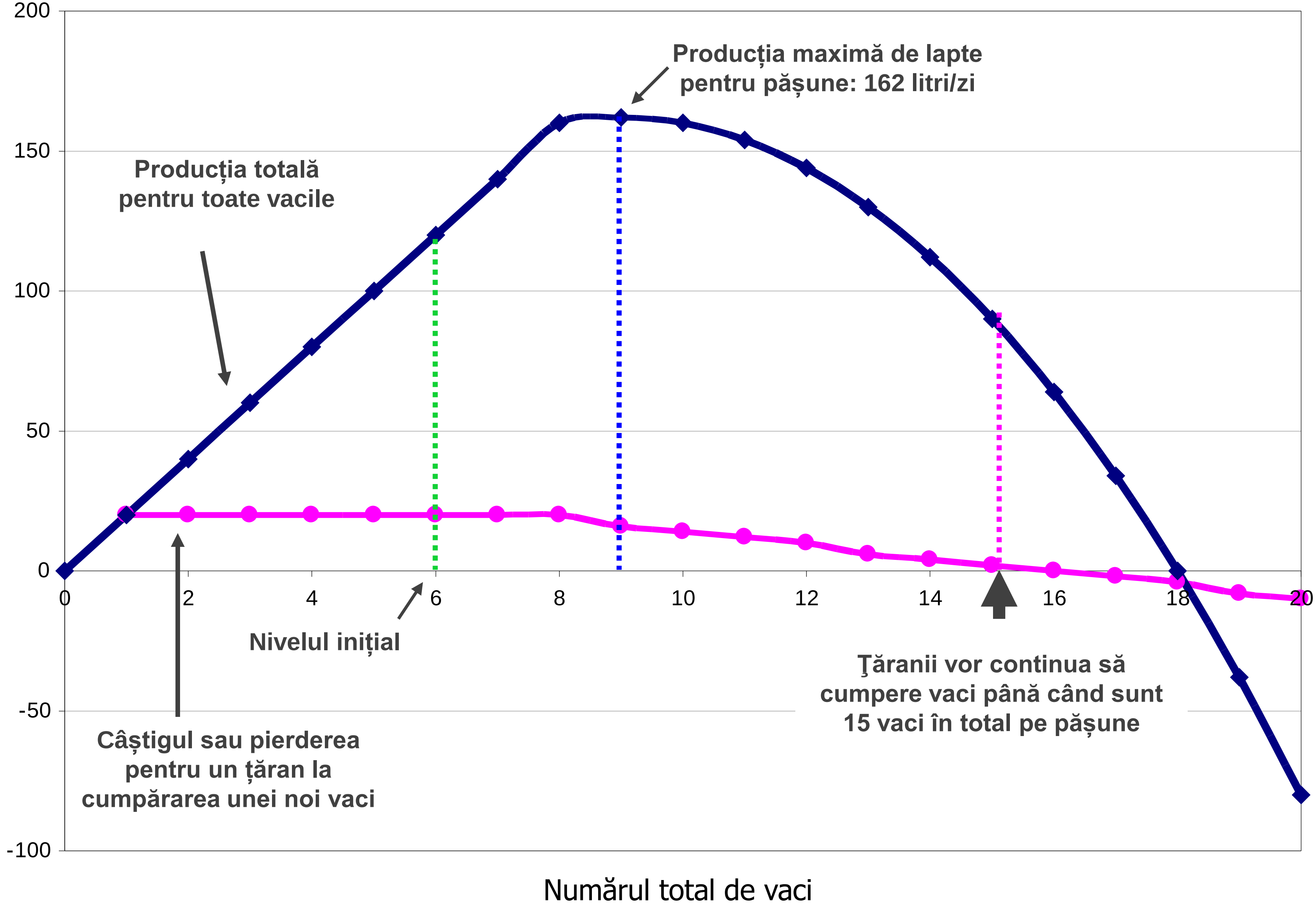


Producția zilnică totală de lapte: 144 litri (12 vaci)

