

---

## Cuprins

1. Introducere .....	1
1.1. Contextul și motivația .....	1
1.2. Structura lucrării .....	2
2. Obiective și contribuție .....	3
3. Studiu bibliografic .....	5
4. Analiză și fundamentare teoretică .....	10
4.1. Optimizare.....	10
4.2. Metaeuristici .....	11
4.3. Ontologie.....	13
4.4. Informații medicale cu privire la necesarul energetic și nutrițional al organismului .....	14
5. Tehnică hibridă inspirată din comportamentul cucilor pentru generarea de meniuri alimentare personalizate.....	17
5.1. Modelul hibrid inspirat din comportamentul cucilor .....	17
5.1.1. Componenta inspirată din comportamentul cucilor .....	17
5.1.2. Componenta hibridă bazată pe principiile geneticii și ale selecției naturale .....	19
5.1.3. Componenta bazată pe Tabu Search .....	20
5.2. Evoluția algoritmului hibrid inspirat din comportamentul cucilor .....	21
5.2.1. Algoritmul clasic inspirat din comportamentul cucilor .....	22
5.2.2. Algoritmul clasic inspirat din comportamentul cucilor îmbunătățit cu o nouă procedură pentru actualizarea soluțiilor cucilor .....	23
5.2.3. Algoritmul hibrid inspirat din comportamentul cucilor și operatorul genetic.....	24
5.2.4. Algoritmul hibrid inspirat din comportamentul cucilor, operatorul genetic și structuri de memorie .....	25
6. Proiectare și implementare .....	28
6.1. Proiectarea prototipului experimental .....	28
6.1.1. Arhitectura conceptuală .....	28
6.1.2. Use Case Model.....	33
6.1.3. Diagrama de activitate .....	33
6.1.4. Diagrama de deployment.....	33
6.1.5. Diagrama de pachete .....	35

---

6.1.6.	Diagrame de clasă .....	37
6.2.	Detalii de implementare .....	40
6.2.1.	Tehnologii și instrumente utilizate .....	40
6.2.2.	Secvențe de cod .....	41
7.	Evaluarea tehnicilor propuse .....	46
7.1.	Scenarii de test.....	46
7.2.	Identificarea parametrilor ajustabili .....	47
7.3.	Analiza rezultatelor experimentale obținute .....	48
7.4.	Metrici de evaluare a performanței algoritmului .....	59
7.5.	Analiză comparativă.....	61
7.5.1.	Analiză comparativă a versiunilor tehnica inspirate din comportamentul cucilor .....	61
7.5.2.	Analiză comparativă între tehnica hibridă inspirată din comportamentul cucilor și tehnica hibridă inspirată din comportamentul cucilor .....	64
8.	Manualul de instalare și utilizare .....	65
8.1.	Manual de instalare .....	65
8.2.	Manual de utilizare .....	65
9.	Concluzii și dezvoltări ulterioare .....	68
	Bibliografie .....	69
	Acronime .....	71
	Anexe .....	72
	Anexa A .....	72
	Anexa B .....	74

## 1. Introducere

Încă din cele mai vechi timpuri, oamenii au fost preocupați de procurarea și prepararea hranei. În urmă cu zeci de mii de ani, sursele principale de hrană erau reprezentate de fructele culese din copaci și de carnea de vânat. În prezent se folosesc din ce în ce mai multe produse rafinate și concentrate, acestea având consecințe grave asupra sănătății. Deoarece s-a constatat că alimentația nesănătoasă reprezintă una din cauzele apariției a numeroase afecțiuni medicale, a fost necesară studierea necesarului energetic și nutrițional al organismului.

Pentru o bună funcționare a organismului uman este necesar ca acesta să se afle într-un echilibru energetic, adică să existe o stabilitate între aportul de alimente și consumul de energie. De asemenea, o alimentație sănătoasă presupune ca alimentele consumate să ofere toate substanțele nutritive esențiale în cantități optime. Substanțele nutritive se clasifică în:

- macronutrienți – aceste substanțe oferă organismului uman necesarul de energie, din aceasta categorie făcând parte glucidele, lipidele și proteinele;
- micronutrienți – aceste substanțe nu au valoare calorică și sunt necesare în cantități mici pentru a îndeplini funcții fundamentale în cadrul proceselor vitale.

Necesarul zilnic de substanțe nutritive diferă în funcție de parametrii ca: vârsta, sexul, greutate corporală, înălțimea și nivelul de activitate depusă. De exemplu, la persoanele de vârstă a treia activitatea fizică și metabolismul bazal scad și ca urmare necesarul energetic se reduce. De asemenea, la această categorie de persoane apare un deficit de vitamina D, de aceea se recomandă creșterea consumului de alimente bogate în vitamina D. Pentru a preveni malnutriția la persoanele de vârstă a treia se recomandă consumul zilnic a următoarelor alimente: un pahar de suc, cereale integrale cu lapte la micul dejun, pește sau carne, un pahar de lapte seara și cel puțin o porție de legume.

### 1.1. Contextul și motivația

Ideea acestui proiect a apărut în contextul în care studiile au arătat că un procent îngrijorător din populația în vârstă din Europa suferă de nutriție proastă sau malnutriție. Conform statisticilor, peste 15% din persoanele de vârstă a treia care locuiesc la domiciliu și peste 60% din vârstnicii care trăiesc în centrele de îngrijire suferă de malnutriție. În cazul acestui grup de persoane, nutriția proastă și malnutriția e cauzată de problemele specifice înaintării în vârstă, cum ar fi: scăderea activității fizice, scăderea metabolismului bazal, pierderea capacității de memorare, scăderea sensibilității, efectele secundare ale medicamentelor sau starea precară a sănătății dentare. Prin utilizarea acestui sistem de către persoanele în vârstă malnutriția va putea fi prevenită sau tratată.

Generarea de recomandări alimentare personalizate constă în căutarea combinației optime de oferte alimentare care să respecte anumite criterii. Această problemă de căutare este de tipul NP dificilă, domeniul de căutare crescând exponențial cu numărul de oferte alimentare. De aceea, timpul necesar căutării cu ajutorul metodelor exhaustive este foarte mare. Acest dezavantaj este eliminat prin utilizarea algoritmilor inspirați din natură, care nu parcurg toate soluțiile posibile. Acești algoritmi nu au ca scop găsirea soluției optime, ci găsire unei soluții optime locale. Din testele efectuate, s-a constatat că optimul local găsit de algoritmi inspirați din natură se apropie foarte mult de soluția optimă. Testele efectuate în acest sens se bazează pe valoarea funcției de fitness.

Până în prezent, strategiile inspirate din natură au fost aplicate cu succes în domenii ca: securitate, inginerie medicală sau rețele de calculatoare. Sursele de inspirație ale acestor algoritmi se bazează pe comportamente și fenomene variate din natură, începând cu interacțiunea dintre microorganisme și continuând cu formarea râurilor, colonizarea buruienilor sau comportamentul furnicilor în căutarea hranei. Aceste surse de inspirație, pot fi clasificate, la nivelul cel mai înalt, în: biologice, chimice sau fizice. Majoritatea algoritmilor fac parte din categoria celor inspirați din biologie.

Există numeroși algoritmi inspirați din natură, care pornesc de la soluții generate în mod aleator, asupra cărora se execută diverse operații până când se ajunge la o soluție optimă. Operațiile care se execută pentru găsirea soluției optime se deosebesc de la un algoritm la altul. Rezultatele obținute diferă în funcție de tipul algoritmului și de contextul problemei. De aceea, nu se poate spune că un algoritm este mai bun decât celelalte, dar putem afirma că pentru un anumit tip de problemă se obțin rezultate mai bune prin utilizarea unui algoritm specific. Totuși, s-a constatat că algoritmi care au capacitatea de a efectua atât o căutare globală, cât și o căutare locală intensivă, sunt mai eficienți. În general proprietățile de exploatare globală a spațiului și de cele de explorare locală a acestuia nu pot fi furnizate de un singur algoritm. Pentru a putea utiliza ambele proprietăți s-a analizat posibilitatea de a combina avantajele mai multor algoritmi, prin utilizarea tehnicii de hibridizare. Algoritmii astfel obținuți au o eficiență mai bună decât algoritmi clasici, precum și o flexibilitate ridicată.

### **1.2. Structura lucrării**

Această lucrare este structurată în nouă capitole care prezintă obiectivele propuse și modul în care acestea au fost atinse. Capitolul 2 prezintă obiectivele și subobiectivele acestui proiect și contribuțiile care au fost aduse acestui proiect. De asemenea, în acest capitol se prezintă și efectele pe care dorim ca proiectul să le aibă asupra calității vieții persoanelor în vârstă. În capitolul 3 se prezintă o documentare bibliografică, cu scopul de a fixa referențialul în care se situează această temă. Capitolul 4 conține o prezentare sumară a următoarelor concepte: metaeuristici, optimizare și ontologii. Acest capitol conține și informații legate de necesarul energetic și nutrițional al organismului la persoanele în vârstă. Capitolul 5 descrie tehnica inspirată din comportamentul cucilor, precum și evoluția pe care o are algoritmului clasic prin adăugarea de componente hibride. În capitolul 6 se prezintă detaliile de proiectare și implementare ale prototipului experimental. Acest capitol conține și prezentarea tehnologiilor utilizate. Capitolul 7 prezintă metodele prin care tehnicile propuse în capitolul 5 sunt evaluate pe diverse scenarii de test. De asemenea, se prezintă și o analiză comparativă între versiunile dezvoltate. Capitolul 8 conține detaliile necesare pentru a putea utiliza și instala produsul care a fost dezvoltat. Ultimul capitol conține un rezumat al contribuțiilor aduse acestui proiect, o analiză sumară a rezultatelor obținute, precum și posibilele dezvoltări și îmbunătățiri ulterioare ale produsului.

## 2. Obiective și contribuție

Principalul obiectiv al acestui proiect este de a dezvolta o tehnică inspirată din biologie pentru generarea de recomandări alimentare personalizate pentru persoanele în vârstă. În vederea generării unei combinații optime de oferte alimentare, se vor lua în considerare următoarele criterii:

- Categoria de alimente pe care utilizatorul dorește să o consume;
- Categoria pe alimente pe care utilizatorul dorește să nu o consume;
- Afecțiunile medicale ale utilizatorului;
- Nivelul zilnic de activitate al utilizatorului.
- Dieta recomandată de un cadru specializat (nutriționist sau medic), în funcție de vârstă, sexul și nivelul zilnic de activitate al utilizatorului.

Dieta recomandată de cadrul specializat va conține valorile zilnice de macronutrienți, micronutrienți și calorii pe care utilizatorul trebuie să le consume. Pentru realizarea acestui proiect se vor lua în considerare următorii nutrienți: proteine, glucide, lipide, fier, sodium, vitamina A, vitamina B1, vitamina C, respectiv kaloriile.

Deoarece există un număr mare de oferte alimentare, serviciul de generare de recomandări personalizate va fi modelat ca o problemă de optimizare combinatorică, care are ca scop găsirea combinației optime de oferte alimentare astfel încât criteriile luate în considerare să fie îndeplinite. Prin alegerea acestui tip de modelare se dorește ca timpul necesar căutării combinației alimentare optime să fie semnificativ mai redus decât în cazul unei căutări exhaustive.

Pentru a putea atinge obiectivul principal, este necesară realizarea următoarelor subobiective:

- Analiza problemei de generare de recomandări alimentare personalizate, prin accentuarea necesității dezvoltării de algoritmi inspirați din biologie.
- Studiul bibliografic al metaeuristicilor existente care pot fi aplicate în contextul acestei probleme.
- Dezvoltarea unei ontologii ce aparține domeniului medical și alimentar.
- Dezvoltarea unei baze de date aparținând domeniului alimentar.
- Generarea de oferte alimentare și generarea de profile medicale.
- Dezvoltarea unui tehnici hibride care integrează un algoritm inspirat din comportamentul de reproducere al cucilor. Algoritmul integrat va fi analizat, vor fi observate avantajele și dezavantajele și se vor înlocui componentele slabe cu componente din alte metaeuristici existente.
- Analiza și dezvoltarea unui prototip experimental care să conțină tehnicile ce au fost dezvoltate și care să ofere posibilitatea de extindere și utilizare pentru alți algoritmi inspirați din natură.
- Evaluarea algoritmului clasic și a versiunilor hibride și analizarea rezultatelor obținute în urma ajustării parametrilor configurabili.
- Stabilirea componentelor care au nevoie de noi hibridizări.

Prin realizarea acestui proiect se dorește îmbunătățirea calității vieții persoanelor în vârstă prin reducerea efectelor negative ale malnutriției, cum ar fi: agravarea bolilor cronice și acute, accelerarea procesului de dezvoltare a bolilor degenerative sau întârzierea recuperării din boala. Toate acestea implică atât o scădere semnificativă a costurilor pentru gestionarea efectelor

malnutriției la persoanele vârstnice, cât și o creștere a numărului de persoane în vârstă capabile de a auto-gestiona comportamentul lor alimentar și dieta.

Contribuțiile aduse acestui proiect sunt: dezvoltarea unui algoritm hibrid inspirat din comportamentul cucilor și a unei aplicații Web care permite testarea acestui algoritm. Aplicația va permite ajustarea parametrilor configurabili, introducerea preferințelor utilizatorului și vizualizarea rezultatelor sub forma unui grafic.

În cadrul activității de cercetare aferente lucrării de diplomă am participat împreună cu colectivul Laboratorului de Cercetare pentru Sisteme Distribuite la elaborarea articolului „Hybrid Invasive Weed Optimization Method for Generating Healthy Meals”.

### 3. Studiu bibliografic

Până în prezent au fost dezvoltate mai multe sisteme care și-au îndreptat atenția către o nutriție personalizată, unele dintre ele fiind concepute special pentru bătrâni. Deși majoritatea acestor sisteme pot fi utilizate de către toate categoriile de persoane, indiferent de vârstă, problemele specifice vârstnicilor în ceea ce privește nutriția și alimentația nu sunt luate în considerare.

În lucrarea [1] sunt prezentate și analizate problemele care ar trebui luate în considerare la proiectarea unui sistem de recomandări alimentare personalizate. Aceste probleme au fost împărțite în două categorii: probleme referitoare la utilizator, respectiv probleme referitoare la algoritmul utilizat de sistem.

Prima categorie de provocări prezentată în [1] e reprezentată de informația despre utilizator pe care sistemul o deține. În general, pentru a putea oferi recomandări alimentare, sistemul trebuie să cunoască necesarul nutrițional de care are nevoie un anumit utilizator, informații legate de ultimele alimente consumate de utilizator, precum și notele pe care utilizatorii le-au acordat anumitor mâncăruri. Este cunoscut faptul că utilizatorii au tendința să nu își amintească tot ce au consumat sau să subestimeze cantitatea pe care au consumat-o, de aceea este relizarea unui istoric alimentar corect este dificilă. O problemă la fel de dificilă o constituie determinarea unei modalități prin care utilizatorul să își dorească să acorde note unui număr cât mai mare de mâncăruri.

A doua categorie de provocări prezentată în [1] e reprezentată de datele de care are nevoie algoritmul pentru a putea oferi recomandări personalizate. Aceste date se referă la informațiile legate de utilizator, baza de date care să conțină rețete și valorile nutriționale ale acestora, precum și un set de constrângeri. Deoarece informațiile despre utilizator pot lipsi uneori sau pot fi inexacte sau incorecte, sistemul ar trebui să fie capabil să corecteze datele greșite sau să completeze datele incomplete. Pentru a putea oferi recomandări variate, care să țină cont de preferințele alimentare ale utilizatorului este nevoie de utilizarea unui număr mare de rețete. Acest lucru va avea efecte negative asupra performanței sistemului. Pentru ca sistemul să fie mai atractiv ar trebui să îi permită utilizatorului să aleagă meniul pentru mai multe zile sau pentru mai multe persoane.

În [2] este prezentat un sistem de recomandare de rețete personalizate, care este destinat persoanelor care doresc să își schimbe stilul alimentar, dar nu știu cum să aplice regulile de nutriție sănătoasă în alegerea meniurilor zilnice. Acest sistem calculează câte un scor pentru fiecare rețetă, în funcție de valoarea nutrițională a respectivei rețete. Pentru ca utilizatorul să înțeleagă mai ușor dacă o rețetă este sănătoasă sau nu, fiecare rețetă are asociată o culoare: roșu pentru rețetele nesănătoase și verde pentru cele sănătoase. Sistemul ia în considerare alimentele care îi plac utilizatorului și cele care nu îi plac, cu scopul de a oferi sugestii personalizate, având grijă să mențină un aport echilibrat al valorilor nutritive. De asemenea, sistemul poate oferi sugestii alimentare pentru perioade variate de timp. În această lucrare se precizează faptul că pentru implementarea acestui sistem s-a utilizat un serviciu web inteligent care înglobează mai mulți algoritmi specializați pe interpretarea și clasificarea rețetelor alimentare. Produsul a fost evaluat de 40 de persoane de gen feminin, pe parcursul a două săptămâni. În urma acestei evaluări s-a constatat că participanți au gătit mâncare mai sănătoasă pe parcursul celor două săptămâni, în comparație cu mâncarea gătită înainte de începerea studiului.

Sistemul prezentat în [3] utilizează conceptul de clustering alimentar, pentru a genera meniuri alimentare personalizate. Clustering-ul alimentar se referă la modalitate prin care sunt

determinate asemănările nutriționale ale diferitelor alimente. Alimentele care se aseamănă din punct de vedere nutrițional sunt alocate în același cluster. Astfel, pentru a diversifica meniul alimentele pot fi înlocuite cu alimente care aparțin aceluiași cluster. Acest sistem poate oferi sugestii pentru toate cele trei mese principale ale zilei (mic dejun, prânz, cină) sau doar pentru o parte din ele. Utilizatorul va putea oferi informații referitoare la alimentele preferate, numărul de meniuri pe care dorește să le primească, sex, greutate, înălțime, vârstă și nivelul de activitate. Pe baza acestor informații, sistemul va genera recomandări alimentare, care să respecte atât preferințele utilizatorului, cât și necesarul zilnic de calorii, proteine, carbohidrați și lipide. Necesarul zilnic energetic se calculează cu ajutorul ecuației Harris-Benedict. Sistemul a fost evaluat de 8 persoane, pentru fiecare generându-se cinci meniuri. În tabelul 3.1 sunt prezentate procentele în care media valorilor nutriționale și energetice conținute în cele cinci meniuri respectă necesarul zilnic de calorii și macronutrienți. Se precizează faptul că pentru utilizatorul cu numărul 2 nu a putut fi generat un meniu care să respecte necesarul zilnic energetic și nutrițional, deoarece acesta este atlet și trebuie să consume produse cu conținut caloric ridicat. Persoanele care au participat la evaluare s-au declarat mulțumite de program și de interfața grafică a acestuia. Dezavantajele acestui sistem sunt date de faptul că acesta nu ia în considerare posibilele afecțiuni medicale ale utilizatorilor și necesarul zilnic recomandat de micronutrienți. Sistemul nu ține cont nici de categoriile speciale de persoane, cum ar fi: sportivi, copii sau persoanele în vârstă.

Nutrition parameters	Energy (Kcal)	Protein (grams)	Fat (grams)	Carbohydrates (grams)
User 1	98.08 %	82.53 %	92.93 %	94.77 %
User 2	91.21 %	74.31 %	94.69 %	87.28 %
User 3	94.73 %	83.20 %	93.20 %	89.78 %
User 4	98.94 %	90.78 %	92.62 %	97.33 %
User 5	98.56 %	94.34 %	94.73 %	94.01 %
User 6	96.25 %	82.75 %	94.26 %	95.48 %
User 7	96.75 %	86.51 %	95.11 %	95.34 %
User 8	97.01 %	83.12 %	92.69 %	91.33 %

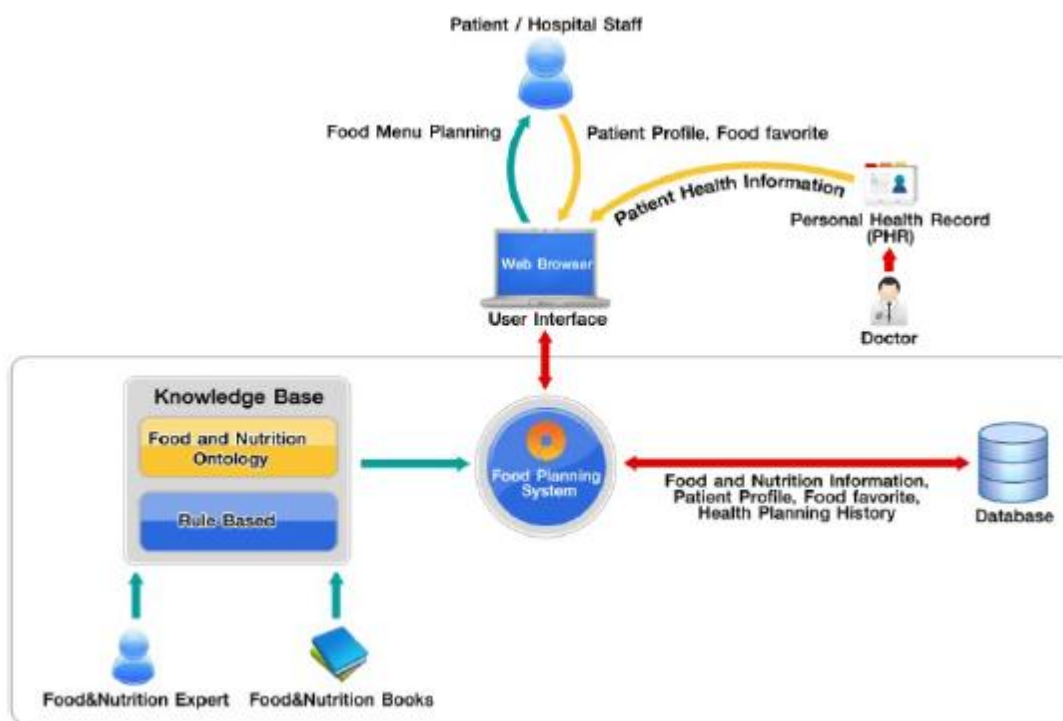
**Tabelul 3.1 Media acurateții procentajelor de calorii și macronutrienți pentru cinci meniuri**

În lucrarea [4] este prezentat un sistem care a fost conceput special pentru persoanele în vârstă. Acesta a apărut cu scopul de a rezolva problemele de alimentație ale persoanelor în vârstă din Thai. Pentru a genera recomandările personalizate, sistemul ia în considerare următoarele criterii: starea de sănătate a utilizatorului, recomandările nutriționale ale unui cadru specializat, alimentul pe care utilizatorul dorește să îl consume, alimentul pe care utilizatorul dorește să nu îl consume și bucătăria regională pe care o preferă. Se folosește o ontologie pentru a stoca informații referitoare la caracteristicile specifice mâncărilor (arome, țara din care provine) și informații referitoare la valorile energetice și nutritive ale mâncărilor, precum și o bază de date pentru a stoca profilul utilizatorului, preferințele acestuia, istoricul alimentar al acestuia, precum și valorile nutritive ale alimentelor. Sistemul este format din cinci componente, ce sunt prezentate în figura 3.1:

- Interfața cu utilizatorul: cu ajutorul acesteia se preiau informațiile referitoare la pacient și se afișează sugestiile alimentare.



- Dosarul personal de sănătate: conține date legate de afecțiunile medicale, precum și informații personale ca numele, vârsta, sexul, greutatea, înălțimea, pulsul și tensiunea arterială.
- Baza de cunoștințe: conține informații legate de alimente și valoarea nutritivă a acestora, precum și reguli pentru obținerea unui meniu personalizat.
- Baza de date: conține informații despre mâncare, nutrienți, profilul personal, mâncarea favorită și istoricul alimentar.
- Sistemul de planificare alimentară: acesta citește informațiile nutriționale din ontologie, informațiile referitoare la starea de sănătate a pacientului și preferințele alimentare ale acestuia. Acest modul folosește celelalte componente pentru a genera sugestia alimentară.

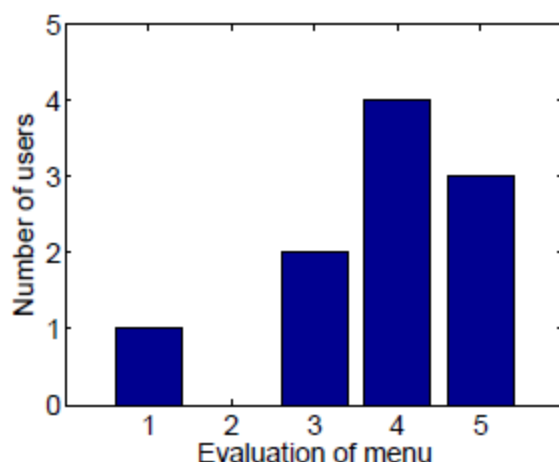


**Figura 3.1** Arhitectura sistemului prezentat în [4]

Utilizatorul poate alege să vizualizeze rezultatele generate pentru o masă, o zi sau o săptămână. Un dezavantaj al acestui sistem este dat de faptul că dieta recomandată nu este variată. Pentru rularea prezentată în lucrare, sistemul sugerează consumarea de trei ori în aceeași zi orez și carne de porc.

În lucrarea [5] este prezentat un sistem care utilizează teoria „rough sets”. Acest sistem generează un meniu alimentar recomandat, pe baza preferințelor alimentare ale utilizatorului, extrase dintr-un chestionar. Chestionarul conține zece feluri de mâncare generate în mod aleator. Pentru fiecare fel de mâncare, utilizatorul trebuie să realizeze o evaluare, adică să stabilească dacă respectivul fel de mâncare îi place sau nu. Din informațiile pe care utilizatorul le furnizează prin completarea chestionarului este extrasă o regulă referitoare la preferințele alimentare ale utilizatorului. Această regulă este extrasă folosindu-se teoria mulțimilor brute și este apoi folosită pentru generarea unui mediu personalizat. Acest sistem a fost evaluat cu ajutorul a zece utilizatori și s-a analizat în ce măsură meniul generat respectă preferințele utilizatorilor.

Rezultatele evaluării pot fi observate în figura 3.2. Un singur utilizator nu a fost mulțumit de meniul recomandat de sistem.

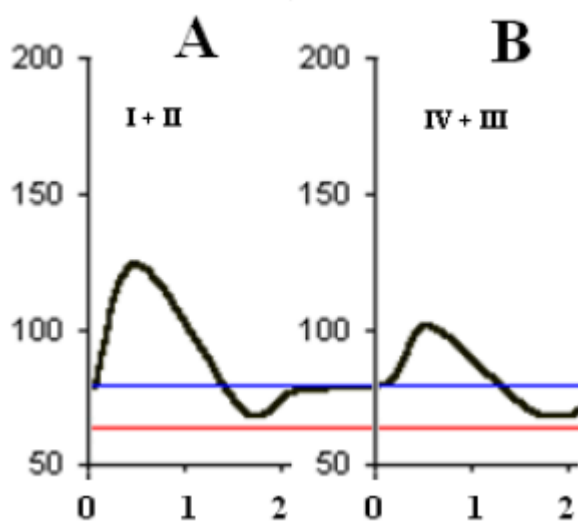


**Figura 3.2 Rezultatele evaluării sistemului [5]**

În lucrarea[6] este prezentat un sistem care generează meniuri alimentare cu scopul de a preveni afecțiunile cardiovasculare. Meniurile vor fi generate astfel încât anumite constrângeri alimentare să fie satisfăcute, iar componentele meniului să poată fi combinate. De asemenea, pentru generarea meniurilor se iau în considerare preferințele alimentare ale utilizatorului precum și alte informații referitoare la utilizator ca, de exemplu, vârsta. Acest sistem utilizează concepte din algoritmi genetici, pentru a genera meniurile recomandate. Un individ este modelat fie ca o singură masă, fie ca un meniu pentru o zi. Evaluarea indivizilor se realizează cu ajutorul unei funcții de fitness bazată pe penalități. Populația de indivizi evoluează prin utilizarea operatorilor de crossover, mutație și selecție. Operatorul de crossover este folosit pentru a interschimba atributele a doi indivizi, utilizându-se un punct de crossover ales aleator. Operator de mutație înlocuiește în mod aleator un fel de mâncare cu un alt fel potrivit. Selecția indivizilor se realizează pe baza funcției de fitness. Sistem utilizează o bază de date, dezvoltată special pentru bucătăria și stilul de viață din Ungaria. Aceasta conține informații referitoare la rețete, ingrediente, clasificarea ingredientelor, precum și următoarele valorile nutritive ale ingredientelor: calorii, proteine, lipide, carbohidrați, fibre, sare, apă și potasiu. Clasificarea ingredientelor se face în funcție de piramida alimentară.

