Proiect SmarTest - Inteligenta Artificiala

- Membrii echipei:

  Matcovici Georgiana
  Gheonea Naomi-Denisa
  Stoleriu Denis-Matei
  Tugui Tudor

- Solutii tehnice posibile:

1. Generarea întrebărilor - Naomi

În literatura de specialitate, metodele de AQG pot fi împărțite în trei mari categorii [2]:

1. **Metode bazate pe reguli (rule-based / template-based)**
2. **Metode bazate pe ontologii**
3. **Metode bazate pe învățare automată (neuronale)**

Fiecare categorie prezintă avantaje și limitări, în funcție de complexitatea datelor și de domeniul de aplicare.

1.1 Metode bazate pe reguli și șabloane

Metodele bazate pe reguli (rule-based) și șabloane (template-based) reprezintă una dintre cele mai vechi și mai accesibile abordări în generarea automată a întrebărilor. Aceste metode se bazează pe reguli lingvistice predefinite, care transformă propozițiile declarative dintr-un text în întrebări, prin aplicarea unor modele sintactice și semantice prestabilite [2].

Într-un sistem tipic de acest tip, procesul de generare include trei etape principale:

1. Analiza lingvistică a textului – identificarea structurii gramaticale, a părților de vorbire și a relațiilor dintre termeni (subiect, predicat, complement).

2. Selectarea informației relevante – alegerea fragmentului de text din care se va construi întrebarea (de exemplu, o propoziție simplă care conține o definiție, o cauză, o proprietate sau o relație logică).

3. Aplicarea șablonului de transformare – folosirea unui model de tip *pattern → întrebare*, în care anumite elemente sunt înlocuite automat pentru a obține o propoziție interogativă.

De exemplu, o regulă simplă pentru propozițiile de tip „X este Y” poate fi formulată ca:

> **Pattern:** `<SUBIECT> este <PREDICAT>`
> **Transformare:** „Ce este <SUBIECT>?”

Astfel, din enunțul *„Algoritmul A* este o metodă euristică de căutare optimă”*, sistemul poate genera întrebarea „Ce este algoritmul A?”*.

Alte exemple frecvente de reguli lingvistice includ:

- Pentru propoziții care exprimă o cauză:
  „X determină Y” → „Ce determină X?” sau *„Ce determină Y?”*

- Pentru relații de apartenență:
  „X face parte din Y” → „Din ce face parte X?”

- Pentru propoziții comparative:
  „X este mai rapid decât Y” → „Care este diferența dintre X și Y?”

Aceste reguli pot fi combinate cu expresii regulate (regex) pentru a recunoaște automat tipul de întrebare și pentru a selecta forma corectă a interogativului (ex: „ce”, „care”, „cum”, „de ce”).
De exemplu, expresia `r"ce\s+este"` identifică întrebări de definiție, iar `r"diferen[ţt]a\s+dintre"` recunoaște întrebări de tip comparativ.

Unele sisteme rule-based moderne folosesc instrumente NLP precum analizoare sintactice (POS taggers) sau dependențe gramaticale (dependency parsing) pentru a extrage structuri mai complexe. În astfel de cazuri, întrebările nu sunt construite doar pe baza șabloanelor rigide, ci și a relațiilor semantice din frază. De exemplu, un sistem poate detecta că *„A* folosește o

funcție de cost”* implică o relație de tip *uses*, generând întrebarea *„Ce funcție de cost folosește algoritmul A*?”*.

1.2 Metode bazate pe ontologii

Metodele bazate pe ontologii reprezintă o abordare structurată și formalizată a generării automate a întrebărilor, bazată pe reprezentarea explicită a **cunoștințelor dintr-un domeniu restrâns**. O **ontologie** descrie conceptele fundamentale, proprietățile și relațiile dintre ele, permițând sistemului să formuleze întrebări coerente din punct de vedere logic și semantic [1].

Sistemul **Medi-Test** [1] este un exemplu clasic de aplicație a metodelor ontologice. Acesta generează întrebări medicale pornind dintr-o ontologie care definește concepte (ex. boli, simptome, tratamente) și relații semantice (*caused_by*, *treated_with*).

Prin analogie, în domeniul AI, conceptele pot fi algoritmi și metode, iar relațiile descriu modul în care aceștia interacționează.

