UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚE ȘI TEHNOLOGIE POLITEHNICA BUCUREȘTI

CENTRUL UNIVERSITAR PITEȘTI

ȘCOALA DOCTORALĂ INTERDISCIPLINARĂ

TEZĂ DE DOCTORAT

MODELAREA, VERIFICAREA ȘI VALIDAREA SISTEMELOR DE PROGRAME ȘI A REȚELELOR

Rezumatul tezei de doctorat

Doctorand:

BOLD Nicolae

Conducător științific:

Prof. univ. dr. BĂLĂNESCU Tudor

PITEȘTI, 2023

CUPRINSUL TEZEI DE DOCTORAT

	Listă figuri	**4**
	Listă tabele	**5**
	Listă abrevieri	**6**
1	**Capitolul I. Introducere**	**8**
1.1	Definire, scop și obiective	8
1.2	Structura tezei	10
1.3	Contribuția tezei	14
2	**Capitolul II. Delimitări conceptuale**	**18**
2.1	Modelarea sistemelor de programe și a rețelelor	18
2.1.1	Definirea conceptului de modelare	18
2.1.2	Modelarea comportamentului unui sistem - sisteme de tranziție	21
2.1.3	Metode și tehnici de modelare. Modelarea formală	23
2.1.4	Automate cu număr finit de stări (FSM)	24
2.1.5	Rețele Petri	28
2.1.6	Modelarea sistemelor de programe și a rețelelor în literatura de specialitate	29
2.2	Verificarea sistemelor de programe și a rețelelor	33
2.2.1	Prezentarea conceptului de verificare	33
2.2.2	Tehnici și instrumente de verificare	34
2.2.3	Verificarea modelelor (model checking)	35
2.2.4	Verificarea modelelor în literatura de specialitate	38
2.3	Validarea implementării sistemelor de programe	39
2.4	Considerații asupra algoritmilor de tip evolutiv	41
3	**Capitolul III. Exemplificarea modelării și implementării unui sistem de programe și a unei rețele**	**44**
3.1	Descrierea sistemului de generare a testelor de evaluare (DMAIR) folosind algoritmi de tip evolutiv și de învățare nesupervizată	44
3.1.1	Stabilirea contextului cercetării	44
3.1.2	Modelarea matematică a sistemului de programe descris	51
3.1.3	Implementarea sistemului de programe	83
3.2	Direcții de cercetare referitoare la modelarea rețelelor	94
4	**Capitolul IV. Modelarea sistemelor de programe și a rețelelor**	**97**

4.1 Modelarea formală a sistemului de programe descris 98
4.1.1 Modelarea folosind UML 98
4.1.2 Modelarea prin intermediul mașinilor cu număr finit de stări (FSM) 100
4.1.3 Modelarea folosind rețele Petri 112
4.2 Direcții de modelare a unui sistem concurent de tip rețea 114
4.3 Implementarea unei mașini cu număr finit de stări în NuSMV 116
5 Capitolul V. Verificarea și validarea sistemelor de producție prin model checking și algoritmi genetici 121
5.1 Verificarea și validarea sistemelor de programe propuse 121
5.1.1 Specificarea proprietăților modelului 121
5.1.2 Verificarea proprietăților modelului 123
5.1.3 Validarea implementării folosind PHPUnit 124
5.2 Folosirea algoritmilor genetici pentru model checking 127
5.2.1 Abordări în literatura de specialitate 127
5.2.2 Determinarea spațiului stărilor sistemului 128
5.2.3 Metodologia folosirii unui algoritm evolutiv pentru model checking 128
5.2.4 Implementarea unui algoritm genetic folosit pentru model checking 130
6 Capitolul VI. Concluzii și activitate viitoare 132
Referințe bibliografice 135
Anexe 145

CUPRINSUL REZUMATULUI

CUPRINSUL TEZEI DE DOCTORAT 2

1 Capitolul I. Introducere 6

1.1 Definire, scop și obiective 6

1.2 Structura tezei 8

1.3 Contribuția tezei 9

2 Capitolul II. Delimitări conceptuale 12

2.1 Modelarea sistemelor de programe și a rețelelor 12

2.1.1 Definirea conceptului de modelare 12

2.1.2 Automate cu număr finit de stări (FSM) 13

2.1.3 Rețele Petri 13

2.1.4 Modelarea sistemelor de programe și a rețelelor în literatura de specialitate 14

2.2 Verificarea sistemelor de programe și a rețelelor 15

2.3 Validarea implementării sistemelor de programe 15

2.4 Considerații asupra algoritmilor de tip evolutiv 16

3 Capitolul III. Exemplificarea modelării și implementării unui sistem de programe și a unei rețele 17

3.1 Descrierea sistemului de generare a testelor de evaluare (DMAIR) folosind algoritmi de tip evolutiv și de învățare nesupervizată 17

3.1.1 Stabilirea contextului cercetării 17

3.1.2 Modelarea matematică a sistemului de programe descris 18

3.1.3 Implementarea sistemului de programe 29

3.2 Direcții de cercetare referitoare la modelarea rețelelor 32

4 Capitolul IV. Modelarea sistemelor de programe și a rețelelor 33

4.1 Modelarea formală a sistemului de programe descris 34

4.1.1 Modelarea folosind UML 34

4.1.2 Modelarea prin intermediul mașinilor cu număr finit de stări (FSM) 35

4.1.3 Modelarea folosind rețele Petri 39

4.2 Direcții de modelare a unui sistem concurent de tip rețea 40

4.3 Implementarea unei mașini cu număr finit de stări în NuSMV 41

5 Capitolul V. Verificarea și validarea sistemelor de producție prin model checking și algoritmi genetici 42

5.1 Verificarea și validarea sistemelor de programe propuse 42

5.1.1 Specificarea proprietăților modelului 42

5.1.2 Verificarea proprietăților modelului 44

5.1.3 Validarea implementării folosind PHPUnit 45

5.2 Folosirea algoritmilor genetici pentru model checking 45

5.2.1 Abordări în literatura de specialitate 45

5.2.2 Metodologia folosirii unui algoritm evolutiv pentru model checking 46

6 Capitolul VI. Concluzii și activitate viitoare 48

Referințe bibliografice 51

1 CAPITOLUL I. INTRODUCERE

Rezumat. *Pe scurt, Capitolul I (Introducere) propune prezentarea definirii pe scurt a problemei de cercetare, a scopului și a obiectivelor acesteia. După această prezentare, se definește structura tezei, formată din șase capitole structurate în trei părți principale. În finalul capitolului, se delimitează o serie de contribuții și elemente de inovație propuse de autor prin procesul de cercetare desfășurat.*

1.1 Definire, scop și obiective

Cercetarea prezentată în această teză își propune formalizarea unor sisteme de programe și a rețelelor dezvoltate pentru sisteme specifice din practică. În acest sens, autorul a pornit de la studiul unui sistem software care implementează generarea unor teste de evaluare educațională luând în calcul restricții (subiectul testului, grad de dificultate, timp de rezolvare etc.). Același sistem realizează și verificarea răspunsurilor primite la test și analiza întrebărilor. Generarea testelor este realizată folosind algoritmi de tip evolutivi, structuri de tip arbore și tehnici de învățare nesupervizată, într-o formă modulară, care simulează un sistem de programe.

Pe baza modelului matematic dezvoltat în urma descrierii sistemului propus, a fost construit un model specific prin aplicarea de tehnici de modelare formală. De asemenea, a fost realizată verificarea modelului obținut folosind metode clasice sau au fost propuse metode care folosesc algoritmi genetici. În același timp, s-a propus și un proces de validare a implementării modelului, cu scopul de a crește robustețea sistemului, având în vedere natura tehnicilor folosite pentru dezvoltarea lui. Această aplicare a tehnicilor de modelare, verificare și validare a fost realizată cu scopul de a studia și rezolva situații complexe și nedorite în cadrul programului. Dintre aceste situații, putem să amintim obținerea de teste care nu corespund restricțiilor dorite, lipsa convergerii algoritmului genetic către o valoare de optim, erori în verificarea răspunsurilor și altele. Mai departe, cercetarea include descrierea unui model matematic al unei rețele aplicat într-un context economic, care poate servi ca punct de plecare pentru studiul unor cazuri de conectivitate a întrebărilor. Astfel, teza are ca scop principal fundamentarea riguros matematică a problemei descrise mai sus prin intermediul modelării matematice și a metodologiilor de modelare formală. Acest scop va avea ca rezultat așteptat obținerea unui model al sistemului de programe care implică componente de tip rețea. Formalizarea prin intermediul creării modelului propus va sta la baza abordării complexității sistemului de programe și la eliminarea erorilor. Obiectivele care se subsumează scopului descris pe scurt pot fi considerate:

O.1. Descrierea complexității problemei propuse și includerea ei în categoria de probleme de optimizare combinatorică dificil de rezolvat în timp polinomial;

O.2. Abordarea complexității problemei prin intermediul modelării acesteia în structura unui sistem de programe;

O.3. Includerea unor aspecte de conectivitate în cadrul problemei prin studiul specific al unui model de rețea și potențialul folosirii ei în cadrul sistemului de programe descris;

O.4. Formalizarea sistemului de programe prin intermediul folosirii metodologiilor de modelare formală și de verificare a proprietăților;

O.5. Folosirea structurilor obținute pentru verificarea implementării practice a sistemului de programe descris;

O.6. Obținerea unei implementări a sistemului de programe care să aibă caracteristica de automatizare prin folosirea instrumentelor de învățare automată.

Prima parte a tezei va cuprinde prezentarea pe scurt a principalelor elemente teoretice care formează tema generală a cercetării. Astfel, pornind de la cele trei direcții tematice generale (modelarea, verificarea și validarea), această parte prezenta fundamentele teoretice care stau la baza cercetării privind aceste trei direcții și metodele și tehnicile folosite (construirea modelului, mașini cu număr finit de stări, specificarea proprietăților, verificarea modelului – model checking, logici temporale etc.). Descrierea lor va servi scopului de fundamentare teoretică a modelării realizate în secțiunile următoare. A doua parte a tezei include rezultatele obținute în cea mai mare parte a perioadei de cercetare. Astfel, în prima parte a perioadei, accentul a fost pus pe primul element tematic, și anume modelarea, cercetarea continuând unele dintre temele începute în perioada pre-doctorală. Cercetarea a continuat cu prezentarea de rezultate obținute din implementarea folosind diverse limbaje și prin studiul unor componente (e.g., analiza itemilor) și cazuri specifice (e.g., restricții, tipuri de întrebări) ale sistemului de programe. Dezvoltarea a conținut rafinarea din punct de vedere al tehnicilor și metodelor de modelare ale temelor cercetate de-a lungul timpului. Rezultate promițătoare au fost obținute prin intermediul formulării din ce în ce mai clare a modelelor și pregătirea lor pentru implementare, verificare și validare. În final, cea de-a treia parte a tezei include descrierea modelării formale a sistemului de programe prin intermediul instrumentelor specifice (limbaje de modelare formală – UML, mașini cu număr finit de stări etc.). În aceeași parte, regăsim specificarea unor proprietăți ale modelului și verificarea lui (model checking) folosind anumite instrumente specifice. Se propune și verificarea automată a proprietăților modelului descris prin intermediul folosirii algoritmilor genetici.

Una dintre direcțiile teoretice ale activității de cercetare a fost reprezentată de realizarea unor modele matematice, construite folosind diverse metode și tehnici de modelare. Dacă inițial aceste tehnici vizau o formă structurală a modelelor, folosind, spre exemplu, dinamica sistemelor, acestea au evoluat spre formulări care să cuprindă elementele tematice ale cercetării doctorale. De altfel, cercetările viitoare se îndreaptă spre consolidarea robusteții implementării sistemului de programe prin dezvoltarea avansată a tehnicilor curente de modelare folosite până în momentul actual.

1.2 Structura tezei

Această teză este structurată pe șase capitole, descrise pe scurt după cum urmează:

- *Capitolul I. Introducere*: acest capitol cuprinde aspecte legate de importanța modelării, verificării și validării sistemelor de programe și a rețelelor. Capitolul mai include și descrierea pe scurt a cercetării din prisma scopului, a obiectivelor, a structurii tezei și a contribuției acesteia.
- *Capitolul II. Delimitări conceptuale*: acest capitol include o trecere în revistă a principalelor concepte folosite în cadrul cercetării. Această scurtă prezentare pornește de la modelarea formală a sistemelor și continuă cu verificarea și validarea lor. Este luat în calcul și studiul dinamicii conceptelor în literatura de specialitate.
- C*apitolul III. Exemplificarea modelării și implementării unui sistem de programe și a unei rețele*: acest capitol include rezultatele obținute în ceea ce privește modelarea și implementarea unui sistem de programe referitor la generarea, verificarea și analiza unor teste de evaluare educațională. Modelarea sistemului este prezentată inițial din perspectivă matematică, apoi fiind descris prin prisma structurii modulare. Modelarea folosește algoritmi euristici (genetici), structuri arborescente și metode și tehnici specifice învățării nesupervizate (Machine Learning). Componenta de conectivitate a sistemului este descrisă pornind de la modelarea unei rețele în context economic folosind teoria grafurilor, cu potențiale aplicații în sistemul de programe descris. **Capitolul se constituie într-o compilație a rezultatelor obținute de-a lungul timpului și a publicării lor în literatură.** Acest capitol este formulat pe baza mai multor lucrări realizate sau în curs de publicare, cu contribuția autorului, listate în teza de doctorat.
- *Capitolul IV. Modelarea sistemelor de programe și a rețelelor*: acest capitol include descrierea unui model formal folosind instrumente specifice (diagrame obținute folosind limbajul UML - Unified Model Language, mașini cu număr finit de stări și rețele Petri). Modelul formal are ca finalitate abordarea complexității sistemului de programe. Capitolul

cuprinde și specificarea proprietăților sistemului și propune direcțiile de acțiune pentru capitolul următor.

- *Capitolul V. Verificarea și validarea sistemelor de producție și a rețelelor prin model checking și algoritmi genetici*: acest capitol include aplicarea procesului de verificare a modelului (model checking) și a proprietăților specificate în limbajul CTL (Computation Tree Logic). În acest scop, se folosește instrumentul de verificare formală NuSMV și se validează implementarea acestuia în practică folosind PHPUnit pentru sistemele descrise în Capitolul III. Capitolul propune și o descriere a unui studiu de caz referitor la folosirea algoritmilor genetici pentru verificarea proprietăților unuia dintre modelele prezentate în Capitolul IV.
- *Capitolul VI. Concluzii și activitate viitoare*: acest capitol include formularea unor concluzii referitoare la subiectele propuse, prezentarea unor limitări referitoare la aspecte întâlnite în cadrul cercetării și stabilirea unor direcții viitoare de studiu referitoare la temă.

1.3 Contribuția tezei

Cele mai importante contribuții formulate în cadrul tezei se înscriu în două componente principale. Cele două componente includ aspecte teoretice, pe de o parte, și practice, pe de cealaltă parte, incluzând descrieri ale modelării formale pentru sisteme de programe specifice. Astfel, caracteristicile acestor sisteme de programe sunt îmbunătățite prin intermediul rezultatelor modelării. În acest fel, se propune o creștere a calității implementării sistemelor de programe în practică. În acest sens, teza furnizează o serie de rezultate referitoare la descrierea, modelarea formală, verificarea și implementarea unui sistem de programe specific. Acest sistem are ca rezultat generarea testelor folosite în evaluarea educațională din întrebări (denumite mai departe itemi) folosind algoritmi evolutivi. Aceste rezultate incluse în cadrul tezei aduc noi contribuții în cadrul comunității academice privind o serie de puncte prezentate în paragrafele următoare.

Primul element de noutate este descrierea unui model matematic al unei probleme de optimizare combinatorică din domeniul evaluării educaționale. Această descriere include o abordare a problemei referitoare la generarea unor secvențe de itemi de evaluare educațională în prezența anumitor restricții. Existența acestor restricții legate de dificultate, subiect sau timp de rezolvare și contextul problemei conduc la o creștere a complexității acesteia. Formulând un context în care se iau în considerare restricții multiple, un proces de învățare individualizat, corelații între itemi și un volum mare de secvențe de itemi generate și stocate (de ordinul a mii de itemi) înainte de generarea testelor, problema poate fi clasificată în categoria problemelor de

optimizare dificil de rezolvat în timp polinomial. Abordarea acestei complexități prin intermediul descrierii unor cazuri specifice este unul dintre elementele de noutate propuse în cadrul tezei.

Un al doilea element contributiv al tezei este folosirea algoritmilor de tip euristic pentru a determina soluții pentru problema propusă. Astfel, descrierea și implementarea unor algoritmi din această clasă pentru găsirea de soluții optimale conferă un element de noutate abordărilor deja stabilite în literatura de specialitate. Mai precis, pentru soluționarea problemei propuse au fost descrise o serie de variante ale unui algoritm evolutiv de tip genetic. În cadrul acestuia, un rol important îl joacă determinarea funcției fitness, a cărei construcție modelează matematic restricțiile propuse pentru generarea testelor de evaluare. Această determinare a unor cazuri specifice ale restricțiilor algoritmului se constituie într-un alt punct contributiv, delimitat în acest mod de cercetări similare din literatura de specialitate. În plus, în finalul tezei este descrisă oportunitatea explorării mai eficiente a spațiului de stări prin intermediul încorporării algoritmilor genetici în verificarea proprietăților modelului descris.

Cel de-al treilea element de noutate îl implică formularea unui sistem de programe pentru abordarea aleasă. Sistemul de programe descris abordează complexitatea algoritmică prin intermediul decompoziției problemei și al implementării euristicii. În ce privește decompoziția problemei, aceasta poate fi descompusă în subprobleme precum generarea itemilor, preselectarea itemilor pentru generare, generarea secvențelor de itemi, analiza itemilor necesară stabilirii validității lor etc.. De asemenea, complexitatea datelor este abordată prin operații de filtrare și selecție a datelor de intrare, precum și prin organizarea lor în structuri specifice algoritmilor genetici. Aceste abordări specifice sistemelor de programe conduc la o creștere a eficienței modelului propus.

