DOI 10.31392/NPU-nc.series 2.2019.21(28).03
УДК: 378.147:519.688]:004.421

Н.М. Кузьміна
кандидат фізико-математичних наук, професор
Національний педагогічний університет імені М.П. Драгоманова;
А.В. Кузьмін
кандидат фізико-математичних наук, доцент
Київський національний університет імені Тараса Шевченка

НАВЧАННЯ ЕВОЛЮЦІЙНИХ АЛГОРИТМІВ СТУДЕНТІВ ІНФОРМАТИЧНИХ СПЕЦІАЛЬНОСТЕЙ

Анотація. У статті розглядаються основні поняття еволюційних алгоритмів та методичні аспекти їх навчання студентів інформатичних спеціальностей. Наведено приклади програмної реалізації алгоритму роєвого інтелекту та описано модульну структуру генетичного алгоритму. Ефективність алгоритмів перевіряється на класичних тестових функціях таких, як функції Розенброка и Растригіна.

Ключові слова: еволюційні алгоритми, алгоритми ройового інтелекту, генетичні алгоритми, функція пристосованості, відбір, схрещування, мутація.

Метою і завданнями навчання таких дисциплін як «Основи теорії і методів оптимізації», «Сучасні методи комп'ютерного моделювання», «Системний аналіз» студентів інформатичних спеціальностей, прикладної математики є ознайомлення й оволодіння класичними методами оптимізації, набуття практичних навичок доцільного, ефективного і педагогічно виваженого використання сучасних інформаційних технологій під час розв'язування задач оптимізації в різних галузях науки і техніки. До предмету навчання окремих розділів або тем відносять огляд основних постановок, методів дослідження і розв'язування прикладних задач оптимізації, до яких застосовують класичні методи умовної або безумовної оптимізації функцій однієї та багатьох змінних, а також сучасні інформаційні системи і технології, які використовують під час їх розв'язування [9].

Доцільним також є розгляд в даних курсах таких оптимізаційних задач, для яких класичні методи і алгоритми оптмізації неефективні:

- задачі, в яких цільова функція, екстремальне значення якої треба знайти, має «складний рельєф», тобто велику кількість близько розташованих локальних екстремумів та одночасно кілька глобальних екстремумів;
- у постановці задачі оптимізації відсутні вимоги гладкості (неперервності та диференційованості), яким повинна задовольняти цільова функція;
- у постановці задачі оптимізації необов'язково знати аналітичний вигляд цільової функції. Достатньо мати процедуру обчислення самої функції для всіх допустимих значень її аргументів тощо.

Для таких задач застосовують еволюційні алгоритми.

Еволюційні алгоритми – це напрям у комп'ютерних науках, в якому використовують принципи біологічної еволюції для розв'язування задач штучного інтелекту. Еволюційні алгоритми відносять до класу природоподібних алгоритмів, у яких програмно імітується пошук найкращих (оптимальних) розв'язків задач «колективною дією сукупності живих істот». В еволюційних алгоритмах імітують біологічну еволюцію, основними принципами якої є поєднання мутацій, природного відбору та відтворення, тобто схрещуванням найкращих представників, відбором їх кращих нащадків і повторним схрещуванням вже цих нащадків [1].

Умовно еволюційні алгоритми можна поділити на два типи: алгоритми ройового інтелекту та генетичні алгоритми.

Зупинимось на особливостях навчання таких алгоритмів.

Ройовий інтелект або *колективний інтелект* (англ. *Swarm intelligence*) – термін, за допомогою якого описують комплексну колективну поведінку децентралізованої системи з самоорганізацією [2]. Його розглядають в теорії штучного інтелекту як метод оптимізації. З точки зору інформатики ройовий інтелект є предметом досліджень комп'ютерних наук, в яких проектуються та вивчаються ефективні числові методи розв'язування задач оптимізації у спосіб, схожий з поведінкою «колективу» живих організмів.

Починати навчання еволюційних алгоритмів ройового інтелекту доцільно з найбільш прозорого і простого у програмній реалізації алгоритму рою частинок.