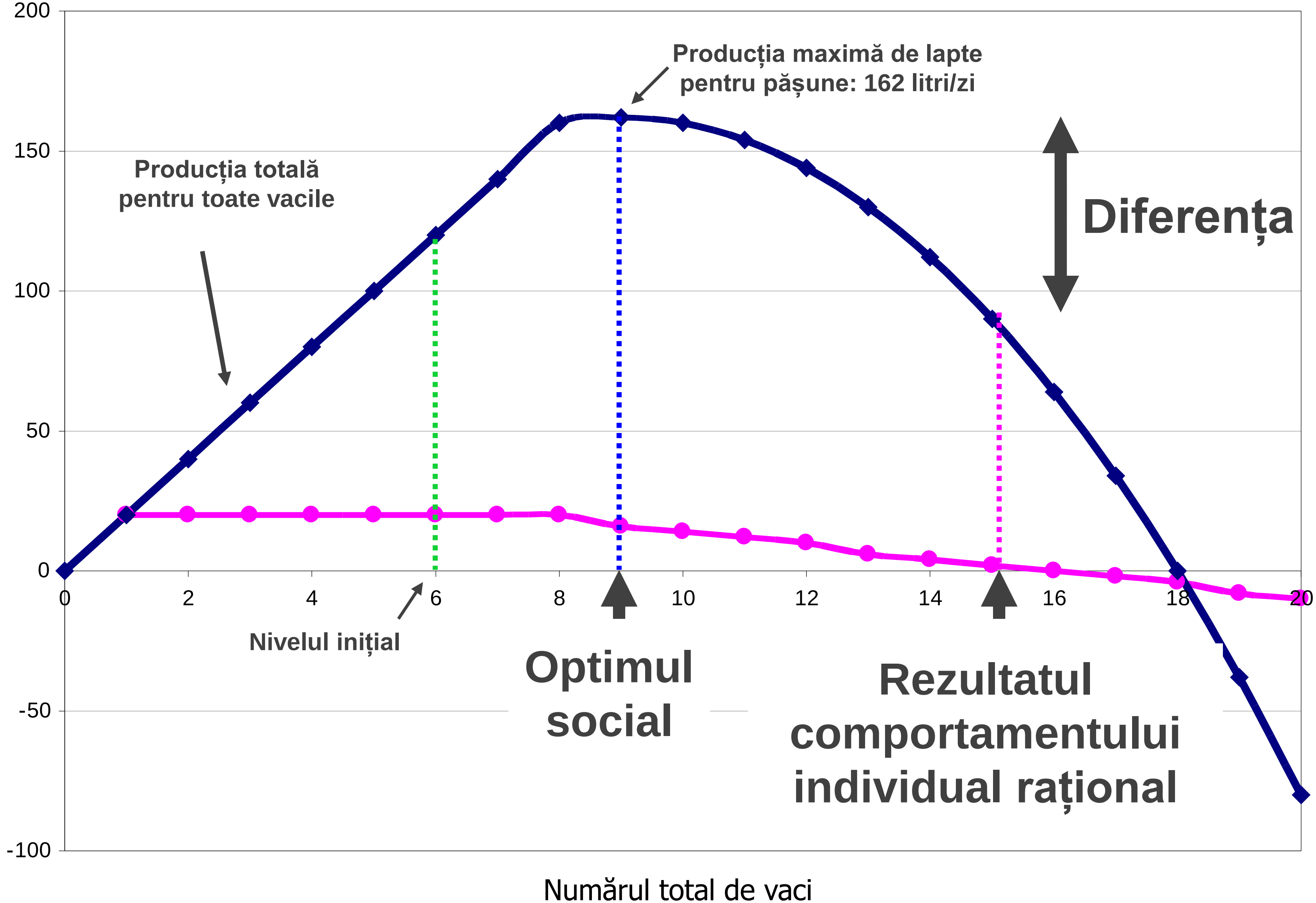
“Încă pot crește producția dacă iau și a treia vacă”



Producția zilnică totală de lapte: 130 litri (10 vaci)



200
150
100
50
0
-50
-100
0
2
4
6
8
10
12
14
16
18
20
Producția maximă de lapte
pentru pășune: 162 litri/zi
Producția totală
pentru toate vacile
Nivelul inițial
Ţăranii vor continua să
cumpere vaci până când sunt
15 vaci în total pe pășune
Câștigul sau pierderea
pentru un țăran la
cumpărarea unei noi vaci
Numărul total de vaci


200
150
100
50
0
-50
-100
0
2
4
6
8
10
12
14
16
18
20
Producția maximă de lapte
pentru pășune: 162 litri/zi
Producția totală
pentru toate vacile
Diferența
Nivelul inițial
Optimul
social
Rezultatul
comportamentului
individual rațional
Numărul total de vaci

Soluții?

- Acord de cooperare între țărani
  - Împărțirea profitului pentru cele 3 vaci în plus
- Consolidare
  - O firmă gestionează toate vacile și devine un singur centru de profit
- Reglementări ale „statului”
  - Stabilirea unui număr maxim de vaci pe pășune sau impunerea redistribuirii profitului

- Proiectarea mecanismelor (*mechanism design*)
  - Stimulente și penalizări pentru agenții individuali astfel încât să fie tentați să atingă optimul social
  - Problemă încă deschisă

Beneficiul social și beneficiul individual

- Partajarea (*sharing*) în rețele P2P
- Poluarea
- ... în general, managementul resurselor din proprietatea comună

Concluzii

- Teoria jocurilor analizează modele abtracte de interacțiuni multi-agent. Domeniul tratează mai multe tipuri de „jocuri” și situații de decizie
- În acest curs, au fost prezente jocuri secvențiale și strategice
- Minimax este un algoritm recursiv care determină mutarea optimă a unui jucător, dată fiind o anumită adâncime de căutare în arborele jocului
- Retezarea alfa-beta este o optimizare des utilizată pentru algoritmul minimax
- O combinație de strategii este un echilibru Nash dacă fiecare jucător își maximizează câștigul, date fiind strategiile folosite de ceilalți jucători. Strategiile din echilibrul Nash sunt stabile