Un alt aspect important adus sistemului de programe este analiza corelării dintre itemii care formează potențialele secvențe de evaluare. Această analiză este abordată din două perspective. Prima dintre ele se referă la corelații deja stabilite de tip taxonomic între itemi, date de legătura dintre subiectele care descriu itemii. În lucrare, această perspectivă este tratată prin intermediul modelării folosind arbori, legăturile fiind preexistente. Cea de-a doua perspectivă se referă abordarea naturii acestor legături, care pornește de la delimitarea legăturilor economice între producători și consumatori într-o piață economică. Acest punct de pornire referitor la studiul rețelelor pentru problema propusă se constituie într-o analiză a naturii legăturilor dintre elementele componente ale unei rețele, punct care va fi aplicat în lucrări viitoare în cadrul sistemului de programe deja propus.

Un al cincilea aspect important al contribuțiilor tezei se constituie în realizarea unui model formal al sistemului de programe descris și verificarea lui. Modelarea formală aduce cu sine o serie importantă de avantaje, precum claritate, structură și documentare a întregului sistem. Mai mult, crearea de niveluri de abstractizare permite gestionarea complexității amintite a sistemului de programe. Acest lucru conduce la concentrarea pe aspecte critice ale sistemului, de exemplu pe generarea optimă a secvențelor de itemi, caz în care a fost realizată modelarea succesiunii operațiilor genetice. De asemenea, verificarea automată a proprietăților modelului descris conduce la identificarea erorilor potențiale. Modelarea formală este propusă în cadrul tezei prin intermediul unor instrumente specifice (diagrame UML, mașini cu număr finit de stări etc.).

Nu în cele din urmă, includerea unor aspecte din domeniul învățării automate este un alt element de noutate în cadrul sistemului descris. Deși literatura de specialitate abordează această problemă, folosirea învățării nesupervizate pentru generarea automată a itemilor și pentru etapele de filtrare și selectare a datelor este un element inovativ pentru sistemul descris. Elemente de acest tip sunt incluse în generarea itemilor folosind procesarea naturală a limbajului (Natural Language Processing) și în formarea de clustere de itemi folosite în cadrul algoritmului genetic. Folosirea învățării automate conduce la reducerea intervenției umane în cadrul sistemului.

Rezumând aceste aspecte, noutatea prezentă în cadrul cercetării constă în această integrare a elementelor descrise mai sus și a unora tradiționale. Această integrare a condus la furnizarea de rezultate în ceea ce privește dezvoltarea unui sistem de programe și a rețelei formate între componentele acesteia. Rețeaua formată poate avea potențialul caracteristic unui sistem distribuit, căruia i se pot aplica modelări specifice. Astfel, această abordare folosește metode și tehnici aplicate care au un caracter complex, cum ar fi algoritmi de tip evolutiv și tehnici de învățare nesupervizată. Un alt element de acest tip a constat în aplicarea modelării formale și a verificării proprietăților sistemului propus în ceea ce privește sistemele descrise, dezvoltând și aplicând metode descrise în literatura de specialitate care are un caracter inovativ.

2 CAPITOLUL II. DELIMITĂRI CONCEPTUALE

Rezumat. *Capitolul II este constituit din integrarea unei serii de concepte care fundamentează teoretic cercetarea descrisă în capitolele ulterioare. În prima secțiune (2.1), sunt prezentate concepte asimilate modelării formale a sistemelor, începând cu definirea procesului de modelare și continuând cu prezentarea unor instrumente specifice (e.g., automate cu număr finit de stări, rețele Petri). În secțiunea a doua (2.2), este detaliat procesul de verificare a sistemelor prin definirea lui și a tehnicilor și instrumentelor de verificare a modelului (model checking). Secțiunea a treia (2.3) cuprinde definirea procesului de validare a implementării unui sistem de programe. În final (2.4), sunt incluse o serie de considerații teoretice referitoare la algoritmii euristici de tip evolutiv (genetici).*

2.1 Modelarea sistemelor de programe și a rețelelor

2.1.1 Definirea conceptului de modelare

În domeniul informaticii, verificarea modelului sau verificarea proprietății este, pentru un anumit model al unui sistem, verificată exhaustiv și automat dacă acest model îndeplinește o specificație dată. În mod tipic, un astfel de model are în vedere sistemele hardware sau software, în timp ce specificațiile conțin cerințe de siguranță, cum ar fi absența blocajelor și a stărilor critice similare care pot provoca căderea sistemului. Verificarea modelului este o tehnică pentru verificarea automată a proprietăților corecte ale sistemelor de stare finită.

Modelarea formală a sistemelor software este o tehnică care implică utilizarea limbajelor și metodelor matematice precise pentru a descrie și analiza sistemele software. Aceasta implică transformarea comportamentului și funcționalității unui sistem software într-o reprezentare formală, care poate fi verificată și analizată riguros pentru a detecta erori, a evalua proprietăți și a asigura corectitudinea sistemului.

În acest sens, modelarea oferă posibilitatea reprezentării schematizate a unui sistem de programe sau a unei rețele, iar rezultatul principal al modelării este obținerea unui model.

Definiție 2.1 [1]. Un model formal este o reprezentare abstractă și precisă a unui sistem de programe sau a unei rețele, care utilizează limbaje și tehnici formale pentru a descrie comportamentul și regulile acestuia.

Modelarea formală a rețelelor implică definirea unei reprezentări precise a rețelei folosind limbaje formale, cum ar fi rețele Petri, algebre de procese, logici temporale, grafuri de dependență, logici modale și altele. Aceste modele permit descrierea comportamentului rețelei,

interacțiunilor dintre componentele sale și proprietăților pe care trebuie să le îndeplinească [12]. Acest tip de modele descrie și analizează o rețea, care poate fi identificată ca un sistem format din elemente interconectate, ca de exemplu rețele de calculatoare, rețele neuronale, rețele de comunicații etc..

2.1.2 Automate cu număr finit de stări (FSM)

O mașină cu număr finit de stări conține un număr finit de stări și produce ieșiri la tranzițiile de stare după ce primește intrări [13]. Mașinile cu număr finit de stări sunt utilizate pe scară largă pentru a modela sisteme în diverse domenii, inclusiv circuite secvențiale, anumite tipuri de programe și, mai recent, protocoale de comunicație [14].

În ceea ce privește automatele cu număr finit de stări, acestea sunt reprezentare finită a unui limbaj formal care poate fi un reprezentat de o mulțime infinită [15]. Definiția formală a unei mașini cu număr finit de stări (Finite State-Machine–FSM) include mai multe componente.

Definiție 2.2 [13]. O mașină cu număr finit de stări (FSM) este reprezentată oficial de o tuplă $(\boldsymbol{S}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, \boldsymbol{S_0}, \boldsymbol{F})$, unde:

- **S este o mulțime finită de stări;**
- **Σ este o mulțime finită de simboluri, numit alfabetul automatului;**
- **δ este funcția de tranziție, adică, $\boldsymbol{\delta}: \boldsymbol{S} \times \boldsymbol{\Sigma} \rightarrow \boldsymbol{S}$;**
- **s_0 este starea de pornire, adică starea automatului înainte ca orice intrare să fie procesată, unde $s_0 \in S$;**
- **F este o mulțime de stări din S (adică $F \subseteq Q$) numite stări finale (de acceptare).**

2.1.3 Rețele Petri

O rețea Petri (eng. Petri net) se compune dintr-un tip particular de graf orientat notat N și o stare inițială 0 M , denumită marcaj inițial (eng. initial marking) [19]. Graful N al unei rețele Petri este o reprezentare grafică a componentelor și relațiilor dintre acestea într-o rețea Petri. Graful este format din noduri și arce, unde nodurile reprezintă locațiile și tranzițiile, iar arcele reprezintă fluxul de trecere între acestea. Nodurile pot fi marcate sau nemarcate, indicând starea curentă a rețelei. Graful oferă o imagine vizuală a comportamentului sistemului și poate fi utilizat pentru analiză și simulare. Graful unei rețele Petri este format din noduri și arce. Nodurile pot fi de două tipuri principale: locații și tranziții.

Definiție 2.3 [19]. O rețea Petri este un cvintuplu, $\boldsymbol{PN} = (\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{F}, \boldsymbol{W}, \boldsymbol{M_0})$, în care:

- $\boldsymbol{P = \{p_1, p_2, \dots, p_m\}}$ **este mulţimea poziţiilor sau locaţiilor (finită);**
- $\boldsymbol{T = \{t_1, t_2, \dots, t_n\}}$ **este mulţimea tranziţiilor (finită);**
- $\boldsymbol{F \subseteq (P \times T) \cup (T \times P)}$ **este mulțimea arcelor;**
- $\boldsymbol{W : F \rightarrow \{1, 2, 3, \dots\}}$ **este funcţia de ponderare a arcelor;**
- $\boldsymbol{M_0 : P \rightarrow \{0, 1, 2, 3, \dots\}}$ **este funcția de marcaj inițial.**

O rețea Petri este formată din locuri, tranziții și arce. Arcurile rulează de la un loc la o tranziție sau invers, niciodată între locuri sau între tranziții. Locurile de la care trece un arc către o tranziție sunt numite locuri de intrare ale tranziției; locurile către care se îndreaptă arcurile de la o tranziție se numesc locuri de ieșire ale tranziției.

2.1.4 Modelarea sistemelor de programe și a rețelelor în literatura de specialitate

Pentru a determina dinamica subiectului în literatura de specialitate, am realizat un scurt studiu bibliografic referitor la categoria de cercetare. Metodologia acestui studiu empiric a constat în folosirea bazei de date Dimensions.ai ca spațiu de căutare și procesarea rezultatelor obținute în programul VOSViewer pentru a regăsi principalii termeni folosiți în cadrul cercetărilor găsite. În urma aplicării metodologiei, rezultatele obținute au fost grupate în forma unei rețele de cuvinte, care poate fi vizualizată în Figura 2.1. Rezultatele obținute în urma căutării au inclus numărul de rezultate specifice căutării în perioada 2014-2023 (844 749 de publicații), grupate pe domenii precum 46 Information and Computing Sciences, 40 Engineering, 35 Commerce, Management, Tourism and Services, 4602 Artificial Intelligence, 4612 Software Engineering, 46 Information and Computing Sciences, 4606 Distributed Computing and Systems Software, 4605 Data Management and Data Science sau 4604 Cybersecurity and Privacy.

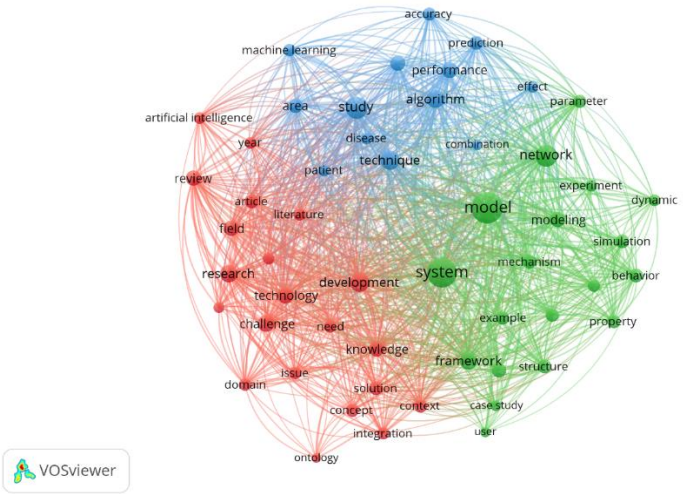


Figura 2.1. Rezultatele analizei cercetărilor legate de subiectul modelării formale în perioada 2014-2023

Se poate observa o delimitare a termenilor în trei clustere, care includ aspecte legate de:

- formularea specifică cercetării științifice, vizualizată în clusterul colorat cu verde, care conține termeni precum system, model, property, network, framework;
- formularea specifică producției, pentru clusterul colorat cu roșu, care conține termeni precum technology, development, research, knowledge, artificial intelligence;
- formularea specifică tendințelor de aplicabilitate ale metodelor formale în practică, pentru clusterul colorat cu albastru, care conține termeni precum performance, machine learning, prediction, accuracy, algorithm.

2.2 Verificarea sistemelor de programe și a rețelelor

Verificarea algoritmilor determină dacă un sistem de tranziție satisface o anumită proprietate. Proprietățile sunt construite din propoziții elementare care pot fi adevărate sau false în starea sistemului. Această idee este formalizată prin intermediul conceptului de *structură Kripke* care extinde sistemul de tranziție cu o interpretare a propozițiilor atomice în stări.

2.3 Validarea implementării sistemelor de programe

Generarea de cazuri de testare bune nu poate fi nici o sarcină simplă, nici rapidă. Când software-ul suferă modificări, suita de teste poate avea nevoie de unele ajustări. Chiar și atunci când o suită de testare acoperă tot codul, adică 100% din acoperire, aceasta nu înseamnă că calitatea cazurilor de testare este cea mai bună odată ce nu poate detecta unele erori. Există câteva abordări care pot evalua calitatea setului de teste.

Există unele lucrări de cercetare în jurul datelor de utilizare a web. Se știe că aceste date sunt foarte importante pentru a înțelege comportamentul utilizatorilor pe site-ul web. Cu aceste

informații, dezvoltatorii sunt capabili să efectueze modificări pe site-ul web în funcție de ceea ce este preluat în aceste date. De exemplu, media traficului pe anumite pagini, etc. poate fi o informație utilă folosită pentru a îmbunătăți capacitatea de utilizare a site-ului și pentru a-l menține interesat pe utilizator.

Utilizarea acestor informații pentru a genera cazuri de testare nu este foarte explorată. Aceste cazuri de testare sunt foarte importante odată ce oferă o consistență bună a sistemului și permit prevenirea și detectarea defecțiunilor noi în sistem. Pentru a le îmbunătăți, poate fi aplicată o tehnică de extindere a acestora și de creștere a acoperirii obținute de cazurile de testare generate și de a evalua calitatea acestora.

Există mai multe moduri de a exprima proiectarea unui sistem sau a unui subsistem [39]. Mașini de stare finită (FSM), graficele de stare, rețelele Petri sunt câteva dintre formalismele care exprimă diverse aspecte ale unui design. Protocoalele, de exemplu, sunt deseori exprimate sub formă de FSM. Proiectele software sunt exprimate folosind graficele de stare. Proiectarea este utilizată ca sursă pentru generarea de teste care sunt folosite ulterior pentru testarea *Implementation Under Test* (IUT) pentru conformitatea cu proiectul.

2.4 Considerații asupra algoritmilor de tip evolutiv

Algoritmii genetici fac parte din familia de algoritmi bio-inspirati și au fost inspirati din legile naturale ale geneticii. Descrierea lor teoretică cuprinde noțiuni legate de legile mendeliane ale eredității, cu influențe din evoluționism, datorită relației sale cu genetica populației și terminologia și comportamentul acestora.

Un algoritm genetic de succes se poate opri atunci când fitness-ul celui mai bun cromozom și cromozomul mediu converg. Această convergență începe pe măsură ce încep să apară noile generații. Astfel, convergența este conectată la echilibrul de legătură descris în Echilibrul lui Hardy-Weinberg, unde variația în cadrul populației limitează o valoare nulă [48].

Pentru informatică, aceste noțiuni teoretice au fost conceptualizate cu ajutorul instrumentelor statistice și a operațiilor matematice. Astfel, o populație ajunge la convergență pe măsură ce atinge echilibrul de legătură [49].

3 CAPITOLUL III. EXEMPLIFICAREA MODELĂRII ȘI IMPLEMENTĂRII UNUI SISTEM DE PROGRAME ȘI A UNEI REȚELE

Rezumat. *Capitolul III este rezultatul compilației dintre rezultatele cercetărilor realizate de către autor care au fost publicate în cadrul diverselor manifestări științifice. Aceste rezultate au fost integrate într-o descriere unitară a sistemului de programe care modelează problema descrisă în Capitolul I. În acest sens, prima secțiune (3.1) cuprinde descrierea din perspective naturale și matematice a sistemului de programe. Sunt descrise contextul cercetării (enunțarea problemei și motivația folosirii algoritmilor genetici) și modelarea matematică a sistemului (structura și funcționalitatea modelului). Descrierea modelării include și descrierea componentelor principale ale modelului (generarea testelor folosind algoritmi genetici și arbori, verificarea răspunsurilor și analiza itemilor). Această secțiune mai cuprinde și o descriere a implementării sistemului de programe. În final, secțiunea a doua (3.2) descrie modelarea unei rețele și a integrării modelului clasic producător-consumator într-un context economic folosind grafuri. Modelul obținut are potențial în studiul conectivității în sistemul de programe descris în secțiunea 3.1.*

3.1 Descrierea sistemului de generare a testelor de evaluare (DMAIR) folosind algoritmi de tip evolutiv și de învățare nesupervizată

3.1.1 Stabilirea contextului cercetării

3.1.1.1 Enunțarea problemei și a scopului cercetării

Proiectarea unui test trebuie să respecte un număr mare de cerințe pentru a fi obiectivă și aliniată scopurilor de învățare. Toate caracteristicile enumerate mai sus sunt îndeplinite în experiența de muncă și practica unui profesor sau evaluator bun, peste medie. Se presupune că profesorul posedă o bună cunoaștere a subiectului și conținutului lecției sau cursului. Un sistem automatizat care să țină cont de toate aceste caracteristici poate utiliza mai multe metode de simulare a unui proces de proiectare care ar acționa aproape de un proces de generare umană.

În ceea ce privește generarea testelor de evaluare, condițiile sunt reprezentate de categoria generală / subiectul testului, gradul de dificultate al testului, raportul teoretic / practic al întrebărilor din test și tipul de întrebare (cu răspunsuri multiple, răspuns scurt sau eseu). Acest model îmbunătățit este o dezvoltare a unui model prezentat și implementat în lucrările de cercetare anterioare, în care generarea itemilor de testare cu cerințe a fost realizată folosind un algoritm genetic. Rafinarea sistemului de generare a fost realizată cu metode bazate pe machine learning (ML), precum clustering sau procesarea limbajului natural (NLP).