Метод *рою частинок* відносять до класу еволюційних алгоритмів агентного типу, які використовують для пошуку глобального екстремуму широкого класу функцій, стосовно яких відсутні вимоги неперервності та диференційовності.

Взагалі кажучи, метод рою частинок можна використовувати для пошуку екстремуму будь-якої функції, значення якої можуть бути обчислені на заданій множині вхідних даних і яка необов'язково має бути заданою в аналітичному поданні.

Метод було запропоновано в середині 90-х років XX сторіччя, його авторами вважають психолога Джеймса Кеннеді та інженера Рассела Еберхарта [3, 4]. В подальшому численні дослідники запропонували різні модифікації цього методу.

Як використовується алгоритм для пошуку екстремального значення деякої цільової функції $F(x_1, x_2, ...x_n)$ на множині $D = \{a_i \le x_i \le b_i, i = 1..n\}$.

Рій частинок розглядається як множина $\{P_j,\ j = 1..L\}$. Кожна частинка і весь рій в цілому характеризуються набором параметрів, за яким визначається їх стан в конкретний момент часу:

$X^{(j)} = (x_1^j, x_2^j, ...x_n^j),\ j = 1..L$ – положення частинки в *n*-вимірному просторі.

Стосовно кожної частинки в кожний момент часу обчислюють значення цільової функції, часто її називають *фітнес функцією* або *функцією пристосованості* $F^{(j)} = F(x_1^j, x_2^j, ...x_n^j),\ j = 1..L$.

Відомі також:

$V^{(j)} = (v_1^j, v_2^j, ...v_n^j),\ j = 1..L$ – швидкості частинки у кожному напрямку;

$XL^{(j)} = (xl_1^j, xl_2^j, ...xl_n^j),\ j = 1..L$ – найкраще положення кожної частинки на поточний момент часу (локальне положення);

$XG = (xg_1, xg_2, ...xg_n)$ – абсолютне найкраще положення рою (глобальне положення).

Алгоритм рою частинок є реалізацією ітераційного процесу з дискретним часом.

На кожній ітерації кожна частинка переміщується з попереднього положення у нове положення за певним законом, при цьому у законі переміщення кожної частинки рою враховується її окреме найкраще (екстремальне положення – локальний екстремум) і найкраще положення найкращої частинки рою (глобальний екстремум).

Для ініціалізації ітераційного процесу початковий стан кожної частинки рою $X^{(j)}$ визначається за випадковою величиною з рівномірним розподілом на множині її значень $D = \{a_i \le x_i \le b_i, i = 1..n\}$.

Початкові швидкості руху кожної частинки також визначаються як випадкові величини з рівномірними розподілами ймовірностей на *n*-вимірному паралелепіпеді – множині її значень $\Pi = \{[-\varepsilon, \varepsilon] \times [-\varepsilon, \varepsilon] ... [-\varepsilon, \varepsilon]\}$, де ε - деяке мале число. Допускається також і нульове значення швидкості.

На кожній ітерації (в кожній момент часу) обчислюється нова швидкість кожної частинки рою за формулою:

$$\hat{V}_i^j = \omega \cdot V_i^j + a1 \cdot rand \cdot (x_i^j - xl_i^j) + a2 \cdot rand \cdot (x_i^j - xg_i),$$

де ω – змінний коефіцієнт інерції, $a1, a2$ – постійні значення прискорень, *rand* – випадкова величина з рівномірним розподілом ймовірностей на множині її значень – відрізку $[-1,1]$.

Після обчислення значення швидкості обчислюється нове положення кожної частинки $\hat{X}^{(j)} = X^{(j)} + \hat{V}^{(j)}$.

Критерієм припинення роботи за алгоритмом може бути досягнення заданого числа ітерацій або будь-який інший критерій, наприклад, відсутність змін для найкращого розв'язку через певну кількість ітерацій.

Під час навчання алгоритму рою частинок необхідно звернути увагу студентів на обов'язкове передбачення в ньому коректного опрацювання події, коли частинка на черговій ітерації може опинитися за межами області пошуку та забезпечити «повернення її в деяку точку області пошуку» за допомогою певних команд.