Un alt sistem care utilizează concepte din algoritmi genetici este prezentat în [7]. Acesta a fost conceput cu scop educațional, fiind destinat persoanelor care suferă de diabet. Sistemul este format din două componente: simulatorul de glucoză și insulină și generatorul de meniuri. Generatorul de meniuri utilizează rezultatele oferite de simulator, precum și preferințele alimentare ale utilizatorului pentru a găsi o combinație de produse și meniuri, care să cauzeze cea mai mică creștere posibilă a nivelului de glucoză. Spre deosebire de alte sisteme similare, acesta permite vizualizarea grafică a nivelelor de glucoză și insulină în raport cu o anumită dietă. În figura 3.3 se poate observa graficul rezultat pentru două combinații diferite de produse alimentare. În graficul notat cu A au fost combinate un produs cu indice glicemic moderat, urmat de un produs cu indice glicemic scăzut. În graficul notat cu B au fost combinate un produs cu indice glicemic ridicat, urmat de un produs cu indice glicemic foarte scăzut. Din cele două grafice se poate remarca faptul că amplitudinea glucozei nu depinde doar de nivelul indicelui

glicemic, ci și de meniurile ce au fost combinate, precum și ordinea ingerării acestora. Acest sistem a fost testat pe un grup de 10 studenți sănătoși (care nu sufereau de diabet), precum și pe 11 femei care se aflau sub tratament cu insulină. Rezultatele au fost satisfăcătoare, considerându-se ca scopul educațional a fost atins.



**Figura 3.3 Simularea fluxului nivelului de glucoză (mg/dl) în timp (h) pentru două combinații alimentare**

## 4. Analiză și fundamentare teoretică

Pentru realizarea acestui proiect se vor utiliza metaheuristici inspirate din biologie, cu scopul de a genera recomandări alimentare care să respecte necesarul energetic și nutrițional al utilizatorului. Deoarece exista un număr mare de oferte alimentare, găsirea combinației optime va fi modelată ca o problemă de optimizare combinatorică.

### 4.1. Optimizare

Optimizarea reprezintă alegerea celui mai bun element dintr-o mulțime de elemente. Conform [8], problemele de optimizare pot fi scrise, în general, sub următoarea formă matematică:

$$\begin{aligned} & \underset{x \in \mathbb{R}^n}{\text{minimizează}} f_i(x), \quad (i = 1, 2, \dots, M), \\ & \text{cu constrângerile } \phi_j(x) = 0, (j = 1, 2, \dots, J), \\ & \psi_k(x) \leq 0, (k = 1, 2, \dots, K), \end{aligned}$$

unde:

- $\mathbb{R}^n$  reprezintă spațiul de căutare;
- $x = (x_1, x_2, \dots, x_n)^T$  reprezintă vectorul variabilelor de decizie;
- componentele  $x_i$  sunt variabile de decizie;
- $f_i(x)$ ,  $\phi_j(x)$  și  $\psi_k(x)$  sunt funcții ale vectorului de decizie  $x$ ;
- $f_i(x)$ , unde  $i = 1, 2, \dots, M$  sunt funcții obiectiv.

Tot în [8] sunt prezentate mai multe criterii în funcție de care poate fi clasificată optimizarea:

- numărul obiectivelor: problema de optimizat poate avea un singur obiectiv ( $M = 1$ ) sau mai multe obiective ( $M > 1$ );
- numărul constrângerilor  $J+K$ : există probleme de optimizat fără constrângeri ( $J = K = 0$ ) sau cu constrângeri ( $J + K > 0$ );
- forma funcției: funcția obiectiv poate fi liniară sau neliniară;
- forma (landscape): funcțiile obiectiv care nu sunt liniare pot fi unimodale (au un singur optim global) sau multimodală (au mai multe optime globale);
- tipul variabilelor de decizie: variabilele de decizie  $x_i$  pot fi discrete, continue sau mixte;
- determinism: sistemul poate fi determinist sau stocastic (atunci când variabilele de decizie, funcțiile obiectiv și/sau constrângerile au o anumită incertitudine sau un anumit zgomot).

Problemele de optimizare care au un singur obiectiv pot fi considerate probleme de optimizare scalare deoarece funcția obiectiv întotdeauna atinge o singură valoare optimă globală sau un scalar. În cazul optimizărilor cu multiple obiective, multiplele funcții obiectiv formează un vector și din acest motiv aceste optimizări se numesc optimizări vectoriale. Optimizarea funcțiilor cu mai multe criterii necesită optimizarea mai multor funcții obiectiv, unele dintre acestea putând fi conflictuale. Problemele de acest tip nu au o soluție unică ce optimizează toate criteriile inițiale, ci există mai multe soluții de compromis care se numesc soluții optime

Pareto. Aceste soluții se caracterizează prin faptul că în spațiul de căutare nu există vectori care să fie mai buni decât ele în raport cu toate criteriile.

Există mai multe metode prin care putem transforma o problema de optimizare multicriterială într-o problemă cu un singur obiectiv. În [8] sunt prezentate doua metode: una care se bazează pe suma ponderată, respectiv o alta care se bazează pe utilități.

În cazul metodei care se bazează pe suma ponderată, toate funcțiile multi-obiectiv pot fi combinate într-un obiectiv scalar dacă se asignează ponderi obiectivelor. Soluțiile vor depinde în acest caz de ponderile alese. Funcția de fitness  $F(x)$  va avea în acest caz următoarea formă matematică:

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \dots + \alpha_p f_p(x),$$

unde coeficienții  $(\alpha_1, \alpha_2, \dots, \alpha_p)$  reprezintă ponderile asociate funcțiilor obiectiv.

Metoda care se bazează pe funcția de utilitate ia în considerare și factori ca gradul de risc și preferințele. Această metodă definește combinații ale valorilor funcțiilor obiectiv  $f_1, f_2, \dots, f_p$  care sunt considerate la fel de convenabile. Deoarece această funcție poate fi construită în multe feluri, definirea ei este considerată dificilă.

## 4.2. Metaeuristici

Începând cu anii 1970, odată cu apariția algoritmilor genetici, s-a acordat din ce în ce mai multă atenție algoritmilor inspirați din procese sau evenimente naturale. Până în prezent au fost publicate numeroase lucrări care prezintă probleme de optimizare rezolvate cu ajutorul algoritmilor metaeuristici inspirați din natură.

În [9] se prezintă caracteristicile generale ale metaeuristicilor, precum și o comparație a strategiilor folosite de diverse metaeuristici. Metaeuristicile sunt prezentate ca fiind strategii de nivel înalt, de explorare a spațiului de căutare în diferite moduri. Deoarece metaeuristicile au un grad mare de nepredictibilitate, este importantă menținerea unui echilibru între explorarea globală și exploatarea locală a spațiului de căutare. Noțiunile de exploatare și explorare sunt întâlnite și sub numele de intensificare, respectiv diversificare. Componenta de intensificare se concentrează pe examinarea soluțiilor aflate în vecinătatea soluțiilor calitative, iar cea de diversificare se bazează pe examinarea regiunilor care nu au fost vizitate anterior. În cadrul unei metaeuristici, diversificarea este necesară pentru a putea identifica zonele din spațiul de căutare care au soluții calitative, iar intensificarea pentru a nu exploata exagerat de mult zonele care au fost deja exploatare sau zonele care nu oferă soluții calitative.

Metaeuristicile hibride sunt definite în [10] ca fiind combinația unei metaeuristici cu componente ale altei metaeuristici sau cu tehnici care aparțin domeniilor de inteligență artificială sau „operation research”. Prin utilizarea unei astfel de metaeuristici se combină punctele tari ale diversilor algoritmi, obținându-se o eficiență sporită și o flexibilitate ridicată. Pentru realizarea unei metaeuristici hibride, algoritmi componentei pot fi combinați în mod colaborativ sau în mod integrativ. Spre deosebire de combinațiile integrative în care se folosește un singur algoritm denumit „master” și cel puțin un algoritm denumit „slave”, în combinațiile colaborative, niciun algoritm nu reprezintă o parte a altui algoritm. În cazul combinațiilor colaborative, algoritmi se pot executa secvențial, intercalat sau în paralel.

Tot în [9] sunt prezentate mai multe metode de clasificare a metaeuristicilor. Cele mai importante criterii în funcție de care se pot clasifica metaeuristicile sunt:

- origine: exista metaeuristici care sunt inspirate din natură sau metaeuristici care nu sunt inspirate din natură.

- numărul de soluții folosite de metaeuristică: algoritmi pot utiliza o singură soluție în orice moment (single point) sau pot utiliza mai multe soluții (population-based).
- modul în care folosesc funcția obiectiv: există algoritmi care păstrează aceeași funcție obiectiv pe parcursul întregii căutări (funcția obiectiv este statică) și algoritmi care modifică această funcție în timpul căutării (funcția obiectiv este dinamică).
- numărul de structuri de vecini: cei mai mulți algoritmi folosesc o singură structură de vecini, dar există și algoritmi care utilizează multiple structuri de vecini pentru a realiza o diversificare a căutării.
- utilizarea istoricului căutărilor: există metaeuristici care determină acțiunea următoare în funcție de starea curentă (metoda memory-less) și metaeuristici care acumulează informații despre căutare (metoda memory usage).

În [9] sunt prezentate mai multe metaeuristici care lucrează cu o singură soluție la un moment dat. Printre acestea se numără și algoritmul de tipul Simulated Annealing. Acesta este unul dintre primii algoritmi care deține o strategie ce rezolvă problema blocării în minime locale. Ideea de bază a acestui algoritm este de a permite efectuarea de operații în urma cărora să rezulte soluții mai slabe calitativ în comparație cu soluția curentă. Soluția mai slabă calitativ înlocuiește soluția curentă cu o anumită probabilitate, calculată, în general, cu ajutorul distribuției Boltzmann. Acest tip de operații sunt realizate cu scopul de a ieși dintr-un punct de minim local. Algoritmul începe cu generarea unei soluții inițiale ( $s$ ) și inițializarea parametrului  $T$  (numit temperatură). După etapa de inițializare, atât timp cât condiția de oprire nu este îndeplinită, se execută în mod iterativ următoarele operații:

- se alege în mod aleator o soluție  $s'$  care se află în vecinătatea soluției  $s$ ;
- se calculează valorile celor două soluții:  $f(s)$ , respectiv  $f(s')$ ;
- soluția  $s'$  devine noua soluție curentă dacă  $f(s') \geq f(s)$ , cu o probabilitate calculată în funcție de  $T$  și  $f(s') - f(s)$ , sau dacă  $f(s') < f(s)$ ;
- se actualizează parametrul  $T$ .

Din categoria metaeuristicilor care lucrează cu mai multe soluții la un moment dat, face parte și algoritmul Ant Colony Optimization, care este inspirat din comportamentul furnicilor în căutarea hranei. În [9] se prezintă faptul că în timp ce furnicile se deplasează între musuroi și sursa de hrană, acestea lasă urme de feromon. Pentru modelarea problemei se consideră un graf  $G = (C, L)$ , ale cărui noduri sunt componentele  $C$ , iar setul  $L$  conectează componentele  $C$ . Elementele setului  $L$  sunt numite conexiuni. Componentele grafului  $G$  au asociat un set de parametri  $T$ , numiți „pheromone trail”. Componentele  $c_i \in C$  au asociat un parametru numit „pheromone trail”, notat cu  $T_i$ , iar conexiunile  $l_{ij} \in L$ , au asociat un parametru notat cu  $T_{ij}$ . Algoritmul începe cu inițializarea setului de parametri  $T$ , apoi se execută în mod iterativ următoarele operații, până când condiția de oprire este îndeplinită:

- fiecare furnică din colonie construiește o nouă soluție prin deplasarea în cadrul grafului  $G$ ; în timpul deplasării, sunt memorate soluțiile parțiale obținute în timpul deplasării în graf;
- în timpul acestei deplasări se actualizează componentele setului  $T$ ;
- opțional, se poate adăuga o cantitate suplimentară de feromon componentelor setului  $T$  care sunt utilizate de cea mai bună soluție.

În [11] este prezentat algoritmul albinelor (Bees Algorithm), care aparține metaeuristicilor bazate pe populație. Acest algoritm este inspirat din comportamentul roilului de albine în căutarea hranei. Algoritmul începe cu plasarea în mod aleator a albinelor „cercetașe” în

spațiul de căutare și evaluarea acestora. După etapa de inițializare, atât timp cât condiția de oprire nu este îndeplinită, se execută în mod iterativ următoarele operații:

- se aleg poziții care se află în vecinătatea albinelor considerate cele mai bune din punct de vedere calitativ;
- se selectează albine, se plasează în pozițiile găsite la pasul anterior, apoi se evaluează albinele;
- pentru fiecare „patch” doar albina cea mai bună calitativ va fi selectată pentru a aparține următoarei populații;
- albinele care nu au o poziție sunt distribuite în mod aleator, apoi sunt evaluate;

### 4.3. Ontologie

În lucrarea [12] sunt prezentate mai multe definiții ale ontologiei în domeniul informatic. În 1993, Gruber afirma că ontologia este „o specificație explicită a conceptualizărilor”. Patru ani mai târziu, Borst definea ontologia ca fiind „o specificație formală a unei conceptualizări comune”. În anul 1998, Studer îmbină definițiile predecesorilor săi, susținând că „o ontologie e o specificație formală, explicită a unei conceptualizări comune”. Toate cele trei definiții prezentate utilizează noțiunea de „conceptualizare”. Conceptualizarea reprezintă o perspectivă abstractă și simplificată a domeniului pe care dorim să îl reprezentăm.

Ontologiile sunt utilizate pentru a descrie concepte și relații dintr-un anumit domeniu. Acestea oferă un vocabular comun pentru a indica tipuri, proprietăți și relația dintre concepte. O ontologie poate conține numeroase componente, printre care:

- clase: reprezintă seturi, colecții, concepte care conțin indivizi;
- indivizi: reprezintă instanțe sau obiecte;
- relații: reprezintă modul în care clasele și indivizii pot fi legate între ele;
- atribute: reprezintă aspecte, proprietăți, caracteristici sau parametri pe care obiectele sau clasele le pot avea;
- axiome: reprezintă afirmații (inclusiv reguli) care cuprind împreună teoria generală pe care ontologia o descrie în domeniul său de aplicare;
- restricții: reprezintă descrierea a ceea ce trebuie să fie adevărat pentru ca unele afirmații să poată fi acceptate ca date de intrare.

Există numeroase metode de clasificare a ontologiilor. Lucrarea [13] prezintă câteva caracteristici în funcție de care putem clasifica ontologiile:

1. Expresivitate: se referă la gradul de formalismului utilizat pentru descrierea ontologiei. Este mai dificil de dezvoltat o ontologie care utilizează un formalism mai ridicat, dar aceasta va exclude interpretările nedorite.
2. Dimensiunea comunității relevante. Ontologiile care sunt dezvoltate pentru a fi utilizate de o comunitate mari trebuie să fie ușor de înțeles, bine documentate și de dimensiuni limitate. Această caracteristică poate influența dinamica cu care ontologia se schimbă sau evoluează.
3. Dinamica conceptuală din domeniu: se referă la cantitatea de elemente conceptuale care apar sau se schimbă într-o perioadă de timp.
4. Numărul de elemente conceptuale din domeniu. Este mai dificil de vizualizat și de revizuit o ontologie care conține multe elemente conceptuale comparativ cu o ontologie mai mică. De aceea, ontologiile cu mai puține elemente conceptuale sunt, deseori, mai utilizate.

5. Gradul de subiectivitate folosit în conceptualizare domeniul: se referă la măsura în care noțiunile unui concept diferă în funcție de actor. Printre domeniile cu un grad mai mare de subiectivitate se numără cultura și religia.
6. Dimensiunea medie a specificației pe element. Această caracteristică se referă la nivelul la care sunt descrise conceptele din ontologie: axiomele, numărul de atribute pe concept etc.

#### **4.4. Informații medicale cu privire la necesarul energetic și nutrițional al organismului**

În lucrarea [14] sunt prezentate informații legate de necesitățile organismului uman. Pentru o bună funcționare a organismului uman este necesar ca acesta să se afle într-un echilibru energetic. Modificările depozitelor energetice ale organismului reprezintă diferența dintre aportul de energie și consumul de energie. Cele mai importante componente ale consumului de energie sunt metabolismul bazal, termogeneza și activitatea fizică. Aportul zilnic recomandat de energie poate fi calculate cu diferite formule ca: ecuația lui Harris-Benedict, ecuația lui Mifflin-St. Jeor, ecuația WHO (World Health Organization) etc. Pe lângă substanțele energetice, există două tipuri de substanțe nutritive: macronutrienți (glucide, lipide și proteine) și micronutrienți (vitamine și minerale).

Glucidele reprezintă cea mai accesibilă sursă de energie pentru organism, având rol structural și energetic. Acestea au rolul de combustibil pentru toate celulele și sunt utilizate la menținerea temperaturii corporale, întreținerea funcțiilor vitale, asigurând totodată energia pentru eforturile musculare. Există mai multe tipuri de glucide, printre care: glucide simple și glucide complexe (polizaharide). Glucidele simple sunt absorbite imediat în sânge și apoi sunt disponibile ca sursă de energie în organism, ceea ce duce la creșterea rapidă a secreției de insulină, urmată de scăderea bruscă a glicemiei. Glucidele complexe sunt mai întâi transformate în glucide simple și apoi sunt absorbite, secreția de insulină fiind mai bine reglată, datorită eliberării progresive a moleculelor de glucoză în sânge. Din acest motiv, specialiștii recomandă consumul de alimente bogate în glucide complexe. Cantitatea de glucide consumată zilnic de către persoanele vârstnice coincide cu doza zilnică recomandată adulților, adică aproximativ 50-55% din aportul energetic zilnic.

Lipidele constituie o familie de compuși insolubili în apă, având rol în furnizarea de energie organismului, în transportul anumitor proteine și a anumitor hormoni în sânge, intrând în constituția tuturor membranelor celulare. Lipidele reprezintă cel mai economic mod de stocare al energiei. Totuși un consum crescut de alimente bogate în lipide duce la apariția dislipidemiilor, care se asociază cu un risc mare de apariție a bolilor cardiovasculare. Specialiștii recomandă ca la persoanele vârstnice doza zilnică de lipide să reprezinte 20-30% din totalul caloric zilnic. Grăsimile saturate nu trebuie să depășească 8% din calorii.

Proteinele sunt molecule de baza, care contribuie la dezvoltarea normală a organismului, fiind alcătuite din aproximativ 20 de aminoacizi. Aproape jumătate din acești aminoacizi se numesc neesențiali, putând fi sintetizați în organism din alți aminoacizi. Proteinele au rol structural, funcțional și energetic, fiind utilizate în creșterea și refacerea țesuturilor, în desfășurarea proceselor metabolice și asigurând aportul energetic necesar organismului. Specialiștii recomandă ca persoanele în vârstă să consume 0,8 g/kgcorp/zi, dar acest aport poate crește până la 1-1,25 g/kgcorp/zi. Creșterea aportului de lipide la peste valoarea recomandată determină apariția afecțiunilor renale și gastrointestinale, iar reducerea aportului duce la apariția malnutriției proteice sau proteocalorice.



Vitaminele sunt clasificate în liposolubile(A, D, E, K) și hiposolubile (grupul B și vitamina C). În cazul ambelor categorii de vitamine pot apărea deficiențe, dar efectele toxice ale unui aport excesiv sunt posibile în special în cazul vitaminelor liposolubile.

Vitamina A se găsește în produsele alimentare, în general în asociere cu lipidele, dar și în margarină. În timpul preparării termice la temperaturi înalte sau prin expunerea la lumina, alimentele își pierd o parte importantă din vitamina A. Aceasta are rol în reglarea funcțiilor sistemului imun și în reproducere.

Vitamina E se găsește în uleiurile vegetale, nuci, cereale, pește, carne și legume verzi. Alimentele își reduc cantitatea de vitamina E prin frigere, dar nu și prin fierbere. Această vitamină e considerată cel mai puternic antioxidant liposolubil, având rolul de a proteja acizii grași polinesaturați din structura membranelor celulare împotriva degradării oxidative.

Vitamina C se găsește atât în produsele de origine vegetală, cât și în cele de origine animală, principalele surse fiind reprezentate de fructe, legume și viscere. Refrigerarea și înghețarea rapidă a alimentelor conservă vitamina C, dar aceasta se pierde în apa de gatit. Vitamina C intervine în sinteza collagenului, carnitinei și a receptorilor colinergici, protecția antiinfecțioasă și împotriva aterosclerozei.

Vitamina D se găsește în lipidele alimentare și în cele vegetale. Alimentele nu își reduc cantitatea de vitamina D prin conservare sau preparare.

Mineralele se clasifică în macromineraie și micromineraie. În cazul macrominerailelor necesarul zilnic depășește 100mg/zi, iar în cazul microminerailelor necesarul zilnic se află sub 100 mg/zi. Absorbția mineralelor este în general mai redusă decât cea a vitaminelor și a macronutrienților. Tabelul 4.1 conține dozele zilnice recomandate de minerale la persoanele vârstnice.

<u>Grup populațional</u>	<u>Aportul zilnic recomandat</u>					
	<u>Ca</u> <u>(mg/zi)</u>	<u>F</u> <u>(mg/zi)</u>	<u>I</u> <u>(μg/zi)</u>	<u>Fe</u> <u>(mg/zi)</u>	<u>Mg</u> <u>(mg/zi)</u>	<u>P</u> <u>(mg/zi)</u>
<u>Femei</u>	1200	3	150	8	320	700
<u>Bărbați</u>	1200	4	150	8	420	700

**Tabel 4.1 Dozele zilnice recomandate de minerale pentru persoanele vârstnice**

Calciul se găsește în produse lactate, legume frunzoase verzi, arpagic, pești cu oase mici, moluște și stridii. Calciul intervine în formarea oaselor și a dinților, în transmiterea impulsului nervos la nivelor sinapselor interneuronale și al plăcii neuromusculare, în menținerea tonusului muscular normal și echilibrului între contracția și relaxarea musculară.

Magneziul se găsește în semințe, cereale neprelucrate, grâul germinat și tărâta de grâu, nucile, leguminoasele, legumele verzi, apa dură, cafea, ceai și cacao. Magneziul este necesar pentru sinteza și metabolismul proteinelor, lipidelor, glucidelor, pentru reproducerea și creșterea celulară și pentru reglarea transportului calciului.

Sodiu se găsește în lapte, brânză, ouă, carne, pește și în cantitate mai mică în legume și cereale. Sodiu intervine în balanța gidrică a organismului, în echilibrul acido-bazic, în permeabilitatea celulară a materialelor metabolice și în excitabilitatea musculară.

Fierul se găsește în ouă, carne slabă, legume, nuci, fructe uscate, cereale și vegetale verzi. Fierul are rol în transportul oxigenului, în oxidarea celulară, în creștere și dezvoltare.

Fosforul se găsește în carnea de pui, pește, carnea roșie și ouă și în cantități mai mici în produsele lactate, carnea slabă, nuci, leguminoase și cereale integrale. Fosforul intervine în formarea oaselor și dinților, absorbția intestinală a glucozei, transportul și metabolismul unor aminoacizi, transportul acizilor grași și metabolismul energetic.

La vârstnici poate apărea malnutriția în două forme: o malnutriție generalizată (caracterizată prin deficiențe nutriționale multiple) și o malnutriție cauzată de deficiențe nutriționale izolate (când un anumit grup de alimente este deficitar în dietă). Pentru a îmbunătăți nutriția la persoanele de vârstă a treia se recomandă consumul următoarelor alimente: un pahar de suc de fructe zilnic, cereale integrale cu lapte la micul dejun, pește sau carne zilnic, un pahar de lapte seara și cel puțin o porție de legume zilnic.

## 5. Tehnică hibridă inspirată din comportamentul cucilor pentru generarea de meniuri alimentare personalizate

### 5.1. Modelul hibrid inspirat din comportamentul cucilor

Modelul hibrid care a fost dezvoltat în cadrul acestei lucrări combină principiile de bază ale comportamentului de reproducere al cucilor, cu elemente de algoritmi genetici, Tabu Search și Reinforcement Learning. Necesitatea hibridizării a apărut în contextul îmbunătățirii afinității soluțiilor obținute. De asemenea, prin hibridizare se încearcă și îmbunătățirea timpului de execuție. Modelul hibrid încearcă să combine avantajele componentelor enumerate, evitând dezavantajele acestora. În cele ce urmează sunt prezentate fiecare dintre componentele modelului hibrid, prin sublinierea modului de adaptare la problema generării de oferte alimentare adecvate.

#### 5.1.1. Componenta inspirată din comportamentul cucilor

Inspirația pentru această componentă se află în mecanismul agresiv de reproducere al cucilor prezentat în [15]. Spre deosebire de alte păsări, femela cuc nu își construiește un cuib și nici nu își clocește ouăle. Femela cuc caută un cuib al altei păsări, își depune un ou în acesta, apoi fură un ou din cuib pentru ca pasărea gazdă să nu observe schimbarea numărului de ouă. De obicei, păsările gazdă nu realizează ceea ce s-a întâmplat și clocesc oul de cuc împreună cu celelalte ouă. Atunci când o pasăre gazdă descoperă ouă care nu îi aparțin, fie aruncă ouăle străine, fie părăsește cuibul. Pentru a minimiza șansele ca ouăle să fie descoperite de pasărea gazdă, unele specii de cuc au evoluat, specializându-se pe mimarea culorilor și modelelor de ouă ale altor păsări. În general, puii de cuc ies din ou înaintea celorlalți pui și se dezvoltă mai repede. Odată ieșit din ou, puiul de cuc are instinctul de a-i omorî pe ceilalți pui din cuib, pentru a primi toată atenția păsărilor gazdă.

Pentru a simplifica mecanismul de reproducere al cucilor, vom aplica următoarele reguli, conform [15]:

1. Fiecare cuc depune un singur ou la un moment dat într-un cuib ales în mod aleator;
2. Cuiburile care au cele mai bune ouă din punct de vedere calitativ vor fi utilizate și în iterațiile următoare;
3. Numărul de cuiburi gazdă disponibile este fix. Fiecare pasăre gazdă descoperă oul de cuc cu o probabilitate  $repCont \in [0,1]$ . Pentru a simplifica această presupunere, un procent  $repCont$  de cuiburi sunt înlocuite cu unele generate aleator.

Pentru dezvoltarea tehnicii inspirate din comportamentul de reproducere al cucilor, a fost necesară realizarea mapării conceptelor implicate în acest process. Această mapare este prezentată în tabelul 5.1. Se încearcă maparea conceptelor cât mai aproape de contextul real al reproducerii cucilor. Un ou de cuc corespunde unei soluții compuse din oferte alimentare, deoarece ouăle de cuc și soluțiile alimentare reprezintă centrul acestei probleme. Modelarea ouălor din cuib nu diferă de modelarea ouălor de cuc. Ouăle sunt evaluate cu ajutorul unei funcții de fitness multi-criteriale ce va fi prezentată ulterior. Pentru a simplifica problema de optimizat presupunem ca un cuib are un singur ou, deci evaluarea cuibului corespunde evaluării oului. De asemenea, un cuc are un singur ou la un moment dat. Depunerea de către cuc a oului propriu în cuib și furtul unui ou din cuib sunt reprezentate prin înlocuirea soluției corespunzătoare cuibului cu soluția corespunzătoare cucului. Descoperirea oului străin în cuib și părăsirea cuibului de

către gazdă duce la contruirea unui nou cuib. Această problemă este modelată prin generarea aleatoare a unei noi soluții corespunzătoare noului cuib.

Conceptele reproducerii cucilor	Conceptele modelul hibrid
Ou de cuc	Soluție compusă din oferte alimentare
Ou din cuib	Soluție compusă din oferte alimentare
Evaluarea oului	Valoare funcția de fitness multi-criteriale
Depunerea unui ou în cuib	Înlocuirea soluției cuibului cu cea a cucului
Construirea unui nou cuib	Generarea aleatoare a unei noi soluții

**Tabel 5.1 Maparea conceptelor care aparțin comportamentului de reproducere al cucilor asupra conceptelor modelului hibrid dezvoltat**

Pentru a evalua soluțiile alimentare considerăm modelarea soluției sub forma unui vector  $Y = [y_1, y_2, \dots, y_m]$ , unde  $y_1, y_2, \dots, y_m$  reprezintă oferte alimentare. Definim funcția de fitness ca fiind:

$$F(Y) = \alpha_1 F_p(Y) + \alpha_2 F_t(Y) + \alpha_3 F_N(Y), \alpha_1 + \alpha_2 + \alpha_3 = 1, \alpha_1, \alpha_2, \alpha_3 \in [0,1].$$

Optimizarea multi-criterială se realizează cu ajutorul care se bazează pe suma ponderată, fiecărui obiectiv fiindu-i se asigănată o pondere  $\alpha_i$ . Cele trei obiective principale sunt definite astfel:

$$\begin{aligned} F_p(Y) &= \text{Norm}(\sum_{y_i \in Y} \text{Pret}(y_i)), \\ F_t(Y) &= \text{Norm}(\max_{y_i \in Y} \text{Timp}(y_i)), \\ F_N(Y) &= \sum_{l=1}^4 \omega_l f_l(Y), \sum_{l=1}^4 \omega_l = 1, \omega_l \in [0,1], \end{aligned}$$

unde:

- $\text{Norm}(A)$  reprezintă normalizarea valorii  $A$ ;
- $\text{Pret}(y_i)$  reprezintă costul mâncării  $y_i$ ;
- $\text{Timp}(y_i)$  reprezintă timpul de livrare al mâncării  $y_i$ ;
- $l \in \{\text{calorii}, \text{proteine}, \text{lipide}, \text{glucide}\}$ ;
- $\omega_l$  reprezintă ponderea asigănată substanței energetice/nutritive  $l$ .