În ceea ce privește verificarea răspunsurilor la itemi, se vor lua în calcul descrieri ale unor lucrări care tratează determinarea similarității a două răspunsuri folosind distanța de editare și tehnici folosite în cadrul procesării limbajului natural.

În ceea ce privește analiza răspunsurilor, se va descrie arhitectura modelului, implementarea și rezultatele acestuia. Pe scurt, modelul constă într-o metodă specifică legată de analiza itemilor de evaluare într-un context educațional specific. În această chestiune, modelul se bazează pe ipoteza că datele statistice legate de un item de evaluare sunt influențate de înțelegerea conceptuală a subiectului itemului. De asemenea, mai mulți factori, cum ar fi factori bazați pe itemi (de exemplu, gradul de dificultate al itemului, natura teoretică / practică a itemului, tipul itemului, numărul articolului), factori statistici și de testare a itemului (de exemplu, medie și standard), abaterea, discriminarea itemilor, încercările de itemi, coeficientul de fiabilitate), factorii centrați pe elev (de exemplu, nivelul de educație al elevului) sau factorii centrați pe grup (de exemplu, media scorului de evaluare), trebuie luați în considerare în analiza răspunsurilor itemului. În acest sens, având în vedere o anumită perioadă de timp de învățământ, cum ar fi un semestru sau un an, și evaluarea periodică a elevilor susținută de un cadru didactic, analiza itemilor stă la baza noțiunilor care pot fi dezvoltate și aprofundate mai mult pe parcursul cursurilor. Această selectare a itemilor care necesită aprofundare se realizează datorită ratelor mai mici de răspunsuri corecte ale acestor itemi care verifică noțiuni specifice în cadrul testelor de evaluare periodică. Modelul este, de asemenea, îmbunătățit prin determinarea unor mecanisme suplimentare de verificare a itemilor folosind metode automate de grupare a lor (clustering). Acest mecanism se folosește pentru a prezice dacă un articol este predispus să aibă rate de răspuns mai scăzute pe baza factorilor luați în considerare, tendință care va fi confirmată de elementul factual. Evident, gruparea automată va oferi rezultate mai fine, luând în considerare un set mai mare de factori.

3.1.2 Modelarea matematică a sistemului de programe descris

3.1.2.1 Descrierea naturală a modelului

Modelul în limbaj natural descrie un sistem mai larg care reunește elemente legate de baze de date, interfețe grafice cu utilizatorul, modele de generare și mecanisme de interpretare și analiză a datelor. Acest sistem este destinat a fi utilizat în scopul evaluării elevilor / studenților. Schema generală a acestui model este prezentată în Figura 3.1.

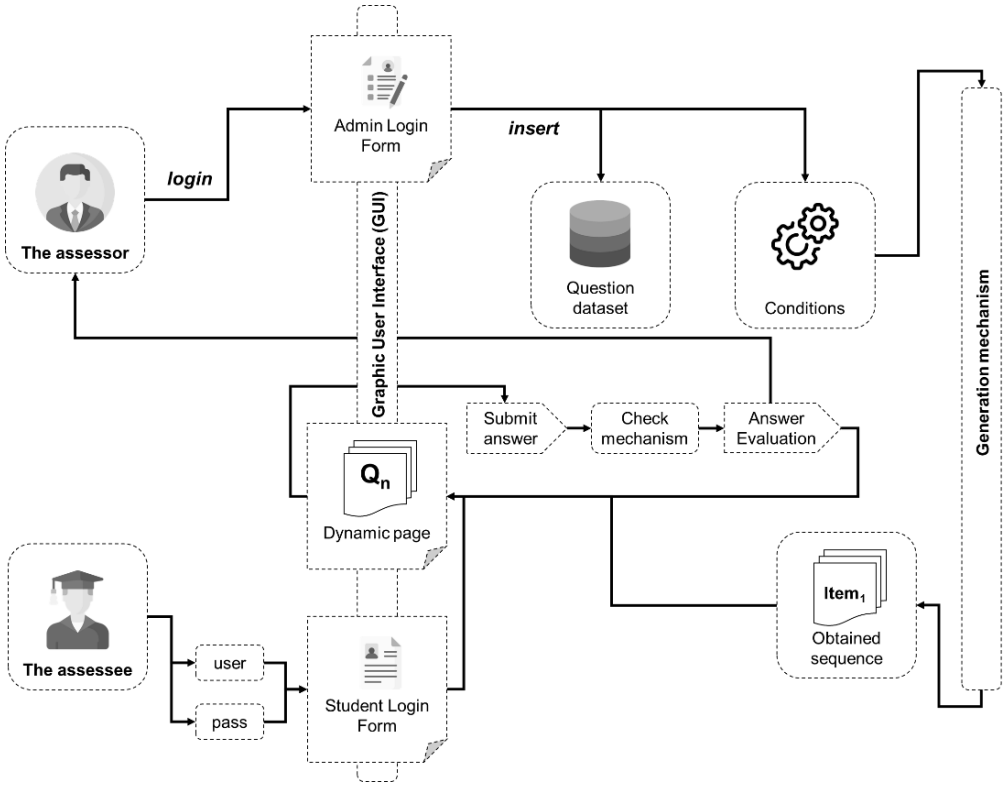


Figura 3.1. Modelul de evaluare DMAIR propus de autor

3.1.2.2 Structura modelului

3.1.2.2.1 Componentele modelului

Modelul DMAIR cuprinde mai multe componente care sunt esențiale pentru un sistem de evaluare. Acest sistem trebuie să fie format din trei funcționalități principale: generarea de itemi (func1 (I)), mecanisme de verificare (func2 (II)) și evaluarea răspunsului (func3 (III)). Modulul responsabil cu generarea itemilor, func1 (I), folosește metode și instrumente de obținere a testelor de evaluare adecvate cerințelor specifice, mecanismul de verificare. Componenta func2 (II) este legată de validarea răspunsurilor date de utilizatori și de modulul de evaluare a răspunsurilor. Componenta func3 (III) introduce dezvoltarea analizei itemilor pentru itemii generați și răspunsurile date și este cel mai legat de analiza învățării (IA).

Modelul cuprinde și o colecție de date sub forma unei baze de date, BD, care include informațiile necesare generării, corectării și analizei itemilor și a răspunsurilor la aceștia. Extinzând funcționalitățile de bază, implementarea include și un modul de interacțiune cu agentul uman, func4 (IV), care constă în interfața grafică (GUI) cu utilizatorul. Reprezentarea vizuală a modelului, inclusiv componenta de interfață grafică cu utilizatorul, este prezentată în Figura 3.1.

3.1.2.2.2 Restricțiile modelului

Pe lângă cele trei componente de bază ale modelului, luăm în calcul și setul de restricții (ConSet) formulate de către utilizator. Aceste restricții constau într-o listă de parametri cărora le pot fi oferite diverse valori și pe baza cărora sunt generate aceste teste. Lista de parametri studiați până în momentul de față în lucrările anterioare include:

- subiectul / subiectele ce doresc a fi tratate în cadrul evaluării;
- gradul de dificultate al secvenței evaluării;
- raportul teoretic / practic al secvenței evaluării;
- tipul predominant de itemi din cadrul secvenței evaluării;
- timpul maxim de rezolvare al secvenței de itemi.

3.1.2.2.3 Variabilele modelului

Variabilele principale ale modelului sunt întrebările, itemi denumiți după procesul de generare, testul, cerințele și mecanismul de generare. Întrebarea este un caz particular al unui item, care poate consta într-o sarcină de lucru sau un exercițiu, acesta fiind motivul pentru care întrebările vor fi considerate cazuri particulare de itemi și ne vom referi în continuare la întrebări, sarcini de lucru, exerciții etc. ca itemi. Un set de itemi cu proprietăți specifice formează un test de evaluare. Pentru a evita confuziile terminologice, vom propune denumirea de Secvență de Itemi (SI –Sequence of Items) pentru un test de evaluare educațională. În plus, pentru a oferi mai multă claritate, vom stabili în cele ce urmează terminologia folosită în cadrul descrierii modelului.

Definiție 3.1. Un item este un tuplu (*id; st; dd; V; tp; t*) format din următoarele componente:

- **numărul de identificare al itemului id_q, care are rolul de a fi identificatorul unic al itemului în faza de implementare;**
- **enunțul st_q , care este format dintr-o frază sau un set de fraze care descrie datele inițiale și solicitările articolului, care trebuie rezolvate;**
- **setul de cuvinte cheie kw_q , care constă în lista de cuvinte cheie care descriu cel mai bine subiectul itemului;**
- **gradul de dificultate dd_q , $dd \in [0, 1]$; gradul de dificultate se calculează ca raport între numărul de răspunsuri incorecte la un anumit item și numărul total de**

răspunsuri. Gradul de dificultate poate fi calculat și folosind metoda prezentată în [22];

- **set de variante V_q (acolo unde este necesar), care poate fi format dintr-o listă de două sau mai multe răspunsuri posibile când tipul de item este multiplu sau este nul când tipul de item este scurt sau eseu;**
- **caracterul teoretic sau practic al itemului tp_q ; $tp \in \{0, 1\}$, unde 0 este teoretic și 1 este practic;**
- **tipul itemului t_q , $t \in$ {„multiplu", „scurt", „eseu"}, ilustrând tipul itemului, dacă are variante sau răspunsul este unul textual, dat de utilizator, în cazul unui item scurt și sau eseu.**

Setul de date despre item, notat în continuare prin BD1, conține articole care sunt generate automat folosind metode NLP sau introduse manual de un profesor. În setul de date BD1 putem vedea și componentele principale ale unui item general: tipul itemului (t_q), instrucțiunea st_q, mulțimea de variante V_q , lista de cuvinte cheie kw_q și gradul de dificultate dd_q.

Definiție 3.2. Funcția-obiectiv *fit*, *fit* : {1,2,…,n} → ℝ, este o funcție matematică ce furnizează o valoare specifică pentru un set de itemi (SI) dat și pentru valori configurate ale setului de restricții oferite de utilizator (ConSet). O formă generală a funcției este prezentată în .

$$\boldsymbol{fit}(\mathbf{OSI}) = \mathbf{0,25} \times \frac{\sum_{i=0}^{S} \boldsymbol{kw_{q_i}}}{\overline{\boldsymbol{S}}} \times (\mathbf{1} - \lceil \boldsymbol{DD_u} - \mathbf{DD}_{\boldsymbol{OSI}} \rceil) \times (\mathbf{1} - \left| \boldsymbol{TP_u} - \sum_{i=0}^{S} \boldsymbol{tp_{q_i}} \right|) \times \sum_{j=1}^{3} (\mathbf{1} - \left| \boldsymbol{QT_i} - \sum_{i=0}^{S} \boldsymbol{QT_{q_i}} \right|)$$

Pe scurt, funcția *fit* calculează valoarea mediei tuturor restricțiilor, după cum urmează:

- cea mai mare valoare medie a asemănării dintre cuvintele cheie date de utilizatori și cuvintele cheie ale itemilor, calculată folosind distanța de editare și metode specifice NLP;
- cea mai mică valoare a diferenței dintre gradul de dificultate dorit (DD_u) și gradul de dificultate calculat (DD_{OSI}) pentru SI;
- cea mai mică valoare a diferenței dintre raportul teoretic/practic dorit (TP_u) și calculat pentru SI;
- cea mai mică valoare a sumei diferențelor dintre componentele valorilor vectorului dorite

(QT_i) și calculate pentru SI care descriu tipul predominant de item.

3.1.2.3 Funcționalitatea modelului

Mecanismul de generare folosește un set predefinit de acțiuni care descriu generarea și evaluarea secvențelor optimizate de itemi generate anterior.

Datele de intrare constau în:

- subiectul dorit dat de setul de cuvinte cheie nr_k generate de utilizator kw = {kw_1 , kw_2 , …, kw_{nr_k} };
- numărul de întrebări necesare pentru fiecare cuvânt cheie nr_{kw} = {nr_kw_1 , nr_kw_2 , …, $nr_kw_{nr_k}$ }
- gradul de dificultate dorit DD_u;
- raportul teoretic-practic dorit TP_u;
- tipul de întrebare predominant dorit QT_u = <qt_m, qt_s, qt_e>.

Funcționalitatea modelului conține următorii algoritmi principali:

- algoritmul de generare a elementelor (care va fi notat în continuare prin *GenTest*), corespondent cu funcționalitatea func1 (I), care conține acțiuni legate de generarea specifică. Această acțiune poate fi stabilită urmând un set specific de pași:
  - **pasul 1**: Fiecare cuvânt cheie este analizat și pentru fiecare dintre ele se formează un grup de întrebări care au cuvinte cheie similare cu cel curent. Asemănarea este calculată folosind metode NLP și clusterele sunt formate folosind tehnica K-means bazată pe ML. Numărul nr_kw_i de întrebări este luat în considerare pentru fiecare cuvânt cheie kw_i.
  - **pasul 2**. Se formează setul de date parțial al întrebărilor C_i care poate fi utilizat pentru generarea testului. Principala cerință luată în considerare este subiectul testului.
  - **pasul 3**. Testul este generat pe baza altor cerințe utilizând un anumit tip de metodă (de exemplu, algoritmi genetici).
- algoritmul mecanismelor de verificare (care va fi notat în continuare cu *ChkItem*), corespondent cu funcționalitatea func2 (II), legată de verificarea automată a răspunsurilor, care va fi dezvoltat în cercetările viitoare;
- algoritmul de evaluare a răspunsului (care va fi notat în continuare cu *EvalStud*),

corespondent cu funcționalitatea func3 (III), care reprezintă partea din model responsabilă cu analiza învățării și care va fi descrisă în următoarele secțiuni. Acest algoritm folosește și algoritmul de predicție a articolului (care va fi notat în continuare cu *ItemPred*) și care va fi prezentat în ultima parte a secțiunii 3.

3.1.2.4 Descrierea componentelor principale ale modelului

3.1.2.4.1 Descrierea componentei func1 (I) folosind algoritmi de tip evolutiv

În această parte a lucrării, prezentăm și implementăm o dezvoltare a unui model care generează o secvență de itemi de evaluare bazat pe mai multe restricții, ale cărei itemi sunt extrași dintr-un set de itemi construit anterior. Dezvoltarea modelului se realizează folosind anumite tipuri de algoritmi evolutivi. Restricțiile pot avea diverse forme pe care le-am amintit anterior și sunt reprezentate de categoria generală/subiectul testului, gradul de dificultate al testului, raportul teoretic/practic al întrebărilor din test și tipul întrebării (cu răspunsuri multiple, răspuns scurt sau eseu). Acest model îmbunătățit este o dezvoltare a unui model prezentat și implementat în lucrările de cercetare anterioare, în care generarea itemilor de testare cu cerințe a fost realizată folosind un algoritm genetic, prin rafinarea sistemului de generare cu metode bazate pe machine learning (ML), precum clustering sau procesarea limbajului natural (NLP). Mecanismul de generare cuprinde datele de intrare, datele de ieșire și principalele subetape ale procesului de generare, împreună cu metodele și tehnicile utilizate pentru generare, și este prezentat în Figura 3.2.

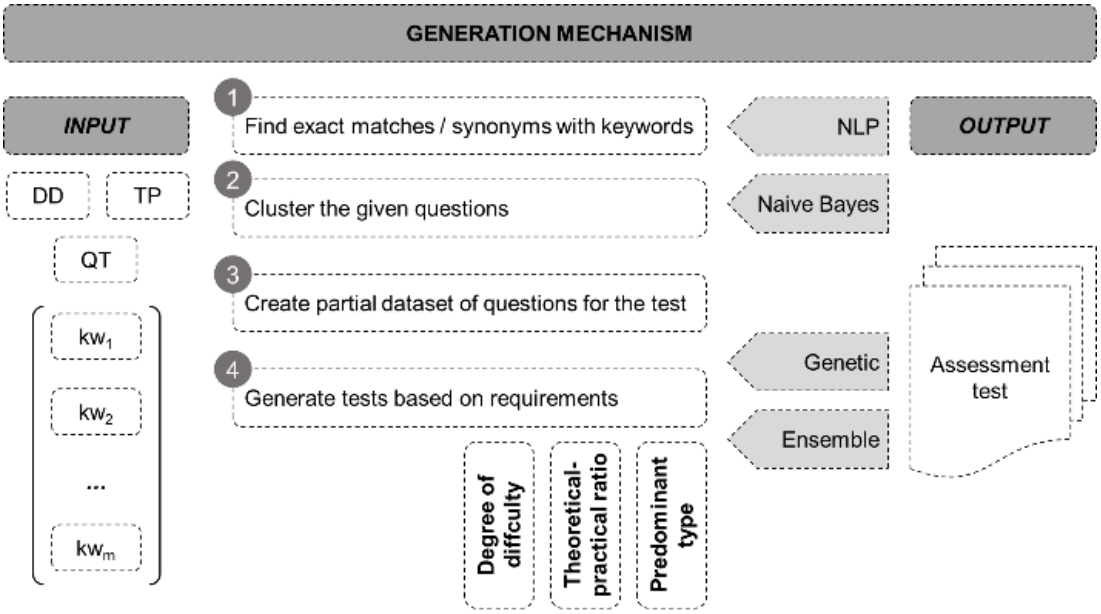


Figura 3.2. Descrierea vizuală a mecanismului de generare

Algoritmul genetic codifică elemente ca gene într-un cromozom, care va reprezenta o

secvență de itemi. În această problemă, vor fi necesare următoarele componente:

- cromozomul, pe care îl vom nota cu *C*, care codifică o secvență de itemi și care este format din următoarele componente:
  - numărul de ordine al cromozomului *id*, $id \in \{1, ..., S\}$, unde *S* este numărul total de itemi din baza de date;
  - setul de gene $G_j = \{g_j \mid j \in \{1, . . . , NrI\}\}$, unde $G = NrI$; $j = 1, nr_T$, $nr_T = n_1 + n_2 + ... + n_i$, $i = 1, n$;
  - funcția de fitness *f*, care codifică în cadrul algoritmului genetic funcția-obiectiv *fit*. Pentru ușurarea exemplificării modelului, considerăm că utilizatorul are în vedere doar restricții legate de gradul de dificultate al secvenței de itemi.
- operatorii genetici, cei utilizați în modelul prezentat fiind:
  - generarea unui cromozom *GenC*, un operator care formează un nou cromozom;

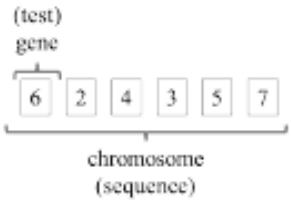


Figura 3.3. Forma generală a unui cromozom

  - mutația unui cromozom *MutC*, care poate fi definită după cum urmează: pentru un cromozom dat *C*, o poziție aleatorie în cromozom *pos*, $pos = 1, NrI$, și o genă selectată aleatoriu g_j, $j = 1, NrI$, operația de mutație este definită ca deplasarea genei g_{pos} cu gena g_j.