De exemplu:

- „A* is_a algoritm de căutare euristică”
- „Minimax related_to Alpha-Beta pruning”
- „Backtracking uses MRV”

Astfel, fiecare întrebare generată derivă dintr-o relație semantică clar definită.

Fluxul de generare bazat pe ontologie urmează etapele:

1. **Construirea ontologiei domeniului**
   Ontologia AI este reprezentată sub formă de fișier JSON sau OWL, conținând:
   - concepte principale: *n-queens*, *Hanoi*, *Backtracking*, *Minimax*, *Nash equilibrium*;
   - relații semantice între concepte:
     - *is_a*: „Backtracking is_a algoritm de căutare”
     - *related_to*: „Minimax related_to Alpha-Beta pruning”
     - *used_in*: „Forward Checking used_in CSP”

- tipul problemei asociate (Search, CSP, Game Theory, Adversarial Search).

2. **Definirea tipurilor de întrebări pentru fiecare categorie**
   Pentru fiecare capitol sau tip de problemă din AI, se definesc **șabloane semantice de întrebare**.

a) Search problem identification
Relații: *is_a*, *applied_to*

- Exemplu: „Dintre metodele BFS, DFS și Backtracking, care este cea mai potrivită pentru rezolvarea problemei n-queens?”

- Șablon: „Pentru instanța [X], care strategie de căutare este cea mai potrivită?”

b) Game theory (normal form)
Relații: *has_equilibrium*, *analyzed_by*

- Exemplu: „Pentru jocul dat în formă normală, există vreun echilibru Nash pur?”

- Șablon: „Pentru jocul definit prin matricea de payoff [M], determină dacă există un echilibru Nash pur.”

c) Constraint satisfaction (CSP)
Relații: *solved_by*, *optimized_by*

- Exemplu: „Care va fi asignarea finală a variabilelor dacă utilizăm Backtracking cu MRV?”

- Șablon: „Dată fiind o problemă CSP cu variabile [V], domenii [D] și constrângeri [C], care este soluția finală dacă se aplică [algoritm + optimizare]?”

d) Adversarial search (Minimax)
Relații: *related_to*, *evaluated_by*

- Exemplu: „Pentru arborele de joc dat, care va fi valoarea rădăcinii și câte noduri vor fi evaluate aplicând Minimax cu Alpha-Beta pruning?”

- Șablon: „Pentru structura de joc [T], determină valoarea nodului rădăcină și numărul de frunze evaluate prin [algoritm].”

Pornind de la ontologie, sistemul parcurge fiecare concept și generează automat întrebări completând șabloanele corespunzătoare. De exemplu:

- Relație *is_a(backtracking, algoritm de căutare)* → „Ce este algoritmul Backtracking și pentru ce tipuri de probleme se utilizează?”

- Relație *related_to(minimax, alpha-beta pruning)* → „Care este legătura dintre algoritmii Minimax și Alpha-Beta pruning?”

- Relație *optimized_by(csp, MRV)* → „Cum influențează utilizarea euristicii MRV performanța algoritmului Backtracking într-o problemă CSP?”

După generare, întrebările sunt verificate gramatical și semantic pentru coerență și redundanță.

1.3 Metode neuronale

Metodele neuronale de generare automată a întrebărilor reprezintă direcția modernă și cea mai complexă în domeniul Automatic Question Generation (AQG). Acestea utilizează modele de învățare profundă (deep learning) capabile să învețe automat tiparele lingvistice și semantice din date, fără a depinde de reguli explicite sau ontologii predefinite [2].

Principiul de bază al acestor metode constă în modelarea sarcinii de generare a întrebărilor ca o problemă de traducere automată (machine translation), în care sistemul „traduce” o propoziție declarativă (sau un concept) într-o întrebare corespunzătoare. De exemplu, propoziția „Algoritmul A* utilizează o funcție de cost pentru a estima distanța până la scop” poate fi „tradusă” în întrebarea „Care este rolul funcției de cost în algoritmul A*?”.

Există mai multe tipuri de modele folosite pentru AQG, fiecare cu particularitățile sale.

a) Modele Seq2Seq (Sequence-to-Sequence)

Primele abordări neuronale de AQG au fost bazate pe arhitectura encoder-decoder cu rețele neuronale recurente (RNN), de obicei folosind LSTM (Long Short-Term Memory) sau GRU (Gated Recurrent Unit).