Pentru fiecare element nutritiv sau energetic funcția obiectiv este calculată cu următoarea formulă:

$$f_l(Y) = \frac{\text{Prag}_l - (\text{ValOptima}_l - \sum_{y_j \in Y} \text{Nutrient}_l(y_j))}{\text{Prag}_l},$$

unde:

- $\text{Prag}_l$  reprezintă limita maximă admisă pentru nutrientul  $l$ ;
- $\text{ValOptima}_l$  reprezintă valoarea optimă pentru nutrientul  $l$ ;
- $\text{Nutrient}_l(y_j)$  reprezintă cantitatea de nutrient  $l$  conținută în mâncarea  $y_j$ .

Soluțiile alimentare trebuie să respecte anumite constrângeri nutritive, ce sunt definite astfel:

$$C_j(y) = \sum_{y_i \in Y} \sum_{k=1}^{nr\_ing(y_i)} q_{ik} * p_{kj} \in [v_{j\ min}, v_{j\ max}],$$

unde:

- $nr\_ing(y_i)$  reprezintă numărul de ingrediente conținute în mâncarea  $y_j$ ;
- $q_{ik}$  reprezintă cantitatea din ingredientul  $k$  conținută în mâncarea  $y_j$ ;
- $p_{kj}$  este cantitatea de nutrient  $j$  conținută în ingredientul  $k$ ;
- $v_{j\ min}$  este necesarul zilnic minim de micronutrientul  $j$ ;

- $v_{j \max}$  este necesarul zilnic maxim de micronutrientul  $j$ ;
- $j \in \{\text{sodiu, fier, vitamina A, vitamina B1, vitamina C}\}$ .

### 5.1.2. Componenta hibridă bazată pe principiile geneticii și ale selecției naturale

Inspirația pentru această componentă se bazează pe principiile geneticii și ale selecției naturale, enunțate de Darwin ("supraviețuiește cel care e cel mai bine adaptat"). Speciile care se adaptează mai ușor la mediu pot supraviețui și evolua peste generații, iar cele care nu reușesc să se adapteze dispar cu timpul, ca urmare a selecției naturale. Există trei tipuri de operatori genetici: selecția, mutația și încrucișarea. Pentru dezvoltarea tehnicii hibride am utilizat operatorul de mutație.

Mutația presupune faptul că indivizii sunt modificați în mod aleator. În cadrul mutației cromozomul curent își modifică una sau mai multe valori într-un singur pas. Scopul acestei operații este de a menține diversitatea în cadrul populației, precum și de a împiedica convergența prematură a soluțiilor. În contextul tehnicii hibride, operatorul de mutație este utilizat pentru actualizarea soluției corespunzătoare oului de cuc. Pentru fiecare componentă din soluție se execută următoarele operații:

- componenta curentă este comparată cu componenta corespunzătoare care aparține soluției optime găsite până în acel moment;
- dacă cele două componente sunt identice nu se execută nicio modificare;
- dacă cele două componente diferă se execută următorii pași:
  - se generează un număr aleator  $x \in [0,1]$ ;
  - dacă  $x < trMut$  atunci nu se execută nicio modificare ( $trMut$  reprezintă un prag prestabilit);
  - dacă  $x \geq trMut$  atunci componenta curentă este înlocuită cu o nouă componentă generată în mod aleator.

Figura 5.1 ilustrează modul în care este utilizat operatorul de mutație în cadrul unor soluții formate din 4 componente. În cadrul soluției rezultate componenta 1 și componenta 4 nu au fost înlocuite pentru că numărul  $x$  generat aleator a fost mai mic decât valoarea pragului  $trMut$ . Componenta 2 nu a fost înlocuită pentru că este identică cu componenta corespunzătoare din soluția optimă. Componenta 3 a fost înlocuită cu o componentă aleatoare deoarece numărul  $x$  generat aleator a fost mai mare sau egal cu valoarea pragului  $trMut$ .

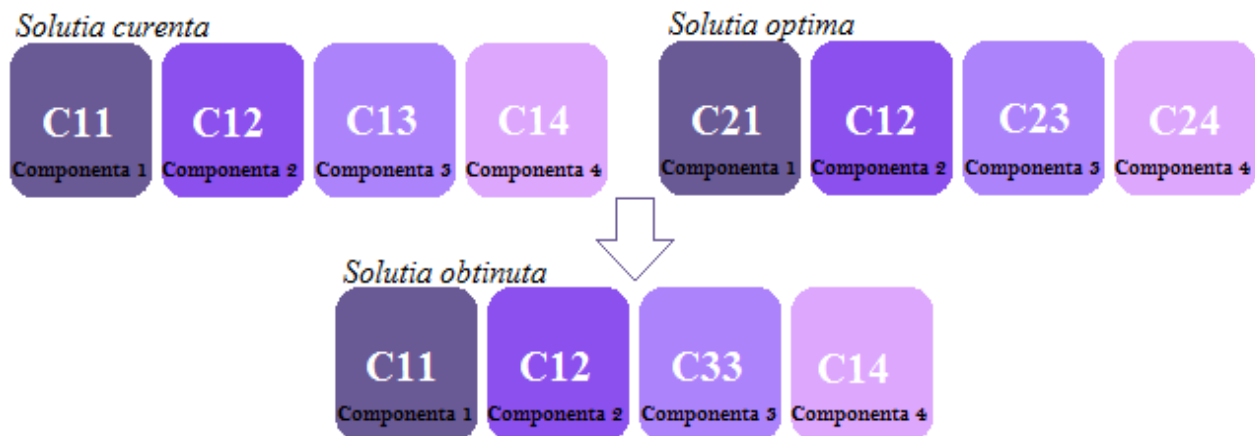


Figura 5.1 Aplicarea operatorului de mutație

### 5.1.3. Componenta bazată pe Tabu Search

Tabu Search, una dintre cele mai cunoscute metaeuristici, utilizează memoria adaptivă cu scopul de a ghida căutarea locală euristică în procesul de explorarea a spațiului de soluții dincolo de optimalitatea locală. Astfel, căutarea soluțiilor se realizează într-un mod mai eficient și mai economic [16].

Insirația biologică a componentei bazate pe Tabu Search e dată de utilizarea memoriei, adică folosirea experiențelor din trecut pentru a lua decizii în prezent. Această componentă a fost introdusă pentru a evita stagnarea și pentru a ajuta la dezvoltarea unei soluții cât mai apropiate de optim.

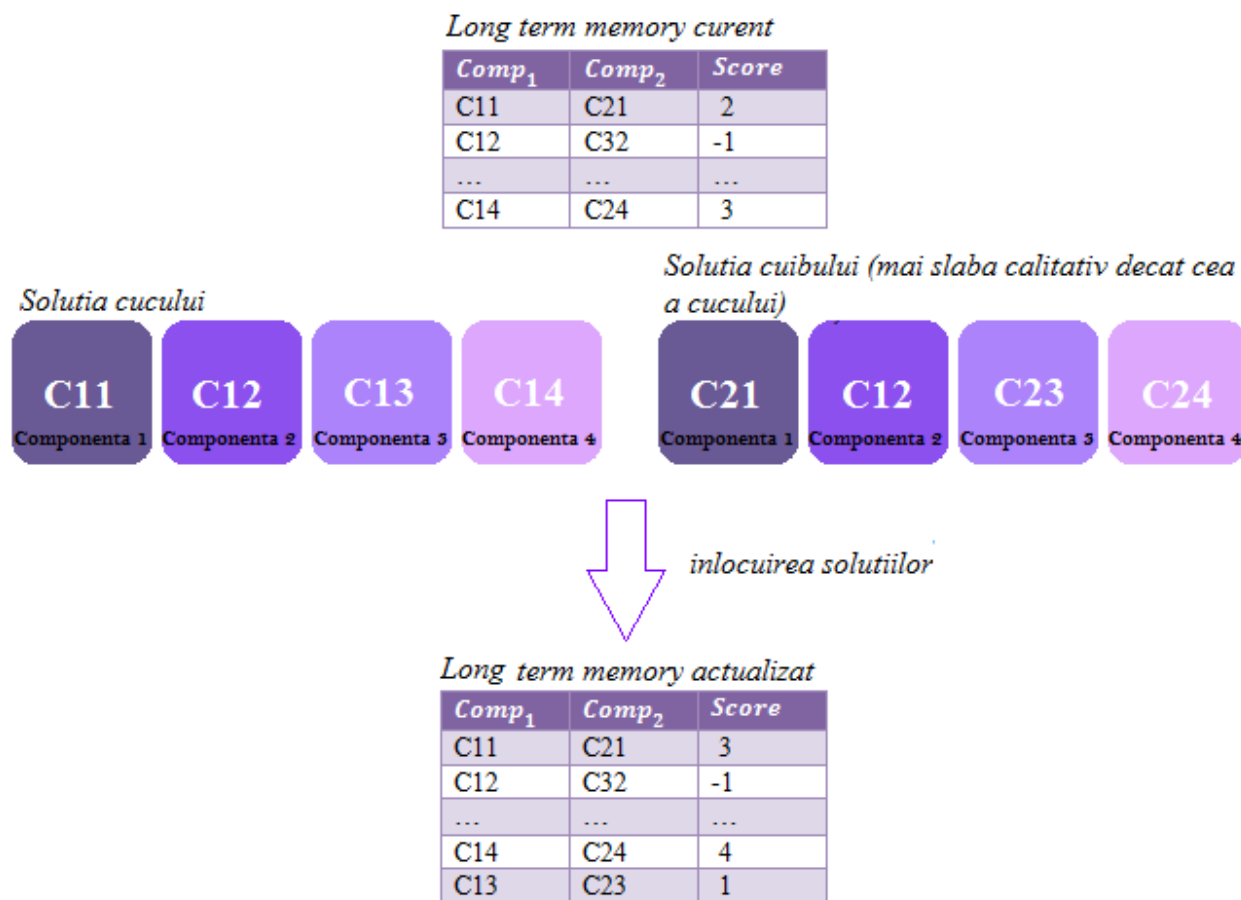
Există două tipuri de memorii Tabu: memorie de scurtă durată, respectiv memorie de lungă durată (memorie de învățare). Principiul de bază al memoriei de scurtă durată presupune întrezicerea utilizării anumitor elemente prin marcarea lor ca fiind tabu. Memoria de învățare are ca sursă de inspirație sistemele neuronale, unde agenții învață în funcție de consecințele produse de acțiunile lor. Pe măsura ce agenții explorează mediul înconjurător, baza de cunoștințe se construiește treptat, efectele acțiunilor agenților fiind cele mai relevante în acest proces. Dacă efectul unei acțiuni este unul pozitiv, în viitor agentul va presupune că aceea este o cale sigură. Dacă efectul este unul negativ, agentul va fi încurajat să încerce alte alternative.

Tehnica hibridă este influențată de componenta Tabu în două puncte: în etapa de alegerea a unui cuib de către cuc, respectiv în etapa de îmbunătățire a oului de cuc prin intermediul operatorului genetic.

Memoria de scurtă durată este consultată în momentul în care cucul alege în mod aleator un cuib pentru a-și depune oul. Această memorie conține tuple de tipul (*sol*, *tabuSize*), unde *sol* reprezintă un cuib care a fost ales anterior, iar *tabuSize* simbolizează numărul de iterații în care cuibul nu mai poate fi ales. Numărul de iterații este setat în mod dinamic în funcție de necesitatea de explorare sau exploatare. Numărul de iterații este mărit atunci când e necesară explorarea și micșorat atunci când este necesară exploatarea.

Memoria de lungă durată este consultată atunci când sunt efectuate mutații asupra unei soluții ce reprezintă oul de cuc, cu scopul îmbunătățirii acesteia. Această memorie conține tuple de tipul (*comp*<sub>1</sub>, *comp*<sub>2</sub>, *score*), *comp*<sub>1</sub> și *comp*<sub>2</sub> sunt componentele unor soluții. *Score* reprezintă valoarea obținută prin înlocuirea componentei *comp*<sub>1</sub> cu *comp*<sub>2</sub>. Atunci când se dorește înlocuirea unei componente din soluție cu o componentă aleatoare este consultată tabela memoriei de lungă durată. Dacă această tabelă conține tuple în care *comp*<sub>1</sub> este componenta pe care dorim să o înlocuim, se alege tupla care are scorul cel mai mare. În acest caz *comp*<sub>1</sub> va fi înlocuită cu *comp*<sub>2</sub>, cu condiția ca scorul să fie pozitiv. Dacă tabela nu conține tuple de forma celor căutate sau scorul este negativ, *comp*<sub>1</sub> va fi înlocuită cu o componentă aleasă în mod aleator din spațiul de căutare.

Memoria de învățare este actualizată în momentul în care un cuc dorește să își depună oul într-un cuib. Dacă soluția cucului este mai bună calitativ decât soluția cuibului, cucul își depune oul în cuib (soluția cuibului este înlocuită cu cea a cucului), iar scorul corespunzător componentelor înlocuite este incrementat. În caz contrar, oul nu este depus în cuib, iar scorul este decrementat. Figura 5.2 ilustrează modul în care este actualizată tabela memoriei de învățare în cazul în care soluția cucului este mai bună decât soluția cuibului în care se dorește depunerea oului.



**Figura 5.2 Actualizarea memoriei tabu de învățare în urma înlocuirii soluției cuibului cu soluția cucului**

## 5.2. Evoluția algoritmului hibrid inspirat din comportamentul cucilor

Algoritmul propus inspirat din comportamentul de reproducere al cucilor și hibridizat prin intermediul operatorului genetic și al elementelor de căutare tabu determină obținerea unei soluții optime sau aproape optime. Dezvoltarea tehnicii hibride a avut loc treptat, pornind de la algoritmul clasic inspirat din comportamentul cucilor (Algoritm\_1). Algoritmul clasic a fost îmbunătățit prin modificarea metodei de actualizare a soluțiilor cucilor (Algoritm\_2). Apoi a fost dezvoltată o nouă versiune la care s-a utilizat operatorul genetic (Algoritm\_3), iar la final au fost utilizate memoria de învățare, respectiv memoria tabu de scurtă durată, rezultând un nou algoritm (Algoritm\_4).

Toate cele patru versiuni ale algoritmilor utilizează următorii parametri de intrare comuni:

- *foodOffers*: totalitatea ofertelor alimentare din spațiul de căutare;
- *userRequest*: preferințele alimentare ale utilizatorului;
- *dietRecommendations*: recomandările nutriționale ale utilizatorului;
- *noNest*: numărul de cuiburi disponibile;
- *noCuck*: numărul de cucii utilizați;
- *repNest*: procentul de cuiburi înlocuite cu unele generate aleator;

- *trMut*: pragul de mutație prestabilit;
- *noIt*: numărul maxim de iterații;
- *diffIt*: diferența de fitness între două iterații ca acestea să poate fi considerate stagnări;
- *noConstIt*: numărul maxim de stragnări succesive;

Algoritmul are un singur element de ieșire: *menus* – meniurile recomandate în urma executării algoritmului.

### 5.2.1. Algoritmul clasic inspirat din comportamentul cucilor

Acest algoritm e compus din două mari etape: etapa de inițializare și cea iterativă. În prima etapă sunt asociate soluții cuiburilor și cucilor în mod aleator. Algoritmul continuă cu etapa iterativă atât timp cât numărul maxim de iterații sau numărul maxim de stagnări succesive nu sunt atinse. În această etapă sunt executați următorii pași:

**Pasul 1.** Un cuc este selectat în mod aleator din mulțimea cucilor *Cuckoos*;

**Pasul 2.** Soluția asociată cucului este actualizată prin aplicarea în mod aleator a mutațiilor asupra soluției cucului curent. Mutațiile sunt aplicate astfel: pentru fiecare componentă din cadrul soluției este generat un număr aleator în intervalul  $[0,1]$  și dacă acest număr este mai mare decât pragul prestabilit *trMut* atunci este executată o mutație aleatoare asupra acestei componente;

Algorithm_1: Cuckoo Search (CS)	
1.	<b>Input:</b> <i>foodOffers, userRequest, dietRecommendations, noNest, noCuck, repNest, trMut, noIt, diffIt, noConstIt</i>
2.	<b>Output:</b> <i>menu</i>
3.	<b>begin</b>
4.	<i>Nests</i> = <b>Initialize_Nests</b> ( <i>noNest, foodOffers, userRequest, dietRecommendations, noNest</i> )
5.	<i>Cuckoos</i> = <b>Initialize_Cuckoos</b> ( <i>noCuck, foodOffers, userRequest, dietRecommendations, noNest</i> )
6.	<i>sol<sub>opt</sub></i> = <b>Get_Best_Solution</b> ( <i>Nests</i> );
7.	<i>iteration</i> = 1
8.	<b>while</b> ( <b>Stop_Condition_Not_Reached</b> ( <i>iteration, diffIt, noConstIt</i> )) <b>do</b>
9.	<i>cuckoo</i> = <b>Select_Cuckoo_Randomly</b> ( <i>Cuckoos</i> )
10.	<i>cuckoo</i> = <b>Update_Cuckoo_Solution</b> ( <i>trMut</i> )
11.	<i>nest</i> = <b>Select_Nest_Randomly</b> ( <i>Nests</i> )
12.	<b>if</b> ( <b>Fitness</b> ( <i>cuckoo.solution</i> ) > <b>Fitness</b> ( <i>nest.solution</i> )) <b>then</b>
13.	<i>nest</i> = <b>Replace_Nest_Solution</b> ( <i>cuckoo.solution</i> )
14.	<b>end if</b>
15.	<i>Nests</i> = <b>Replace_Worst_Nests</b> ( <i>Nests, repNest, foodOffers, userRequest, dietRecommendations</i> )
16.	<i>sol<sub>opt</sub></i> = <b>Get_Best_Solution</b> ( <i>Nests</i> )
17.	<i>iteration</i> = <i>iteration</i> + 1
18.	<b>end while</b>
19.	<b>return</b> <i>sol<sub>opt</sub></i>
20.	<b>end</b>

Figura 5.2 Algorithm\_1 – pseudocod



**Pasul 3.** Se alege în mod aleator un cuib din mulțimea cuiburilor *Nests*;

**Pasul 4.** Dacă soluția cucului este mai bună calitativ decât soluția cuibului atunci soluția cuibului este înlocuită de soluția cucului;

**Pasul 5.** Un procent *repNest* de cuiburi care au cele mai slabe soluții calitativ sunt înlocuite de cuiburi cu soluții generate aleator;

**Pasul 6.** Soluția optimă este actualizată dacă este cazul.

Pseudocodul acestui algoritm poate fi observat în figura 5.3.

### 5.2.2. Algoritm clasic inspirat din comportamentul cucilor îmbunătățit cu o nouă procedură pentru actualizarea soluțiilor cucilor

Algoritmul clasic inspirat din comportamentul cucilor a fost îmbunătățit prin modificarea procedurii de actualizare a soluțiilor cucilor. Atât etapa de inițializare, cât și cea iterativă au fost modificate. În etapa inițială sunt asociate soluții cuiburilor în mod aleator, apoi mulțimea cuiburilor e ordonată descrescător în funcție de calitatea soluțiilor lor. În această etapă nu sunt asociate soluții cucilor. La fel ca algoritmul clasic și acesta continuă cu etapa iterativă atât timp cât numărul maxim de iterații sau numărul maxim de stagnări succesive nu sunt atinse. În această etapă sunt executați următorii pași:

**Pasul 1.** Pentru fiecare cuc din populația cucilor sunt parcurse următoarele etape:

- Se alege în mod aleator un cuib *nest* din mulțimea cuiburilor *Nests*;
- Cât timp mai există cuiburi care pot fi procesate în intervalul  $[(\text{indexul soluției } nest \text{ în mulțimea } Nests) + 1, |Nests| - 1]$  (elementele din mulțimea *Nests* sunt ordonate descrescător în funcție de valoarea funcției de fitness) se execută următoarele operații:
  - **Se generează o nouă soluție:** cucul artificial va genera o nouă soluție  $sol_c$  pe baza soluției asociate cuibului selectat *nest*. Noua soluție va fi generată prin aplicarea mutațiilor în mod aleator asupra cuibului selectat, după cum urmează: pentru fiecare componentă din cadrul soluției asociate cuibului este generat un număr aleator în intervalul  $[0,1]$  și dacă acest număr este mai mare decât pragul prestabilit *trMut* atunci este executată o mutație aleatoare asupra acestei componente;
  - **Depunerea soluției:** cucul va selecta aleator un cuib cu indexul în intervalul  $[(\text{indexul soluției } nest \text{ în mulțimea } Nests) + 1, |Nests| - 1]$  cu scopul de a-și depune soluția. Apoi, variabila *nest* e actualizată cu un nou cuib selectat. Dacă soluția cucului are o funcție de fitness mai bună decât soluția cuibului selectat atunci cuibul selectat va fi înlocuit cu soluția cucului. Acest proces este similar cu depunerea unui ou în cuib de către cuc.

**Pasul 2.** Un procent *repNest* de cuiburi care au cele mai slabe soluții calitativ sunt înlocuite de cuiburi cu soluții generate aleator;

**Pasul 3.** Mulțimea cuiburilor e ordonată descrescător în funcție de calitatea soluțiilor lor;

**Pasul 4.** Soluția optimă este actualizată dacă este cazul.

Pseudocodul acestui algoritm poate fi observat în figura 5.4.

<b>Algorithm_2: Improved Cuckoo Search (CSU)</b>	
1.	<b>Input:</b> <i>foodOffers, userRequest, dietRecommendations, noNest, noCuck, repNest, trMut, noIt, diffIt, noConstIt</i>
2.	<b>Output:</b> <i>menu</i>
3.	<b>begin</b>
4.	<i>Nests</i> = <b>Initialize_Nests</b> ( <i>noNest, foodOffers, userRequest, dietRecommendations, noNest</i> )
5.	<i>Nests</i> = <b>Sort_Descending</b> ( <i>Nests</i> )
6.	<i>iteration</i> = 1
7.	<b>while</b> ( <b>Stop_Condition_Not_Reached</b> ( <i>iteration, diffIt, noConstIt</i> )) <b>do</b>
8.	<b>foreach</b> <i>cuckoo</i> <b>in</b> <i>Cuckoos</i> <b>do</b>
9.	<i>nest</i> = <b>Select_Nest_Randomly</b> ( <i>Nests</i> )
10.	<b>while</b> ( <b>Nest_to_Process</b> ( <i>nest, Nests</i> )) <b>do</b>
11.	<i>cuckoo</i> = <b>Generate_Solution_for_Cuckoo</b> ( <i>nest</i> )
12.	<i>nest</i> = <b>Select_Nest_Randomly</b> ( <i>Nests</i> )
13.	<b>if</b> ( <b>Fitness</b> ( <i>cuckoo.solution</i> ) > <b>Fitness</b> ( <i>nest.solution</i> )) <b>then</b>
14.	<i>nest</i> = <b>Replace_Nest_Solution</b> ( <i>cuckoo.solution</i> )
15.	<b>end if</b>
16.	<b>end while</b>
17.	<b>end foreach</b>
18.	<i>Nests</i> = <b>Replace_Worst_Nests</b> ( <i>Nests, repNest, foodOffers, userRequest, dietRecommendations</i> )
19.	<i>Nests</i> = <b>Sort_Descending</b> ( <i>Nests</i> )
20.	<i>sol<sub>opt</sub></i> = <b>Get_Best_Solution</b> ( <i>Nests</i> )
21.	<i>iteration</i> = <i>iteration</i> + 1
22.	<b>end while</b>
23.	<b>return</b> <i>sol<sub>opt</sub></i>
24.	<b>end</b>

Figura 5.4 Algoritm\_2 – pseudocod

### 5.2.3. Algoritmul hibrid inspirat din comportamentul cucilor și operatorul genetic

Algoritmul hibrid inspirat din comportamentul cucilor și algoritmi genetici are la bază ideile menționate în secțiunea anterioară 5.1.2. Etapa de inițializare coincide cu cea a algoritmului anterior (Algoritm\_2), la fel și condiția de oprire. Etapa iterativă este asemănătoare cu cea a algoritmului anterior, singura diferență fiind la etapa de generare a unei noi soluții din pasul 1. În cadrul acestui algoritm această etapă se execută astfel: pentru fiecare componentă din soluție corespunzătoare cucului se execută următoarele operații:

- componenta curentă este comparată cu componenta corespunzătoare care aparține soluției optime găsite până în acel moment;
- dacă cele două componente sunt identice nu se execută nicio modificare;
- dacă cele două componente diferă se execută următorii pași:
  - se generează un număr aleator  $x \in [0,1]$ ;
  - dacă  $x < trMut$  atunci nu se execută nicio modificare;

- dacă  $x \geq trMut$  atunci atunci este executată o mutație aleatoare asupra acestei componente.

Pseudocodul acestui algoritm poate fi observat în figura 5.5.

<b>Algorithm_3: Improved Cuckoo Search With Random Genetic Operator (CSURG)</b>	
1.	<b>Input:</b> <i>foodOffers, userRequest, dietRecommendations, noNest, noCuck, repNest, trMut, noIt, diffIt, noConstIt</i>
2.	<b>Output:</b> <i>menu</i>
3.	<b>begin</b>
4.	<i>Nests</i> = <b>Initialize_Nests</b> ( <i>noNest, foodOffers, userRequest, dietRecommendations, noNest</i> )
5.	<i>Nests</i> = <b>Sort_Descending</b> ( <i>Nests</i> )
6.	<i>iteration</i> = 1
7.	<b>while</b> ( <b>Stop_Condition_Not_Reached</b> ( <i>iteration, diffIt, noConstIt</i> )) <b>do</b>
8.	<b>foreach</b> <i>cuckoo</i> <b>in</b> <i>Cuckoos</i> <b>do</b>
9.	<i>nest</i> = <b>Select_Nest_Randomly</b> ( <i>Nests</i> )
10.	<b>while</b> ( <b>Nest_to_Process</b> ( <i>nest, Nests</i> )) <b>do</b>
11.	<i>cuckoo</i> = <b>Generate_Solution_for_Cuckoo</b> ( <i>nest, sol<sub>opt</sub></i> )
12.	<i>nest</i> = <b>Select_Nest_Randomly</b> ( <i>Nests</i> )
13.	<b>if</b> ( <b>Fitness</b> ( <i>cuckoo.solution</i> ) > <b>Fitness</b> ( <i>nest.solution</i> )) <b>then</b>
14.	<i>nest</i> = <b>Replace_Nest_Solution</b> ( <i>cuckoo.solution</i> )
15.	<b>end if</b>
16.	<b>end while</b>
17.	<b>end foreach</b>
18.	<i>Nests</i> = <b>Replace_Worst_Nests</b> ( <i>Nests, repNest, foodOffers, userRequest, dietRecommendations</i> )
19.	<i>Nests</i> = <b>Sort_Descending</b> ( <i>Nests</i> )
20.	<i>sol<sub>opt</sub></i> = <b>Get_Best_Solution</b> ( <i>Nests</i> )
21.	<i>iteration</i> = <i>iteration</i> + 1
22.	<b>end while</b>
23.	<b>return</b> <i>sol<sub>opt</sub></i>
24.	<b>end</b>

Figura 5.5 Algoritm\_3 – pseudocod

#### 5.2.4. Algoritmul hibrid inspirat din comportamentul cucilor, operatorul genetic și structuri de memorie

Algoritmul hibrid inspirat din comportamentul cucilor, algoritmi genetici și structuri de memorie tabu are la bază ideile menționate în secțiunea anterioară 5.1.3. Pe lângă datele de intrare utilizate de celelalte variante ale algoritmului, acesta folosește și parametrul  $it_{tabu}$ : dimensiunea memoriei tabu de scurtă durată (numărul de iterații în care un cuib este tabu).

În comparație cu etapa de inițializare a algoritmului anterior (Algoritm\_3), etapa acestui algoritm conține și operația de inițializare a celor două memorii tabu: cea de scurtă durată, respectiv cea de învățare. Condiția de oprire coincide cu cea a algoritmului anterior (Algoritm\_3).