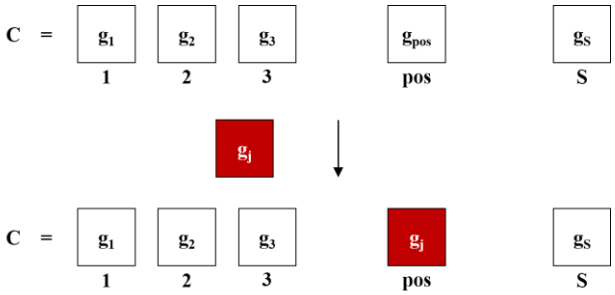


Figura 3.4. Reprezentarea vizuală a operatorului MutC

  - încrucișarea unui cromozom *CsvC*, care poate fi definită după cum urmează: pentru doi cromozomi dați C_1 și C_2 o poziție aleasă aleatoriu în cadrul celor doi cromozomi *pos*, prima parte a cromozomului C_1 până la gena g_{pos} este combinată cu a doua parte

a cromozomului C_2 din gena g_{pos} până la sfârșitul C_2 și prima parte a cromozomului C_2 până la gena g_{pos} este combinată cu a doua parte a cromozomului C_1 de la gena g_{pos} până la sfârșitul C_1, rezultând doi cromozomi noi C_1' și C_2'.

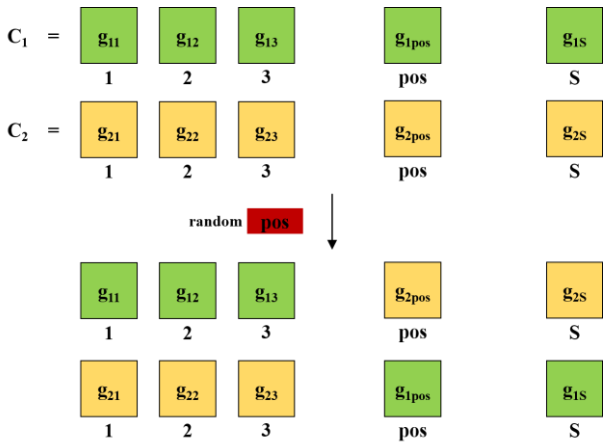


Figura 3.5. Reprezentarea vizuală a operatorului CsvC

- operatorul de sortare *SortC*, care sortează cromozomii în funcție de valoarea funcției fitness;

- parametrii algoritmului genetic: populația inițială de cromozomi (NrPop), numărul de generații (NG), rata mutației și rata de crossover.

Metodologia pentru algoritmul genetic cuprinde următorii pași:

1. Se citesc datele de intrare (numărul total de itemi din baza de date, numărul de itemi dorit din secvența de itemi, mărimea populației inițiale *NrPop*, cuvintele-cheie dorite de către utilizator și numărul lor, cuvintele-cheie ale fiecărui item).
2. Populația inițială este generată folosind în mod repetat operatorul *GenC*. Pentru fiecare cromozom, se calculează valoarea de fitness.
3. Se aplică operatorii genetici:
   a. Operatorul crossover *CsvC* este aplicat pentru doi cromozomi C_{k1} și C_{k2} aleși aleatoriu, $k = 1, NrPop$.
   b. Operatorul de mutație *MutC* este aplicat pentru un cromozom C_k ales aleatoriu, $k = 1, NrPop$.
   c. Valorile de fitness sunt calculate pentru cromozomii nou obținuți și se face sortarea pe baza acestor valori. Operația de sortare se face crescător pe baza valorii de fitness deoarece valoarea optimă a funcției de fitness este zero sau tinde către aceasta pentru un test optim care să respecte cerința legată de gradul de dificultate.

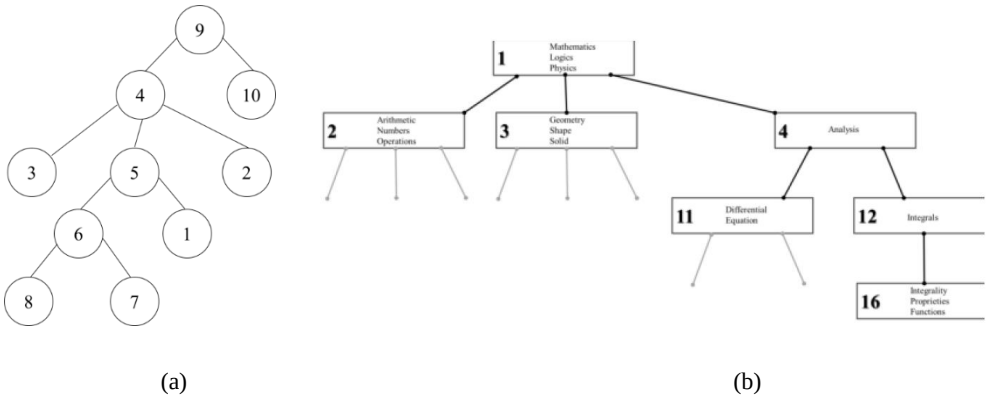
d. Subpașii 3(a) și 3(b) sunt aplicați pentru NG timpi.

4. Se extrage primul cromozom obținut.

3.1.2.4.2 Descrierea componentei func1 (I) folosind structuri de tip rețea (arbori)

Folosirea unei structuri de tip rețea pentru problema descrisă în secțiunea 3.1.1.1 este utilă în situația în care se urmărește generarea unei secvențe de itemi SI având în vedere apartenența itemilor în cadrul taxonomic al unui subiect. În acest sens, modelarea se rezumă la formarea unei rețele între itemii generați, itemi între care se stabilește o relație de ierarhie, în funcție de clasificarea taxonomică a lor sau de gradul secvențial de dificultate. Astfel, această dezvoltare a componentei func1 (I) este utilă în situațiile în care se dorește evaluarea unor cunoștințe pornind de la general la particular în ceea ce privește generalizarea unor concepte care au fost predate pe baza unui suport de text, pe care îl vom numi în continuare *corpus*. O altă situație care găsește soluții în această abordare este evaluarea generată pe baza unei creșteri graduale a gradului de dificultate al itemilor în cadrul secvenței de itemi. Vom prezenta în partea următoare a subsecțiunii o variantă actualizată a unor lucrări anterioare ale autorului [99], care prezintă un model și o implementare a acestuia bazată pe formularea unei baterii de secvențe de itemi. În această lucrare, modelarea a fost actualizată prin considerarea bateriei de SI ca fiind o SI și a secvențelor din arbore ca fiind itemi. De asemenea, modelarea a inclus alegerea unei metode din cadrul procesării limbajului natural (NLP) și a ontologiilor digitale pentru crearea dinamică a rețelei de itemi. Această cercetare este propusă pentru studii viitoare, unde se va lua în considerare și modelarea de relații ca cele descrise în secțiunile tezei referitoare la rețele.

În acest caz, se obține o structură arborescentă (un arbore) a itemilor, realizându-se o ierarhie bazată pe criterii specifice, cele studiate raportându-se la clasificarea taxonomică a subiectelor itemilor și la delimitarea succesivă a gradului de dificultate al itemilor. O reprezentare vizuală a unor astfel de arbori este prezentată în Figura 3.6.



(a) (b)

Figura 3.6. Exemple de reprezentări sub formă de graf ale arborilor utilizați în algoritmul care arată structura arborescentă

a relațiilor dintre itemi: (a) arbore folosit pentru gradul de dificultate; (b) arbore folosit pentru clasificarea taxonomică.

Având la dispoziție o astfel de reprezentare a itemilor, problema la care se rezumă această descriere este găsirea unui arbore parțial între nodul-rădăcină și orice nod-frunză. Rădăcina are cel mai mic grad de dificultate (deci, este cel mai ușor item), iar frunzele sunt cei mai dificili itemi. Rezultatul modelului este o secvență de itemi care îndeplinește două condiții:

- formează un arbore parțial al arborelui principal;
- numărul de muchii lipsă între nodurile din subarborele generat este zero sau minim, pe baza conexiunilor din arborele principal.

Modelul prezentat în această subsecțiune caută un arbore parțial format dintr-un număr dat de noduri care păstrează proprietatea arborescenței. Această proprietate înseamnă că nodurile (itemii) sunt legate între ele într-o conexiune de dificultate (ceea ce înseamnă că o muchie conectează un item cu un alt item mai dificil decât el) sau de taxonomie. Dificultatea ar putea fi măsurată folosind gradul de dificultate (care este raportul dintre răspunsurile corecte la test și numărul total de răspunsuri la test). O altă condiție a acestei căutări este ca un nod să poată fi atins numai prin nodul său părinte.

3.1.2.4.3 Descrierea componentei func2 (II)

Această componentă a modelului include verificarea răspunsurilor date de către utilizatori la itemii generați în cadrul altor componente. În cazul itemilor de tip răspuns multiplu, această verificarea automată a răspunsurilor este un proces trivial, realizându-se prin confruntarea răspunsurilor date de către respondenți la item cu răspunsurile corecte ale itemului.

Pentru celelalte două tipuri de itemi luate în calcul în cadrul modelului, verificarea răspunsurilor este un aspect complex, dată fiind natura răspunsurilor. În acest sens, scopul componentei modelului se rezumă la realizarea unei verificări automate cât mai apropiate de către un utilizator uman. Acest lucru poate fi realizat în mod curent folosind metode specifice ale Machine Learning (ML) dezvoltate în literatura de specialitate, bazate pe prelucrarea naturală a limbajului (NLP). În lucrare este descrisă o încercare de stabilire a unei verificări automate pentru răspunsuri deschise de tip scurt realizată într-o lucrare anterioară, care folosește distanța de editare pentru a determina gradul de similaritate dintre două răspunsuri.

3.1.2.4.4 Descrierea componentei func3 (III)

Componenta func3 (III) reprezintă o parte a modelului DMAIR, fiind unul dintre cele trei module principale care au fost prezentate la sfârșitul subsecțiunii precedente. În acest sens, acest model constă în modulul de evaluare a răspunsului. Acest modul urmărește determinarea

performanței de evaluare, în special în ceea ce privește performanța elevului pentru un anumit item.

Componenta func3 (III) este concepută pornind de la premisa că un răspuns incorect la un item poate indica faptul că subiectul itemului nu este pe deplin înțeles, mai ales în anumite condiții (de exemplu, elevul răspunde corect la alți itemi din SI-ul din cate face parte itemul, itemul primește în mod repetat răspunsuri greșite pentru mai mulți studenți etc.). În această chestiune, modelul ia în considerare mai mulți factori pentru a determina cauzalitatea directă dintre slaba înțelegere a subiectului și răspunsul incorect la un item cu subiectul respectiv. Modelul are o premisă simplă și este construit pe faza de generare a articolelor. La scurt timp, după aplicarea algoritmului *GenTest*, se face evaluarea itemilor, folosind metodologia descrisă anterior pentru *EvalStud*. O reprezentare vizuală a modelului poate fi văzută în Figura 3.7.

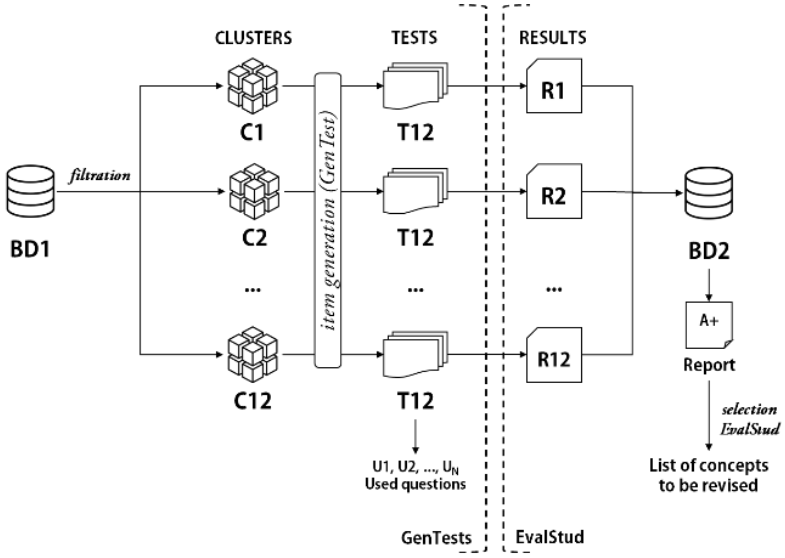


Figura 3.7. Reprezentarea vizuală a componentei func3 (III)

Funcționalitatea modelului constă în acțiunile care pot fi efectuate în cadrul modelului. Principalele două acțiuni constau în:

1. determinarea subiectelor care trebuie revizuite pe baza răspunsurilor date de elevi la evaluarea periodică, notate cu *EvalStud*;
2. determinarea probabilității ca un item să fie răspuns corect de către un elev sau de către un grup utilizând k-means clustering, notat cu *ItemPred*.

3.1.3 Implementarea sistemului de programe

3.1.3.1 Rezultate ale implementării

3.1.3.1.1 Descrierea implementării func1 (I) folosind structuri arborescente

Acum, vom lua un al doilea exemplu, care simulează o situație reală. Subiectele itemilor sunt legate de noțiuni referitoare la IT și calculatoare. Variabilele de intrare sunt: n=35, t=(0 1 1 2 15 15 3 3 4 4 4 4 9 25 2 10 11 11 12 7 7 8 8 8 8 5 5 6 6 20 20 21 22 23 24), kw = (hardware, PC, harddisk, memory, unit, external, reading, peripheral, software, application, browser, Internet). Arborele care poate fi construit cu datele prezentate anterior și cu toate cuvintele-cheie aferente nodurilor este prezentat în Figura 3.8.

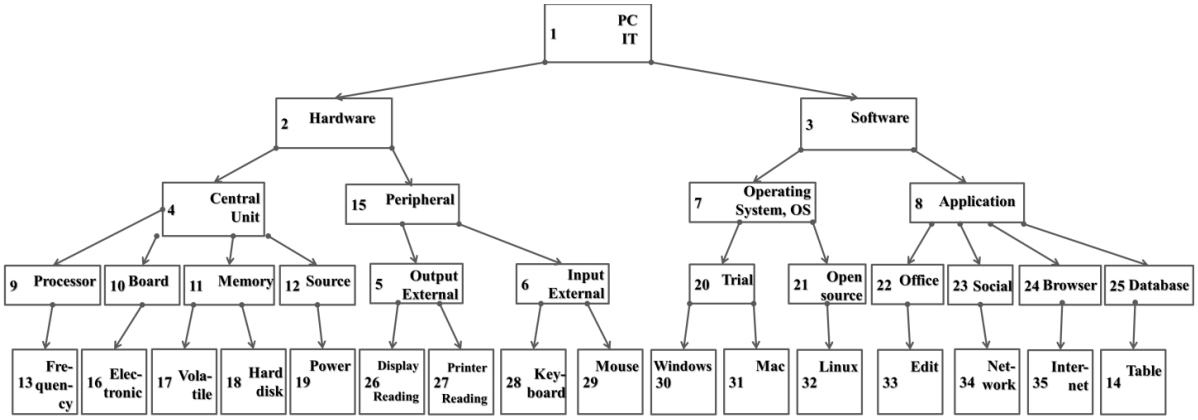


Figura 3.8. Arborele pentru exemplul dat

După ce rulăm programul, secvențele pe care le obținem sunt:

- L=(5,1,2,4,11,18) - hardware 1 times; PC 1 times; harddisk 1 times; memory 1 times; unit 1 times
- L=(5,1,2,15,5,27) - hardware 1 times; PC 1 times; external 1 times; reading 1 times; peripheral 1 times
- L=(5,1,2,15,6,29) - hardware 1 times; PC 1 times; external 1 times; reading 1 times; peripheral 1 times
- L=(5,1,3,8,25,36) - PC 1 times; software 1 times; application 1 times; browser 1 times; Internet 1 times

3.1.3.1.2 Descrierea implementării func1 (I) folosind algoritmi genetici

Înainte de a lua în calcul generarea testului, vom include o serie de rezultate referitoare la determinarea cuvintelor-cheie folosind tehnici de procesare a limbajului natural. Rezultatele se referă și la compararea cuvintelor-cheie stabilite de utilizator pentru un item cu cele generate automat prin această tehnică, așa cum a fost enunțat în secțiunea Anexe. Pentru exemplificarea

acestei componente a modelului, s-a realizat o implementare în context de laborator, care ia în considerare mai mult partea computațională decât o interfață cu utilizatorul. În timp ce pentru celelalte implementări a fost folosit un context de programare Java, pentru această implementare a fost preferat un algoritm Python, în principal din cauza ușurinței de lucru cu bibliotecile ML în acest mediu de programare. Setul de date de întrebări a luat în considerare 50 de întrebări cu caracteristicile lor, așa cum se arată în secțiunea Anexe. Pentru a arăta eficiența extragerii automate NLP a cuvintelor cheie, am folosit biblioteca WordNet pentru a extrage automat două cuvinte cheie din fiecare enunț al întrebărilor. Versiunile comparate ale cuvintelor cheie sunt prezentate în secțiunea Anexe.