На рисунку 1 наведено програмну реалізацію методу рою частинок у середовищі системи комп'ютерної математики Maple 18, за допомогою якого реалізовано пошук максимального значення функції Розенброка [5] $F(x_1, x_2, x_3) = -((1 - x_1)^2 + (1 - x_2)^2 + 100(x_2 - x_1)^2 + 100(x_3 - x_2^2)^2)$ в області $\{-3 \le x_i \le 3, i = 1..3\}$. Глобальний максимум даної функції досягається в точці $[\ 1.003083751\ \ 1.006167513\ \ 1.012381485\ ]$, в якій значення функції дорівнює -0.00004756385126.

Algoritm roya chastok

#Procedura poshuku optymalnogo elementa

```
ABest := proc(X1, N, M)
 global jmax, gmax :
 local i :
  gmax := -10^10 :
 jmax := 1 :
 for i from 1 to N do
 if X1[i, M] > gmax then gmax := X1[i, M] :
jmax := i :
 end if:
 end do:
 end proc:
```

#Pochatkovi dani

```
> with(LinearAlgebra) :
> with(RandomTools) :
> L := 3 :
> N := 300 :
> Func := (x1, x2, x3) → -((1 - x1)^2 + (1 - x2)^2 + 100·(x2 - x1^2)^2 + 100·(x3 - x2^2)^2) :
> ITER := 1000 :
> XMIN := Array([-3,-3,-3]) :
> XMAX := Array([3, 3, 3]) :
> adaptmean := Array(1 ..ITER) :
> ωmax := 0.9 : ωmin := 0.4 :
> a1 := 1.49 : a2 := 1.49 :
> X := Matrix(1 ..N, 1 ..L + 1) :
> XX := Matrix(1 ..N, 1 ..L) :
> Xbest := Matrix(1 ..N, 1 ..L + 1) :
> V := Matrix(1 ..N, 1 ..L) :
```

#Pochatkova inicializaciya

```
>
   for i from 1 to N do
    for j from 1 to L do
   X[i,j] := Generate(float(range = XMIN[j] ..XMAX[j])) :
   V[i,j] := Generate(float(range = -1 ..1.)) :
    end do:
   X[i, L + 1] := Func(seq(X[i, l], l = 1 ..L)) :
    end do:
> Xbest := X :
> ABest(X, N, L + 1) :
> jmax :
> gmax :
> t1 := time( ) :
```

Iteracii

```
for iter from 1 to ITER do
ω := ωmax + ((ωmin − ωmax)·(iter − 1))/ITER :
adaptmean[iter] := (add(Xbest[i, L + 1], i = 1 ..N))/N :
for i from 1 to N do
for j from 1 to L do
rand1 := evalf(Generate(integer(range = -10^10 ..10^10))/(2·10^10 + 1)) :
rand2 := evalf(Generate(integer(range = -10^10 ..10^10))/(2·10^10 + 1)) :
V[i,j] := ω·V[i,j] + rand1·a1·(X[i,j] − Xbest[i,j]) + rand2·a2·(X[i,j] − Xbest[jmax,
    j]) :
end do:
end do:
for i from 1 to N do
for j from 1 to L do
XX[i,j] := X[i,j] + V[i,j] :
X[i,j] := piecewise(XX[i,j] ≥ XMIN[j] and XX[i,j] ≤ XMAX[j], XX[i,j], Xbest[i,j]) :
X[i, L + 1] := Func(seq(X[i, l], l = 1 ..L)) :
end do:
end do:
for i from 1 to N do
if X[i, L + 1] > Xbest[i, L + 1] then
Xbest[i, 1 ..L + 1] := X[i, 1 ..L + 1]
end if:
end do:
#jmax:='jmax':gmax:='gmax':
ABest(Xbest, N, L + 1) :
end do:
>

> t2 := time( ) − t1;

                    t2 := 153.536

> ABest(Xbest, N, L + 1) :
> V : X : Xbest;
> # Rezultat
> gmax; jmax; Xbest[jmax, 1 ..L];

                    -0.00004756385126
                    122
[ 1.003083751  1.006167513  1.012381485 ]
```

Рис.1

На рисунку 2 наведено графічне зображення поверхні, що визначається через функцію Розенброка $F(x_1,x_2,x_2)=((1-x_1)^2+(1-x_2)^2+100(x_2-x_1)^2+100(x_2-x_2^2)^2)$, яка набуває мінімального значення в точці $(x_1,x_2)=(1,1)$.