**Algorithm\_4:** Improved Cuckoo Search With Random Genetic Operator And Memory Structures (CSUCGM)

```

1. Input: foodOffers, userRequest, dietRecommendations, noNest, noCuck, repNest, trMut,
   noIt, diffIt, noConstIt, ittabu
2. Output: menu
3. begin
4.   Nests = Initialize_Nests(noNest, foodOffers, userRequest, dietRecommendations,
   noNest)
5.   Nests = Sort_Descending (Nests)
6.    $M_t = M_l = \phi$ 
7.   iteration = 1
8.   while (Stop_Condition_Not_Reached(iteration, diffIt, noConstIt)) do
9.     foreach cuckoo in Cuckoos do
10.      nest = Select_Nest_Randomly(Nests)
11.       $M_t = \text{Update\_TMemory}(M_t, nest)$ 
12.      while (Nest_to_Process(nest, Nests)) do
13.         $sol_c = \text{Generate\_Solution\_for\_Cuckoo}(nest, sol_{opt}, M_l)$ 
14.        nest = Select_Nest_Randomly(Nests)
15.        if (Fitness( $sol_c.solution$ ) > Fitness(nest.solution)) then
16.           $M_l = \text{Reward}(sol_c, nest, M_l)$ 
17.          nest = Replace_Nest_Solution(cuckoo.solution)
18.        else
19.           $M_l = \text{Penalize}(sol_c, nest, M_l)$ 
20.        end if
21.      end while
22.    end foreach
23.    Nests = Replace_Worst_Nests(Nests, repNest, foodOffers, userRequest,
    dietRecommendations)
24.    Nests = Sort_Descending (Nests)
25.     $sol_{opt} = \text{Get\_Best\_Solution}(Nests)$ 
26.     $M_t = \text{Reset\_Tabu\_Memory}(M_t)$ 
27.     $M_l = \text{Reset\_Learning\_Memory}(M_l)$ 
28.    iteration = iteration + 1
29.  end while
30.  return  $sol_{opt}$ 
31. end
    
```

Figura 5.6 Algoritm\_4 – pseudocod

În cadrul acestui algoritm se execută următorii pași iterativi:

**Pasul 1.** Pentru fiecare cuc din populația cucilor sunt parcurse următoarele etape:

- Se alege în mod aleator un cuib *nest* din mulțimea cuiburilor *Nests* astfel încât acesta să nu fie tabu. Cuibul ales devine tabu și va fi înregistrat în memoria tabu de scurtă durată. Această memorie este utilizată doar atunci când un cuc selectează primul cuib;

- Cât timp mai există cuiburi care pot fi procesate în intervalul  $[(\text{indexul soluției } nest \text{ în mulțimea } Nests) + 1, |Nests| - 1]$  (elementele din mulțimea  $Nests$  sunt ordonate descrescător în funcție de valoarea funcției de fitness) se execută următoarele operații:

- **Se generează o nouă soluție:** cucul artificial va genera o nouă soluție  $sol_c$  pe baza soluției asociate cuibului selectat  $nest$ . Noua soluție va fi generată la fel ca la algoritmul anterior, dar atunci când se dorește aplicarea unei mutații aleatoare va fi consultată tabela memoriei de învățare așa cum a fost prezentat în secțiunea 5.1.3;
- **Depunerea soluției:** cucul va selecta aleator un cuib cu indexul în intervalul  $[(\text{indexul soluției } nest \text{ în mulțimea } Nests) + 1, |Nests| - 1]$  cu scopul de a-și depune soluția. Apoi, variabila  $nest$  e actualizată cu un nou cuib selectat. Dacă soluția cucului are o funcție de fitness mai bună decât soluția cuibului selectat atunci cuibul selectat va fi înlocuit cu soluția cucului. În cadrul acestei etape va fi actualizată memoria de învățare conform prezentării din secțiunea 5.1.3;

**Pasul 2.** Înlocuirea unor cuiburi - analog algoritmului anterior;

**Pasul 3.** Ordonarea cuiburilor - analog algoritmului anterior;

**Pasul 4.** Actualizarea soluției optime - analog algoritmului anterior;

**Pasul 5.** Memoria tabu și cea de învățare sunt resetate cu scopul de a evita stagnarea.

Pseudocodul acestui algoritm poate fi observat în figura 5.6.

## 6. Proiectare și implementare

Capitolul curent descrie aplicație web distribuită ce a fost dezvoltată pentru testarea tehnicii hibride. Această permite vizualizarea atât a recomandărilor alimentare, cât și a unui grafic care prezintă evoluția în timp a celor mai bune soluții.

### 6.1. Proiectarea prototipului experimental

Proiectarea sistemului presupune descrierea de design și analiză a aplicației, prin intermediul diagramelor UML, precum și prin descrierea acestora în detaliu.

#### 6.1.1. Arhitectura conceptuală

Pentru a valida metoda de optimizare bazată pe Hybrid Cuckoo Search a fost dezvoltat și utilizat un prototip experimental (figura 6.1).

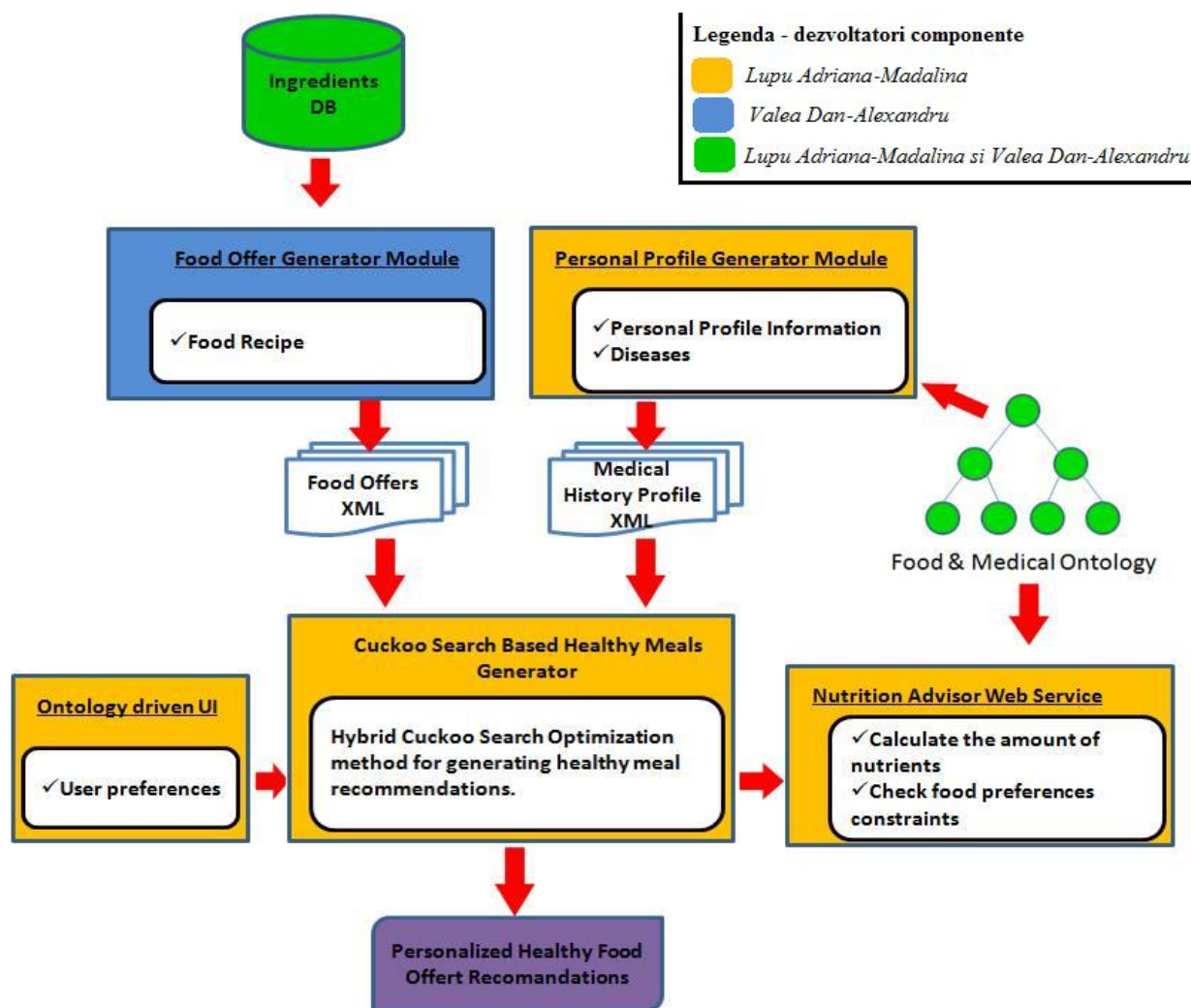


Figura 6.1 Arhitectura conceptuală a prototipului experimental

Prototipul experimental este compus din următoarele module și resurse: Interfață Grafică Bazată pe Ontologie, Ontologie Alimentară și Medicală, Generator de Profile Personale, Bază de Date a Ingredientelor Alimentare, Generator de Oferte Alimentare, Serviciu Web de Consultanță Nutrițională și Generator de Meniuri Alimentare Bazat pe Cuckoo Search.

Interfață Grafică Bazată pe Ontologie are rolul de a-l ghida pe utilizator în procesul de specificare a preferințelor alimentare. Acesta poate selecta categoria de alimente pe care dorește să o consume, precum și categoria pe care dorește să nu o consume. Categoriile alimentare sunt furnizate de ontologie. De asemenea, utilizatorul poate specifica și nivelul său de activitate pentru ziua respectivă.

Ontologia conține informații referitoare la diverse tipuri de mâncare, profilul personal și afecțiunile medicale ale utilizatorilor. Pentru fiecare fel de mâncare sunt specificate toate ingredientele conținute și pentru fiecare ingredient este prezentată cantitatea de calorii, lipide, carbohidrați, proteine, vitamine și minerale. Profilul personal al pacientului conține informații legate de nume, prenume, data nașterii, sex, înălțime, greutate, precum și afecțiunile medicale ale acestuia. Pentru fiecare afecțiune medicală sunt specificate anumite recomandări nutriționale, obținute din [17,18]. În figura 6.2 se poate observa faptul că supa de pui cu găluște conține opt ingredient: carne grasă de pui, morcovi, rădăcină de pătrunjel, rădăcină de țelina, ceapă uscată, griș, ouă de gaină și piper negru. Această supă aparține următoarelor categorii: carne de pui, griș și amestec de legume. Figura 6.3 ilustrează valorile nutritive ale morcovului. Acesta conține 45kcal, 0.3g lipide, 8.8g carbohidrați, 1.5g protein, 100mg sodiu, 1mg fier, 65mg calciu, 7.9mg vitamina A, 0.05mg vitamina B1, 4mg vitamina C pe 100 grame.

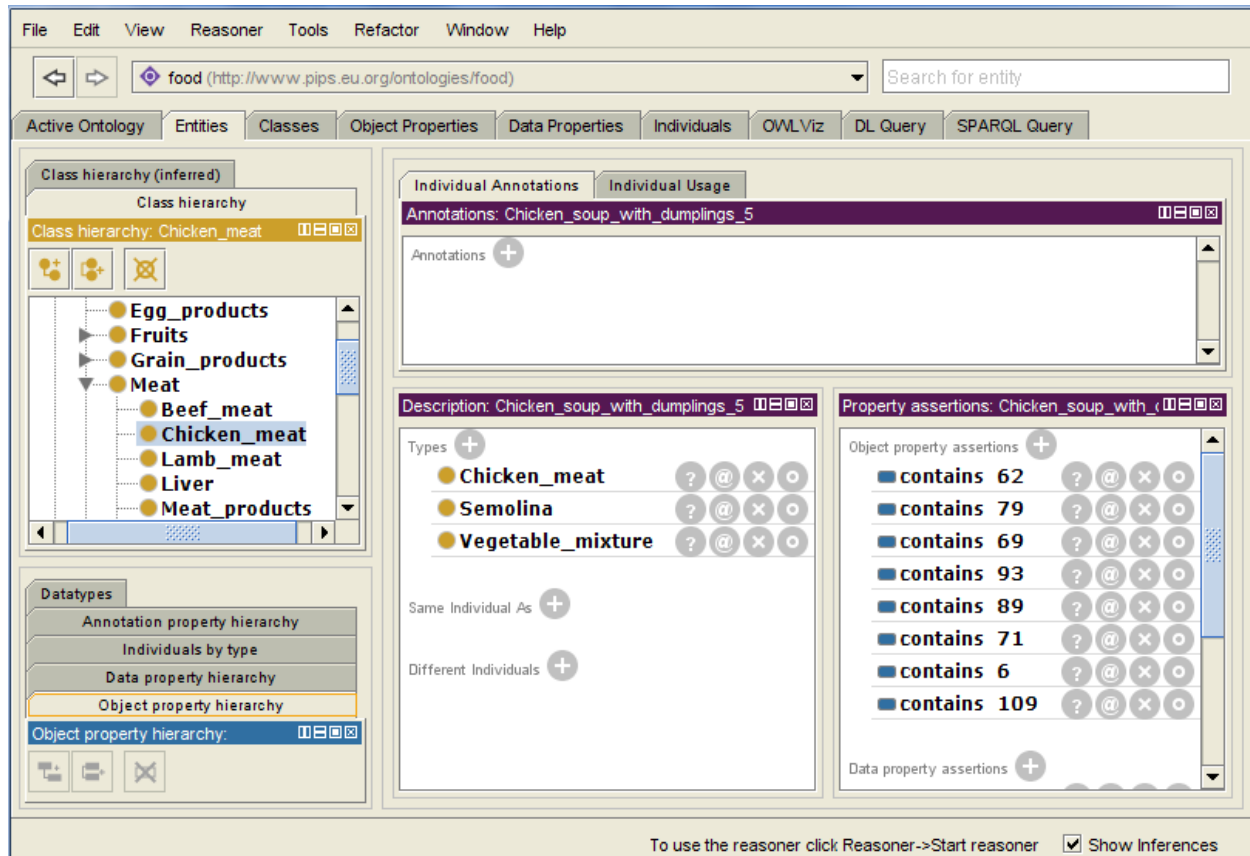


Figura 6.2 Exemplu din ontologia alimentară ce descrie supa de pui cu găluște

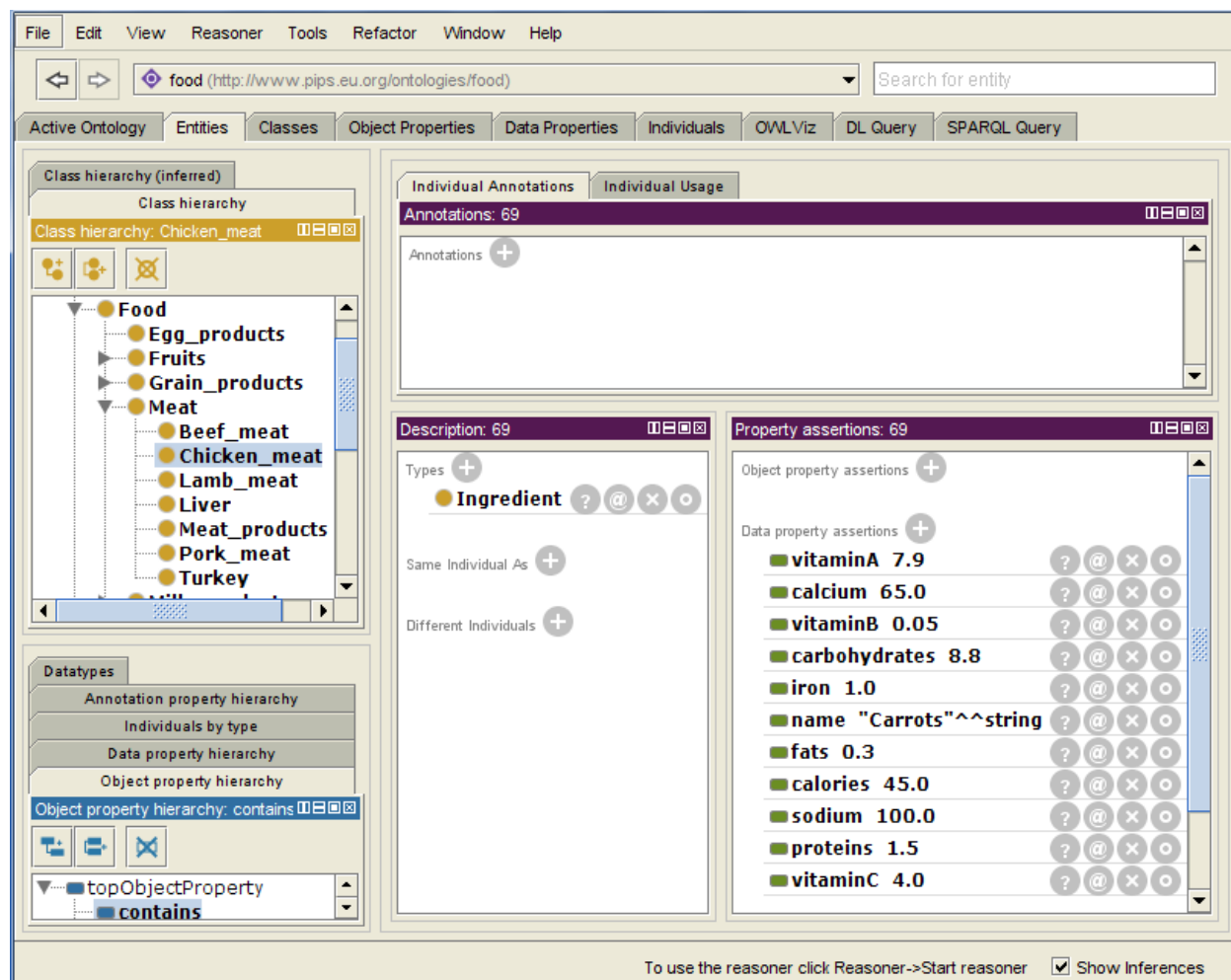


Figura 6.3 Exemplu din ontologia alimentară ce morcovul conținut în supă de pui cu găluște

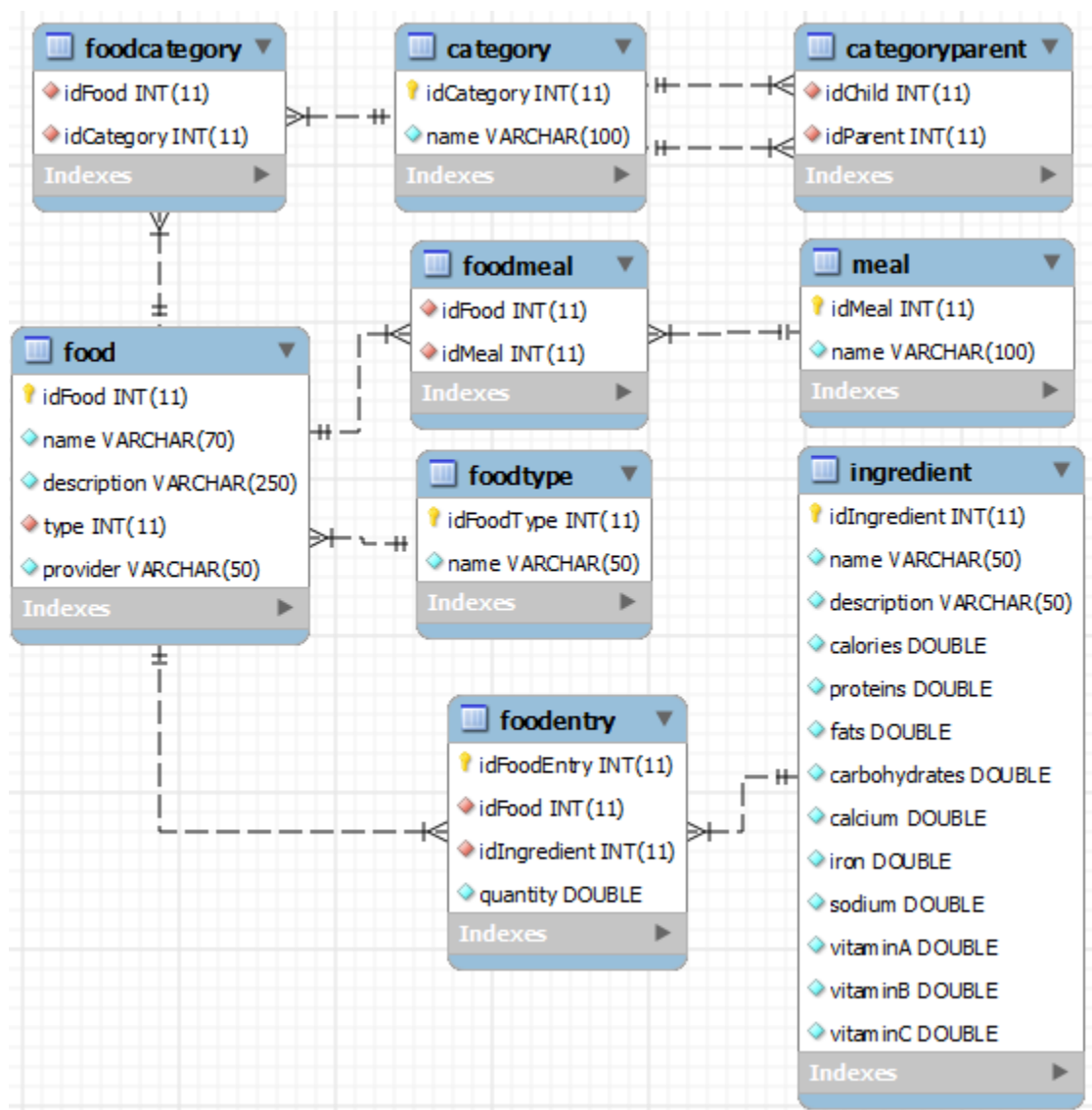
Modulul de generare de profile personale crează profilul personal al unei persoane pe baza informațiilor aflate în ontologie. Profilul personal al unei persoane este stocat într-un fișier XML și conține informații personale ale pacientului, precum și informații referitoare la afecțiunile medicale ale acestuia. Un exemplu de astfel de profil este ilustrat în figura 6.4.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<patient_profile>
  <first_name>Andrei</first_name>
  <last_name>Enescu</last_name>
  <gender>male</gender>
  <date_of_birth>1945-02-08</date_of_birth>
  <height>165.0</height>
  <weight>70.0</weight>
  <diseases>
    <disease>Diabetes Type I</disease>
    <disease>Hypertension</disease>
  </diseases>
</patient_profile>
```

Figura 6.4 Exemplu de profil medical



Bază de Date a Ingredientelor Alimentare conține informații referitoare la tipul mâncărilor și rețetele acestora (ingredientele necesare și cantitatea necesară pentru fiecare ingredient). Rețetele au fost realizate pe baza informațiilor furnizate în [19]. De asemenea, această bază de date conține și informații nutriționale pentru fiecare ingredient preluate din [14]. Diagrama bazei de date este ilustrată în figura 6.5.



**Figura 6.5** Diagrama bazei de date alimentare

Generatorul de Oferte Alimentare crează fișiere cu oferte alimentare în format XML. Aceste oferte conțin informații referitoare la numele mâncării, furnizorul acesteia, precum și ingredientele conținute împreună cu cantitățile corespunzătoare. Informațiile alimentare conținute în fișiere sunt preluate din baza de date alimentară. În figura 6.6 este ilustrat un exemplu de ofertă alimentară de ciorbă de crap.