După cum se poate observa, implementarea alege întrebările principale care sunt legate de cuvintele cheie care sunt generate de implementarea automată. Astfel, o mare importanță trebuie acordată perfecționării generării de cuvinte cheie, cu caracteristica suplimentară de a genera sinonime la cuvintele cheie găsite. De asemenea, se poate observa că există diferențe semnificative între cuvintele cheie ale utilizatorului și cuvintele cheie generate, datorită relației strânse dintre vocabularul enunțului și cuvintele cheie generate. Acest aspect poate fi îmbunătățit prin rafinarea algoritmului cu adăugarea de a găsi sinonime sau cuvinte apropiate la fel de înțeles ca cele din enunț.

Menționăm faptul că, pentru această implementare, s-a procedat la determinarea diferită a funcției fitness, care ia în calcul doi parametri: gradul de dificultate al itemilor și timpul de rezolvare al acestora. Astfel, definiția funcției $f: [0, +\infty) \rightarrow (0, \frac{1}{2}]$ este următoarea:

$$(C_j) = \frac{1}{|\sum_1^S g_i - DD_u| + |\sum_1^S g_i - T|} = \frac{1}{h}$$

Rolul funcției de fitness este de a verifica dacă combinația sumei gradelor de dificultate ale fiecărei întrebări din cadrul unui cromozom este mai mică și la fel de apropiată de valoarea DD_u și suma timpilor de rezolvare a fiecărei întrebări dintr-un cromozom este mai mică decât valoarea lui T, care ia valoarea timpului maxim de rezolvare dorit de către utilizator.

3.1.3.1.3 Descrierea implementării func3 (III)

Implementarea a fost realizată folosind limbajul de programare web PHP, iar interfața a fost creată folosind biblioteca Bootstrap, care se bazează pe limbaje HTML, CSS și JavaScript. O reprezentare a interfeței pentru generarea componentei de teste și a componentei de analiză a articolului este prezentată în figura următoare.

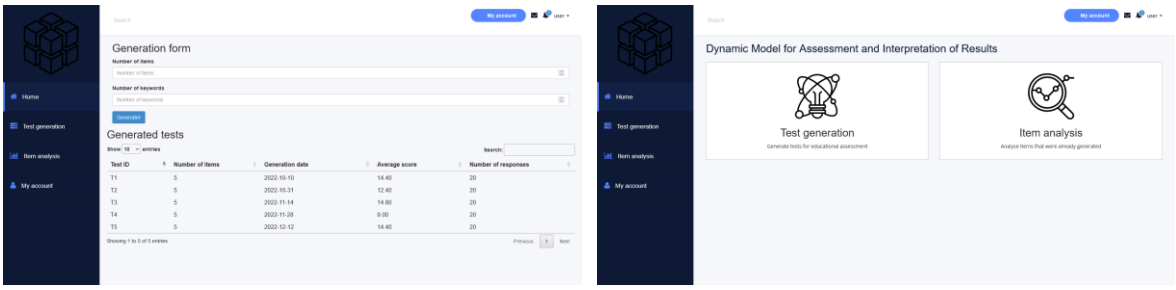


Figura 3.9. Capturi de ecran ale implementării func3 (III)

În ceea ce privește rezultatele obținute legate de implementare, a fost luat în considerare un context specific cu mai mulți parametri. Setul de date BD1 nu este prezentat în această lucrare din cauza cantității mari de date, dar este disponibil într-un depozit [30].

În urma răspunsurilor, mai mulți itemi au fost determinați ca fiind mai dificili decât ceilalți și lista subiectelor care pot fi revizuite care a fost obținută în urma analizei rezultatelor. Lista conține subiecte precum sisteme de operare, Windows OS, programare, Microsoft Word, formatare, algoritm, caracteristicile algoritmului și aplicații practice legate de programare. În această chestiune, numărul de articole care au fost selectate a fost de aproximativ 27% din numărul total de articole. Selecția articolelor s-a făcut pe baza unui prag care are o semnificație statistică, ca în cazul numărului superior-inferior, care reprezintă 27% din numărul total de articole. Pentru seturi mari de articole, vor fi selectații itemii care au obținut un punctaj mai mic de 27% din punctajul maxim al testului. Itemii care au generat aceste subiecte revizuibile au fost Q2 din Testul 1, Q4 și Q5 din Testul 2 și Q2 și Q5 din Testul 4, care au obținut cel mai mic număr de răspunsuri corecte.

Tabelul 3.1. Elementele care au fost selectate și caracteristicile acestora

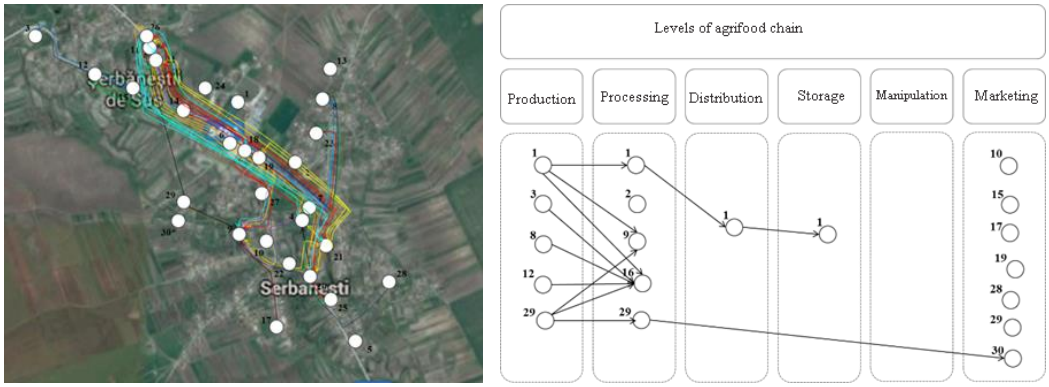
	Scor	**dd_q**	**sd_q**	**d_q**	**pbs_q**	**m_q**	**l_q**
T1Q2	4	0,80	0,40	0,6	0,06	0,20	4
T2Q4	6	0,70	0,46	0,8	0,28	0,30	6
T2Q5	5	0,75	0,43	0,8	0,55	0,25	5
T4Q2	5	0,75	0,43	0,2	-0,22	0,25	5
T4Q5	4	0,80	0,40	0,4	0,11	0,20	4

Analiza itemilor a confirmat că itemii menționați au avut cel mai mare grad de dificultate. Ceilalți parametri legați de validitatea testului au arătat că majoritatea întrebărilor au fost concepute corespunzător.

3.2 Direcții de cercetare referitoare la modelarea rețelelor

Un alt exemplu care poate fi luat în calcul pentru a studia modelarea rețelelor pornește de la o situație complexă din contextul economic. Acest studiu este util și relevant pentru sistemul de programe descris în secțiunea anterioară prin intermediul aplicării lui în analiza conectivității itemilor. Descrierea aplicării lui este un punct important în activitatea de cercetare viitoare. Introducem descrierea acestei cercetări prin prisma prezentării unui studiu de caz referitor la modelarea sistemelor concurente. În acest sens, secțiunea curentă prezintă un model formulat pentru rețeaua de întreprinderi economice dintr-o anumită piață economică. Harta economică a unei piețe geografice este structurată sub forma unui graf ordonat. Vârfurile grafului sunt reprezentate de întreprinderi, iar muchiile sunt considerate relații între întreprinderi. Această reprezentare scoate în evidență câteva caracteristici cheie ale sistemului: dualitate de vârfuri, contractilitate, capacitate de expansiune, non-izolare și interferență în comerțul on-line. Elementele de bază ale modelului sunt formate din bazele matematice ale teoriei grafurilor și din unele metode utilizate în domeniu pentru a determina diferite proprietăți ale grafurilor (Roberts, Fred S, 1989). În această privință, piața este modelată ca un grafic direcționat, vârfurile reprezentând întreprinderile, iar muchiile relațiile economice care sunt sau pot fi stabilite între vârfuri (Quoniam, L. et al, 2007). Metodologia modelului este compusă din următorii pași:

1. se analizează întreprinderile şi se găsesc parametrii necesari acestor întreprinderi.
2. se constată afinităţile între perechile de întreprinderi.
3. se stabilesc relaţiile potenţiale între întreprinderi.
4. unele caracteristici ale pieței sunt găsite, analizate și studiate cu ajutorul proprietăților grafurilor.
5. în final, sunt formulate sugestii legate de dezvoltarea economică.



. Figura 3.10. (a) Harta economică sub formă de graf pentru o piață geografică; (b) Vizualizarea lanțurilor agroalimentare obținute în urma selecției

4 CAPITOLUL IV. MODELAREA SISTEMELOR DE PROGRAME ȘI A REȚELELOR

Rezumat. *Capitolul IV cuprinde descrierea unui model formal al sistemului descris în Capitolul III, corespondent modelului matematic propus. Structurat pe trei secțiuni principale, acest capitol descrie formalizarea sistemului folosind metodologii specifice. Secțiunea 4.1. începe cu prezentarea procesului de modelare formală a sistemului folosind limbajul UML, având ca rezultat o serie de diagrame ale modelului. Procesul continuă cu modelarea sistemului folosind mașini cu număr finit de stări și rețele Petri, luând în calcul structura modulară a sistemului. Nu în ultimul rând, analiza continuă cu secțiunea 4.2, care cuprinde descrierea modelării formale a rețelei propuse în Capitolul III. În final, secțiunea 4.3 realizează descrierea implementării modelului formal al sistemului conceput în secțiunea 4.1 în mediul de modelare NuSMV.*

Pentru ca verificarea modelelor să verifice un model al unui sistem de programe, trebuie să se efectueze trei sarcini principale:

- Prima sarcină este modelarea, în care proiectarea sistemelor este transformată într-un formalism acceptat de instrumentul de verificare a modelului. În unele cazuri, aceasta este o sarcină simplă, în timp ce sistemele complexe pot necesita utilizarea abstractizării pentru a elimina detaliile nerelevante sau neesențiale ale sistemului.
- A doua sarcină este specificarea; adică declararea proprietăților modelului pe care sistemul trebuie să le satisfacă. Instrumentele de verificare a modelelor utilizează de obicei logica temporală, care poate afirma cum evoluția comportamentului sistemului în timp.
- Sarcina finală este verificarea. În mod ideal, această sarcină se realizează într-un mod complet automatizat. Instrumentul de verificare a modelului va oferi o urmă de eroare (contra-exemplu) care ajută la localizarea unde a apărut o eroare în cazul unui rezultat negativ.

Modelarea, verificarea și validarea sistemului de programe descris folosind metodele și tehnicile consacrate este următorul pas în ceea ce privește cercetarea temei propuse în Capitolul III. Această continuare este necesară în vederea formalizării și stabilirii caracterului riguros al dezvoltării practice a sistemului de programe propus. În acest sens, în acest capitol se vor avea în vedere modelarea formală a sistemului de programe descris în capitolul anterior folosind tehnicile de modelare din literatura de specialitate. Acest proces include modelarea unora dintre componentele sistemului folosind Unified Model Language (UML), modelarea sistemului de

programe prin intermediul mașinilor cu număr finit de stări (Finite State Machine – FSM) și prin intermediul rețelelor Petri. De asemenea, se va avea în vedere modelarea componentei descrise în secțiunea 0 ca un sistem concurent marcat în forma unei rețele de itemi. După realizarea modelării, vom include în cadrul cercetării verificarea și validarea prin intermediul tehnicilor de model checking a unor proprietăți care vor fi specificate în funcție de aspecte ale proiectării modelului propus în capitolul anterior și care au o însemnătate ridicată în determinarea unor neajunsuri ale sistemului de programe. Aspectele legate de verificarea și validarea modelului vor fi prezentate prin intermediul reprezentării modelului în instrumente specifice, cum ar fi NuSMV.

4.1 Modelarea formală a sistemului de programe descris

4.1.1 Modelarea folosind UML

Vom realiza în această secțiune modelarea componentei de generare a secvențelor printr-un algoritm genetic folosind limbajul UML (Unified Model Language). Pentru aceasta, vom formula diagrame de stări și diagrame de clase pentru modelul matematic propus. Înainte de a stabili aceste modele de detaliu, vom schița un model al întregului sistem folosind o diagramă de componente. Aceasta poate fi vizualizată în Figura 4.1.

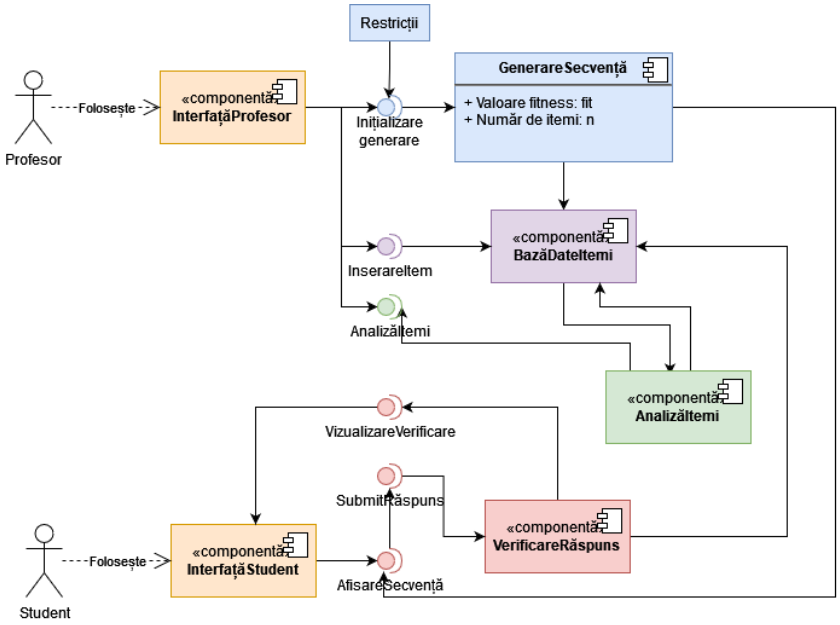


Figura 4.1. Diagrama de componente pentru sistemul propus

Vom stabili în continuare comportamentul sistemului în ceea ce privește componenta GenerareSecvențe, stabilind astfel aspectele principale legate de generarea secvențelor de itemi. În Figura 4.2 se poate vizualiza diagrama de stări a componentei de generare a secvențelor de itemi folosind algoritmi genetici.

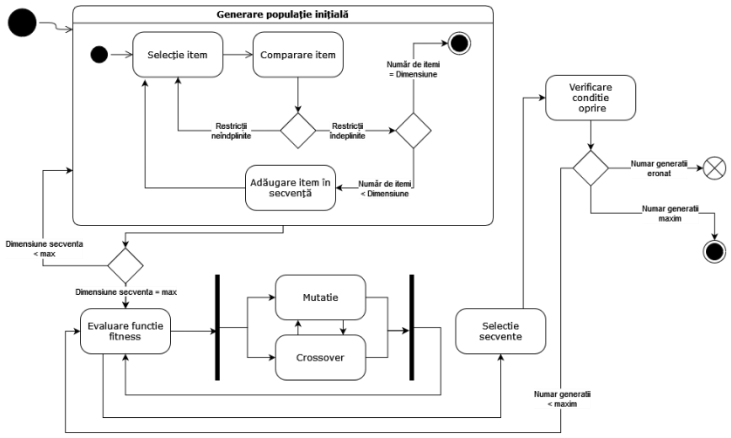


Figura 4.2. Diagrama de stări a componentei de generare a unei secvențe de itemi folosind algoritmi genetici

Diagrama prezentată arată în mod detaliat stările generatorului de secvențe de itemi, pornind de la generarea populației inițiale și având ca rezultat final o secvență optimizată de itemi. Am inclus în cadrul acestei diagrame subcomponenta referitoare la generarea unui item, care se realizează în cadrul activității generale de generare a populației inițiale.

4.1.2 Modelarea prin intermediul mașinilor cu număr finit de stări (FSM)

Modelarea formală prin intermediul mașinilor cu număr finit de stări implică determinarea elementelor constituente ale unei astfel de mașini: stări, tranziții între stări și acțiuni care determină aceste tranziții. Pentru modelul prezentat în Capitolul III, vom stabili pentru cele trei componente principale câte o mașină cu număr finit de stări: prima dintre ele modelează obținerea unei secvențe optime de itemi folosind un algoritm genetic (o vom nota cu GFSM-1), cea de-a doua modelează verificarea răspunsurilor date la o secvență rezolvată (o vom nota cu GFSM-2) și cea de-a treia modelează analiza răspunsurilor pentru un item (notată cu GFSM-3). Pentru detalierea primei mașini, vom construi mașina cu număr finit de stări GFSM-1.1, care modelează obținerea unei singure secvențe de itemi. Mai mult, pentru a studia în mod specific anumite proprietăți, includem și modelarea unei secvențe de itemi prin intermediul creării mașinii GFSM-1.2. Pentru ușurința modelării, s-a ales reprezentarea FSM sub formă de mașini Moore.

4.1.2.1 Mașina cu număr finit de stări GFSM-1

Mașina cu număr finit de stări GFSM-1 modelează obținerea unei secvențe optime de itemi folosind un algoritm genetic. Acest model va fi folosit pentru a studia anumite proprietăți ale algoritmului genetic, cum ar fi proprietatea de asigurare a obținerii unei secvențe optime. Componentele principale ale mașinii GFSM-1 sunt:

- alfabetul Σ: $\Sigma = \{NG, NP, rm, rc\}$;

- mulțimea de stări S: $S = \{INIT, s_1, s_2, s_3, s_4, s_5, s_6, STOP, ERR\}$;
- starea inițială s_0: s_0 = INIT;
- mulțimea de acțiuni ACT: $ACT = \{act_i | 1 \leq i \leq 13\}$;
- mulțumea de tranziții corespondente acțiunilor T: $T = \{T_i | 1 \leq i \leq 13\}$;
- mulțimea de stări finale F: $F = \{STOP, ERR\}$.