Другий тип еволюційних алгоритмів – генетичні – також застосовуються для розв'язування широкого спектра оптимізаційних задач [6-8]. За допомогою генетичних алгоритмів знаходять прийнятні розв'язки складних, аналітично нерозв'язних задач, або задач, які не можуть бути розв'язані за класичними чисельними процедурами оптимізації. У *генетичному алгоритмі* застосовують послідовний добір і комбінування невідомих значень аргументів функції з метою покращення значення функції пристосованості з використанням механізму, що нагадує біологічну еволюцію в процесі зміни поколінь.

Термін «генетичний алгоритм» і його класичну схему вперше було запропоновано у 1975 году Дж. Холландом у роботі «Adaptation in Natural and Artificial Systems» [6].

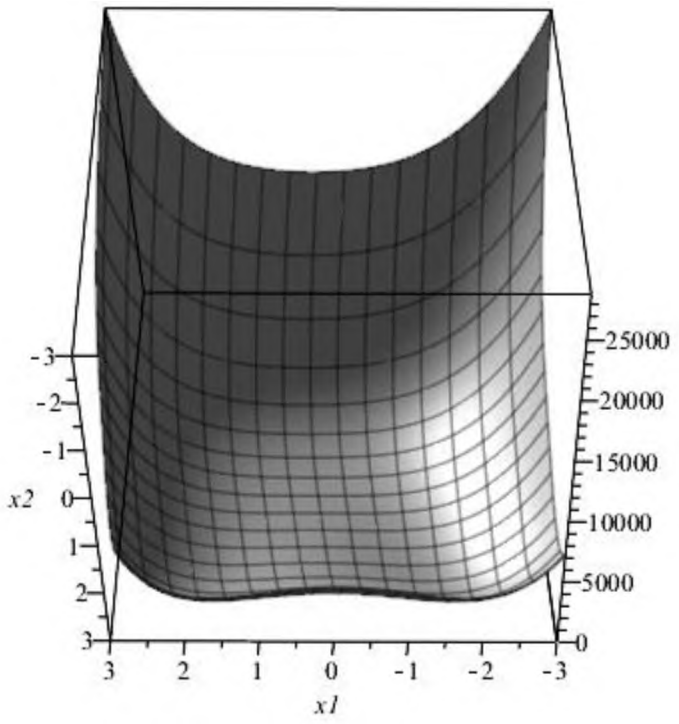

Рис.2

Для застосування генетичних алгоритмів задачу оптимізації розглядають як задачу знаходження максимуму (мінімуму) деякої функції $f(x_1, x_2, ...x_n)$, значення якої обчислюються у замкненому паралелепіпеді її визначення $\{X\min_i \le x_i \le X\max_i,\ i = 1...n\}$, яку називають *функцією пристосованості* (*fitness function*). Векторний аргумент функції пристосованості кодується із заданою в алгоритмі точністю s в заданому паралелепіпеді рядком бітів $(x_1, x_2, ...x_n)$ =>(10001,11001,…,11010). Універсальність генетичних алгоритмів полягає в тому, що від постановки конкретної задачі залежать тільки спосіб обчислення значень функції пристосованості і кодування розв'язків. Всі інші кроки алгоритмів для всіх задач виконуються типово.

У генетичних алгоритмах оперують із сукупністю «особин», ця сукупність називається *популяціями*, самі «особини» є бінарними рядками – кодами можливих розв'язків задачі, і кожна «особина», крім бінарного коду, характеризується значенням функції пристосованості.