```

<?xml version="1.0" encoding="UTF-8"?>
<food>
  <name>Carp sour soup</name>
  <provider>Provider_33</provider>
  <meal>Lunch</meal>
  <type>Soup</type>
  <price>4,45</price>
  <delive_time>63</delive_time>
  <ingredients>
    <ingredient>
      <name>Carp</name>
      <quantity>80</quantity>
    </ingredient>
    <ingredient>
      <name>Dried onion</name>
      <quantity>15</quantity>
    </ingredient>
    <ingredient>
      <name>Carrots</name>
      <quantity>11</quantity>
    </ingredient>
    <ingredient>
      <name>Tomatoes</name>
      <quantity>45</quantity>
    </ingredient>
    <ingredient>
      <name>Parsley roots</name>
      <quantity>25</quantity>
    </ingredient>
    <ingredient>
      <name>Lemons</name>
      <quantity>3</quantity>
    </ingredient>
  </ingredients>
</food>

```

**Figura 6.6 Exemplu de ofertă alimentară**

Serviciul Web de Consultanță Nutrițională calculează cantitatea de calorii și nutrienți conținute într-un set de oferte alimentare. Acesta consultă ontologia alimentară pentru a prelua informațiile nutriționale ale ingredientelor. Prin intermediul acestui serviciu web se poate verifica dacă un set de oferte alimentare respectă preferințele utilizatorului, adică dacă mâncărurile aparțin categoriilor de alimente corespunzătoare. De asemenea, această componentă furnizează recomandările nutriționale pentru diverse afecțiuni.

Generator de Meniuri Alimentare Bazat pe Cuckoo Search furnizează recomandări personalizate de meniuri sănătoase folosind o tehnică hibridă de optimizare bazată pe comportamentul de reproducere al cucilor. Pentru generarea meniurilor acesta folosește preferințele alimentare ale utilizatorului furnizate prin intermediul interfeței grafice, ofertele alimentare disponibile, profilul personal al utilizatorului, precum și informațiile oferite de serviciul web.

### 6.1.2. Use Case Model

Modelul cazurilor de utilizare al acestui sistem conține un singur caz de utilizare: generarea de meniuri alimentare. Acest caz de utilizare conține și partea de autentificare. Actorul acestui caz de utilizare este orice utilizator obișnuit. Precondiția acestui caz de utilizare presupune că utilizatorul nu trebuie să fie deja autentificat. Se vor parcurge următorii pași:

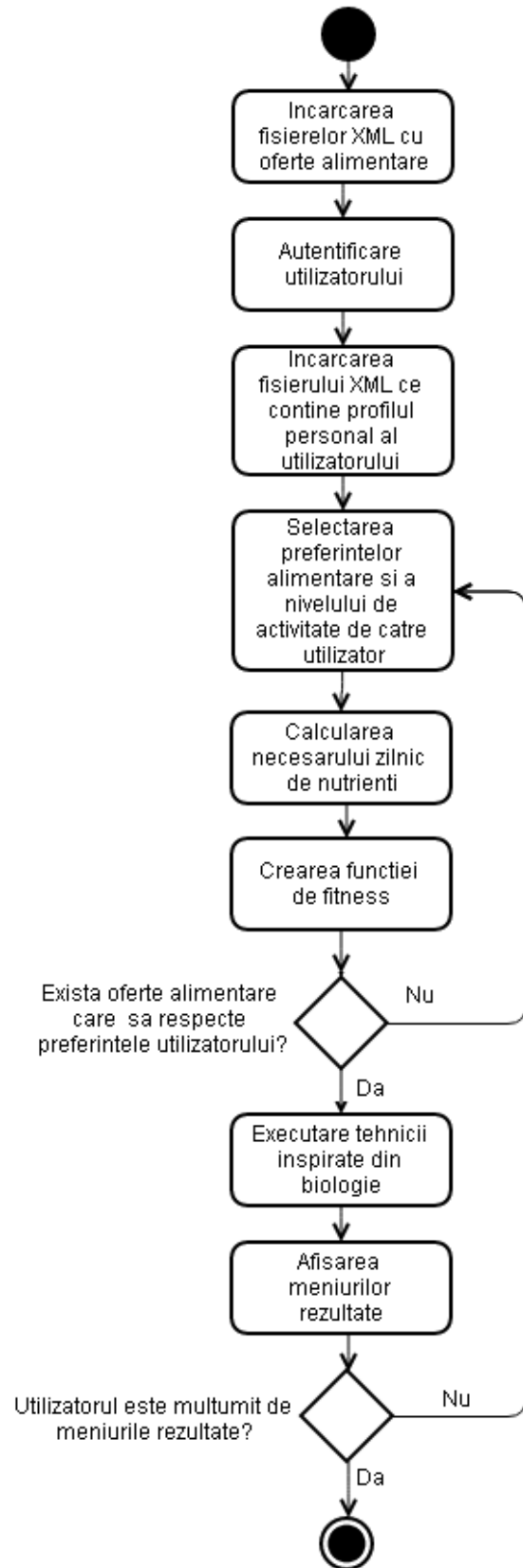
1. Utilizatorul introduce username-ul și parola;
2. Sistemul validează credențiale și afișează pagina principală a aplicației ce conține profilul personal al utilizatorului;
  - 2a. Credențiale invalide:
    1. Sistemul afișează un mesaj care indică faptul că username-ul sau parola sunt incorecte și sugerează reintroducerea acestora;
    2. Utilizatorul continuă cu pasul 1;
3. Utilizatorul selectează subcategoriile și/sau categoriile de alimente pe care dorește să le consume și pe cele pe care dorește să nu le consume, precum și nivelul de activitate pentru ziua respectivă;
4. Sistemul afișează un set de meniuri personalizate recomandate și informațiile nutriționale ale acestora;
  - 4a. Preferințele alimentare ale utilizatorului sau nivelul acestuia de activitate nu sunt corespunzătoare cu ofertele existente:
    - Sistemul afișează un mesaj care sugerează schimbarea preferințelor sau a nivelului de activitate;
    - Utilizatorul continuă cu pasul 3;

### 6.1.3. Diagrama de activitate

Diagrama de activitate evidențiază fluxul programului de la punctul de start, până la cel final. În primă etapă sunt încărcate toate fișierele ce conțin ofertele ce conțin oferte alimentare. Această etapă este urmată de autentificarea utilizatorului și încărcarea fișierului ce conține profilul personal al acestuia. Apoi sunt selectate de către utilizator preferințele alimentare și nivelul de activitate. Pe baza profilului personal și a nivelului de activitate sunt calculate dozele zilnice necesare de nutrienți. Pentru a putea ordona soluțiile alimentare este creată funcția de fitness. Dacă ofertele alimentare nu respectă preferințele utilizatorului se afișează un mesaj de eroare și utilizatorul este rugat să revină la selectarea preferințelor sau a nivelului de activitate. Altfel este executată tehnica hibridă inspirată din biologie și sunt afișate meniurile recomandate. Dacă utilizatorul nu este mulțumit de aceste meniuri poate reveni la pasul de schimbare a preferințelor alimentare sau a nivelului de activitate. Figura 6.7 ilustrează această diagramă de activitate.

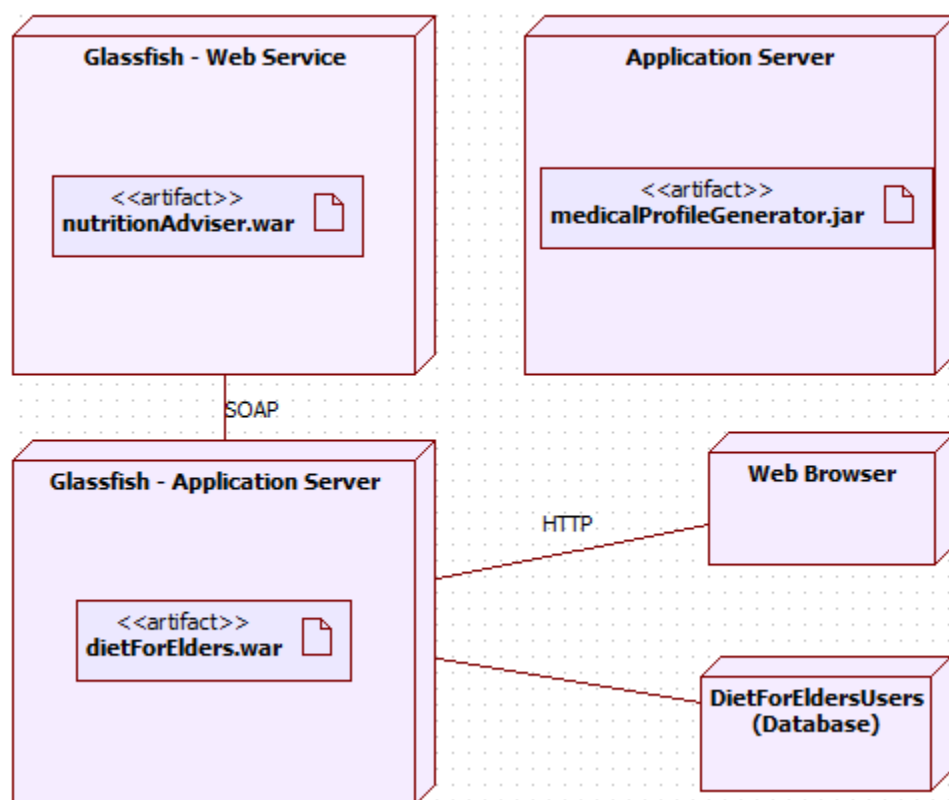
### 6.1.4. Diagrama de deployment

Diagrama de deployment descrie sistemul din perspectiva fizică. Clientul, un browser web, comunică cu serverul prin intermediul protocolul HTTP în mod remote. Componenta principală, care conține interfața utilizator și algoritmul hibrid inspirat din biologie, se află într-o arhivă war ce rulează pe un server Glassfish. Aceasta comunică cu serviciul care oferă consultanță nutrițională prin intermediul protocolului SOAP.



**Figura 6.7** Diagrama de activitate a aplicației

Serviciul web de consultanță nutrițională rulează pe un server Glassfish. Componenta de generare de profile personale se află într-o arhivă jar și rulează independent de componenta principală și de serviciul web. Diagrama de deployment este ilustrată în figura 6.8.



**Figura 6.8 Diagrama de deployment**

### 6.1.5. Diagrama de pachete

În această secțiune sunt prezentate cele mai relevante pachete și este descris modul în care acesta funcționează. Componenta principală conține șase pachete principale: model, convertor, service, creator, searchAlgorithm și web. Interacțiunea dintre pachete e ilustrată în figura 6.9.

În pachetul model se află entitățile sistemului care sunt folosite pentru modelarea persoanelor, afecțiunilor medicale, preferintelor alimentare, ofertelor alimentare, recomandărilor și restricțiilor medicale. Clasele acestui pachet nu depind de niciun alt pachet.

Pachetul converter are rolul de a transforma entitățile aflate în pachetul model în entități de tipul celor utilizate de serviciul web.

Pachetul creator se ocupă cu crearea spațiului de căutare, a funcției de fitness, precum și a profilelor personale ale utilizatorilor. Acest pachet utilizează clasele aflate în pachetul model.

Clasele aflate în cadrul pachetului service sunt responsabile de configurarea parametrilor ajustabili ai algoritmului, verificarea soluțiilor alimentare și aflarea recomandărilor și restricțiilor medicale ale utilizatorului.

Pachetul cel mai important al sistemului este searchAlgorithm. Acesta conține mai multe pachete care modelează algoritmul de căutare, componente hibride, constrângeri și funcția de

optimizare. Acest pachet interacționează cu celelalte pachete (cu excepția pachetului web) pentru găsirea unor meniuri alimentare optime.

Pachetul web este responsabil de interfața grafică a sistemului, conținând la rândul său diverse pachete. Acesta interacționează cu pachetele model și searchAlgorithm.

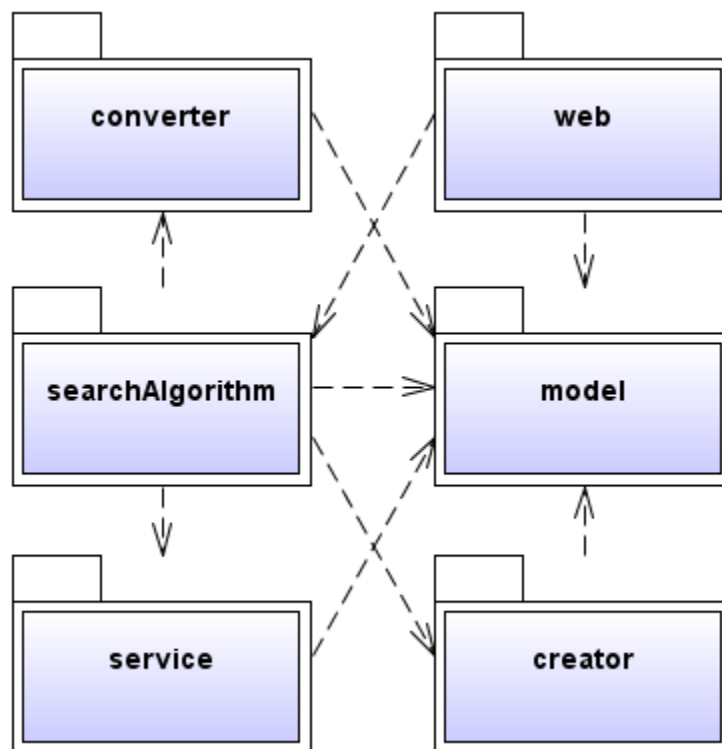


Figura 6.9 Diagrama de pachete a componentei principale

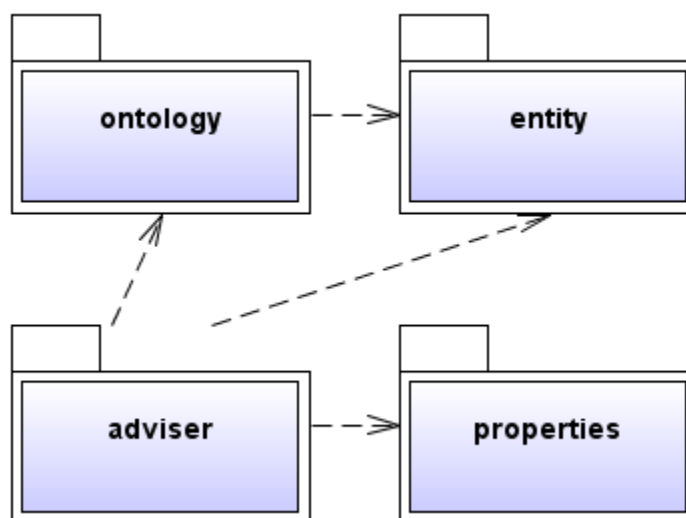


Figura 6.10 Diagrama de pachete a serviciului web

Serviciul web e realizat prin intermediul a patru pachete principale. Pachetul entity conține modelarea entităților aflate în ontologie: categorii de alimente, alimente, persoane, restricții nutriționale. Pachetul ontology e responsabil cu crearea unei conexiuni cu ontologia și interogarea acesteia. Pachetul properties se ocupă de încărcarea fișierelor de proprietăți, iar pachetul adviser conține metodele serviciului web. Modul în care aceste pachete interacționează poate fi observat în figura 6.10.

#### 6.1.6. Diagrame de clasă

Această secțiune descrie modul în care interacționează clasele principale ale pachetelor relevante.

Cele mai importante clase din modulul model sunt responsabile de modelarea soluțiilor alimentare și sunt clase ce reprezintă starea obiectelor, acestea având doar metode de setare sau returnare a variabilelor membru. Modul în care interacționează aceste clase este ilustrat în figura 6.11.

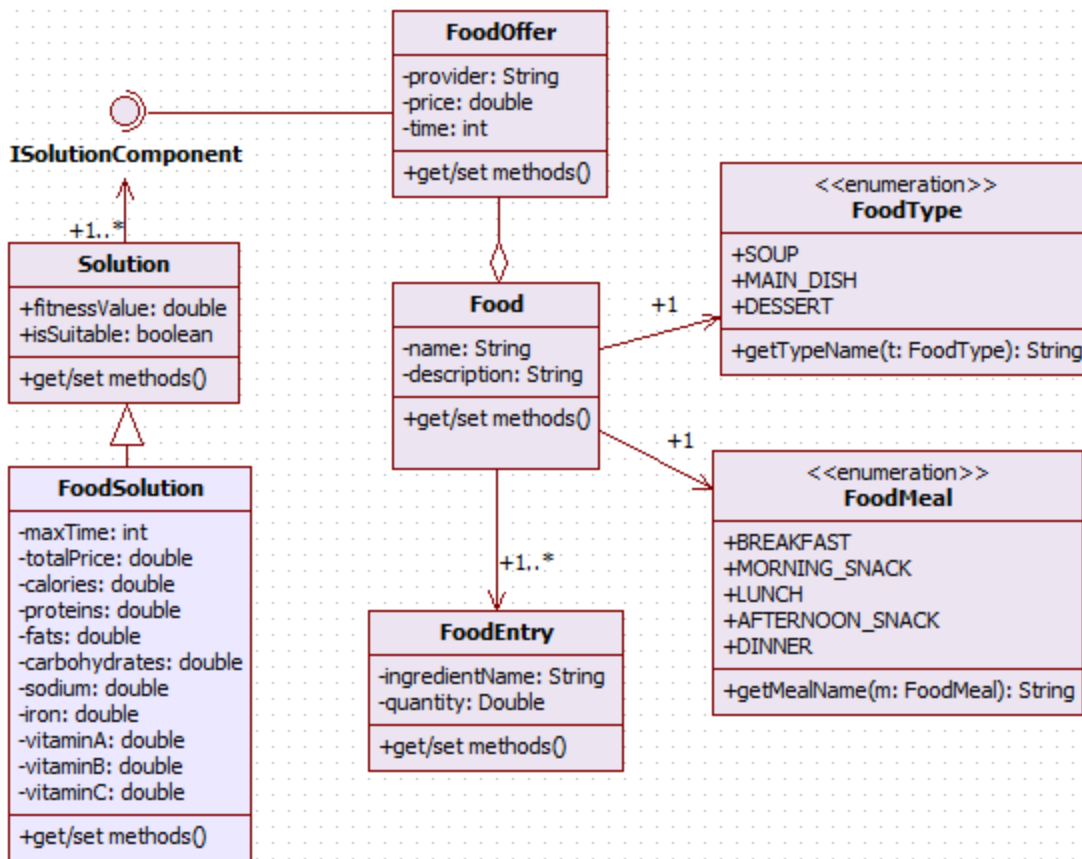
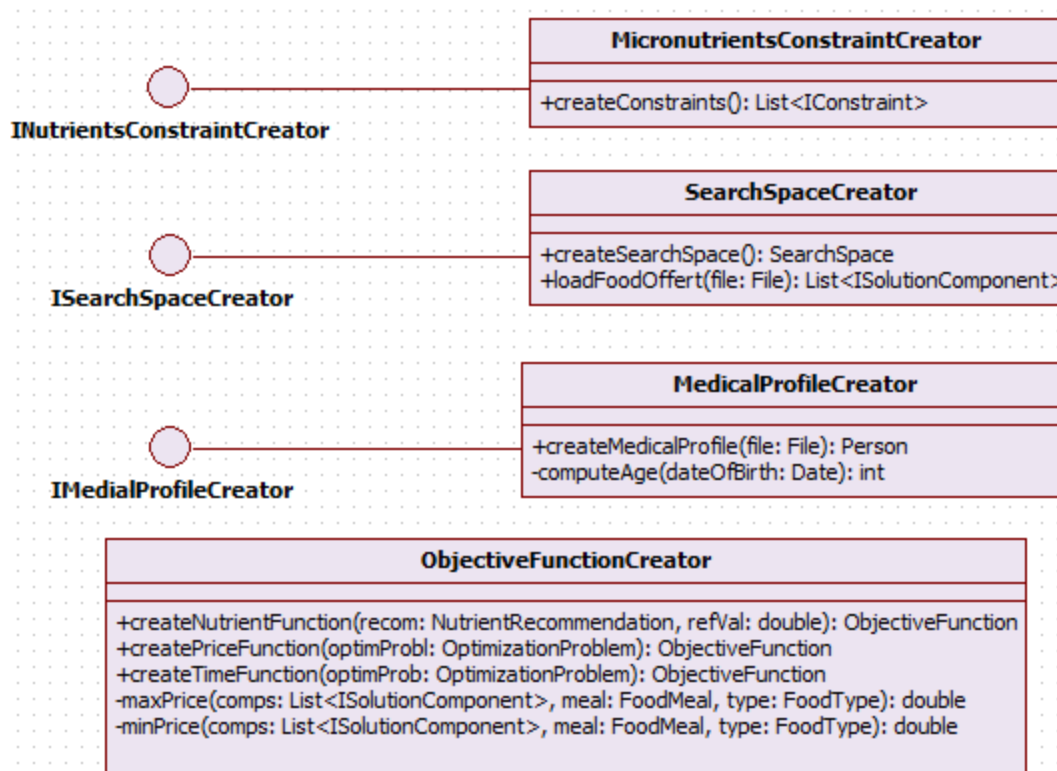


Figura 6.11 Clasele principale ale pachetului model

Figura 6.12 prezintă clasele pachetului creator. Acesta conține clase responsabile cu crearea spațiului de cautare, a funcției obiectiv, a constrângerilor nutriționale, și a profilului medical, precum și interfețele corespunzătoare. Clasele acestui pachet sunt clase de comportament.



**Figura 6.12 Clasele principale ale pachetului creator**

Pachetul `searchAlgorithm` conține clasele care modelează logica algoritmului de căutare. Clasa principală este `Metaheuristic` ce conține metode abstracte de executare a algoritmului și de actualizarea a celei mai bune soluții. Clasa are un obiect de tipul `OptimizationProblem` cu ajutorul căreia este încărcat spațiul de căutare, este creată funcția obiectiv și sunt adăugate contrângerile. Această clasă este extinsă de clasa `CuckooAlgorithm` care oferă implementarea algoritmului `Algoritm_1` și care conține metodele comune ale algoritmului, ca: înlocuirea celor mai slabe cuiburi, executarea pasului de inițializare a populației, sortarea cuiburilor etc. Clasa este extinsă de `CuckooAlgorithmImproved` ce suportă implementarea algoritmului `Algoritm_2`. Clasa `HybridCuckooAlgorithm` extinde clasa `CuckooAlgorithmImproved` și oferă posibilitatea utilizării componentelor hibride: operatorul genetic și structuri de memorie. Clasa `Population` are o listă de soluții ce reprezintă populația cuiburilor, atunci când aparține clasei `Metaheuristic`, respectiv mulțimea de cucu când aparține clasei `CuckooAlgorithm`.

De asemenea, acest pachet conține clasa `NutritionAdviserWebProcessor` care comunică cu serviciul web de consultanță nutrițională. Deoarece realizarea conexiunii cu serviciul web este costisitoare în ceea ce privește timpul, am restricționat numărul de instanțieri ale clasei la un singur obiect. Restricționarea a fost posibilă prin intermediul design patternului `Singleton`, care permite construirea unui singur obiect din aceasta clasă. Diferența majoră dintre o clasă cu metode și atribute statice și o clasă `Singleton` este dată de faptul că aceasta din urmă permite instanțierea târzie, memoria fiind utilizată doar în momentul în care este necesar și este apelată metoda `getNutritionAdviserWebProcessorInstance()`.

Figura 6.13 prezintă clasele pachetului `searchAlgorithm` și modul în care acestea interacționează.



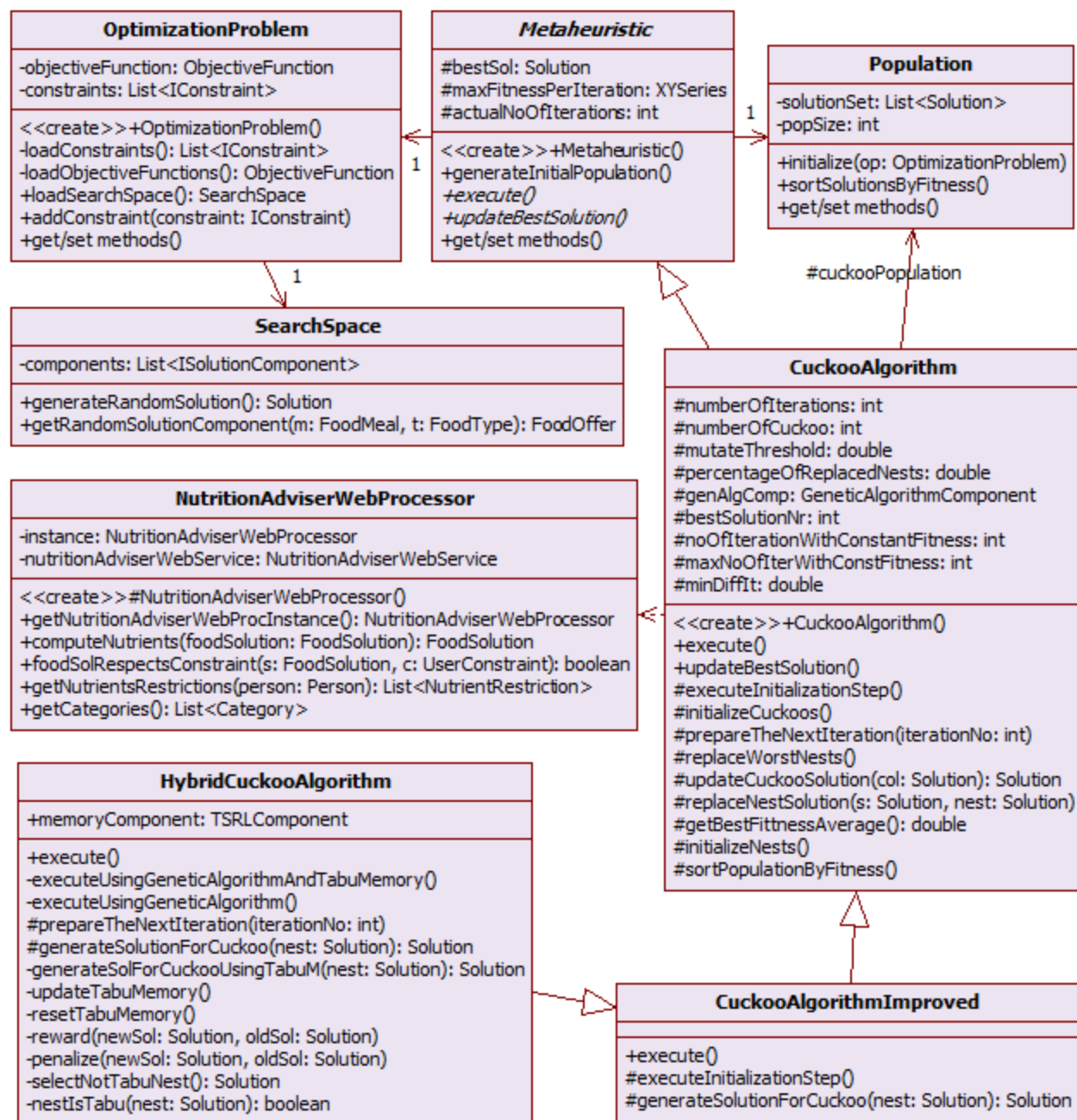


Figura 6.13 Clasele principale ale pachetului searchAlgorithm

Pachetul objectiveFunction aparține pachetului searchAlgorithm și conține clasele cu ajutorul cărora este creată funcția de fitness. Pentru crearea acestei funcții s-a utilizat design patternul structural composite, care permite tratarea obiectelor individuale și compuse într-un mod uniform. În figura 6.14 sunt ilustrate mai multe „părți componente” ale acestui design pattern:

- ObjectiveFunction: reprezintă abstractizarea tuturor elementelor (izolate și ierarhice);
- PriceObjectiveFunction, TimeObjectiveFunction, MacronutrientObjectiveFunction: sunt elemente „izolate”; acestea sunt numite „leaf” și nu prezintă o structură ierarhică;

- **ComplexObjectiveFunction**: este elementul „compozit” cu structură ierarhică, ce agregă alte elemente compozite sau noduri „izolate”;

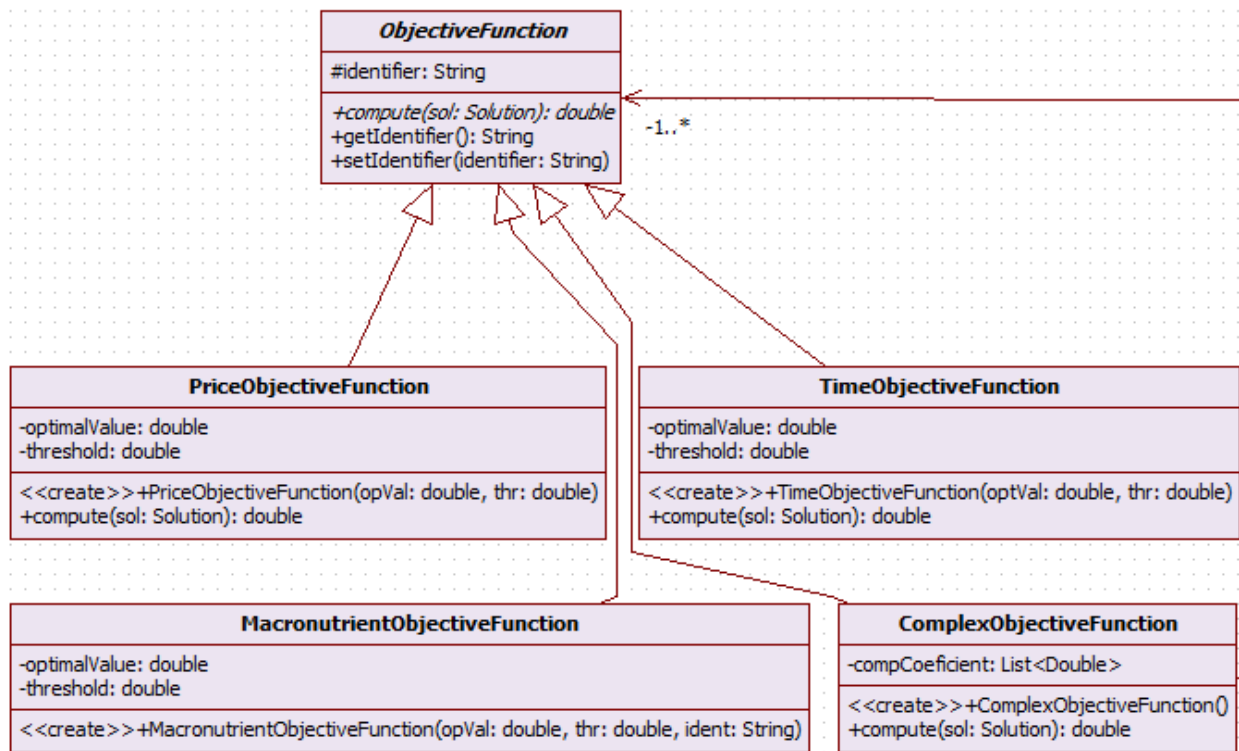


Figura 6.14 Clasele pachetului objectiveFunction

## 6.2. Detalii de implementare

### 6.2.1. Tehnologii și instrumente utilizate

Pentru implementarea acestei aplicații au fost utilizate următoarele tehnologii și tool-uri:

- **Protégé** este un tool care conține un set bogat de structuri de modelare a cunoștințelor ce ajută la dezvoltarea, vizualizarea și procesarea ontologiilor. Ontologiile dezvoltate în Protégé pot fi exportate în diverse formate, inclusiv XML Schema, RDF, RDFS sau OWL. Pentru acest proiect a fost utilizată versiunea 3.5.
- **MySQL** este un sistem populat ce permite dezvoltarea și gestionarea bazelor de date relaționale. Bazele de date MySQL pot fi administrate prin intermediul liniei de comandă sau cu ajutorul unei interfețe grafice. În cadrul acestui proiect am utilizat instrumentul grafic MySQL Workbench.
- **NetBeans** este un mediu integrat de dezvoltare, un instrument ce permite scrierea, compilarea, testarea, depanarea, proiectarea și instalarea programelor. Acesta conține și șabloane de cod, sfaturi de codare, precum și instrumente de refactorizare. Pentru realizarea acestui proiect a fost folosită versiunea 7.4.
- **Apache Maven** este un instrument folosit pentru build-ul și managementul proiectelor. Într-un fișier XML, numit pom.xml, sunt specificate dependențele proiectului, modulele

sau alte component utilizate în cadrul proiectului, precum și alte informații referitoare la realizarea build-ului. Bibliotecile descrie în cadrul dependențele din fișierul pom.xml sunt descărcate în mod dinamic.

- Glassfish este un server web utilizat la scara larga pentru găzduirea aplicațiilor web dezvoltate în Java. Acesta le permite programatorilor să creeze aplicații portabile și scalabile, fiind suportat de către majoritatea mediilor integrate de dezvoltare, inclusiv NetBeans.
- Apache Wicket este un framework folosit pentru dezvoltarea aplicațiilor web. Acesta gestionează întreaga parte de server a fiecărei pagini, fără ca programatorul să fie nevoit să utilizeze în mod direct un obiect de tipul HttpSession sau alte obiecte similar. De asemenea, se oferă o separare clară a markup-ului (reprezentat de fișierele html sau xml) de logica aplicației (reprezentată de codul Java). Pentru realizarea acestui proiect a fost utilizată versiunea 6.9.1.
- Apache Jena este un API Java folosit pentru modelarea, parsarea și persistența datelor în fișiere RDF sau OWL. De asemenea, Jena include și suport pentru limbajul de interogare SPARQL.
- JFreeChart este o librărie Java ce permite realizarea unei varietăți mari de grafice interactive sau non-interactive, printre care grafice X-Y, grafice de tipul Pie, de tipul Bar etc. JFreeChart desenează automat scalele axelor, precum și legendele. În cadrul acestui proiect a fost folosită versiunea 1.0.13.
- StarUML este un instrument folosit pentru crearea diagramelor în limbajul Unified Modeling Language. Cu ajutorul acestuia se pot realiza diferite diagrame printre care diagrame de clasă, de secvență, de colaborare, de componente, de activitate, de deployment etc. Pentru realizarea diagramelor din cadrul acestui proiect a fost utilizată versiunea 5.0.2.
- GitHub este un serviciu ce permite găzduirea aplicațiilor web pentru proiectele software care doresc utilizarea unui sistem de control al versiunilor. Prin intermediul acestuia serviciu se poate realiza partajarea codul între mai multe persoane. Toate aceste persoane pot efectua modificări asupra acelorași fișiere fără ca munca lor să fie stearsă sau suprascrisă.

### 6.2.2. Secvențe de cod

În această secțiune vor fi prezentate și explicate câteva secvențe de cod din cadrul celor trei componente implementate.

În cadrul componentei principale, ce conține algoritmul de căutare a meniurilor personalizate, a fost utilizat conceptul de reflection. Acesta permite examinarea claselor, a câmpurilor sau a metodelor la runtime, fără a cunoaște numele claselor, câmpurilor sau metodelor la compilare. Pentru a calcula funcția de fitness a macronutrienților ar fi trebuit să parcurgem fiecare macronutrient și cu structuri condiționale am fi putut apela metoda corespunzătoare de returnare a cantității din macronutrientul respectiv. Pentru a crea o metodă mai ușor de înțeles și pentru a permite adaugarea unui nou nutrient cu un efort minim am utilizat Java Reflection.

Pentru fiecare macronutrient din enumerarea Macronutrient se va executa secvența de cod din figura 6.15. *Identifier* reprezintă numele macronutrientului definit în enumerarea Macronutrient, iar *GET\_METHOD* este o constantă cu valoarea „get”. *FoodSol* este un obiect de

tipul `FoodSolution` ce conține metode deget pentru fiecare macronutrient. Metoda `getDeclaredMethod()` returnează un obiect de tipul `Method` care reflectă metoda specificată, declarată în clasa reprezentată de obiectul care a apelat metoda `getDeclaredMethod()`. În cazul nostru obiectul `method` va reflecta metoda `getMacronutrientName()` declarată în clasa `FoodSolution`, unde `MacronutrientName` reprezintă numele unui macronutrient (ex: Fats). Parametrul format din `GET_METHOD + identifier` e un `String` ce reprezintă numele metodei, iar `null` reprezintă un array ce conține tipul parametrilor formali ai metodei, în ordinea în care sunt declarați. În cazul nostru metoda de `get` nu are niciun parametru formal. Metoda `invoke(foodSol, null)` returnează rezultatul apelului metodei de către obiectul `foodSol`, neavând parametri actuali (valoarea parametrilor actuali este `null`). În cazul în care sunt aruncate excepții, acestea sunt prinse și logate.