- **Encoderul** citește textul sursă (de exemplu, o propoziție din curs sau o descriere a unui algoritm) și îl transformă într-o reprezentare vectorială latentă.

- **Decoderul** generează, pas cu pas, întrebarea în limbaj natural.

Aceste modele pot fi antrenate pe perechi de tip *(context, întrebare)* sau *(concept, întrebare)*, extrase din dataseturi educaționale sau manual adnotate.
De exemplu:

- Input: „Algoritmul Backtracking explorează toate soluțiile posibile revenind asupra deciziilor când apare o constrângere.”

- Output: „Cum funcționează algoritmul Backtracking?”

Avantajul acestor modele este capacitatea de a învăța structuri lingvistice complexe fără intervenție umană directă. Totuși, ele tind să aibă dificultăți în a genera întrebări precise din punct de vedere semantic, mai ales în domenii tehnice.

b) Modele Transformer

Modelele de tip **Transformer** au înlocuit aproape complet arhitecturile RNN, datorită capacității lor superioare de a capta relații contextuale între cuvinte, indiferent de poziție [2].

1.4. Fluxul propus pentru generarea întrebărilor bazată pe ontologie

Metoda propusă utilizează o **ontologie dedicată domeniului inteligenței artificiale**, care conține concepte, relații semantice și tipuri de întrebări asociate. Fluxul general este inspirat de arhitectura Medi-Test [1] și include următorii pași:

1. **Preprocesarea și curățarea datelor**
   Exemplele de examene din anii anteriori sunt prelucrate pentru consistență: eliminarea semnelor de punctuație, a caracterelor speciale și a cuvintelor de legătură, urmată de normalizare (litere mici), tokenizare și lematizare.
2. **Construirea ontologiei AI**
   Ontologia este reprezentată într-un format JSON sau OWL și include:
   - *Concepte principale*: „backtracking”, „minimax”, „Nash equilibrium”, etc.;
   - *Relații semantice*:
     - *is_a* – relație de generalizare (ex. „backtracking is_a algoritm de căutare”);
     - *related_to* – relații asociative (ex. „minimax related_to alpha-beta pruning”);
     - *part_of* sau *uses* – relații de compoziție.
3. Intrebările sunt construite automat pornind de la structura ontologică și de la relațiile dintre concepte.
   Exemple:

- Relație *is_a*: „Ce este algoritmul Backtracking și ce tip de probleme rezolvă?”
- Relație *related_to*: „Care este legătura dintre algoritmii Minimax și Alpha-Beta pruning?”
- Tip *definiție*: „Definește conceptul de echilibru Nash într-un joc cu doi jucători.”

4. Întrebările generate sunt verificate pentru coerență și redundanță, fiind eliminate cele duplicat. Opțional, o verificare umană poate asigura calitatea finală a întrebărilor.

Bibliografie:
[1] I. Pistol, D. Trandabăț, M. Răschip, “Medi-Test: Generating Tests from Medical Reference Texts,” *Data*, vol. 3, no. 4, p. 70, 2018.

[2] N. Mulla, P. Gharpure, “Automatic question generation: a review of methodologies, datasets, evaluation metrics, and applications,” *Progress in Artificial Intelligence*, vol. 12, pp. 1–32, 2023.

2. Generarea răspunsului corect - Matei

Inițial iau ca input întrebarea și cursurile, normalizez atât întrebarea cât și cursurile (transform caracterele în litere mici, elimin semnele de punctuație și conjuncțiile) pentru a avea potriviri cât mai precise.

Următorul pas este determinarea tipului de întrebare folosind Pattern-uri (regex) - (caut anumite cuvinte cheie din întrebare) :

- încerc să identific toate posibilitățile din întrebare pentru un răspuns cât mai precis.