Alfabetul cuprinde numărul de generații al algoritmului genetic, variabilă care, de altfel, determină oprirea algoritmului. Tabelul 4.1 cuprinde descrierea stărilor amintite pentru mașina GFSM-1.

Tabelul 4.1. Descrierea stărilor pentru mașina GFSM-1

#	Stare	Descrierea stării	Valoare *ng*
1.	*INIT*	În această stare, se poate începe generarea secvențelor de itemi. Sunt inițializate valorile parametrilor algoritmului genetic.	0
2.	s_1	Este generat un cromozom în populația inițială de cromozomi.	0
3.	s_2	Este evaluată valoarea funcției fitness pentru fiecare cromozom din populația de cromozomi.	1..NG
4.	s_3	Este realizată operația de mutație a cromozomilor, pentru a forma un caracter de diversitate al populației.	1..NG
5.	s_4	Este realizată operația de recombinare (crossover) a cromozomilor.	1..NG
6.	s_5	Este realizată selecția celor mai buni cromozomi prin intermediul sortării.	1..NG
7.	s_6	Este realizată verificarea atingerii condiției de oprire a algoritmului (atingerea numărului maxim de generații).	1..NG
8.	*STOP*	Este atinsă valoarea maximă a numărului de generații.	NG
9.	*ERR*	Este starea în care se numărul de generații are o valoare în afara mulțimii de valori obișnuite.	ng ∉ {0, …, NG}

Mai departe, vom descrie mulțimea de acțiuni care descriu tranzițiile dintre stări, în Tabelul 4.2.

Tabelul 4.2. Descrierea acțiunilor și a tranzițiilor pentru mașina GFSM-1

#	Acțiune	Descrierea acțiunii	Tranziție		
			Stare inițială	Stare finală	Input

1.	act_1	Descrie inițializarea populației primare de cromozomi și stabilirea parametrilor algoritmului genetic.	*INIT*	s_1	$NG = 0$ $NP = 0$ $rm = 0$ $rc = 0$
2.	act_2	Descrie evaluarea valorii funcției de fitness a cromozomilor din populația de cromozomi.	s_1	s_2	$NG = 0$ $NP = max$ $rm = 0$ $rc = 0$
3.	act_3	Descrie generarea următorului cromozom din populația de cromozomi.	s_1	s_1	$NG < max$ $NP < max$ $rm = 0$ $rc = 0$
4.	act_4	Descrie realizarea operației de mutație a cromozomilor.	s_2	s_3	$NG < max$ $NP = max$ $rm = 1$ $rc = 0$
5.	act_5	Descrie realizarea operației de recombinare (crossover) a cromozomilor.	s_2	s_4	$NG < max$ $NP = max$ $rm = 0$ $rc = 1$
6.	act_6	Descrie realizarea operației de selectare a cromozomilor, realizată prin sortarea lor în funcție de valoarea caracteristică a funcției fitness.	s_2	s_5	$NG < max$ $NP = max$ $rm = 1$ $rc = 1$
7.	act_7	Descrie evaluarea valorii de fitness pentru un cromozom mutat.	s_3	s_2	$NG < max$ $NP = max$ $rm = 1$ $rc = 1$
8.	act_8	Descrie realizarea operației de recombinare (crossover) a cromozomilor, provenind din mutație.	s_3	s_4	$NG < max$ $NP = max$ $rm = 1$ $rc = 0$
9.	act_9	Descrie evaluarea valorii de fitness pentru un cromozom rezultat din crossover.	s_4	s_2	$NG < max$ $NP = max$ $rm = 0$ $rc = 1$
10.	act_{10}	Descrie realizarea operației de mutație a cromozomilor, provenind din crossover.	s_4	s_3	$NG < max$ $NP = max$ $rm = 0$ $rc = 1$
11.	act_{11}	Descrie verificarea condiției de oprire a algoritmului și incrementarea numărului de generații curent.	s_5	s_6	$NG < max$ $NP = max$ $rm = 1$ $rc = 1$
12.	act_{12}	Descrie acțiunea de finalizare a algoritmului și afișarea secvenței optime de itemi.	s_6	*STOP*	$NG = max$ $NP = max$ $rm = 1$

					rc = 1
13.	act_{13}	Descrie depășirea numărului de generații ale algoritmului.	s_6	*ERR*	*NG > max* *NP = max* *rm = 1* *rc = 1*

Reprezentarea vizuală a mașinii GFSM-1 poate fi vizualizată în Figura 4.3.

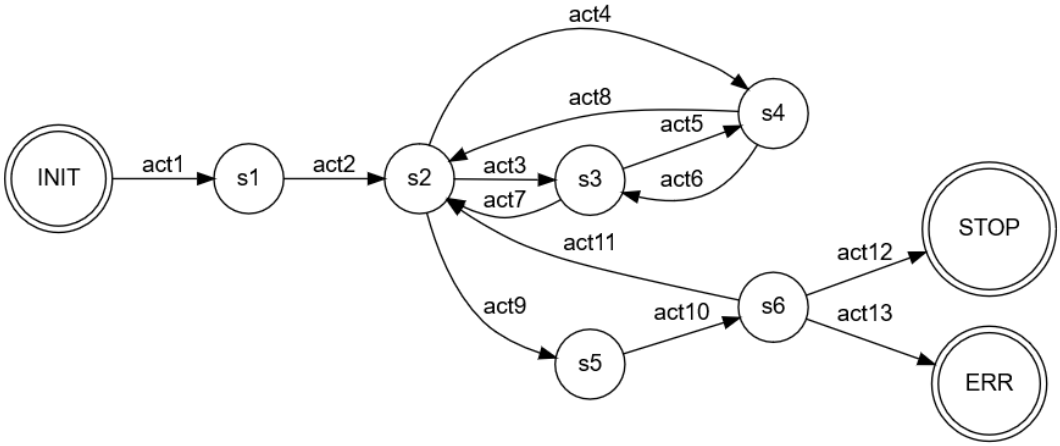


Figura 4.3. Reprezentarea vizuală a mașinii GFSM-1

4.1.2.2 Mașina cu număr finit de stări GFSM-1.2

Mașina cu număr finit de stări GFSM-1.2 modelează un cromozom care trece prin diverse stări, luând în calcul valoarea de optim a funcției de fitness. Componentele principale ale mașinii GFSM-1.2 sunt:

- alfabetul Σ: $\Sigma = \{fitness, max, rm, rc, M\}$;
- mulțimea de stări S: $S = \{INIT, NG, G, M, P, C, O, NS\}$;
- starea inițială s_0: s_0 = NG;
- mulțimea de acțiuni ACT: $ACT = \{act_i | 1 \leq i \leq 19\}$;
- mulțimea de tranziții corespondente acțiunilor T: $T = \{T_i | 1 \leq i \leq 19\}$;
- mulțimea de stări finale F: $F = \{O, NS\}$.

Alfabetul cuprinde valorile de la 1 la numărul de itemi din baza de date. Reprezentarea vizuală a mașinii GFSM-1.2 poate fi vizualizată în Figura 4.4.

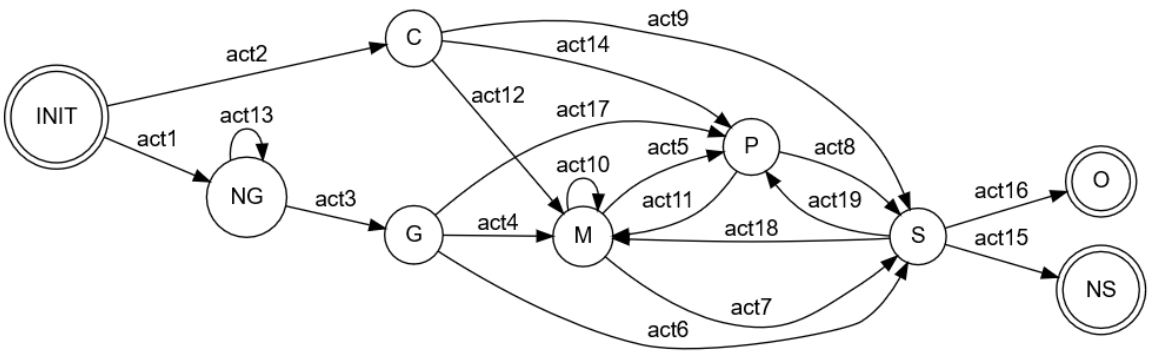


Figura 4.4. Reprezentarea vizuală a mașinii GFSM-1.2

Trebuie menționat faptul că unele tranziții depind de variabile. Astfel, tranzițiile legate de generarea unei secvențe depind de dimensiunea maximă a secvenței, iar tranzițiile corespondente activităților 15 și 16 se realizează doar în momentul în care a fost atins numărul maxim de generații. De asemenea, operația de selectare depinde de cardinalul unei mulțimi de selectare, egală cu dimensiunea populației inițiale, care se schimbă cu fiecare generație. Operația de selectare depinde, de asemenea, de comparațiile dintre valorile funcției fitness calculate pentru fiecare cromozom. În starea finală *O* ajunge o singură secvență sau, după caz, în funcție de parametrul legat de selecția finală a cromozomilor, un număr dorit de secvențe.

Mașina cu număr finit de stări GFSM-1.2 poate fi folosită mai departe ca model pentru verificarea unor proprietăți specifice comportamentului algoritmului genetic care țin de performanța acestuia. Proprietățile se pot referi la valoarea maximă a funcției fitness sau de convergere a algoritmului către un optim global. Mulțimea de proprietăți include și proprietăți de safety, cum ar fi aceea că nicio secvență-părinte nu poate deveni o secvență-copil.

4.1.3 Modelarea folosind rețele Petri

Modelarea propusă anterior stabilește comportamentul sistemului pentru componentele propuse în secțiunile anterioare. Este, însă, utilă și realizarea unui model al comportamentului sistemului propus din perspectiva resurselor folosite și a caracterului concurent al elementelor acestuia. Din acest punct de vedere, modelarea folosind o rețea Petri va permite studiul unor componente dintr-o perspectivă unitară referitor la componentele modelului.

Vom defini rețeaua Petri folosită pentru sistemul descris, pe care o vom nota cu G-PN, prin intermediul componentelor ei. Aceste componente sunt:

- mulțimea pozițiilor / locațiilor Loc: $Loc = \{ P_i \mid 0 \leq i \leq \}$;
- mulțimea tranzițiilor Tr: $Tr = \{ T_i \mid 1 \leq i \leq \}$;
- mulțimea arcelor A: $A \subseteq (Loc \times Tr) \cup (Tr \times Loc)$;

- mulțimea marcajelor M: $M: P \rightarrow \{0, 1, 2, \dots\}$

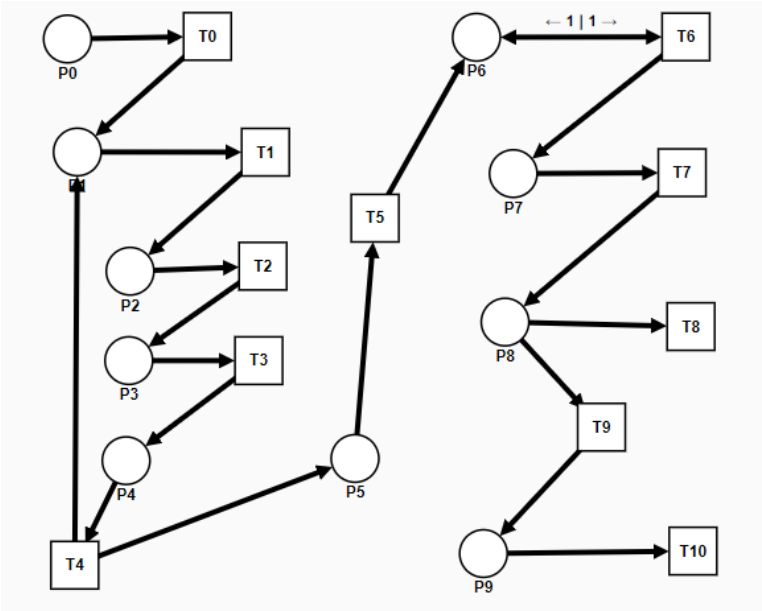


Figura 4.5. Rețeaua Petri obținută pentru sistemul descris

4.2 *Direcții de modelare a unui sistem concurent de tip rețea*

Mai departe, pentru a include aspecte legate de sistemul concurent, vom studia modificarea acestei rețele pentru doi consumatori. Rețeaua Petri obținută conține următoarele componente:

- mulțimea de locuri P: $P = \{p_i \mid 1 \leq i \leq 7\}$;
- mulțimea de tranziții T: $T = \{t_i \mid 1 \leq i \leq 6\}$
- mulțimea de arce A;
- funcția de marcaj M: $M: P \rightarrow \mathbb{N}$.

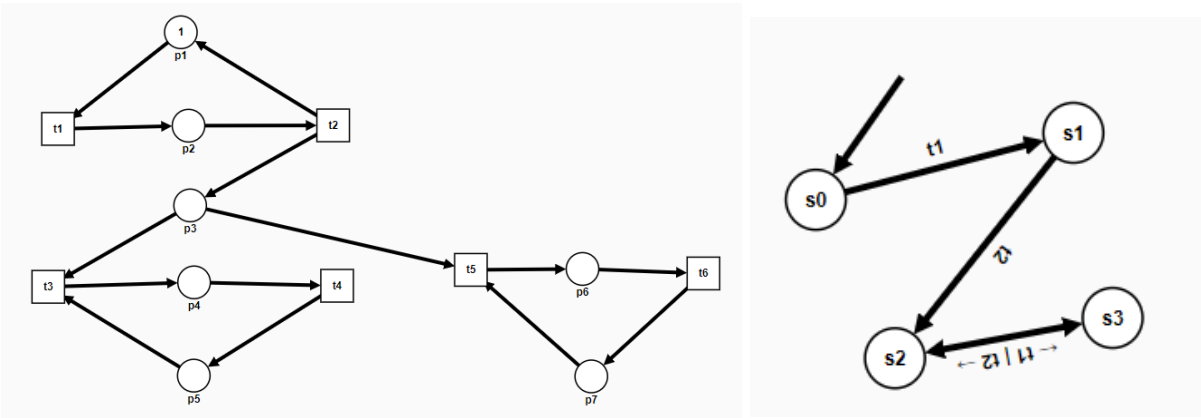


Figura 4.6. Rețeaua Petri obținută și caracteristicile ei pentru doi consumatori

În lipsa altor precizări, cei doi consumatori au drepturi egale de a consuma. Pentru a

schimba acest element și a introduce reguli referitoare la consum, putem introduce un arc inhibitor de la p5 la t5, ceea ce va indica faptul că t5 va fi validată numai dacă p3 nu conține token (jeton). Astfel, modelul poate fi modificat pentru o serie de consumatori, stabilind astfel studiul comportamentului agenților în cadrul sistemului economic în contextul stabilirii relațiilor economice dintre aceștia.

4.3 Implementarea unei mașini cu număr finit de stări în NuSMV

Pentru a realiza verificarea modelului, am realizat implementarea mașinii GFSM-1, prezentată în secțiunea 4.1.2.1, în limbajul NuSMV.

În cadrul implementării, se poate observa că se iau în calcul diverse variabile referitoare la trecerea dintr-o stare în alta, în așa fel încât să se evite situații de deadlock sau de buclare infinită dintr-o stare în alta. Am luat în calcul faptul că algoritmul de generare depinde de dimensiunea populației inițiale, de dimensiunea unei secvențe, de rata mutației și de rata de recombinare (crossover). În cadrul modelului, am stabilit o serie de valori pentru variabilele amintite mai sus. În cadrul modelului, variabilele *mut* și *crs* codifică rata mutației și rata de recombinare (crossover), în forma unei fracțiuni din dimensiunea populației inițiale. Totodată, ele împiedică ciclarea la infinit a unei secvențe de stări care să le cuprindă pe ambele. Celelalte două variabile codifică numărul de generații ale algoritmului genetic (ales în model ca 10), care coincide și cu alfabetul mașinii de număr finit de stări GFSM-1, și dimensiunea unei secvențe (aleasă în model ca 5), care înseamnă dimensiunea unui test de evaluare pentru elev.

În urma construirii modelului, am obținut un număr de stări „reachable” de 85:

```
##########################################################
system diameter: 85
reachable states: 85 (2^6.40939) out of 2376
(2^11.2143)##################################################
```

Pentru a putea verifica unele aspecte legate de verificarea modelului, vom stabili unele proprietăți pe care le vom verifica în cadrul celor două modele în capitolul următor.

5 CAPITOLUL V. VERIFICAREA ȘI VALIDAREA SISTEMELOR DE PRODUCȚIE PRIN MODEL CHECKING ȘI ALGORITMI GENETICI

Rezumat. *Capitolul V este o continuare firească a părții anterioare a tezei, care include descrierea proceselor de verificare a modelului și de validare a implementării acestuia. Este structurat pe două secțiuni principale care delimitează rezultatele obținute din aplicarea acestor procese. Secțiunea 5.1 integrează acțiunile de specificare a unor proprietăți ale modelului formal obținut în capitolul anterior, de verificare a lor și de validare a implementării. Specificarea proprietăților modelului include generarea unor proprietăți, inclusiv de tip safety și liveness, folosind logici temporale. Verificarea acestor proprietăți a fost realizată în mediul NuSMV prin transpunerea lor în limbajul specific și rularea acestora în mediul de modelare. Nu în cele din urmă, a fost realizată validarea implementării sistemului, realizată în limbajul de scripting PHP, folosind cadrul de testare PHPUnit pentru o serie de clase și metode implementate. În final, secțiunea 5.2 propune o direcție de dezvoltare viitoare a verificării automate a proprietăților modelului descris prin folosirea algoritmilor genetici pentru spațiul de stări existent al modelului.*

5.1 Verificarea și validarea sistemelor de programe propuse

5.1.1 Specificarea proprietăților modelului

Pentru a putea face o verificare a modelului, vom propune o serie de proprietăți folosind logici temporale. Aceste proprietăți vor include proprietăți de „safety” (nimic rău nu se poate întâmpla) și „liveness” (se va întâmpla ceva bun la un moment dat). În această parte descriem aceste proprietăți, mai întâi folosind limbajul formal, și mai apoi vom verifica aceste proprietăți în cadrul interpretorului NuSMV.