Серед усіх «особин» популяції (розв'язків) виокремлюють найбільш пристосовані (з найбільшим значенням функції пристосованості), такі «особини» мають високу ймовірність «схрещуватись» і «давати гарне потомство», та найгірші, які мають низькі шанси «продукувати потомство» і поступово вилучаються з популяції. Таким чином, пристосованість нового покоління (середнє значення функції пристосованості) в середньому вище за попереднє.

У класичному генетичному алгоритмі початкова популяція формується випадковим чином; чисельність популяції (кількість особин) фіксується, не змінюється під час роботи за алгоритмом і є параметром алгоритму; кожна «особина» кодується k-бітним рядком, де k – довжина коду «особини» – є однаковою для всіх особин и визначається виключно за діапазоном зміни аргументу (числа з плаваючою комою) та точністю подання цього числа у вигляді бінарног коду.

Далі започатковується ітераційний процес, згідно якого стосовно кожної нової популяції послідовно організовуються процедури відбору, формування батьківських пар, схрещування, мутації, формування нового покоління. На кожному кроці алгоритму є такі стадії:

- *генерування наступної популяції* (*intermediate generation*) шляхом *відбору* (*selection*) наявного покоління. У класичному генетичному алгоритмі використовується пропорційний відбір (*proportional selection*) попадання «особини» до проміжної популяції з врахуванням значення функції пристосованості. Деякі найбільш пристосовані «особини» можуть бути зараховані в наступну популяцію кілька разів, а найменш пристосовані – взагалі в неї не потрапити;
- *формування батьківських пар* передбачає випадковий вибір зі сформованої популяції двох різних «особин», які утворюють «батьківську пару»;
- *схрещування* (*recombination*) – «батьківська пара» породжує двох нащадків шляхом процедури *кроссовера* (*crossover*) – обмін частиною бінарного коду;
- *мутація* нового покоління. До отриманого в результаті схрещування нового покоління застосовують оператор мутації, який полягає в інверсії одного з двійкових разрядів. Вибір

номера розряду та застосування до нього інверсії відбувається випадковим чином із заданим рівнем ймовірності. Мутація необхідна для виходу популяції з локального екстремуму, а також для захисту від передчасної збіжності алгоритму.

Як критерій зупинки роботи з аалгоритмом використовують задану кількість поколінь або умову *сходження* (*convergence*) популяції (всі рядки популяції знаходяться в області деякого екстремума і майже однакові), а це означає, що досягнутий варіант рішення близький до оптимального.

Під час вивчення генетичних алгоритмів доцільно розглянути такі моделі генетичних алгоритмів:

- *Genitor (Whitley)*, де використовується специфічна стратегія відбору – на кожному кроці тільки одна пара випадкових «батьків» створює тільки одну «дитину», яка замінює не батька, а одну з гірших особин популяції;
- *CHC (Eshelman)* (Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation). Для нового покоління обирають *N* кращих різних особин серед батьків і дітей. В процесі схрещування кожному нащадку переходить рівно половина бітів кожного з батьків;
- *Hybrid algorithm (Davis)*. У гібрідному алгоритмі об'єднуються переваги генетичних алгоритмів з перевагами класичних методів;
- *Island Models (Острівна модель)* – модель паралельного генетичного алгоритму та інші.

Студентам інформатичних спеціальностей, прикладної математики доцільно давати завдання стосовно програмної реалізації генетичних алгоритмів з подальшим їх тестуванням як на відомих задачах оптимізації, які можна досліджувати і розв'язувати за класичними методами оптимізації з використанням, наприклад, систем комп'ютерної математики, так і на таких задачах, як оптимізація або апроксимація складних функцій, на різноманітних задачах на графах, на задачах налаштування і навчання штучних нейронних мереж, складання розкладів, знаходження різних ігрових стратегій, на задачах біоінформатики тощо.

Для покрокового відтворення модульної структури генетичного алгоритму студентам пропонується створити і реалізувати такі процедури у програмних середовищах за власним вибором (табл.1).

Таблиця 1.