De exemplu:

```
PATTERN_TIP_INTREBARE_REGEX = {
    "definitie": [
        r"ce\s+este",
        r"defin\w+",
        r"reprezintă",
        r"explică",
        r"explicați",
    ],
    "diferenta": [r"diferen[țt]a\s+dintre", r"deosebirea\s+dintre"],
    "caracteristici": [r"caracteristicile", r"avantajele", r"proprietățile"],
    "problema": [r"calculează", r"determină", r"rezolvă"],
}
```

Folosesc expresii regulate (regex) deoarece permit potrivirea mai multor forme gramaticale ale aceluiași cuvânt (ex: defin\w+ identifică „definiția”, „definește”, „definită”, „definirea”).

Astfel, sistemul poate recunoaște mai ușor tipul întrebării chiar dacă formularea diferă.

Întrebarea poate fi de teorie sau de rezolvare a unor probleme. O întrebare se încadreaza în categoria ***Teorie*** dacă întrebarea conține o definiție, diferență sau caracteristici. În caz contrar este de ***Calcul***.

După ce identific tipul de întrebare, voi identifica termenii, respectiv expresia pentru care va trebui să generez răspunsul (elimin cuvintele care determină tipul de întrebare, iar cei rămași vor fi termenii pentru care voi genera un răspuns). Apoi determin cursurile în care se regăsesc.

Ultimul pas este generarea răspunsului corect. Vom reutiliza încă o dată expresii regulate separate pentru răspuns în funcție de tipul de întrebare, extrăgând din cursuri fraza unde se regăsesc termenii până la următorul semn de sfârșit de punctuație.

De exemplu:

```
PATTERN_RASPUNS = {
    "definitie": [
        r"{termen}\s+este\s+(.*?)(?:\.|\n)",
        r"{termen}\s+reprezintă\s+(.*?)(?:\.|\n)",
        r"prin\s+{termen}\s+se\s+înțelege\s+(.*?)(?:\.|\n)"
    ],
    "diferenta": [
        r"diferența\s+dintre\s+{a}\s+și\s+{b}\s+este\s+(.*?)(?:\.|\n)",
        r"{a}\s+se\s+deosebește\s+de\s+{b}\s+prin\s+(.*?)(?:\.|\n)"
    ],
    "caracteristici": [
        r"caracteristicile\s+{termen}\s+sunt\s+(.*?)(?:\.|\n)",
        r"{termen}\s+se\s+caracterizează\s+prin\s+(.*?)(?:\.|\n)"
    ]
}
```

Pattern-urile vor fi declarate într-un fișier Python separat.

Bibliografie:

https://realpython.com/structural-pattern-matching/#what-is-pattern-matching

https://ardsite.medium.com/introduction-in-using-machine-learning-for-pattern-recognition-in-python-892104422df2

https://medium.com/@dr.booma19/a-python-guide-to-crafting-dynamic-question-answer-generation-with-t5-and-bert-d14f1c4b6fd0

3. Introducerea și evaluarea răspunsurilor

Solutii posibile existente:

1. Selvi, K., & Banerjee, S. (2010). *Automatic Short Answer Grading System (ASAGS)*
    a. - isi propune sa automatizeze evaluarea raspunsurilor scurte (1-3 propozitii) oferite de studenti la intrebarile teoretice: foloseste o ontologie pentru domeniul respectiv, utilizeaza modele de similaritate pentru scor numeric, identifica sinonime si relatii semantice
2. “Text-to-text Semantic Similarity for Automatic Short Answer Grading”
    a. Autori: Michael Mohler, Rada Mihalcea
    b. Publicat în: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2009)*-scorul final al unui raspuns se bazeaza pe similaritatea semantica dintre raspunsul studentului si cel corect folosindu-se de potrivirea exacta a cuvintelor, relatii de sens, structuri de dependenta si ordinea cuvintelor
3. *Graph semantic similarity-based automatic assessment for programming exercises*.
    Autori: Chengguan Xiang, Ying Wang, Qiyun Zhou, Zhen Yu.
    Publicat: „Sci Rep.”, 2024, vol.14(1):10530- Propune un algoritm pentru evaluarea automată a exerciţiilor de programare, fără a executa codul studentului — în schimb, se bazează pe graful de dependenţă al programului (Program Dependency Graph, PDG) şi pe similaritatea semantică între structura grafului codului studentului şi cel de referinţă

Metode propuse:

- folosirea unei ontologii pentru a identifica numarul de concepte din raspunsul corect acoperite de raspunsul primit, la fel ca la MediTest si 1)
- similaritatea textului fata de raspunsul corect: 1), 2)
- implementarea algoritmilor si compararea rezultatelor cu cele primite, ideea de la 3) ar fi mai complicat de implementat

Ideea de rezolvare:

Aplicatia permite introducerea unui raspuns prin intermediul unei interfete grafice, evalueaza raspunsul procentual (0-100%) si afiseaza raspunsul corect.