Pentru modelul GFSM-1, dintre proprietățile de safety, propunem mulțumea de proprietăți $\phi_{GFSM-1S} = \{\phi_1, \phi_2\}$, formulate în logica CTL:

- $\phi_1: AG\neg(state = ERR)$, care corespunde proprietății conform căreia nu se va atinge starea de eroare în niciuna dintre căi. În limbajul NuSMV, proprietatea este tradusă astfel: $AG\,!(state\ =\ ERR)$.
- $\phi_2: AG(NP < 5) \rightarrow (state = s1)U(NG = 0)$: această proprietate implică faptul că în orice stare a modelului în care numărul curent de itemi din secvență este mai mic decât dimensiunea secvenței, trebuie ca starea curentă să fie starea s1 și numărul de generații al algoritmului să fie 0.

Dintre proprietățile de liveness pentru aceeași mașină, propunem mulțimea de proprietăți $\phi_{GFSM-1L} = \{\phi_3, \phi_4\}$, formulate în logica CTL:

- $\phi_3: EF(state = STOP)$, care corespunde proprietății conform căreia starea STOP, care descrie atingerea unei soluții, este atinsă pe una dintre căile mașinii. Se poate traduce prin faptul că algoritmul are cel puțin o soluție viabilă. În limbajul NuSMV, proprietatea este tradusă astfel: $F(state = STOP)$.
- $\phi_4: EF(NG = 10)\&\&AG(AF(state = STOP))$: această proprietate conține două sub-proprietăți, prima dintre ele verificând dacă într-un moment viitor numărul de generații atinge numărul maxim propus în model, iar cea de-a doua verifică dacă pentru fiecare stare a sistemului există o cale care, în orice moment ulterior, atinge starea STOP.

Stabilim și o proprietate de tip fairness și una de tip deadlock în mulțimea de proprietăți $\phi_{GFSM-1P} = \{\phi_5, \phi_6\}$, pentru modelul GFSM-1:

- $\phi_5: AG\ (state = s3 \rightarrow EX(state = s4\ \&\&\ EX(state = s3))$, care verifică proprietatea de a nu ajunge într-o stare de deadlock între mutație și crossover.
- $\phi_6: AG\ (state = INIT \rightarrow AF(state = STOP))$: această proprietate asigură că, dacă sistemul ajunge în starea inițială INIT, în orice moment ulterior aceasta va ajunge în starea finală dorită.

Pentru modelul GFSM-1.2, dintre proprietățile de safety, propunem mulțumea de proprietăți $\phi_{GFSM-1.2S} = \{\phi_7, \phi_8\}$, formulate în logica CTL:

- $\phi_7: AG\neg(state = NS)$, care corespunde proprietății conform căreia nu se va atinge starea de eroare în niciuna dintre căi.
- $\phi_8: \neg\ (state = P\ \rightarrow\ state = C)$, care propune verificarea faptului că din starea de secvență-părinte nu se poate ajunge în starea de secvență-copil pentru operația de recombinare (crossover).

Dintre proprietățile de liveness pentru aceeași mașină, propunem mulțimea de proprietăți $\phi_{GFSM-1.2L} = \{\phi_9, \phi_{10}\}$, formulate în logica CTL:

- $\phi_9: EF(state = OP)$, care corespunde proprietății conform căreia starea OP (O), care descrie atingerea unei soluții, este atinsă pe una dintre căile mașinii. Se poate traduce prin faptul că algoritmul are cel puțin o soluție viabilă.
- $\phi_{10}: AG\ (max > fit)$: proprietatea verifică faptul că întotdeauna valoarea funcției fitness

va avea un maxim.

5.1.2 *Verificarea proprietăților modelului*

Pentru fiecare dintre aceste proprietăți, am rulat modelele în NuSMV și am obținut datele din Tabelul 5.1.

Tabelul 5.1. Rezultatele obținute pentru proprietățile definite

Formulă	**Sintaxă în NuSMV**	**Rezultat**	**Contraexemplu**
$\phi_1: AG\neg(state = ERR)$	AG !(state = ERR)	true	-
$\phi_2: AG(NP < 5) \rightarrow (state = s1)U(NG = 0)$	AG (NP < 5) -> (state = s1) & AG(NG=0)	true	-
$\phi_3: EF(state = STOP)$	EF state = STOP	true	-
$\phi_4: EF(NG = 10)\&\&AG(AF(state = STOP))$:	(EF NG = 10 & AG (AF state = STOP))	true	-
$\phi_5: AG\ (state = s3 \rightarrow EX(state = s4\ \&\&\ EX(state = s3))$	AG (state = s3 -> EX (state = s4 & EX state = s3))	false	-> State: 1.1 <- INIT -> State: 1.2 <- state = s1 -> State: 1.3 <- NP = 1 -> State: 1.4 <- NP = 2 -> State: 1.5 <- NP = 3 -> State: 1.6 <- NP = 4 -> State: 1.7 <- NP = 5 -> State: 1.8 <- state = s2 -> State: 1.9 <- state = s3
$\phi_6: AG\ (state = INIT \rightarrow AF(state = STOP))$:	AG (state = INI -> AF state = STOP)	true	-
$\phi_7: AG\neg(state = NS)$	AG !(state = NS)	true	-
$\phi_8: \neg\ (state = P \rightarrow state = C)$		false	-> State: 1.1 <-
$\phi_9: EF(state = OP)$	EF state = OP	true	-
$\phi_{10}: AG\ (max > fit)$	AG maxx > fit	false	-> State: 2.1 <-

5.1.3 *Validarea implementării folosind PHPUnit*

Pentru a valida implementarea descrisă în secțiunile anterioare, propunem testarea codului folosit pentru a realiza această implementare. În acest sens, preluăm două dintre clasele construite pentru implementare, clasa Cromozom și clasa Item, și funcția de generare a populației inițiale. Codul propus este construit în limbajul de programare web PHP, aplicația fiind într-un stadiu avansat de implementare sub forma unei aplicații web.

Rularea testelor a fost făcută folosind utilitarul PHPUnit, pentru care s-au generat o serie de teste specifice limbajului PHP pentru testarea celor două clase și a funcției amintite.

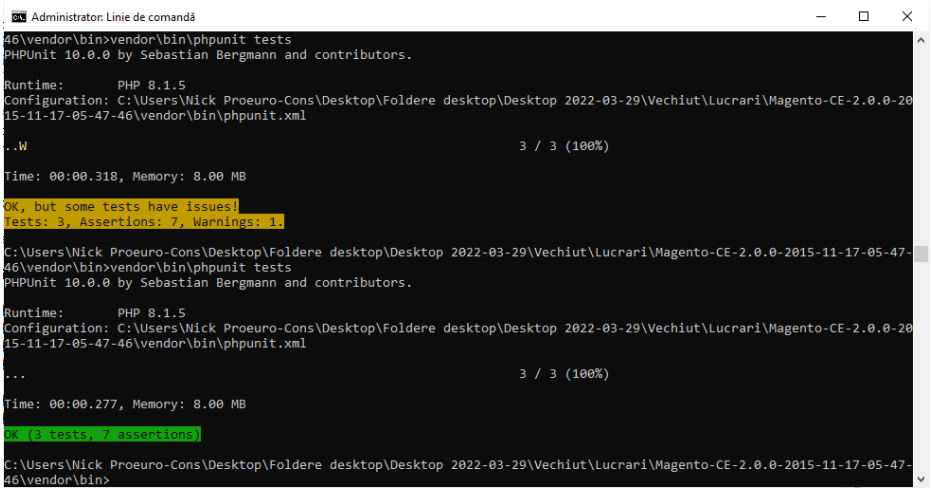


Figura 5.1. Rezultatul rulării testelor în PHPUnit

Conform testelor, am rulat dacă pentru ambele clase constructorul inițializează corect o serie de variabile. Pentru funcția de generare, am determinat dacă rezultatul returnat este unul de tip corect. Astfel, pentru o serie de teste, am determinat dacă implementarea modelelor este fidelă acestora și dacă modul în care se comportă codul folosit în implementare este unul valid, iar parametrii acestuia se inițializează în mod corect.

5.2 ***Folosirea algoritmilor genetici pentru model checking***

5.2.1 *Abordări în literatura de specialitate*

În acest sens, ne dorim să prezentăm în această secțiune o modalitate de verificare a unui model formal prin verificarea unor proprietăți date, care se poate aplica inclusiv pentru modele cu un spațiu al stărilor extins. Mai mult, având în vedere natura verificării modelului, trebuie luată în calcul verificarea proprietăților pentru întregul spațiu de stări, ceea ce conduce la îngreunarea procesului de verificare pentru unele situații. Astfel, pentru o serie de proprietăți date și o modelare a sistemului mai sus amintit sub forma unei mașini cu număr finit de stări,

vom încerca să generăm secvențe de stări folosind un algoritm genetic și vom aplica verificarea proprietăților stabilite pentru secvențele obținute. În acest fel, se pot genera secvențe de stări care să ia în calcul numărul de proprietăți satisfăcute sau nesatisfăcute, algoritmul conducând către o stare de optim, care implică satisfacerea unui număr maxim de condiții. Această abordare nu este una inovativă, existând lucrări care descriu o astfel de metodologie bazată pe rezultate ([101], [102]). Totuși, posibilitatea aplicării ei în cadrul acestui model și extinderea folosirii algoritmilor genetici pentru problema propusă pot fi considerate elemente de noutate raportate la lucrările din literatura de specialitate.

5.2.2 Metodologia folosirii unui algoritm evolutiv pentru model checking

Pentru a determina modalitatea în care vom aplica metoda propusă, stabilim elementele inițiale ale problemei. Pentru un model al unui sistem, dat prin mulțimea stărilor, a variabilelor sistemului și a domeniilor lor de definiție, se dorește obținerea unei mașini cu număr finit de stări care conține stările propuse și care verifică o proprietate sau o mulțime de proprietăți.

Metodologia unui astfel de algoritm cuprinde următorii pași:

S1. Definirea reprezentării genomului: un cromozom poate fi reprezentat ca o mașină de număr finit de stări, o succesiune de stări, acțiuni și tranziții ale acestei mașini cu număr finit de stări;

S2. Inițializarea populației inițiale de cromozomi: aceasta se poate face prin generare aleatorie sau prin orice altă modalitate de generare.

S3. Evaluarea valorilor de fitness pentru populația inițială: se stabilesc valorile funcției de fitness pentru populația de cromozomi obținută.

S4. Aplicarea operației de mutație: se aplică, în acest caz operația de mutație prin selectarea unei gene aleatorii și înlocuirea ei cu o altă genă.

S5. Aplicarea operației de crossover: se aplică, în acest caz operația de crossover prin selectarea unei a doi cromozomi aleatorii și a unui punct de recombinare și combinarea celor doi cromozomi în funcție de punctul de recombinare.

S6. Selecția elitelor: această operație se face prin sortarea cromozomilor în funcție de valoarea caracteristică a funcției fitness și alegerea acelora cu valori maxime.

S7. Repetarea pașilor S3 – S6 de un număr de ori egal cu numărul de generații ale algoritmului.

S8. Selecția cromozomului optim: se alege cromozomul cu valoarea funcției de fitness maximă și se oprește algoritmul.

Ca urmare a rulării unui astfel de cod și după transformarea cromozomului într-o mașină cu număr finit de stări, se va obține, în funcție de performanța algoritmului, o succesiune de stări care îndeplinește numărul maxim de proprietăți propuse spre verificare. Pentru implementarea totală a unui astfel de algoritm în Python, este nevoie de folosirea de librării specifice verificării modelului, în așa fel încât valoarea funcției fitness să fie calculată conform definiției.

Găsirea unui model creat corect pentru o mulțime de stări și tranziții pe baza verificării modelului are o însemnătate deosebită, în special în ceea ce privește modelele complexe, în care un număr mare de stări împiedică scopul realizării și implementării unui model formal.

6 CAPITOLUL VI. CONCLUZII ȘI ACTIVITATE VIITOARE

Rezumat. *Capitolul VI include o serie de considerații rezumative referitoare la teză. Totodată, este inclusă și o scurtă enumerare a limitărilor cercetării realizate până în momentul de față. Componenta finală a capitolului o constituie o serie de propuneri pentru activitatea de cercetare care se dorește a fi întreprinsă în legătură cu dezvoltarea viitoare a sistemului de programe, a modelării, verificării și implementării acestuia.*

Cea mai importantă parte a performanței evaluării unui test educațional este legată de buna proiectare a testului de evaluare. În acest sens, implementarea acestui model oferă un instrument cu adevărat util pentru o bună proiectare a itemilor testului și, în același timp, oferă informații legate de subiecte care pot fi revizuite într-o perioadă de timp în context educațional. În această chestiune, această implementare poate fi extrem de utilă pentru determinarea materiilor care necesită timp suplimentar pentru predare și înțelegere. Modelul se dovedește a fi unul viabil, datorită naturii problemei la care răspunde și metodelor folosite pentru a rezolva această problemă. Dat fiind faptul că evaluarea este în prezent unul dintre subiectele cele mai cercetate în educație, modelul și implementarea lui pot fi considerate ca fiind importante pentru a obține un test bine conceput, după ce implementarea poate fi scalată pentru medii mai generale.

Aplicarea modelelor formale, verificarea modelului și validarea realizate pentru o astfel de aplicație care folosește un algoritm genetic este una extrem de importantă, mai ales din perspectiva naturii complexe a problemei propuse, dar și a necesității obținerii unui rezultat cât mai performant. În acest sens, verificarea proprietăților unor modele formale este un pas determinant în îmbunătățirea acestei performanțe, mai ales în ceea ce privește convergența algoritmului genetic și a unor valori ale funcțiilor de fitness cât mai mari.

Limitările de natură formală includ restrângerea modelării formale la o serie de componente încă nedezvoltate pe deplin. De asemenea, limitări referitoare la modelarea matematică și de implementare includ componente ale algoritmilor euristici, referitoare la operanzii folosiți, la metodologia algoritmului folosit sau la rafinarea sistemului de determinare a funcției de fitness. În cele din urmă, alte limitări pot fi considerate acele procese de modelare aflate în curs de finalizare a unor componente ale sistemului și includerea lor potențială în implementarea practică a sistemului Deși limitările cercetării sunt încă în proces de redefinire, rezultatele obținute până în momentul actual sugerează un potențial extrem de bine definit pentru identificarea de soluții folosind cercetările realizate până în acest moment.

Pe măsură ce sistemele devin complexe, un flux continuu devine imperativ. O metodă formală ajută la reducerea erorilor, a costurilor și la asigurarea faptului că sistemul dezvoltat va îndeplini toate așteptările. Înțelegerea rolului instrumentelor și limbajelor, indiferent de domeniul aplicației, va fi invariabil un avantaj important pentru toți cei implicați. Rolul modelelor de verificare în proiectarea și verificarea sistemelor cu instrumente și limbaje relevante este prezentat în detaliu. Caracterul adecvat al instrumentelor și limbajelor de verificare și validare a modelelor în diferite domenii de aplicații se bazează în principal pe puține caracteristici ale domeniului, inclusiv secvențiale, paralele, temporizate, fără timp etc.

Deși testarea este de departe cea mai importantă tehnică de validare practică pentru sistemele informatice utilizate de industrie, a fost mult timp neglijată ca domeniu de cercetare serioasă. Cu toate acestea, în ultimul deceniu, studiul utilizării modelelor (formale) pentru generarea și executarea sistematică a suitelor de testare sonore, validarea suitelor de testare și interpretarea rezultatelor testelor a devenit un câmp de cercetare stabilit. Acest lucru a condus la dezvoltarea de noi teorii și instrumente pentru a sprijini testarea sistemelor software care au fost aplicate cu succes în practică. În ciuda acestui succes inițial, practicile standard de testare ale industriei în general sunt încă scăzute. Motivele pentru aceasta sunt:

1. Lipsa informației / educației: echipele industriale nu sunt conștiente de natura și potențialul metodelor și instrumentelor mai avansate. Acest lucru trebuie abordat prin campanii de transfer de cunoștințe bine concentrate.
2. Tehnicile actuale au un mare potențial, dar trebuie îmbunătățite și abordează o serie de probleme practic relevante. O preocupare principală este scalabilitatea, de ex. în ceea ce privește numărul de componente și structura sistemului și în legătură cu parametrii sistemului variind pe domenii mari sau infinite. Cu sistemele încorporate, această problemă este agravată de necesitatea de a lua în considerare caracteristicile fizice ale mediului sistemului și integrarea strânsă a electronicii, mecanicii și software-ului de control, necesitând astfel manipularea proprietăților în timp real, stocastic și hibrid în timpul modelării, generării testelor și executare. Acest lucru necesită eforturi considerabile de cercetare pentru a rafina și extinde teoriile și instrumentele existente.

Urmărind principiile de dezvoltare ale elementelor din domeniul temei, putem pune, astfel, bazele cercetării pornind de la elementele particulare ale direcțiilor avute până în acest moment și raportate la temă. Un punct important pentru cercetarea viitoare este extinderea unor componente ale sistemului prin formularea de soluții pentru cazuri extinse ale sistemului (e.g., adăugarea de noi restricții, tipuri noi de itemi, cazuri de conectivitate a itemilor) și prin

automatizarea implementării folosind învățarea automată. În acest fel, direcțiile viitoare de cercetare includ îmbunătățirea modelelor prezentate în această lucrare, dezvoltarea extinsă a implementării sistemului de programe, includerea de noi funcționalități, dezvoltarea și finalizarea implementării aplicației într-un mediu real, precum și determinarea folosirii algoritmilor genetici în procesele de modelare și verificare formală.