```
try {
    Class<?> clazz = FoodSolution.class;
    Method method = clazz.getDeclaredMethod (GET_METHOD+
        identifier, null);
    Double nutrientQuantity=(Double)method.invoke (foodSol, null);
    .....
} catch (    NoSuchMethodException | SecurityException |
    IllegalAccessException | IllegalArgumentException |
    InvocationTargetException ex) {
    Logger.getLogger(MacronutrientObjectiveFunction.class.getName(
    )).log(Level.SEVERE,null, ex);
}
```

**Figura 6.15 Utilizarea Java Reflection pentru aflarea valorii macronutrienților**

Pentru fiecare soluție generată în mod aleator sau în urma modificării anumitor componente se verifică dacă aceasta respectă constrângerile (atât cele nutritive, cât și cele impuse de utilizator). Dacă o soluție nu respectă una sau mai multe constrângeri, aceasta nu va fi folosită în cadrul algoritmului.

Verificarea constrângerilor se realizează cu metoda prezentată în figura 6.16. Metoda apelează serviciul web pentru calcularea cantităților din fiecare nutrient conținute în soluție, apoi parcurge fiecare constrângere și verifică dacă este respectată. În cazul în care o constrângere nu este respectată, câmpului `isSuitable` al obiectului `solution` îi este asignată valoarea `false` și metoda este părăsită fără a se verifica și celelalte constrângeri.

Evaluarea soluțiilor se realizează prin combinarea funcției multi-obiectiv într-un obiectiv scalar. Valoarea funcțiilor multi-obiectiv este calculată în cadrul claselor care extind clasa `ObjectiveFunction`. În cadrul `ComplexObjectiveFunction` este definită metoda `compute(Solution sol)`, prin intermediul căreia mai multe funcții multi-obiectiv sunt combinate într-un obiectiv scalar (figura 6.17). Obiectivul scalar este calculat ca o sumă ponderată. Clasa `ComplexObjectiveFunction` conține o listă de `ObjectiveFunction`, ce reprezintă componentele funcției de fitness, precum și o listă ce reprezintă ponderile asociate componentelor. Metoda din această clasă este apelată pentru a calcula valoarea funcției de fitness corespunzătoare nutrienților, dar și pentru a calcula funcția principală de fitness. În ultimul caz lista de componente conține următoarele tipuri de obiecte:

- `PriceObjectiveFunction`: conține metodă de calcul a funcției obiectiv ce depinde de timp;

- TimeObjectiveFunction: conține metoda de calcul a funcției obiectiv ce depinde de preț;
- ComplexObjectiveFunction: această conține la rândul ei patru obiecte de tipul MacronutrientObjectiveFunction, câte un obiect pentru fiecare tip de nutrient: calorii, proteine, lipide și carbohidrați.

```
public static Solution checkSolutionNutrientsAndConstraints (Solution
solution, OptimizationProblem optimProblem) {
    solution = NutritionAdviserWebProcessor.
    getNutritionAdviserWebProcessorInstance ().computeNutrients ((
    FoodSolution) solution);

    if (optimProblem.getConstraints () != null && optimProblem.
    getConstraints ().size () > 0) {
        for (IConstraint constr : optimProblem.getConstraints ()) {
            if (!constr.verify(solution)) {
                solution.setIsSuitable(false);
                return solution;
            }
        }
    }

    solution.setIsSuitable(true);
    return solution;
}
```

Figura 6.16 Verficarea respectării constrângerilor de către o soluție

```
@Override
public double compute(Solution sol) {
    double computedValue = 0.0;
    if (components != null && !components.isEmpty()) {
        for (int i = 0; i < components.size(); i++) {
            ObjectiveFunction comp = components.get(i);
            double soFunctionValue = comp.compute(sol);
            computedValue += compCoefficient.get(i) * soFunctionValue;
        }
    }
    return computedValue;
}
```

Figura 6.17 Calcularea funcției de fitness multi-obiectiv

În cadrul algoritmului hibrid inspirat din comportamentul cucilor, operatorul genetic și structuri de memorie se folosește conceptul de acordare de recompense sau penalizări. În figura 6.18 este ilustrată metoda prin intermediul căreia se acordă o recompensă obținută în urma înlocuirii soluției *oldNest* cu *newNest*. Pentru fiecare pereche de componente din cadrul celor două soluții se face o căutare în elementele memoriei de lungă durată dacă cele două componente nu sunt identice. Atunci când perechea de componente este găsită scorul este incrementat. Dacă

în elementele memoriei de lungă durată nu se găsește perechea căutată se construiește un nou element care este apoi adăugat la memorie.

```
private void reward(Solution newNest, Solution oldNest) {
    for (int i = 0; i < oldNest.getComponents().size(); i++) {
        if (!oldNest.getComponents().get(i).equals(newNest.getComponents().
            get(i))) {
            boolean elementFound = false;
            for (LTMElement ltmE : memoryComponent.getLTMemory().
                getLTMElements()) {
                if (ltmE.getFirstComponent().equals(oldNest.getComponents().
                    get(i)) && ltmE.getSecondComponent().equals(oldNest.
                        getComponents().get(i))) {
                    int initialScore = ltmE.getScore();
                    initialScore++;
                    ltmE.update(initialScore);
                    elementFound = true;
                    break;
                }
            }
            if (!elementFound) {
                LTMElement ltmElement = new LTMElement(oldNest.
                    getComponents().get(i), newNest.getComponents().get(i));
                memoryComponent.addToLongTermMemory(ltmElement);
            }
        }
    }
}
```

**Figura 6.18** Metoda de acordare de recompense în urma înlocuirii a două soluții

În cadrul componentelor de consultanță nutrițională și de generare a profilelor medicale au fost executate câteva query-uri SPARQL. Două dintre acestea fi prezentate în continuare. În ontologie entitățile și indivizii au următorul prefix: <http://www.pips.eu.org/ontologies/food#> care a fost notat cu *foaf*. Atunci când este utilizat în cadrul query-ului prefixul este urmat de „:” și cuvântul pe care dorim să îl concatenăm la prefix. În cazul nostru *foaf:name* e echivalent cu <http://www.pips.eu.org/ontologies/food#name>.

În figura 6.19 este prezentat un query ce returnează toate ingredientele din ontologie împreună cu valorile nutritive ale acestora. Pentru realizarea query-ului au fost utilizate pattern-uri triple de tipul: subiect, predicat, obiect. De exemplu, „?ingredient foaf:name ?ingredientName” presupune găsirea tuturor subiectelor (*?ingredient*) și a obiectelor (*?ingredientName*) care sunt unite prin predicatul *foaf:name*. Punctul folosit după obiect și înaintea următorului subiect permite conectarea împreună a triplelor, astfel creându-se un graf interconectat. Query-ul va returna valorile tuturor variabilelor specificate după cuvântul cheie *select*.

```
private static final String SELECT_ALL_INGREDIENTS_QUERY = "PREFIX foaf: <
http://www.pips.eu.org/ontologies/food#>"
+ "SELECT ?ingredientName ?calories ?proteins ?carbohydrates ?fats"
+ " ?iron ?sodium ?vitaminA ?vitaminB ?vitaminC"
+ " WHERE { "
+ "?ingredient foaf:name ?ingredientName ."
+ "?ingredient foaf:calories ?calories ."
+ " .?ingredient foaf:proteins ?proteins ."
+ " .?ingredient foaf:carbohydrates ?carbohydrates ."
+ " .?ingredient foaf:fats ?fats ."
+ " .?ingredient foaf:iron ?iron ."
+ " .?ingredient foaf:sodium ?sodium ."
+ " .?ingredient foaf:vitaminA ?vitaminA ."
+ " .?ingredient foaf:vitaminB ?vitaminB ."
+ " .?ingredient foaf:vitaminC ?vitaminC ."
+ "}";
```

**Figura 6.19** Query SPARQL ce returnează ingredientele și valorile nutritive ale acestora

În figura 6.19 este ilustrat un query ce returnează toate persoanele împreună cu informațiile personale și afecțiunile medicale ale acestora. Spre deosebire de query-ul prezentat anterior, acesta utilizează și clauza „optional”. Aceasta este folosită atunci când vrem să includem variabile suplimentare în căutare, dar vrem să permit rezultate și atunci când nu există valori pentru aceste variabile. În cazul nostru informațiile personale ale pacientului vor fi returnate chiar și atunci când persoana respectivă nu are nicio afecțiune medicală.