Putem adapta ideea de la MediTest, vom avea o ontologie pentru Inteligenta Artificiala in loc de una medicala, care include concepte, algoritmi, termeni relatii, folosind fisiere json.

```
{
  "concept_name": {
    "definition": ->text explicativ
    "key_words": ->lista de termeni importanti
    "relationships": {
      "is_a": ->categorie mai generala
      "related_to": ->alte concepte
    },
    "caracteristici": …
  }
}
```

Exemplu:

```
"iterative_deepening_search": {
    "definitie": "Algoritm care combina avantajele BFS si DFS. Exploreaza starile pana la o adancime limitata, crescand limita treptat pentru a evita buclele infinite si pentru a gasi solutia optima.",
    "cuvinte_cheie": [
      "adancime", "limita", "crestere", "bfs", "dfs", "iterativ"
    ],
    "relatii": {
      "is_a": "algoritm de cautare",
      "related_to": ["depth_first_search", "breadth_first_search"]
    },
    "caracteristici": [
      "evita buclele infinite",
      "gaseste solutia optima",
      "foloseste cautare limitata",
      "reia explorarea cu adancime mai mare"
    ]
  }
```

Normalizam raspunsul primit (litere mici, fara semne de punctuatie), identificam ce concepte din ontologia noastra apar in raspuns. Aplicatia genereaza raspunsul corect la intrebare, si calculeaza scorul raspunsului primit bazandu-se pe

- similaritatea textului (cosine similarity) intre acesta si raspunsul corect
- numarul de concepte din raspunsul corect acoperite de raspunsul primit

Pentru diferite tipuri de intrebari:

- ➔ ,,Pentru problema identificată dintr-o listă de minim 4 (n-queens, generalised Hanoi, graph coloring, knight's tour) și o instanță sau un set de instanțe, care este cea mai potrivită strategie de rezolvare, dintre cele menţionate la curs?'' avem stocate in baza de cunostinta strategiile clasice de rezolvare pentru fiecare problema + eventual strategii partial corecte si putem genera un scor al raspunsului: Ex: n-queens-corect:backtracking(100%), partial corect:DFS(50%)
- ➔ ,,Pentru jocul dat în forma normală (în matricea atașată), există echilibru Nash pur?''-aici aplicatia calculeaza toate echilibrele Nash pure si verifica daca raspunsul este corect sau partial corect
- ➔ ,,Care va fi asignarea variabilelor rămase date fiind variabilele, domeniile, constrângerile și asignarea parțială, dacă am utiliza algoritmul Backtracking cu optimizarea (FC, MRV sau AC-3) pentru continuarea rezolvării?'' aplicatia aplica algoritmul si compara rezultatul cu raspunsul primit
- ➔ ,,Pentru arborele dat, care va fi valoarea din rădăcină și câte noduri frunze vor fi vizitate în cazul aplicării strategiei MinMax cu optimizarea Alpha-Beta?'' aplicatia contruieste arborele, aplica algoritmul, compara rezultatele cu raspunsul primit

Deci pentru evaluarea raspunsurilor care necesita aplicarea unor algoritmi, aplicatia aplica algoritmul si compara rezultatele cu raspunsul primit, iar pentru evaluarea unor intrebari teoretice, de rationamentm se poate folosi o ontologie de concepte, termeni relatii.

Bibliografie:

https://www.researchgate.net/publication/47702502_Automatic_Short_-Answer_Grading_System_ASAGS

https://arxiv.org/abs/1011.1742

https://aclanthology.org/E09-1065.pdf

https://pubmed.ncbi.nlm.nih.gov/38719952/

https://www.ibm.com/think/topics/ai-in-software-development
https://drive.google.com/file/d/19gMnMliEqMGBmyGt-q1OtLk2cP7u-rqj/
https://drive.google.com/file/d/1zzbYqujb0gZ-kzrKsDenLQ7yVXX2r2bi/