REFERINȚE BIBLIOGRAFICE

[1] G. Schneider, „Model Checking: A Complement to Test and Simulation," 2018. [Interactiv]. Available: http://www.cse.chalmers.se/~gersch/model-checking/What-is-Model-Checking.html.

[2] M. Y. Alzahrani, Model Checking Web Applications, School of Mathematical and Computer Sciences, Heriot-Watt University, p. 116, 2015.

[3] M. Minea, „Model Checking. Noțiuni de bază. Verificare formală. Curs 2," 2004.

[4] D. Jackson, Software Abstractions: logic, language, and analysis, MIT press, 2012.

[5] C. Baier și J.-P. Katoen, Principles of model checking, Cambridge, Massachusetts, London: The MIT Press, 2008.

[6] F. Wagner, R. Schmuki, T. Wagner și P. Wolstenholme, Modeling software with finite state machines: a practical approach, CRC Press, 2006.

[7] M. Stănescu, S. Caramihai, A. Curaj, O. Popescu, P. D.C. și D. Catana, Sisteme dinamice cu evenimente discrete, București: Universitatea "Politehnica", 1996.

[8] C. Edmund, O. Grumberg și K. Hamaguchi, Another look at LTL model checking, Carnegie-Mellon University. Department of Computer Science, 1994.

[9] H. Kugler, D. Harel, A. Pnueli, Y. Lu și B. Yves, „Temporal logic for scenario-based specifications," Tools and Algorithms for the Construction and Analysis of Systems: 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, vol. 11, pp. 445-460, 2005.

[10] J. R. Abrial, Modeling in Event-B: system and software engineering, Cambridge University Press, 2010.

[11] D. Jackson, Software Abstractions: logic, language, and analysis, MIT press, 2012.

[12] K. Jayaraman, N. Bjørner, G. Outhred și C. Kaufman, „Automated analysis and debugging of network connectivity policies," Microsoft Research, pp. 1-11, 2014.

[13] J. E. Hopcroft, R. Motwani și J. D. Ullman, Introduction to Automata Theory Languages and Computation, 3rd Edition Cornell University, California: Stanford University, 2001.

[14] D. Lee și M. Yannakakis, „Principles and methods of testing finite state machines-a survey," Proceedings of the IEEE, vol. 84, nr. 8, pp. 1090-1123, 1996.

[15] A. Gill, Introduction to the theory of finite-state machines, 1962.

[16] F. Wagner, R. Schmuki, T. Wagner și P. Wolstenholme, Modeling software with finite state machines: a practical approach, CRC Press, 2006.

[17] G. Schneider, „Model Checking: A Complement to Test and Simulation," [Interactiv]. Available: http://www.cse.chalmers.se/~gersch/model-checking/What-is-Model-Checking.html. [Accesat mai 2018].

[18] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson și W. Yi, „UPPAAL—a tool suite for automatic verification of real-time systems," Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control: verification and control, p. 232–243, 1996.

[19] O. Păstrăvanu, M. Matcovschi și C. Mahulea, APLICAȚII ALE REȚELELOR PETRI ÎN STUDIEREA SISTEMELOR CU EVENIMENTE DISCRETE, 2002.

[20] P. Antsaklis și J. Moody, Supervisory Control of Discrete Event Systems Using Petri Nets,, Kluwer Academic Publishers, 1998.

[21] B. K. Aichernig și C. A. Furia, „Formal Methods: State of the Art and Future Directions," ACM Computing Surveys, vol. 51, nr. 3, pp. 1-36, 2018.

[22] W. Ahrendt și B. Beckert, „A Survey of Formal Methods in Software Development," ACM Computing Surveys, vol. 47, nr. 2, pp. 1-56, 2015.

[23] M. Butler și J. Colley, „Formal Methods in Practice: Domain Modeling for Distributed Real-Time Systems," ACM Transactions on Software Engineering and Methodology, vol. 27, nr. 2, pp. 1-45, 2018.

[24] M. Huhn și R. Grosu, „Formal Methods in Safety-Critical Railway Systems," ACM Computing Surveys, vol. 44, nr. 1, pp. 1-47, 2012.

[25] J. Hatcliff, D. Kroening și L. V. Hong, „Formal Methods for Cyber-Physical Systems," ACM Transactions on Embedded Computing Systems, vol. 11, nr. 3, pp. 1-28, 2012.

[26] S. Bensalem, M. Bozga și J. Combaz, „Modeling and Verification Techniques for Cyber-Physical Systems: A Survey," Proceedings of the IEEE, vol. 100, nr. 1, pp. 12-28, 2011.

[27] T. Balanescu, R. Nicolescu și H. Wu, „Asynchronous P systems," International Journal of Natural Computing Research (IJNCR), vol. 2, nr. 2, pp. 1-18, 2011.

[28] A. Cimatti și C. Piazza, „Formal Methods for the Design of Real-Time and Hybrid Systems," Annual Review of Control, Robotics, and Autonomous Systems, vol. 1, nr. 1, pp. 171-193, 2018.

[29] H. Garavel, M. ter Beek și J. van de Pol, „The 2020 Expert Survey on Formal Methods," în FMICS 2020 - 25th International Conference on Formal Methods for Industrial Critical Systems, Vienna, Austria, 2020.

[30] A. E. K. Sobel și M. R. Clarkson, „Formal methods application: An empirical tale of software development," IEEE transactions on software engineering, vol. 28, nr. 3, pp. 308-320, 2002.

[31] E. M. Clarke, O. Grumberg și D. Peled, „Model checking," MIT Press, 1999.

[32] J. P. Queille și J. Sifakis, „A temporal logic to deal with fairness in transition systems," în 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 1982.

[33] J. Hajek, „Automatically verified data transfer protocols," în 4th International Conference on Computer Communication (ICCC), IEEE Computer Society Press, 1978.

[34] C. H. West, „An automated technique for communications protocol validation," IEEE Transactions on Communications, vol. 26, nr. 8, p. 1271–1275, 1978.

[35] K. R. Apt și D. Kozen, „Limits for the automatic verification of finite-state concurrent systems," Information Processing Letters, vol. 22, nr. 6, p. 307–309, 1986.

[36] G. J. Holzmann, Design and Validation of Computer Protocols, Prentice-Hall, 1990.

[37] K. L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.

[38] R. Kurshan, Computer-aided Verification of Coordinating Processes: The Automata-Theoretic Approach, Princeton University Press, 1999.

[39] A. P. Mathur, Foundations of Software Testing, Addison-Wesley Professional, 2008.

[40] N. A. Barricelli, Numerical testing of evolution theories, Acta Biotheoretica, 1962.

[41] A. S. Fraser, „Simulation of genetic systems by automatic digital c omputers. I. Introduction," Aust. J. Biol. Sc i, vol. 10, p. 484, 1957.

[42] J. H. Holland, „Genetic Algorithms," Scientific American, vol. 267, nr. 1, p. 66–73, 1992.

[43] J. Carr, „An Introduction to Genetic Algorithms," 2014.

[44] J. R. Koza, „Advances in Genetic Programming," în Introduction to Genetic Programming, Cambridge: MIT Press, 1994, pp. 21-41.

[45] M. Mitchell, An Introduction to Genetic Algorithms, Cambridge: MIT Press, 1996.

[46] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine, Reading: Addison-Wesley, 1989.

[47] X.-S. Y. I. F. J. B. D. F. Iztok Fister Jr., „A Brief Review of Nature-Inspired Algorithms for Optimization," Elektrotehni\v{s}ki vestnik, vol. 80, nr. 3, 2013.

[48] D. R. B. ,. R. R. M. David Beasley, „An Overview of Genetic Algorithms: Part 1, Fundamentals," University Computing, pp. 55-69, 1993.

[49] H. Muhlenbein și T. Mahnig, „Mathematical analysis of evolutionary algorithms for optimization," Essays and surveys in Metaheuristics, pp. 525-556, 2002.

[50] S. L. Craig, S. J. Smith și B. B. Frey, „Professional development with universal design for learning: supporting teachers as learners to increase the implementation of UDL," Professional Development in Education, vol. 48, nr. 1, pp. 22-37, 2019.

[51] L. R. Ketterlin-Geller, „Knowing what all students know: Procedures for developing universal design for assessment," The Journal of Technology, Learning and Assessment, vol. 4, nr. 2, 2005.

[52] D. Clow, „An overview of learning analytics," Teaching in Higher Education, vol. 18, nr. 6, pp. 683-695, 2013.

[53] L. Bokander, „Psychometric Assessments," The Routledge Handbook of Second Language Acquisition and Individual Differences, pp. 454-465, 2022.

[54] T. Moses, „A Review of Developments and Applications in Item Analysis," Methodology of Educational Measurement and Assessment, pp. 19-46, 2017.

[55] M. Webb, D. Gibson și A. Forkosh-Baruch, „Challenges for information technology supporting educational assessment," Journal of Computer Assisted Learning, vol. 29, nr. 5, pp. 451-462, 2013.

[56] E. Popescu, „Adaptation Provisioning with respect to Learning Styles in a Web-Based Educational System: An Experimental Study," Journal of Computer Assisted Learning, vol. 26, nr. 4, pp. 243-257, 2010.

[57] C. Holotescu, „A conceptual model for Open Learning Environments," în International Conference on Virtual Learning – ICVL, Bucharest, 2015.

[58] C. Denham și A. Lieberman, Time to Learn (Eds.), Washington: National Institute of Education, 1980.

[59] R. Dolan și T. Hall, „Universal design for learning: Implications for large scale assessment," IDA Perspectives, vol. 27, nr. 4, p. 22–25, 2001.

[60] J. M. Chappuis, „Thoughtful Assessment with the Learner in Mind," Educational Leadership, vol. 71, pp. 20-26, 2014.

[61] Z. S. Siddiqui, „Framework for an effective assessment: From rocky roads to silk route," Pak J Med Sci., vol. 33, nr. 2, pp. 505-509, 2017.

[62] C. I. Popîrlan, T. I. V., C. C. Dinu, G. Stoian, C. Popîrlan și D. Dănciulescu, „Influence of Certain Social Aspects on Most Likely Internet Usage Scenarios," Sustainability, vol. 14, nr. 12, p. 7129, 2022.

[63] C. I. Popîrlan, I. V. Tudor, C. C. Dinu, G. Stoian, C. Popîrlan și D. Dănciulescu, „Hybrid Model for Unemployment Impact on Social Life," Mathematics, vol. 9, nr. 18, p. 2278, 2021.

[64] N. S. Ofiesh, C. M. Rojas și R. A. Ward, „Universal Design and the Assessment of Student Learning in Higher Education: Promoting Thoughtful Assessment," Journal of Post-secondary Education and Disability, vol. 19, nr. 2, pp. 173-181, 2006.

[65] A. Ben-Simon și R. E. Bennett, „Towards more substantively meaningful automated essay scoring," Journal of Teaching, Learning and Assessment, vol. 6, nr. 1, pp. 4-44, 2007.

[66] P. Deane, „On the relation between automated essay scoring and modern views of the writing construct," Assessing Writing, vol. 18, nr. 1, pp. 7-24, 2013.

[67] J. Gardner, M. O'Leary și L. Yuan, „Artificial intelligence in educational assessment: 'Breakthrough? Or buncombe and ballyhoo?'," Journal of Computer Assisted Learning, vol. 37, nr. 5, pp. 1207-1216, 2021.

[68] D. Bidyut, M. Mukta, P. Santanu și A. S. Arif, „Multiple-choice question generation with auto-generated distractors for computer-assisted educational assessment," Multimedia Tools and Applications, vol. 80, nr. 21, pp. 31907-31925, 2021.

[69] R. C. Dhawaleswar și K. S. Sujan, „Automatic Multiple Choice Question Generation From Text: A Survey," IEEE Transactions on Learning Technologies, vol. 13, nr. 1, pp. 14-25, 2020.

[70] Z. Bowei, L. Pengfei, P. Liangming și T. A. Ai, „Automatic True/False Question Generation for Educational Purpose," în Proceedings of the 17th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2022), 2022.

[71] D. Bidyut și M. Mukta, „Factual open cloze question generation for assessment of learner's knowledge," International Journal of Educational Technology in Higher Education, vol. 14, nr. 1, 2017.

[72] A. Malafeev, „Automatic Generation of Text-Based Open Cloze Exercises," Communications in Computer and Information Science, pp. 140-151, 2014.

[73] A. Husam, C. Yllias și A. Sadid, „Automatic Question Generation from Sentences," în Actes de la 17e conférence sur le Traitement Automatique des Langues Naturelles, 2010.

[74] X. Zheng, Automatic question generation from freeform text, Nanyang Technological University, Singapore: Master's thesis, 2022.

[75] S. Burrows, I. Gurevych și B. Stein, „The eras and trends of automatic short answer grading," International Journal of Artificial Intelligence in Education, vol. 25, nr. 1, p. 60–117, 2014.

[76] J. A. A. Mohd, D. A. Fatimah, A. A. G. Abdul și M. Ramlan, „Automated marking system for Short Answer Examination (AMS-SAE)," în 2009 IEEE Symposium on Industrial Electronics &; Applications, 2009.

[77] V. Zhong, W. Shi, W.-T. Yih și L. Zettlemoyer, „RoMQA: A Benchmark for Robust, Multi-evidence Multi-answer Question Answering," în arXiv, 2022.

[78] K. S. Sujan și R. C. Dhawaleswar, „Development of a practical system for computerized evaluation of descriptive answers of Middle School Level Students," Interactive Learning Environments, vol. 30, nr. 2, p. 215–228, 2019.

[79] R. Ganz, „An individualistic approach to item analysis," Readings in mathematical social science, pp. 89-108, 1966.

[80] K. H. Aqeel și M. A.-H. Aqeel, „Testing & the impact of item analysis in improving students' performance in end-of-year final exams," English Linguistics Research, vol. 11, nr. 2, p. 30, 2022.

[81] M. R. Novick, „The axioms and principal results of classical test theory," Journal of Mathematical Psychology, vol. 3, nr. 1, p. 1–18, 1966.

[82] D. J. Weiss și M. E. Yoes, „Item response theory," Advances in Educational and Psychological Testing: Theory and Applications, p. 69–95, 1991.

[83] R. K. Hambleton și R. W. Jones, „An NCME instructional module on Comparison of classical test theory and item response theory and their applications to test development," Educational Measurement: Issues and Practice, vol. 12, nr. 3, p. 38–47, 1993.

[84] G. Abdelrahman, Q. Wang și B. Nunes, „Knowledge Tracing: A Survey," ACM Comput. Surv., vol. 55, nr. 11, 2023.

[85] S. V. I. Alharbi, „A Genetic Algorithm Based Approach for Solving the Minimum Dominating Set of Queens Problem," Journal of Optimization, 2017.

[86] A. D. M. M. V. Colorni, „A Genetic Algorithm To Solve The Timetable Problem".

[87] D. Constantin, „A new model for estimating the risk of bankruptcy of the insurance companies based on the artificial neural networks," în Proceedings of the 16th edition of the SGEM International GeoConferences (International Multidisciplinary Scientific GeoConferences), 2016.

[88] B. Moon, „The Subset Sum Problem: Reducing Time Complexity of NP-Completeness with Quantum Search," Undergraduate Journal of Mathematical Modeling: One + Two, vol. 4, nr. 2, 2012.

[89] M. Yildirim, „A genetic algorithm for generating test from a question bank," Comput. Appl. Eng. Educ., vol. 18, pp. 298-305, 2010.

[90] M. Aktaş, Z. Yetgin, F. Kılıç și Ö. Sünbül, „Automated test design using swarm and evolutionary intelligence algorithms," Expert Systems, vol. 39, nr. 4, 2021.

[91] D. Nijloveanu, N. Bold și A.-C. Bold, „A Hierarchical Model of Test Generation within a Battery of Tests," în International Conference on Virtual Learning, Timișoara, 2015.

[92] D. A. Popescu, N. Bold și D. Nijloveanu, „A method based on genetic algorithms for generating assessment tests used for learning," în 17th International Conference on Intelligent Text Processing and Computational Linguistics, 2016.

[93] D. A. Popescu, N. Bold și O. Domșa, „Generating assessment tests with restrictions using genetic algorithms," în 12th IEEE International Conference on Control & Automation, Kathmandu, Nepal, 2016.

[94] O. Domșa și N. Bold, „Generator of variants of tests using the same questions," în The 12th International Scientific Conference eLearning and Software for Education, Bucharest, 2016.

[95] B. Flanagan, C. Yin, S. Hirokawa, K. Hashimoto și Y. Tabata, „An automated method to generate e-learning quizzes from online language learner writing," International Journal of Distance Education Technologies (IJDET), vol. 11, nr. 4, p. 63–80, 2013.

[96] R. Mitkov, H. L. E. An și N. Karamanis, „A computer-aided environment for generating multiple-choice test items," Natural Language Engineering, vol. 12, nr. 2, p. 177–194, 2006.

[97] G. Deena și K. Raja, „Objective Type Question Generation using Natural Language Processing," International Journal of Advanced Computer Science and Applications, vol. 13, nr. 2, 2022.

[98] C. T.H., L. C.E., R. R.L. și S. C., Introduction to algorithms, third edition, MIT Press, 2009.

[99] D. A. Popescu, N. Bold și O. Domșa, „A generator of sequences of hierarchical tests which contain specified keywords," în International Symposium on Applied Computational Intelligence and Informatics, IEEE Conference Proceedings, Timișoara, 2016.

[100] M. Harman și B. F. Jones, „Search-based software engineering," Information & Software Technology, vol. 43, nr. 14, p. 833–839, 2001.

[101] R. Lefticaru, F. Ipate și C. Tudose, „Automated Model Design using Genetic Algorithms and Model Checking".

[102] G. Katz și D. Peled, „Model checking-based genetic programming with an application to mutual exclusion," Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08). Springer, p. 141–156, 2008.