```
private static final String GET_PERSON_QUERY = " PREFIX foaf: <http://www.
pips.eu.org/ontologies/food#> "
+ " SELECT  ?first_name ?last_name ?weight ?height ?gender ?date_of_birth ?
disease_name"
+ " WHERE { ?person foaf:first_name ?first_name "
+ " .?person foaf:last_name ?last_name"
+ " .?person foaf:weight ?weight"
+ " .?person foaf:height ?height"
+ " .?person foaf:gender ?gender"
+ " .?person foaf:date_of_birth ?date_of_birth"
+ " OPTIONAL{?person foaf:hasDiseases ?disease OPTIONAL{?disease foaf:
diseaseName ?disease_name}}"
+ "}";
```

**Figura 6.20** Query SPARQL ce returnează informații referitoare la profilele pacienților

## 7. Evaluarea tehnicilor propuse

Metodele dezvoltate pe parcursul evoluției tehnicii hibride prezentate în secțiunea 5.2 au fost testate pe diverse scenarii ce sunt prezentate în această secțiune. Astfel, pe baza rezultatelor obținute se poate realiza o analiză comparativă a tehnicilor propuse. De asemenea, se va realiza o analiză comparativă între tehnica prezentată în cadrul acestei lucrări și o tehnică hibridă bazată pe colonizarea buruienilor, dezvoltată de Valea Dan Alexandru.

### 7.1. Scenarii de test

Pentru a efectua o testare minuțioasă am folosit trei scenarii de test, care se deosebesc în funcție de necesarul zilnic de nutrienți. Pentru toate scenariile de test dimensiunea spațiului de căutare este de 68.400.000.000.000.000 de meniuri alimentare. Sistemul pe care au fost testate scenariile are următoarele configurații:

- Model procesor: Pentium(R) Dual-Core CPU;
- Frecvența procesorului: 2.10 GHz ;
- Arhitectura: 64 biți;
- Memorie RAM: 3,00 GB.

În tabelul 7.1 sunt prezentați parametrii primului scenariu de test (Scenariu\_1) împreună cu necesarul zilnic de nutrienți. Pacient prezentat prezintă două afecțiuni medicale, ceea ce afectează dozele zilnice recomandate de sodiu și de carbohidrați.

Nume	Enescu	Calorii	1690kcal
Prenume	Andrei	Proteine	116g
Vârstă	69ani	Lipide	48,6g
Greutate	70kg	Carbohidrați	157,5g
Înălțime	165cm	Fier	8-45mg
Sex	Masculin	Sodiu	500-1500mg
Nivel de activitate	Activitate ușoară	Vitamina A	1,2-7,5mg
Afecțiuni	Hipertensiune, Diabet tip I	Vitamina B1	1,6-5mg
Ingredient preferat	Carne de pui	Vitamina C	75-1000mg
Ingredient nedorit	Carne de vită		

**Tabel 7.1 Parametrii pentru Scenariu\_1 și dozele nutriționale zilnice recomandate**

În tabelele 7.2 și 7.3 sunt ilustrați parametrii pentru scenariile Scenariu\_2, respectiv Scenariu\_3, precum și dozele zilnice de nutrienți recomandate. Aceste scenarii au aceeași parametri, cu excepția nivelului de activitate. Se poate observa faptul că nivelul de activitate influențează dozele zilnice recomandate de calorii, proteine și lipide și carbohidrați. Necesarul zilnic recomandat de sodium este influențat de faptul că pacientul suferă de hipertensiune arterială.

Pentru Scenariu\_2 s-a încercat efectuarea unei căutări exhaustive. Căutarea a fost oprită după aproximativ cinci zile, iar la finalul acestei perioade spațiul de căutare nu era parcurs în totalitate, iar funcția maximă de fitness găsită până în acel moment avea valoarea aproximativ 0,50.



Nume	Pop
Prenume	Ioan
Vârstă	64 ani
Greutate	66 kg
Înălțime	160 cm
Sex	Masculin
Nivel de activitate	Activitate moderată
Afectiuni	Hipertensiune
Ingredient preferat	Fructele de pădure
Ingredient nedorit	Brânza

Calorii	1800kcal
Proteine	123g
Lipide	50g
Carbohidrați	202,4g
Fier	8-45mg
Sodiu	500-1500mg
Vitamina A	1,2-7,5mg
Vitamina B1	1,6-5mg
Vitamina C	75-1000mg

**Tabel 7.2 Parametrii pentru Scenariu\_2 și dozele nutriționale zilnice recomandate**

Nume	Pop
Prenume	Ioan
Vârstă	64 ani
Greutate	66 kg
Înălțime	160 cm
Sex	Masculin
Nivel de activitate	Activitate ridicată
Afectiuni	Hipertensiune
Ingredient preferat	Orez
Ingredient nedorit	Ciuperci

Calorii	1927,8kcal
Proteine	132,5g
Lipide	53,5g
Carbohidrați	216,9g
Fier	8-45mg
Sodiu	500-1500mg
Vitamina A	1,2-7,5mg
Vitamina B1	1,6-5mg
Vitamina C	75-1000mg

**Tabel 7.3 Parametrii pentru Scenariu\_3 și dozele nutriționale zilnice recomandate**

## 7.2. Identificarea parametrilor ajustabili

Parametri ajustabili folosiți în cadrul tehnicilor CS, CSU, CSURG, CSUCGM sunt următorii:

- *noNest*: numărul de cuiburi disponibile pe parcursul executării tehnicii.
- *noCuck*: numărul de cucii utilizați;
- *repNest*: procentul de cuiburi înlocuite cu unele generate aleator. Acest procent poate avea valori în intervalul  $[0,1]$ . Pentru valoarea 0 nu se va înlocui niciun cuib la sfârșitul iterației, iar pentru valoarea 1 se vor înlocui toate soluțiile la sfârșitul iterației, deci nu se va beneficia de îmbunătățirile aduse cuiburilor pe parcursul iterațiilor;
- *trMut*: pragul de mutație prestabilit. Acest parametru poate avea valori în intervalul  $[0,1]$ . Atunci când acest parametru are valoarea 0 vor efectua mutații asupra tuturor componentelor, iar atunci când acesta are valoarea 1 nu se vor efectua mutații, soluția rămânând neschimbată.
- *noIt*: numărul maxim de iterații;
- *diffIt*: diferența minimă de fitness între două iterații ca acestea să fie considerate stagnări. Numărul de iterații parcurse crește proporțional cu valoarea parametrului.
- *noConstIt*: numărul maxim de stagnări succesive. Cu cât valoarea acestui parametru este mai mare cu atât tehnica va parcurge un număr mai mare de iterații.

Algoritmul CSUCGM utilizează un parametru ajustabil suplimentar:  $it_{\text{tabu}}$ , care reprezintă dimensiunea memoriei tabu de scurtă durată. Aceasta trebuie să fie cuprinsă în intervalul  $[0, noNest]$ , unde în cazul utilizării valorii 0 nu se va utiliza memoria de scurtă durată.

Pentru ca rezultatele obținute în urma testării să fie cât mai aproape de rezultatele reale s-au executat 20 de rulari pentru fiecare configurație și s-a făcut o medie a acestora. Aspectele analizate sunt următoarele:

- $fit_{\text{avg}}$  reprezintă media funcțiilor de fitness a primelor celor mai bune cuiburi;
- $t_{\text{avg}}(\text{sec})$  reprezintă media timpului de execuție măsurat în secunde;
- $it_{\text{avg}}$  reprezintă media numărului de iterații executate;

### 7.3. Analiza rezultatelor experimentale obținute

În cele ce urmează se vor prezenta rezultatele obținute în urma rulării tehnicilor CS, CSU, CSURG și CSUCGM. Configurația colorată cu roșu are cel mai bun fitness, iar cea colorată cu verde reprezintă cel mai bun compromis între fitness și timp.

no	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg</sub> (sec)	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,502	3,9	10,5
2	25	30	20	0,8	0,2	6	0,01	0,47	3	11,1
3	25	30	20	0,6	0,4	6	0,01	0,507	5,9	11,85
4	25	30	20	0,6	0,2	6	0,01	0,481	3,45	9,85
5	25	30	15	0,8	0,4	6	0,01	0,498	3,6	10,45
6	25	30	15	0,8	0,2	6	0,01	0,491	3,2	11,85
7	25	30	15	0,6	0,4	6	0,01	0,499	3,3	11,95
8	25	30	15	0,6	0,2	6	0,01	0,477	2,55	10,75
9	25	25	20	0,8	0,4	6	0,01	0,489	3,3	11,75
10	25	25	20	0,6	0,4	6	0,01	0,496	3,1	12,4
11	25	25	15	0,8	0,4	6	0,01	0,5	3,65	12,95
12	25	25	15	0,6	0,4	6	0,01	0,492	2,5	10,05
13	20	30	20	0,8	0,4	6	0,01	0,498	3,5	10,2
14	20	30	20	0,6	0,4	6	0,01	0,502	3,7	11,6
15	20	30	15	0,8	0,4	6	0,01	0,499	5,5	10,45
16	20	30	15	0,6	0,4	6	0,01	0,504	5,6	11,75
17	20	25	20	0,8	0,4	6	0,01	0,493	3,3	11,75
18	20	25	20	0,6	0,4	6	0,01	0,499	3,1	12,85
19	20	25	15	0,8	0,4	6	0,01	0,489	2,55	11,6
20	20	25	15	0,6	0,4	6	0,01	0,482	2,44	10,45
21	20	25	15	0,6	0,4	10	0,001	0,493	4,1	15,4

Tabel 7.4 Rezultatele obținute pentru algoritmul CS, Scenariu\_1

În tabelul 7.4 pot fi observate rezultatele obținute pentru utilizarea primului scenariu în cadrul algoritmului CS. Pentru valorile parametrilor din rândul 3 se obțin soluțiile cele mai bune în ceea ce privește funcția de fitness, dar timpul de procesare al acestor soluții este cel mai mare comparativ cu timpul celorlalte configurații din tabel. Cea mai bună configurație obținută ca un compromis între fitness și timp poate fi observată în rândul 18. Configurația pentru care media numărului de iterații executate este cea mai mare se află în rândul 21 deoarece spre deosebire de celelalte configurații aceasta permite executarea algoritmului atunci când apare un număr mare de stagnări consecutive.

no	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg</sub> (sec)	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,588	6,65	15,15
2	25	30	20	0,8	0,2	6	0,01	0,570	4,25	15,7
3	25	30	20	0,6	0,4	6	0,01	0,584	6,4	14,4
4	25	30	20	0,6	0,2	6	0,01	0,572	5,15	13,5
5	25	30	15	0,8	0,4	6	0,01	0,581	6,1	12,25
6	25	30	15	0,8	0,2	6	0,01	0,564	2,15	9,05
7	25	30	15	0,6	0,4	6	0,01	0,578	3,7	10,95
8	25	30	15	0,6	0,2	6	0,01	0,561	2,05	10,1
9	25	25	20	0,8	0,4	6	0,01	0,572	3,1	10,95
10	25	25	20	0,6	0,4	6	0,01	0,573	3,15	11,3
11	25	25	15	0,8	0,4	6	0,01	0,580	3,3	12,65
12	25	25	15	0,6	0,4	6	0,01	0,579	3,3	12,3
13	20	30	20	0,8	0,4	6	0,01	0,573	3,65	10,65
14	20	30	20	0,6	0,4	6	0,01	0,579	3,7	11,55
15	20	30	15	0,8	0,4	6	0,01	0,586	4,1	12,4
16	20	30	15	0,6	0,4	6	0,01	0,584	3,75	11,75
17	20	25	20	0,8	0,4	6	0,01	0,571	2,85	10,55
18	20	25	20	0,6	0,4	6	0,01	0,577	3,3	12,1
19	20	25	15	0,8	0,4	6	0,01	0,565	2,55	9,25
20	20	25	15	0,6	0,4	6	0,01	0,580	3,45	12,45
21	20	25	15	0,6	0,4	10	0,001	0,587	4,9	19,45

**Tabel 7.5 Rezultatele obținute pentru algoritmul CS, Scenariu\_2**

În tabelul 7.5 pot fi observate rezultatele obținute pentru cel de-al doilea scenariu. Deoarece pacientul utilizat pentru acest scenariu suferă de o singură afecțiune medicală și constrângerile nutriționale impuse de afecțiunile medicale sunt mai puține, recomandările alimentare pentru acest scenariu sunt mai bune calitativ comparative cu cele din primul scenariu. Cea mai bună configurație în ceea ce privește funcția de fitness se află în rândul 1, dar timpul de procesare pentru soluțiile acestei configurații este cel mai mare comparativ cu timpul celorlalte

configurații din tabel. Cea mai bună configurație obținută ca un compromis între fitness și timp apare în rândul 16. La fel ca în cazul scenariului anterior, configurația pentru care media numărului de iterații executate este cea mai mare se află în rândul 21.

No	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg</sub> (sec)	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,575	4,9	11,7
2	25	30	20	0,8	0,2	6	0,01	0,553	3,3	9,25
3	25	30	20	0,6	0,4	6	0,01	0,574	6,4	12,25
4	25	30	20	0,6	0,2	6	0,01	0,563	4,3	10,4
5	25	30	15	0,8	0,4	6	0,01	0,571	4,45	11,05
6	25	30	15	0,8	0,2	6	0,01	0,565	3,25	9,9
7	25	30	15	0,6	0,4	6	0,01	0,585	4,8	11,45
8	25	30	15	0,6	0,2	6	0,01	0,569	3,45	11,85
9	25	25	20	0,8	0,4	6	0,01	0,578	5,05	12,6
10	25	25	20	0,6	0,4	6	0,01	0,576	5,25	14,5
11	25	25	15	0,8	0,4	6	0,01	0,571	4,7	12,25
12	25	25	15	0,6	0,4	6	0,01	0,568	4,2	11,85
13	20	30	20	0,8	0,4	6	0,01	0,586	6,9	10,85
14	20	30	20	0,6	0,4	6	0,01	0,579	4,25	12,05
15	20	30	15	0,8	0,4	6	0,01	0,575	3,55	10,4
16	20	30	15	0,6	0,4	6	0,01	0,571	3,85	11,2
17	20	25	20	0,8	0,4	6	0,01	0,57	3,15	10,9
18	20	25	20	0,6	0,4	6	0,01	0,572	3,45	11,7
19	20	25	15	0,8	0,4	6	0,01	0,573	2,85	10,3
20	20	25	15	0,6	0,4	6	0,01	0,568	3,1	11,1
21	20	25	15	0,6	0,4	10	0,001	0,592	5,2	19,25

**Tabel 7.6 Rezultatele obținute pentru algoritmul CS, Scenariu\_3**

În tabelul 7.6 pot fi observate rezultatele obținute pentru cel de-al treilea scenariu. Deoarece pentru acest scenariu a fost utilizat același pacient ca în cazul scenariului anterior, meniurile alimentare recomandate în cazul acestor două scenarii sunt aproximativ la fel de bune calitativ. De asemenea, meniurile recomandate în cadrul acestui scenariu sunt mai bune calitativ în comparație cu cele recomandate în cadrul primului scenariu.

Pentru cel de-al treilea scenariu, cele mai bune soluții din punctul de vedere al funcției de fitness sunt obținute pentru configurația descrisă în rândul 21, dar, spre deosebire de celelalte două scenarii, timpul de procesare al acestor soluții nu este cel mai mare comparativ cu timpul celorlalte configurații din tabel. Cea mai bună configurație obținută ca un compromis între fitness și timp apare în rândul 13. La fel ca în cazul scenariilor anterior, configurația pentru care media numărului de iterații executate este cea mai mare se află în rândul 21.

No	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,597	22,3	18,4
2	25	30	20	0,8	0,2	6	0,01	0,619	21,8	19,65
3	25	30	20	0,6	0,4	6	0,01	0,598	17,85	16,65
4	25	30	20	0,6	0,2	6	0,01	0,585	18	15,65
5	25	30	15	0,8	0,4	6	0,01	0,593	15	18,15
6	25	30	15	0,8	0,2	6	0,01	0,589	12,4	16,45
7	25	30	15	0,6	0,4	6	0,01	0,573	13,4	15,45
8	25	30	15	0,6	0,2	6	0,01	0,575	12,65	16,75
9	25	25	20	0,8	0,4	6	0,01	0,594	16,05	16,55
10	25	25	20	0,6	0,4	6	0,01	0,59	15,45	15,4
11	25	25	15	0,8	0,4	6	0,01	0,593	13,95	18,5
12	25	25	15	0,6	0,4	6	0,01	0,569	12,15	15,15
13	20	30	20	0,8	0,4	6	0,01	0,588	20,5	16,4
14	20	30	20	0,6	0,4	6	0,01	0,599	16,35	15,95
15	20	30	15	0,8	0,4	6	0,01	0,567	12,1	14,2
16	20	30	15	0,6	0,4	6	0,01	0,586	10,8	12,95
17	20	25	20	0,8	0,4	6	0,01	0,604	16,55	17,55
18	20	25	20	0,6	0,4	6	0,01	0,585	13,9	14,3
19	20	25	15	0,8	0,4	6	0,01	0,58	12,55	16,3
20	20	25	15	0,6	0,4	6	0,01	0,572	11,3	15,4
21	20	25	15	0,6	0,4	10	0,001	0,587	16,8	18,6

**Tabel 7.7 Rezultatele obținute pentru algoritmul CSU, Scenariu\_1**

În tabelul 7.7 pot fi observate rezultatele obținute pentru utilizarea primul scenariu în cadrul algoritmului CSU. Cele mai bune soluții în ceea ce privește funcția de fitness se obțin prin utilizarea configurației aflate în rândul 2, dar timpul de procesare al acestor soluții este al doilea cel mai mare comparativ cu timpul celorlalte configurații din tabel. În cadrul acestei configurații media numărului de iterații executate este cea mai mare din tabel.

Cea mai bună configurație obținută ca un compromis între fitness și timp apare în rândul 17, funcția de fitness pentru această configurație fiind cu doar 0,015 mai mică comparativ cu configurația pentru care se obțin cele mai bune rezultate în ceea ce privește funcția de fitness. Comparativ cu aceeași configurație, timpul necesar procesării este cu aproximativ 0,25% mai mic.

În tabelul 7.8 sunt ilustrate rezultatele obținute pentru cel de-al doilea scenariu. Deoarece pacientul utilizat pentru acest scenariu suferă de o singură afecțiune medicală și constrângerile nutriționale impuse de afecțiunile medicale sunt mai puține, recomandările alimentare pentru acest scenariu sunt mai bune calitativ comparative cu cele din primul scenariu. Pentru acest

scenariu cea mai bună configurație din punctul de vedere al funcției de fitness se află în rândul 9, timpul de procesare al acestor soluții fiind unul dintre cei mai mari din tabel.

Cea mai bună configurație obținută ca un compromis între fitness și timp apare în rândul 17, la fel ca în cadrul scenariului anterior. Pentru configurația aflată în rândul 21, media numărului de iterații executate este cea mai mare, fiind egală cu numărul maxim de iterații permise.

No	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg</sub> (sec)	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,676	27,4	17,55
2	25	30	20	0,8	0,2	6	0,01	0,672	25,4	17,5
3	25	30	20	0,6	0,4	6	0,01	0,654	22,8	14,65
4	25	30	20	0,6	0,2	6	0,01	0,659	23,2	15,35
5	25	30	20	0,8	0,4	6	0,01	0,655	26	17,45
6	25	30	15	0,8	0,2	6	0,01	0,658	22,1	18,25
7	25	30	15	0,6	0,4	6	0,01	0,637	18,15	14,45
8	25	30	15	0,6	0,2	6	0,01	0,647	15,9	14,55
9	25	25	20	0,8	0,4	6	0,01	0,679	25,6	17,8
10	25	25	20	0,6	0,4	6	0,01	0,658	22,4	15,9
11	25	25	15	0,8	0,4	6	0,01	0,654	18,85	17,4
12	25	25	15	0,6	0,4	6	0,01	0,639	14,85	13,45
13	20	30	20	0,8	0,4	6	0,01	0,660	23,5	15,45
14	20	30	20	0,6	0,4	6	0,01	0,654	22,75	15,15
15	20	30	15	0,8	0,4	6	0,01	0,642	17,5	14,65
16	20	30	15	0,6	0,4	6	0,01	0,640	18,45	15,05
17	20	25	20	0,8	0,4	6	0,01	0,673	22,1	15,85
18	20	25	20	0,6	0,4	6	0,01	0,649	19,85	14,5
19	20	25	15	0,8	0,4	6	0,01	0,658	18	16,6
20	20	25	15	0,6	0,4	6	0,01	0,638	16,2	14,75
21	20	25	15	0,6	0,4	10	0,001	0,656	22,5	20

**Tabel 7.8 Rezultatele obținute pentru algoritmul CSU, Scenariu\_2**

În tabelul 7.9 sunt ilustrate rezultatele obținute pentru cel de-al treilea scenariu. Acest scenariu utilizează același pacient ca în cazul scenariului anterior, de aceea meniurile alimentare recomandate în cazul acestor două scenarii sunt aproximativ la fel de bune calitativ. De asemenea, meniurile recomandate în cadrul acestui scenariu sunt mai bune calitativ în comparație cu cele recomandate în cadrul primului scenariu.

Cea mai bune soluții cu privire la funcția de fitness sunt obținute prin folosirea configurației din rândul 2, dar, spre deosebire de celelalte două scenarii, timpul de procesare al acestor soluții nu se află printre cele mai mari din tabel. În rândul 4 apare cea mai bună

configurație obținută ca un compromis între fitness și timp, funcția de fitness a acesteia fiind cu 0,026 mai mică comparativ cu configurația pentru care se obțin cele mai bune rezultate în ceea ce privește funcția de fitness. Comparativ cu aceeași configurație, timpul necesar procesării este cu aproximativ 0,20% mai mic. Pentru configurația aflată în rândul 21, media numărului de iterații executate este cea mai mare, fiind egală cu numărul maxim de iterații permise, la fel ca în cazul scenariului anterior.

no	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg</sub> (sec)	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,668	27,5	18,45
2	25	30	20	0,8	0,2	6	0,01	0,683	26,5	18,45
3	25	30	20	0,6	0,4	6	0,01	0,654	22,5	14,7
4	25	30	20	0,6	0,2	6	0,01	0,657	20,65	14,8
5	25	30	20	0,8	0,4	6	0,01	0,633	18,3	14,7
6	25	30	15	0,8	0,2	6	0,01	0,649	19,4	15,9
7	25	30	15	0,6	0,4	6	0,01	0,650	21,05	16
8	25	30	15	0,6	0,2	6	0,01	0,638	19	14,8
9	25	25	20	0,8	0,4	6	0,01	0,669	29,95	17,35
10	25	25	20	0,6	0,4	6	0,01	0,656	26,15	15,45
11	25	25	15	0,8	0,4	6	0,01	0,660	24,35	17,7
12	25	25	15	0,6	0,4	6	0,01	0,652	22,3	16,8
13	20	30	20	0,8	0,4	6	0,01	0,662	28,4	16,35
14	20	30	20	0,6	0,4	6	0,01	0,660	26,95	16,4
15	20	30	15	0,8	0,4	6	0,01	0,654	21,85	16,75
16	20	30	15	0,6	0,4	6	0,01	0,643	19,5	15,4
17	20	25	20	0,8	0,4	6	0,01	0,673	27,15	17,4
18	20	25	20	0,6	0,4	6	0,01	0,650	20,35	13,75
19	20	25	15	0,8	0,4	6	0,01	0,644	16,05	13,7
20	20	25	15	0,6	0,4	6	0,01	0,644	17,85	14,95
21	20	25	15	0,6	0,4	10	0,001	0,656	24	20

**Tabel 7.9 Rezultatele obținute pentru algoritmul CSU, Scenariu\_3**

Pentru toate cele trei scenarii pentru care a fost evaluat algoritmul CSU, cele mai bune rezultate au fost obținute prin utilizarea unui număr mare de cucu. Acest lucru este datorat faptului că atunci când este utilizat un număr mai mare de cucu sunt executate mai multe operații asupra populației de cuiburi în cadrul aceleași iterații.

În tabelul 7.10 pot fi observate rezultatele obținute pentru utilizarea primului scenariu în cadrul algoritmului CSURG. Cea mai bună soluție în ceea ce privește funcția de fitness sunt obținute în urma utilizării configurației din rândul 3. Cea mai bună configurație obținută ca un

compromis între fitness și timp apare în rândul 21, iar cea mai mare medie a numărului de iterații executate este obținută pentru prima configurație.

No	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg</sub> (sec)	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,602	20,95	18,95
2	25	30	20	0,8	0,2	6	0,01	0,597	24,25	18
3	25	30	20	0,6	0,4	6	0,01	0,617	17,85	17,55
4	25	30	20	0,6	0,2	6	0,01	0,614	22,4	17,75
5	25	30	15	0,8	0,4	6	0,01	0,576	19	17,3
6	25	30	15	0,8	0,2	6	0,01	0,581	15,5	17,45
7	25	30	15	0,6	0,4	6	0,01	0,587	13	16,25
8	25	30	15	0,6	0,2	6	0,01	0,578	11,75	17,2
9	25	25	20	0,8	0,4	6	0,01	0,611	19,8	19,35
10	25	25	20	0,6	0,4	6	0,01	0,61	16,55	17,35
11	25	25	15	0,8	0,4	6	0,01	0,586	15,2	18
12	25	25	15	0,6	0,4	6	0,01	0,593	11,15	16,4
13	20	30	20	0,8	0,4	6	0,01	0,592	18,75	16,95
14	20	30	20	0,6	0,4	6	0,01	0,593	20,05	16,45
15	20	30	15	0,8	0,4	6	0,01	0,556	11,2	15,2
16	20	30	15	0,6	0,4	6	0,01	0,561	12,8	16,25
17	20	25	20	0,8	0,4	6	0,01	0,59	19,95	15,85
18	20	25	20	0,6	0,4	6	0,01	0,599	16,6	17,2
19	20	25	15	0,8	0,4	6	0,01	0,583	12,45	16,9
20	20	25	15	0,6	0,4	6	0,01	0,59	10,2	16,4
21	20	25	15	0,6	0,4	10	0,001	0,611	16,3	18,4

**Tabel 7.10 Rezultatele obținute pentru algoritmul CSURG, Scenariu\_1**

În tabelul 7.11 sunt ilustrate rezultatele obținute pentru cel de-al doilea scenariu. Pacientul utilizat pentru acest scenariu suferă de mai puține afecțiuni medicale comparativ cu pacientul primului scenariu și constrângerile nutriționale impuse de afecțiunile medicale sunt mai puține. De aceea, recomandările alimentare pentru acest scenariu sunt mai bune calitativ comparativ cu cele din primul scenariu. Cea mai bună configurație cu privire la funcția de fitness se află în rândul 3, la fel ca în cazul scenariului anterior.

Cea mai bună configurație obținută ca un compromis între fitness și timp apare în rândul 21, media numărului de iterații executate în cadrul acestei configurații fiind cea mai mare din tabel. Funcția de fitness pentru această configurație este cu 0,016 mai mică comparativ cu configurația pentru care se obțin cele mai bune rezultate în ceea ce privește funcția de fitness. În comparație cu aceeași configurație, timpul necesar procesării este cu aproximativ 0,17% mai mic.



No	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,678	25,9	16,3
2	25	30	20	0,8	0,2	6	0,01	0,669	23,4	18
3	25	30	20	0,6	0,4	6	0,01	0,685	24	18,65
4	25	30	20	0,6	0,2	6	0,01	0,683	24,15	19,2
5	25	30	20	0,8	0,4	6	0,01	0,649	21,15	14,6
6	25	30	15	0,8	0,2	6	0,01	0,656	22,25	17,45
7	25	30	15	0,6	0,4	6	0,01	0,656	20,25	17,05
8	25	30	15	0,6	0,2	6	0,01	0,644	16,25	15,05
9	25	25	20	0,8	0,4	6	0,01	0,656	21,55	16
10	25	25	20	0,6	0,4	6	0,01	0,670	21,85	16,5
11	25	25	15	0,8	0,4	6	0,01	0,660	18,7	17,6
12	25	25	15	0,6	0,4	6	0,01	0,641	15,5	14,3
13	20	30	20	0,8	0,4	6	0,01	0,675	25,4	17,25
14	20	30	20	0,6	0,4	6	0,01	0,660	22,85	15,45
15	20	30	15	0,8	0,4	6	0,01	0,641	16,2	13,8
16	20	30	15	0,6	0,4	6	0,01	0,644	16,7	14,25
17	20	25	20	0,8	0,4	6	0,01	0,658	20,3	15,25
18	20	25	20	0,6	0,4	6	0,01	0,656	18,7	13,95
19	20	25	15	0,8	0,4	6	0,01	0,646	16,05	15
20	20	25	15	0,6	0,4	6	0,01	0,646	15,85	14,65
21	20	25	15	0,6	0,4	10	0,001	0,669	20,1	19,6

**Tabel 7.11 Rezultatele obținute pentru algoritmul CSURG, Scenariu\_2**

În tabelul 7.12 pot fi observate rezultatele obținute pentru cel de-al treilea scenariu. În cadrul acestui scenariu este folosit același pacient ca în cazul scenariului anterior, de aceea meniurile alimentare recomandate în cazul acestor două scenarii sunt aproximativ la fel de bune calitativ, dar sunt mai bune calitativ în comparație cu cele recomandate în cadrul primului scenariu.

Cea mai bune valori ale funcției de fitness sunt obținute în urma folosirii configurației din rândul 4, timpul de procesare al acestor soluții aflându-se printre cele mai mari din tabel. În rândul 18 apare cea mai bună configurație obținută ca un compromis între fitness și timp, funcția de fitness a acesteia fiind cu doar 0,002 mai mică comparativ cu configurația din rândul 4. Comparativ cu aceeași configurație, timpul necesar procesării este cu aproximativ 0,20% mai mic. Pentru configurația aflată în rândul 21, media numărului de iterații executate este cea mai mare, fiind egală cu numărul maxim de iterații permise.

no	nolt	noNest	noCuck	trMut	repNest	noConstlt	diffIt	fit <sub>avg</sub>	t <sub>avg</sub> (sec)	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	0,669	31,85	18,4
2	25	30	20	0,8	0,2	6	0,01	0,678	30,3	18,95
3	25	30	20	0,6	0,4	6	0,01	0,652	22,2	14,65
4	25	30	20	0,6	0,2	6	0,01	0,679	29,6	19,6
5	25	30	20	0,8	0,4	6	0,01	0,656	22,2	18,2
6	25	30	15	0,8	0,2	6	0,01	0,662	18,75	16,3
7	25	30	15	0,6	0,4	6	0,01	0,656	21,1	16,45
8	25	30	15	0,6	0,2	6	0,01	0,647	20,1	16,5
9	25	25	20	0,8	0,4	6	0,01	0,674	27,05	17,25
10	25	25	20	0,6	0,4	6	0,01	0,673	27,35	17,85
11	25	25	15	0,8	0,4	6	0,01	0,644	20,1	15,75
12	25	25	15	0,6	0,4	6	0,01	0,648	20,8	14,65
13	20	30	20	0,8	0,4	6	0,01	0,672	28,45	16,9
14	20	30	20	0,6	0,4	6	0,01	0,658	23,9	15,1
15	20	30	15	0,8	0,4	6	0,01	0,646	19,8	15,65
16	20	30	15	0,6	0,4	6	0,01	0,634	17,3	13,5
17	20	25	20	0,8	0,4	6	0,01	0,660	24,9	15,75
18	20	25	20	0,6	0,4	6	0,01	0,677	23,45	16,65
19	20	25	15	0,8	0,4	6	0,01	0,655	19,65	17
20	20	25	15	0,6	0,4	6	0,01	0,650	17,45	15,4
21	20	25	15	0,6	0,4	10	0,001	0,659	23,25	20

Tabel 7.12 Rezultatele obținute pentru algoritmul CSURG, Scenariu\_3

Pentru toate cele trei scenarii pentru care a fost evaluat algoritmul CSURG, cele mai bune rezultate în privința funcției de fitness au fost obținute prin utilizarea a 30 de cuiburi, 20 de cucu și maxim 25 de iterații. Cele mai bune rezultate ca un compromis între funcția de fitness și timp au fost dobândite prin utilizarea a 25 de cuiburi și cel mult 20 de iterații.

Rezultate obținute în urma utilizării primului scenariu în cadrul algoritmului CSUCGM sunt ilustrate în tabelul 7.13. Cea mai bună configurație în ceea ce privește funcția de fitness se află în rândul 3, timpul de procesare pentru aceasta fiind printre cele mai mari din tabel. Cea mai bună configurație obținută ca un compromis între fitness și timp apare în rândul 18, funcția de fitness a acesteia fiind cu doar 0,01 mai mică comparativ cu configurația din rândul 3, iar timpul necesar procesării este cu aproximativ 0,25% mai mic. Media cea mai mare a numărului de iterații executate este obținută prin utilizarea configurației aflate în rândul 5.

No	nolt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	2	0,667	17,8	19,25
2	25	30	20	0,8	0,2	6	0,01	2	0,653	17,6	18,85
3	25	30	20	0,6	0,4	6	0,01	2	0,67	17,25	17,9
4	25	30	20	0,6	0,2	6	0,01	2	0,663	16,4	16,75
5	25	30	15	0,8	0,4	6	0,01	2	0,661	16,8	21,55
6	25	30	15	0,8	0,2	6	0,01	2	0,65	13,75	21,05
7	25	30	15	0,6	0,4	6	0,01	2	0,669	16,35	20,05
8	25	30	15	0,6	0,2	6	0,01	2	0,658	13,45	19,6
9	25	25	20	0,8	0,4	6	0,01	2	0,652	15,95	18,65
10	25	25	20	0,6	0,4	6	0,01	2	0,646	14,5	16,95
11	25	25	15	0,8	0,4	6	0,01	2	0,626	12,95	18,95
12	25	25	15	0,6	0,4	6	0,01	2	0,64	12,5	17,2
13	20	30	20	0,8	0,4	6	0,01	2	0,662	17,1	18,25
14	20	30	20	0,6	0,4	6	0,01	2	0,665	16,75	16,65
15	20	30	15	0,8	0,4	6	0,01	2	0,648	15	18,9
16	20	30	15	0,6	0,4	6	0,01	2	0,652	14,3	17,2
17	20	25	20	0,8	0,4	6	0,01	2	0,65	14,75	17,45
18	20	25	20	0,6	0,4	6	0,01	2	0,66	12,85	14,75
19	20	25	15	0,8	0,4	6	0,01	2	0,632	12,7	19,05
20	20	25	15	0,6	0,4	6	0,01	2	0,638	10,5	16,65
21	20	25	15	0,6	0,4	10	0,001	2	0,649	17,1	18,55
22	20	25	15	0,6	0,4	6	0,01	5	0,626	10,8	14,9

**Tabel 7.13 Rezultatele obținute pentru algoritmul CSUCGM, Scenariu\_1**

În tabelul 7.14 sunt ilustrate rezultatele obținute pentru cel de-al doilea scenariu. La fel ca în cazul celorlalte variante ale algoritmului, recomandările alimentare pentru acest scenariu sunt mai bune calitativ comparative cu cele din primul scenariu. Cea mai bună configurație cu privire la funcția de fitness se află în rândul 3, la fel ca în cazul scenariului anterior.

Cea mai bună configurație obținută ca un compromis între valoarea funcția fitness și timp apare în rândul 10, funcția de fitness a acesteia fiind cu 0,02 mai mică comparativ cu configurația din rândul 3, iar timpul necesar procesării este cu aproximativ 0,25% mai mic. Pentru configurația aflată în rândul 21, media numărului de iterații executate este cea mai mare, fiind egală cu numărul maxim de iterații permise.

No	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	2	0,721	20,2	17,95
2	25	30	20	0,8	0,2	6	0,01	2	0,724	16,95	18,25
3	25	30	20	0,6	0,4	6	0,01	2	0,751	16,8	18
4	25	30	20	0,6	0,2	6	0,01	2	0,738	15,15	16,2
5	25	30	15	0,8	0,4	6	0,01	2	0,711	15,55	21,45
6	25	30	15	0,8	0,2	6	0,01	2	0,699	10,9	20
7	25	30	15	0,6	0,4	6	0,01	2	0,725	12,4	17,55
8	25	30	15	0,6	0,2	6	0,01	2	0,711	9,75	17,5
9	25	25	20	0,8	0,4	6	0,01	2	0,731	13,8	20,35
10	25	25	20	0,6	0,4	6	0,01	2	0,731	12,3	17
11	25	25	15	0,8	0,4	6	0,01	2	0,712	10,8	19,05
12	25	25	15	0,6	0,4	6	0,01	2	0,715	9,95	16,65
13	20	30	20	0,8	0,4	6	0,01	2	0,721	13,9	18
14	20	30	20	0,6	0,4	6	0,01	2	0,724	12,9	16,2
15	20	30	15	0,8	0,4	6	0,01	2	0,700	11,9	18,45
16	20	30	15	0,6	0,4	6	0,01	2	0,725	12,05	17,95
17	20	25	20	0,8	0,4	6	0,01	2	0,709	11	16,1
18	20	25	20	0,6	0,4	6	0,01	2	0,723	10,6	14,85
19	20	25	15	0,8	0,4	6	0,01	2	0,711	10,75	18,65
20	20	25	15	0,6	0,4	6	0,01	2	0,720	10,05	16,7
21	20	25	15	0,6	0,4	10	0,001	2	0,719	11,8	20
22	20	25	15	0,6	0,4	6	0,01	5	0,714	12	16,1

Tabel 7.14 Rezultatele obținute pentru algoritmul CSUCGM, Scenariu\_2

În tabelul 7.15 sunt prezentate rezultatele obținute pentru cel de-al treilea scenariu. La fel ca în cazul celorlalte variante ale algoritmului, recomandările alimentare pentru acest scenariu sunt mai bune calitativ comparative cu cele din primul scenariu. Cea mai bună configurație cu privire la funcția de fitness se află în rândul 3, timpul necesar procesării acesteia fiind cel mai mare din tabel. Cea mai bună configurație obținută ca un compromis între valoarea funcția fitness și timpul necesar procesării apare în rândul 20, funcția de fitness a acesteia fiind cu doar 0,002 mai mică comparativ cu configurația din rândul 3, iar timpul necesar procesării este cu aproximativ 0,30% mai mic.

Pentru toate cele trei scenarii pentru care a fost evaluat algoritmul CSUCGM, cele mai bune rezultate în privința funcției de fitness au fost obținute prin utilizarea configurației aflate în rândul 3. Cele mai bune rezultate ca un compromis între funcția de fitness și timp au fost dobândite prin utilizarea a 25 de cuiburi.

no	noIt	noNest	noCuck	trMut	repNest	noConstIt	diffIt	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
1	25	30	20	0,8	0,4	6	0,01	2	0,736	18,7	19,3
2	25	30	20	0,8	0,2	6	0,01	2	0,732	19,3	20,2
3	25	30	20	0,6	0,4	6	0,01	2	0,738	19,6	16,2
4	25	30	20	0,6	0,2	6	0,01	2	0,726	15,65	15,7
5	25	30	15	0,8	0,4	6	0,01	2	0,706	14	19,9
6	25	30	15	0,8	0,2	6	0,01	2	0,728	12,55	20,65
7	25	30	15	0,6	0,4	6	0,01	2	0,730	13,55	18,15
8	25	30	15	0,6	0,2	6	0,01	2	0,725	9,85	16
9	25	25	20	0,8	0,4	6	0,01	2	0,719	15,75	19,1
10	25	25	20	0,6	0,4	6	0,01	2	0,731	12,2	13,75
11	25	25	15	0,8	0,4	6	0,01	2	0,709	12,7	18,6
12	25	25	15	0,6	0,4	6	0,01	2	0,718	14,8	19,7
13	20	30	20	0,8	0,4	6	0,01	2	0,717	15,55	17,3
14	20	30	20	0,6	0,4	6	0,01	2	0,725	13,95	16,05
15	20	30	15	0,8	0,4	6	0,01	2	0,707	12,95	17,95
16	20	30	15	0,6	0,4	6	0,01	2	0,715	13	17,1
17	20	25	20	0,8	0,4	6	0,01	2	0,728	15,05	18,45
18	20	25	20	0,6	0,4	6	0,01	2	0,710	11,55	14,25
19	20	25	15	0,8	0,4	6	0,01	2	0,692	10,5	16,65
20	20	25	15	0,6	0,4	6	0,01	2	0,736	11,35	16,75
21	20	25	15	0,6	0,4	10	0,001	2	0,733	13,25	19,95
22	20	25	15	0,6	0,4	6	0,01	5	0,712	11,05	15,65

Tabel 7.15 Rezultatele obținute pentru algoritmul CSUCGM, Scenariu\_3

#### 7.4. Metrice de evaluare a performanței algoritmului

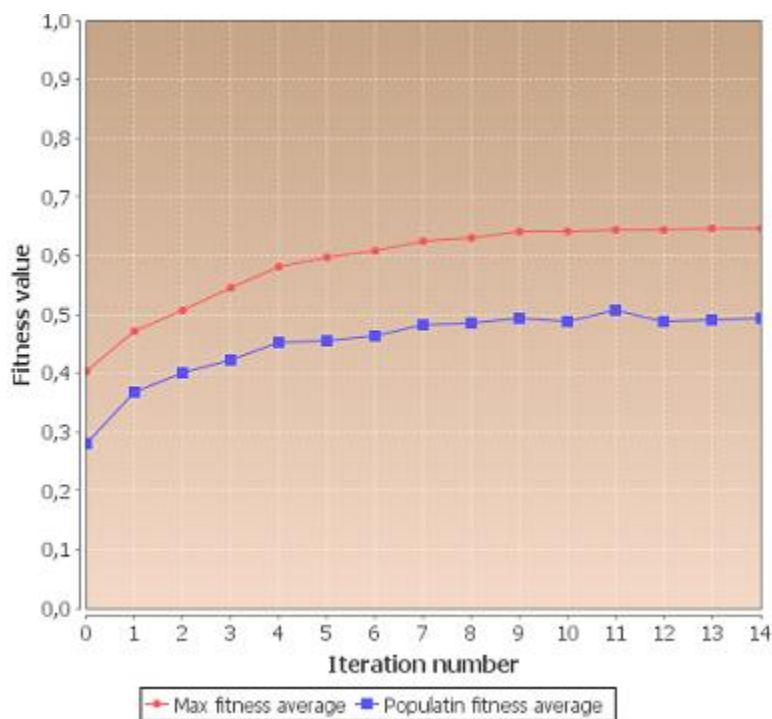
Pentru a ilustra eficiența algoritmilor s-au generat grafice, care, conform[20], prezintă următoarele tipuri de informații:

- media celor mai bune valorile ale funcției de fitness în timpul iterațiilor;
- media valorilor funcției de fitness a întregii populații de cuiburi în timpul iterațiilor;

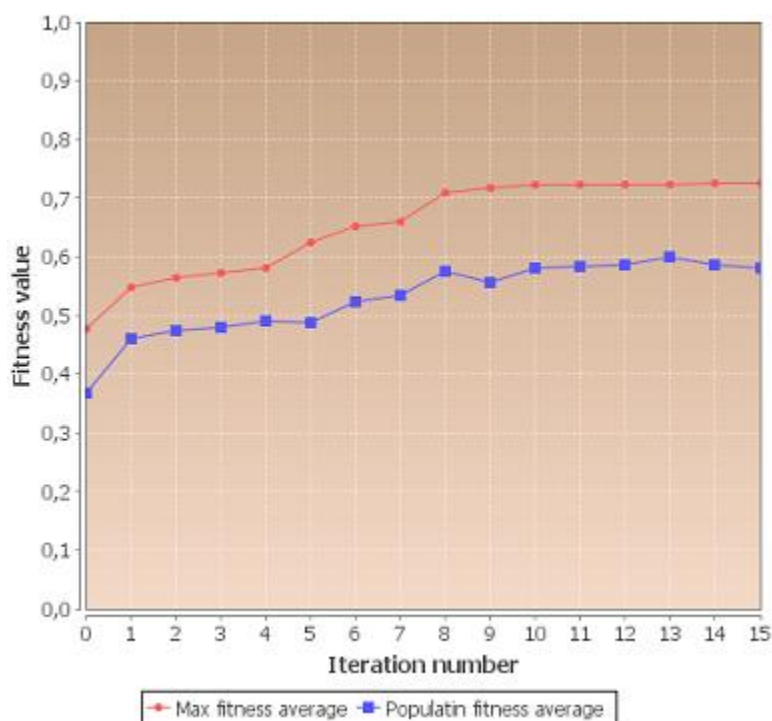
Graficele au fost colectate pentru fiecare scenariu, pentru cea mai bună configurație, din punctul de vedere al funcției de fitness, prezentate în secțiunea 7.3.

Pentru primul scenariu configurația cea mai bună se află pe rândul cu numărul 2 al algoritmului CSUCGM. Se poate observa în graficul din figura 7.1 diferența mare între media funcției de fitness a tuturor cuiburilor și media funcției de fitness a celor mai bune cuiburi. De asemenea, se poate remarca faptul că există iterații în urma cărora media funcției de fitness a tuturor cuiburilor este micșorată deoarece la sfârșitul fiecărei iterații un procent relativ mare din populația

cuiburilor este înlocuită cu noi cuiburi generate aleator. După 14 iterații algoritmul se oprește deoarece au existat 6 iterații care nu au adus îmbunătățiri considerabile celor mai bune cuiburi.

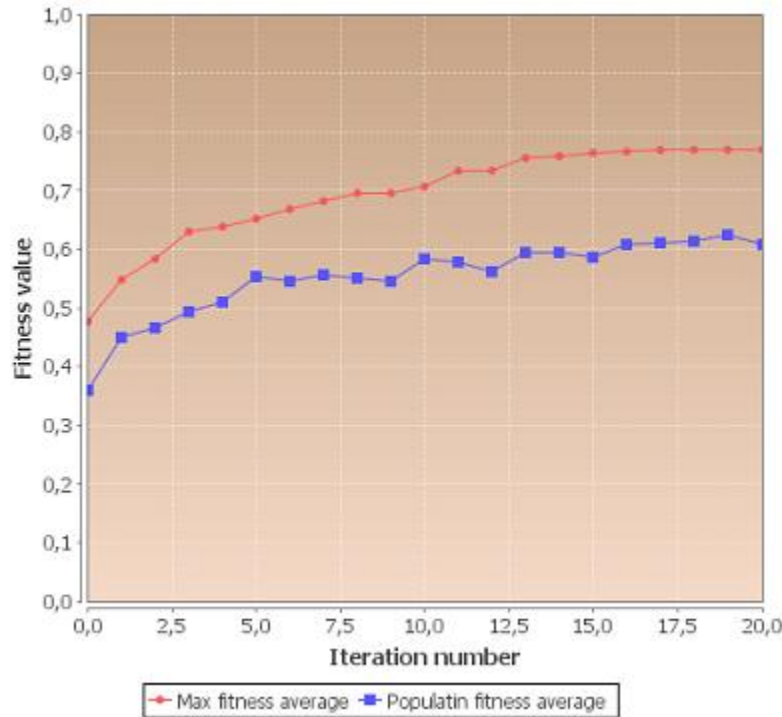


**Figura 7.1** Graficul obținut în urma executării celei mai bune configurații a algoritmul CSUCGM pentru Scenariu\_1



**Figura 7.2** Graficul obținut în urma executării celei mai bune configurații a algoritmul CSUCGM pentru Scenariu\_2

Pentru cel de-al doilea scenariu configurația cea mai bună se află pe rândul cu numărul 2 al algoritmului CSUCGM, la fel ca în cazul primului scenariu. Din graficul din figura 7.2 se poate observa diferența mare între media funcției de fitness a tuturor cuiburilor și media funcției de fitness a celor mai bune cuiburi. Această diferență se mărește de la o iterație la alta. La fel ca în scenariul anterior, se poate remarca faptul că există iterații în urma cărora media funcției de fitness a tuturor cuiburilor scade. Spre deosebire de scenariul anterior, în cadrul acestui scenariu, după etapa de inițializare valoarea funcțiilor de fitness ale cuiburilor este mai mare.



**Figura 7.3** Graficul obținut în urma executării celei mai bune configurații a algoritmul CSUCGM pentru Scenariu\_3

Graficul celui de-al treilea scenariu este ilustrat în figura 7.3. Acesta este obținut în urma aceleiași configurații ca în cazul scenariilor precedente. Evoluția valorilor funcției de fitness este asemănătoare cu cea a scenariilor prezentate anterior, dar spre deosebire de acestea, în cadrul acestui grafic algoritmul execută un număr mai mare de iterații.

## 7.5. Analiză comparativă

### 7.5.1. Analiză comparativă a versiunilor tehnica inspirate din comportamentul cucilor

Pentru a putea realiza analiza comparativă se vor lua în considerare rezultatele obținute în secțiunea 6.3. Se compară rezultatele obținute pentru cele mai bune configurații ale parametrilor ajustabili, atât din punctul de vedere al valorii funcției de fitness, cât și în ceea ce privește compromisul între funcția de fitness și timpul de procesare. Această comparație este efectuată pentru toate cele trei scenarii prezentate anterior.

Pentru cazul primului scenariu, din tabelul 7.16 se observă că în ceea ce privește funcția de fitness, cele mai bune soluții se obțin prin utilizarea algoritmului CSUCGM, iar cele mai slabe prin algoritmului CS. Valorile obținute în urma rulării algoritmilor CSU și CSURG sunt aproximativ aceleași deoarece diferența între acești doi algoritmi este una minoră: în cadrul primului algoritm în se realizează mutații doar dacă componentele soluției diferă de cele ale soluției optime din acea iterație. Un număr relativ mic de componente sunt identice cu cele ale soluției optime.

algoritm	nolt	noNest	noCuck	trMut	repNest	noConstit	diffit	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
<b>CS</b>	25	30	20	0,6	0,4	6	0,01	-	0,507	5,9	11,85
<b>CSU</b>	25	30	20	0,8	0,2	6	0,01	-	0,619	21,8	19,65
<b>CSURG</b>	25	30	20	0,6	0,4	6	0,01	-	0,617	17,85	17,55
<b>CSUCGM</b>	25	30	20	0,6	0,4	6	0,01	2	0,67	17,25	17,9

**Tabel 7.16 Configurațiile cu cele mai bune valori ale funcției de fitness, Scenariu\_1**

În tabelul 7.17 se poate observa faptul că valorile funcției de fitness și ale timpului de procesare se îmbunătățesc cu fiecare versiune. De asemenea, în urma ultimei hibridizări este adusă o îmbunătățire în ceea ce privește timpul de procesare.

algoritm	nolt	noNest	noCuck	trMut	repNest	noConstit	diffit	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
<b>CS</b>	20	25	20	0,6	0,4	6	0,01	-	0,499	3,1	12,85
<b>CSU</b>	20	25	20	0,8	0,4	6	0,01	-	0,604	16,55	17,55
<b>CSURG</b>	20	25	15	0,6	0,4	10	0,001	-	0,611	16,3	18,4
<b>CSUCGM</b>	20	25	20	0,6	0,4	6	0,01	2	0,66	12,85	14,75

**Tabel 7.17 Configurațiile cu cele mai bune valori ale funcției de fitness și ale timpului de procesare, Scenariu\_1**

Tabelul 7.18 prezintă cele mai bune configurații ale scenariului 2. Se poate observa creșterea progresivă a funcției de fitness. La fel ca în cazul scenariului anterior, valoarea funcției de fitness a algoritmului CSURG nu este semnificativ mai bună decât cea a versiunii anterioare.

algoritm	nolt	noNest	noCuck	trMut	repNest	noConstit	diffit	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
<b>CS</b>	25	30	20	0,8	0,4	6	0,01	-	0,588	6,65	15,15
<b>CSU</b>	25	25	20	0,8	0,4	6	0,01	-	0,679	25,6	17,8
<b>CSURG</b>	25	30	20	0,6	0,4	6	0,01	-	0,685	24	18,65
<b>CSUCGM</b>	25	30	20	0,6	0,4	6	0,01	2	0,751	16,8	18

**Tabel 7.18 Configurațiile cu cele mai bune valori ale funcției de fitness, Scenariu\_2**

În tabelul 7.19 se poate observa o îmbunătățire a valorii funcției de fitness în raport cu timpul de procesare pe parcursul celor patru versiuni. Comparativ cu versiunea inițială, ultima versiune a algoritmului are funcția de fitness cu 0,147 mai bună iar timpul este de peste trei ori



mai mare. Timpul de procesare al algoritmului CSUCGM este cu aproximativ 70% mai bun decât celelalte două versiuni anterioare ale algoritmului.

algoritm	nolt	noNest	noCuck	trMut	repNest	noConstlt	diffIt	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
CS	20	30	15	0,6	0,4	6	0,01	-	0,584	3,75	11,75
CSU	20	25	20	0,8	0,4	6	0,01	-	0,673	22,1	15,85
CSURG	20	25	15	0,6	0,4	10	0,001	-	0,669	20,1	19,6
CSUCGM	25	25	20	0,6	0,4	6	0,01	2	0,731	12,3	17

**Tabel 7.19** Cofigurațiile cu cele mai bune valori ale funcției de fitness și ale timpului de procesare, Scenariu\_2

Tabelul 7.20 prezintă cele mai bune configurații ale scenariului 3. La fel ca în cazul scenariului anterior, valoarea funcției de fitness a algoritmului CSURG este aproximativ egală cu cea a versiunii anterioare. De asemenea, se observă îmbunătățiri în ceea ce privește funcția de fitness între prima versiune și ultima versiune, dar timpul ultimei versiuni este semnificativ mai mare.

algoritm	nolt	noNest	noCuck	trMut	repNest	noConstlt	diffIt	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
CS	20	25	15	0,6	0,4	10	0,001	-	0,592	5,2	19,25
CSU	25	30	20	0,8	0,2	6	0,01	-	0,683	26,5	18,45
CSURG	25	30	20	0,6	0,2	6	0,01	-	0,679	29,6	19,6
CSUCGM	25	30	20	0,6	0,4	6	0,01	2	0,738	19,6	16,2

**Tabel 7.20** Cofigurațiile cu cele mai bune valori ale funcției de fitness, Scenariu\_3

În tabelul 7.21 sunt ilustrate cele mai bune configurații cu privire la valoarea funcției de fitness în raport cu timpul de procesare. Se observă o îmbunătățire semnificativă a funcției de fitness pe parcursul celor patru versiuni. Cea mai bună configurație în ceea ce privește un compromis între funcția de fitness și timpul de procesare este cea a algoritmului CSUCGM.

algoritm	nolt	noNest	noCuck	trMut	repNest	noConstlt	diffIt	it <sub>tabu</sub>	fit <sub>avg</sub>	t <sub>avg(sec)</sub>	it <sub>avg</sub>
CS	20	30	20	0,6	0,4	6	0,01	-	0,579	4,25	12,05
CSU	25	30	20	0,6	0,2	6	0,01	-	0,657	20,65	14,8
CSURG	20	25	20	0,6	0,4	6	0,01	-	0,677	23,45	16,65
CSUCGM	20	25	15	0,6	0,4	6	0,01	2	0,736	11,35	16,75

**Tabel 7.21** Cofigurațiile cu cele mai bune valori ale funcției de fitness și ale timpului de procesare, Scenariu\_3

**7.5.2. Analiză comparativă între tehnica hibridă inspirată din comportamentul cucilor și tehnica hibridă inspirată din comportamentul cucilor**

## 8. Manualul de instalare și utilizare

### 8.1. Manual de instalare

Există mai multe perspective în ceea ce privește instalarea aplicației în funcție de tipul persoanei ce dorește să efectueze instalarea:

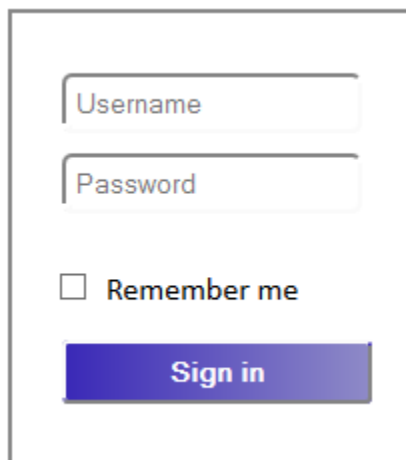
1. Utilizatorul: folosește aplicația web pentru a genera meniuri alimentare personalizate. Acesta trebuie să aibă acces la un server care găzduiește aplicația. De asemenea, acesta are nevoie de un browser web ca Firefox sau Chrome.
2. Integratorul de sistem și dezvoltatorul software: folosesc codul sursă a aplicației, așadar este necesar ca aceștia să utilizeze serverul Glassfish pe deploy pentru cele două componente web (serviciul web și componenta ce conține tehnica hibridă). Toate cele trei componente au fost realizate folosind NetBeans, deci se pot importa în acest IDE astfel: File -> Open Project. Înaintea rulării componentei de generare de profile medicale trebuie efectuată următoarea configurare: în fișierul aflat în locația `PROJECT_PATH/src/properties/configuration-file.config` este necesară setarea căii către fișierul cu extensia owl ce conține ontologia. De asemenea, este necesară adăugarea bibliotecilor Jena la proiect. Acestea pot fi adăugate astfel: se selectează din NetBeans fișierul Libraries din cadrul proiectului, se execută click dreapta pe acest fișier, se alege opțiunea Add JAR/Folder, se navighează și se selectează bibliotecile Jena, apoi se apasă Open. Componenta poate fi rulată prin selectarea proiectului în NetBeans, executarea click dreapta pe acest proiect și alegerea opțiunii Run. Fișierele XML generate se vor afla în locația `PROJECT_PATH/medicalProfiles`, unde `PROJECT_PATH` reprezintă calea către fișierul ce conține componenta de generare de profile medicale. Pentru a putea utiliza serviciul web este necesară executarea configurării precizate anterior (setarea căii spre ontologie și adăugarea bibliotecilor necesare). Apoi, pentru a face deploy serviciului web acesta trebuie selectat în NetBeans, executat click dreapta pe proiect și aleasă opțiunea Deploy. Instalarea componentei principale presupune utilizarea instrumentului Maven pentru build. Acesta va descărca în mod dinamic bibliotecile necesare. Este foarte important ca serviciul web să fie pornit înainte ca aplicația principală să fie rulată. Pentru a face deploy aplicației se vor executa aceiași pași ca în cazul serviciului web.

### 8.2. Manual de utilizare

Prima pagină din cadrul aplicației de generare de meniuri recomandate este cea aferentă autentificării utilizatorului. Aceștia se vor autentifica prin intermediul unui username și a unei parole. Formularul de autentificare poate fi observat în figura 8.1. În cazul în care credențialele introduse de utilizator nu sunt corecte va fi afișat un mesaj de eroare, iar autentificarea nu va avea loc.

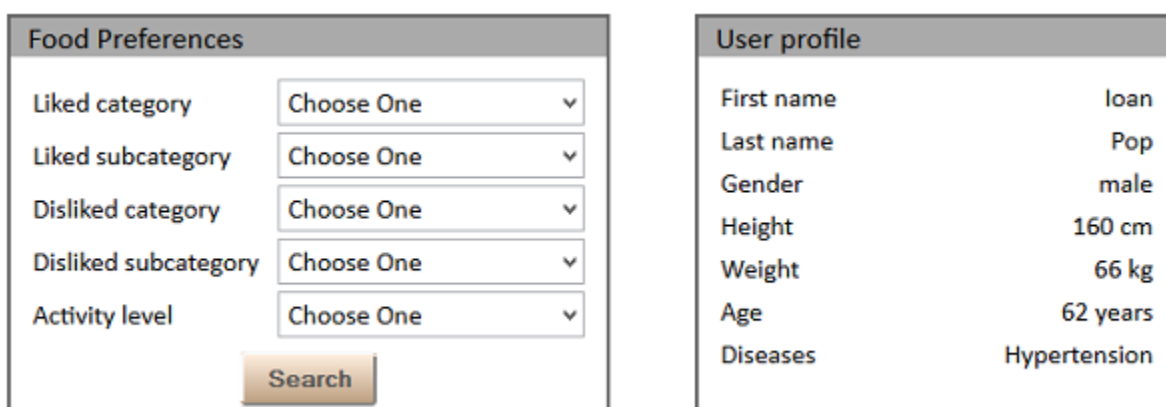
Dupa autentificare utilizatorul va fi redirecționat la pagina principală a aplicației, unde va fi afișat profilul personal al acestuia. Tot în cadrul acestei pagini, utilizatorului i se oferă posibilitatea de a selecta categoria și subcategoria preferată, precum și cea pe care nu dorește să o consume. De asemenea, utilizatorul trebuie să specifice și nivelul său de activitate, apoi poate apăsa butonul „Search” pentru generarea meniurilor personalizate. În cazul în care utilizatorul nu selectează cel puțin nivelul de activitate, categoria preferată și categoria pe care nu dorește să o

consume va fi afișat un mesaj de atenționare, iar generarea meniurilor recomandate nu va fi efectuată. În figura 8.2 sunt ilustrate informațiile care apar după autentificarea utilizatorului.



Formularul de autentificare este prezentat într-un cadru alb cu o bordură gri. Conține două câmpuri de text: 'Username' și 'Password', ambele cu text gri. Sub acestea este un câmp de checkbox cu eticheta 'Remember me'. În partea de jos este un buton albastru cu textul 'Sign in' în alb.

**Figura 8.1 Formularul de autentificare**



Informațiile ce apar după autentificarea utilizatorului sunt prezentate în două panouri separate.

**Food Preferences**

Liked category	Choose One
Liked subcategory	Choose One
Disliked category	Choose One
Disliked subcategory	Choose One
Activity level	Choose One

Search

**User profile**

First name	Ioan
Last name	Pop
Gender	male
Height	160 cm
Weight	66 kg
Age	62 years
Diseases	Hypertension

**Figura 8.2 Informațiile ce apar după autentificarea utilizatorului**

După efectuarea generării meniurilor în pagină vor fi adăugate următoarele secțiuni:

- necesarul zilnic nutrițional al pacientului;
- meniurile recomandate împreună cu cantitatea de nutrienți conținută de acestea;
- un grafic care prezintă evoluția funcției maxime de fitness și evoluția funcției de fitness a populației în cadrul iterațiilor algoritmului;

Figura 8.3 prezintă o parte din informațiile afișate pe pagina principală după generarea meniurilor: cantitatea zilnică de nutrienți necesară și meniurile personalizate împreună cu cantitatea de nutrienți conținută de acestea. Utilizatorul are posibilitatea de a vizualiza mai multe meniuri prin intermediul butoanelor „Next” și „Previous”. Aceste meniuri sunt aranjate în mod descrescător în funcție de valoarea funcției de fitness, utilizatorul putând alege meniul care i se potrivește mai bine. În cazul în care utilizatorul nu este mulțumit de meniurile generate acesta poate apăsa din nou butonul „Search”. De asemenea, acesta își poate modifica preferințele alese la pasul anterior.

Daily nutritional recommendations			
Calories		2,032.28 Kcal	
Proteins		139.719 g	
Fats		56.452 g	
Carbohydrates		228.632 g	
Iron		8 - 45 mg	
Sodium		500 - 1,500 mg	
VitaminA		1.2 - 7.5 mg	
VitaminB		1.6 - 5 mg	
VitaminC		75 - 1,000 mg	

Recommended menus			
Fitness: 0.772		<b>Nutrients</b>	
Total price: 60.69		Calories	1,876.67 Kcal
Maximum time: 76		Proteins	132.668 g
Scrambled eggs and orange juice	Provider_13	Fats	52.296 g
Cottage cheese with cherry gem	Provider_49	Carbohydrates	213.438 g
Greek sour soup with beef	Provider_31	Sodium	707.735 mg
Boiled potatoes with grilled sirloin beef	Provider_13	Iron	19.618 mg
Pancakes with cherry jam	Provider_17	Vitamin A	1.4 mg
Blueberry	Provider_15	Vitamin B	2.927 mg
Boiled potatoes with grilled sirloin pork	Provider_35	Vitamin C	201.865 mg

Previous

Next

Figura 8.3 Doza zilnică de nutrienți recomandată pentru un utilizator și meniurile generate

## 9. Concluzii și dezvoltări ulterioare

În această lucrare a fost propusă o tehnică hibridă inspirată din biologie pentru rezolvarea problemei de generare de meniuri alimentare personalizate, precum și metode de testare și analizare a acestei tehnici.

Tehnica propusă se numește Tehnică hibridă inspirată din comportamentul de reproducere al cucilor și combină avantajele următoarelor componente: componenta bazată pe comportamentul cucilor, componenta bazată pe principiile geneticii și ale selecției naturale și componenta bazată pe strategii de căutare Tabu. Aceste componente au ca sursă de inspirație atât elementele naturale, cât și elemente ce nu sunt inspirate din natură. Tehnica hibridă a fost dezvoltată în mod incremental, astfel au fost implementate mai multe versiuni ce folosesc diverse componente hibride.

Toate versiunile tehnicii propuse au fost testate pentru diverse scenarii și combinații ale parametrilor ajustabili. Criteriile care au fost luate în calcul pentru analiza acestor tehnici sunt: media valorii funcției de fitness a celor mai bune soluții, timpul de procesare și numărul de iterații executate. Pentru ca rezultatele obținute să fie cât mai apropiate de cele reale, pentru fiecare configurație a parametrilor ajustabili a fost rulată versiunea tehnicii de douăzeci de ori și s-a făcut o medie a rezultatelor obținute în urmă rulărilor.

Astfel prin dezvoltarea acestei tehnici s-au obținut soluții care respectă necesarul nutrițional al organismului într-un timp foarte scurt comparativ cu cel necesar rulării metodei exhaustive. De asemenea, prin hibridizările aduse componentei de bază s-au observat îmbunătățiri considerabile ale valorii funcției de fitness.

Tehnica inspirată din comportamentul cucilor ar putea fi îmbunătățită prin determinarea în mod dinamic a numărului de cuiburi înlocuite la sfârșitul fiecărei iterații. Acest număr ar putea depinde de numărul de cuiburi care au fost înlocuite cu soluțiile ouălor de cuci în cadrul unei iterații. În urma testării și analizării s-au observat avantajele și dezavantajele aduse de fiecare componentă. Punctele slabe ale componentelor pot fi diminuate prin efectuarea de modificări sau adăugarea de noi hibridizări.

Aplicația de generare de meniuri recomandate poate fi dezvoltată prin:

- posibilitatea de generare de meniuri pentru diverse perioade de timp (o singură masă sau mai multe zile);
- adăugarea unui criteriu suplimentar pentru generarea recomandărilor: istoricul medical al pacientului care să conțină evoluția stării de sănătate a pacientului (de exemplu: creșterea în greutate, scăderea cantității de calciu din organism);
- adăugarea unui nou tip de utilizator: cadru medical.

De asemenea, analiza și testarea aplicației ar putea fi îmbunătățită prin adăugarea unor criterii suplimentare de evaluare (de exemplu: numărul total de soluții procesate).

## Bibliografie

- [1] S. Mika, Challenges for Nutrition Recommender Systems, Context Aware Intelligent Assistance, 2011.
- [2] P. Nachtigall, G. Geleihnse, J. Hoonhout, A. van Halteren, N. Gros, Toward an Intelligent Nutritional Support System, CHI 2010 Workshop on Wellness Informatics, 2010.
- [3] Y. Usthasopha, S. Phimoltares, N. Cooharajanane, Nutrition Counseling System and Food Menu Planning, International Conference on Information Science and Applications, 2010.
- [4] S. Sivilai, C. Snae, M. Brückner, Ontology-Driven Personalized Food and Nutrition Planning System for the Elderly, The 2nd International Conference in Business Management and Information Sciences, 2012.
- [5] T. Kashima, S. Matsumoto, H. Ishii. Decision support system for menu recommendation using rough sets, International Journal of Innovative Computing, Information and Control, v. 7, p. 2799-2808, 2011.
- [6] B. Gaál, I. Vassányi, G. Kozmann, A Novel Artificial Intelligence Method for Weekly Dietary Menu Planning, Methods of Information in Medicine, v. 44, p. 655-664, 2005.
- [7] J. Bulka, A. Izworski, J. Koleszynska, J. Lis, I. Wochlik, Automatic meal planning using artificial intelligence algorithms in computer aided diabetes therapy, Proceedings of the 4th International Conference on Autonomous Robots and Agents, 2009.
- [8] X. S. Yang, Engineering Optimization - an Introduction with Metaheuristic Applications, Cap. 2, 18, p. 15-19, 233-239, 2010.
- [9] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys, 2003.
- [10] C. Blum, A. Blesa, J. Maria, A. Roli, M. Sampels, Hybrid Metaheuristics - An Emerging Approach to Optimization, v. 114, Cap. 1, 2, p. 17-20, 33, 2008.
- [11] D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, M. Zaidi, The Bees Algorithm - A Novel Tool for Complex Optimisation Problems, Proceedings of IPROMS 2006 Conference, 2006.
- [12] N. Guarino, D. Oberle, S. Staab, 'What Is an Ontology?' in S Staab & R Studer (eds), Handbook on Ontologies (International Handbooks on Information Systems) second edition, p. 1-20, 2009.
- [13] M. Hepp, P. de Leenheer, A. de Moor, Y. Sure, Ontology Management: Semantic Web, Semantic Web Services, and Business Applications, p. 3-22, 2008.
- [14] M. Graur, Ghid pentru alimentatie sanatoasa, Performatica, 2006.
- [15] X.-S. Yang, S. Deb, Cuckoo search via Lévy flights, World Congress on Nature & Biologically Inspired Computing, 2009.
- [16] F. Glover, J. Kelly, M. Laguna, Genetic algorithms and tabu search: Hybrids for optimization, Computers & Operations Research, v. 22, p. 111-134, 1995.
- [17] M. Stibich, Salt and High Blood Pressure, <http://longevity.about.com/od/abouthighbloodpressure/p/sodium.htm>.

- [18] AMERICAN Diabetes Association, Help for Diabetics? Healthy eating politics, <<http://www.healthy-eating-politics.com/american-diabetes-association.html>>.
- [19] Good Food Romania, <<http://www.goodfood.ro/>>.
- [20] D. Floreano, C. Mattiussi, Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies (Intelligent Robotics and Autonomous Agents series), MIT Press, Cap. 1, p. 29-33, 2008.



## Acronime

- SOAP - Simple Object Access Protocol
- XML - Extensible Markup Language
- UML - Unified Modelling Language
- HTTP - Hypertext Transfer Protocol
- IDE - Integrated Development Environment
- API - Application Programming Interface
- JAR - Java ARchive
- WAR - Web application ARchive
- CS - tehnica de căutare inspirată din comportamentul cucilor
- CSU - tehnica de căutare inspirată din comportamentul cucilor îmbunătățită
- CSURG - tehnica hibridă inspirată din comportamentul cucilor și operatorul genetic
- CSUCGM - tehnica hibridă inspirată din comportamentul cucilor, operatorul genetic și structure de memorie Tabu

## Anexe

### Anexa A

<!--Secțiune din fișierul pom.xml ce conține dependențele proiectului și o parte din specificațiile de build-->

```
<?xml version="1.0" encoding="UTF-8"?>
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>diet4elders</groupId>
  <artifactId>DietForElders</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>DietForElders</name>
  <properties>
    <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.wicket</groupId>
      <artifactId>wicket-core</artifactId>
      <version>6.9.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.wicket</groupId>
      <artifactId>wicket-spring</artifactId>
      <version>6.9.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.wicket</groupId>
      <artifactId>wicket-extensions</artifactId>
      <version>6.9.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.wicket</groupId>
```

```
<artifactId>wicket-auth-roles</artifactId>
<version>6.9.1</version>
</dependency>
<dependency>
  <groupId>jfree</groupId>
  <artifactId>jfreechart</artifactId>
  <version>1.0.13</version>
</dependency>
</dependencies>
<build>
  <resources>
    <resource>
      <filtering>>false</filtering>
      <directory>src/main/resources</directory>
    </resource>
    <resource>
      <filtering>>false</filtering>
      <directory>src/main/java</directory>
      <includes>
        <include>**</include>
      </includes>
      <excludes>
        <exclude>**/*.java</exclude>
      </excludes>
    </resource>
  </resources>
</build>
</project>
```

**Anexa B**

```
/*
 * Tehnica bazată pe comportamentul cucilor (fragmente)
 */
package diet4elders.dietforelders.searchAlgorithm;

import
diet4elders.dietforelders.searchAlgorithm.hybridcomponent.GeneticAlgorithmComponent;
import diet4elders.dietforelders.model.FoodSolution;
import diet4elders.dietforelders.model.Solution;
import diet4elders.dietforelders.service.AlgorithmConfigurationService;
import diet4elders.dietforelders.service.ApplicationService;
import diet4elders.dietforelders.service.SolutionService;

/**
 *
 * @author Mada
 */
public class CuckooAlgorithm extends Metaheuristic {

    protected double mutateThreshold;
    protected intnumberOfIterations;
    protected booleanhasParameters;
    protected intnumberOfCuckoo;
    protected double percentageOfReplacedContainers;
    protected GeneticAlgorithmComponentgenAlgComponent;
    protected Population cuckoos;
    protected intbestSolutionNr;
    protected intnoUpdatedCont;
    protected intnoOfIterationWithConstantFitness;
    protected intmaxNoOfIterWithConstFitness;
    protected double minDiffIt;
    protected double firstConstantSolutionFitness;

    public CuckooAlgorithm() {
        population = new Population();
        bestSol = new FoodSolution();
        hasParameters = false;
        genAlgComponent = new GeneticAlgorithmComponent(this.optimProblem);
    }

    @Override
    public void execute() {
        executeInitializationStep();
    }
}
```

```

intiterationNo = 1;
while (iterationNo <= numberOfIterations && noOfIterationWithConstantFitness <
maxNoOfIterWithConstFitness - 1) {
    Solution selectedCuckoo = cuckoos.getSolutionSet().get(ApplicationService.
getRandom().nextInt(cuckoos.getSolutionSet().size()));
    selectedCuckoo = updateCuckooSolution(selectedCuckoo);

    Solution selectedContainer = population.getSolutionSet().get(ApplicationService.
getRandom().nextInt(population.getSolutionSet().size()));
    if (selectedCuckoo.getFitnessValue() > selectedContainer.getFitnessValue()) {
        if (!population.getSolutionSet().contains(selectedCuckoo)) {
            replaceContainerSolution(selectedCuckoo, selectedContainer);
        }
    }
    prepareTheNextIteration(iterationNo);
    iterationNo++;
}
actualNoOfIterations = iterationNo - 1;
}

protected void prepareTheNextIteration(intiterationNo) {
    sortPopulationByFitness();
    replaceWorstContainers();
    sortPopulationByFitness();
    updateBestSolution();
    double bestFitnessAverage = getBestFitnessAverage();
    if (bestFitnessAverage - firstConstantSolutionFitness <= getMinDiffIt()) {
        noOfIterationWithConstantFitness++;
    } else {
        firstConstantSolutionFitness = bestFitnessAverage;
        noOfIterationWithConstantFitness = 0;
    }
    getMaxFitnessPerIteration().add(iterationNo, bestFitnessAverage);
    getAvgFitnessPerIteration().add(iterationNo, getNestsFitnessAverage());
}

public double getNestsFitnessAverage() {
    double average = 0.0;
    for (int i = 0; i < population.getPopSize(); i++) {
        average += population.getSolutionSet().get(i).getFitnessValue();
    }
    return average / population.getPopSize();
}

protected void executeInitializationStep() {
    checkParameters();
}

```

```

initializeContainers();
initializeCuckoos();
sortPopulationByFitness();
updateBestSolution();
firstConstantSolutionFitness = getBestFitnessAverage();
getMaxFitnessPerIteration().add(0, getBestFitnessAverage());
getAvgFitnessPerIteration().add(0, getNestsFitnessAverage());
}

```

```

protected void replaceWorstContainers() {
    for ( int i = (int) (population.getSolutionSet().size() -Math.round(population.
        getSolutionSet().size() * percentageOfReplacedContainers)); i < population.
        getSolutionSet().size(); i++) {
        Solution foodSol = new Solution();
        foodSol.setIsSuitable(false);
        boolean foodSolIsAlreadyContained = true;
        while (!foodSol.isIsSuitable() || foodSolIsAlreadyContained) {
            foodSol = optimProblem.getSearchSpace().generateRandomSolution();
            if (population.getSolutionSet().contains(foodSol)) {
                foodSolIsAlreadyContained = true;
            } else {
                foodSolIsAlreadyContained = false;
                foodSol = SolutionService.checkSolution(foodSol, optimProblem);
            }
        }
        population.setSolutionToSet(i, foodSol);
    }
}

```

```

protected Solution updateCuckooSolution(Solution selectedCuckoo) {
    selectedCuckoo.setIsSuitable(false);
    while (selectedCuckoo.isIsSuitable() == false) {
        selectedCuckoo = SolutionService.checkSolution(genAlgComponent.
            mutate(selectedCuckoo, mutateThreshold), optimProblem);
    }
    return selectedCuckoo;
}
}

```