№ п/п	*Назва процедури*	*Призначення, обчислення, вказівки*
1.	***GenerationDec***	Заповнення матриці заданої розмірності $G(1..N,1..M+1)$ випадковими десятковими числами із заданого відрізка. Останній стовпчик матриці заповнюються значеннями функції, аргументами якої є всі попередні елементи рядка матриці.
2.	***Mutation***	Мутація нового покоління. Опрацьовується кожний бінарний елемент прямокутної матриці *G* заданної розмірності за таким правилом: генерується випадкове число *p* (*p* – дійсне число $0<p<<1$ – ймовірність мутації); якщо згенероване випадкове число менше заданого порогового значення *P*, то відповідний елемент матриці інвертують (0 в 1, 1 в 0); у протилежному випадку – елемент матриці залишають без змін.
3.	***Crossover***	*«Схрещування* особин» проміжної популяції шляхом *кроссовера* (*crossover*), що забезпечує формування нового покоління. Опрацьовується прямокутна матриця з парною кількістю $2N$ рядків і будь-якою кількістю стовпчиків, з бінарними елементами. Рядки матриці об'єднуються у пари, пари утворюються за значеннями номерів рядків матриці і отримуються з процедури ***Parens***. Кожна пара рядків матриці опрацьовується таким чином. Генерирується випадкове ціле число *cros* ($1<= cros <= L$), де *L* – кількість стовпчиків матриці. В рядках матриці, з яких утворено пару, обмінюють місцями елементи з номерами від 1 до *cros*.
4.	***Parens***	Послідовність натуральних чисел від 1 до $2N$ ділиться випадковим чином на два списки за умови, що елементи списка з однаковими номерами не співпадають (в разі співпадання елементів відбувається присвоювання випадкового числа до виконання умови).

Продовження таблиці 1.

5.	***BinDecParam***	Обчислення параметрів прямого й оберненого перетворень дійсних чисел заданого інтервалу у бінарну послідовність із заданою точністю *eps*. Допоміжня процедура до процедур ***CodBinary*** і ***CodDecimal***. В результаті необхідно отримати: *nn[i], i=1..M* – список цілих чисел, у якому містяться значення кількості бінарних розрядів, необхідних для кодування будь-якого дійсного числа з інтервала [*Xmin*[*i*], *Xmax*[*i*]] з точністю *eps*. *dd*[*i*], *i*=1..*M* – дійсне число, за яким визначається значення дискретності для подання дійсного числа із заданого інтервала цілим числом. $nn[i] = \left[\log_2 \left(\frac{X\max[i] - X\min[i]}{eps} \right) \right] + 1,\ i = 1..M$ $dd[i] = \frac{X\max[i] - X\min[i]}{2^{nn[i]}} \le eps,\ i = 1..M$ $NN[j+1] = \sum_{i=1}^{j} nn[i],\ j = 1..M,\ NN[1] = 0$
6.	***CodBinary***	Виконується кодування будь-якого дійсного числа *xdec* із заданого інтервала [*Xmin..Xmax*] із заданою точноістю *eps* у двійкову послідовність фіксованої довжини, *l* – ціле число, максимальна кількість двійкових розрядів у такому поданні, *d* – крок дискретності кодування числа *xdec* цілим числом. Ціле число частин величини *d* стосовно заданого числа *xdec* обчислюється за формулою: $xx = \left[\frac{xdec - x\min}{d} \right]$. Ціле число *xx* записується у двійковій формі й доповнюється нулями в старших розрядах, якщо їх кількість менша за *l*.
7.	***CodDecimal***	Перетворення двійкового подання десяткового числа із заданого інтервала [*Xmin..Xmax*] із заданою точністю *eps* у його десяткове подання. Виконується обернене перетворення до отриманого за процедурою ***CodBinary***.
8.	***ACodBinary***	Послідовне перетворення дійсних чисел – елементів матриці *Gdec(1..N,1..M+1)* – у двійкові коди. Перетворення виконується стосовно елементів перших *M* стовпчиків за допомогою процедур ***CodBinary*** та ***BinDecParam***. Результати перетворень зберігаються в матртці *Gbin*. Кожному стовпчику матриці *Gdec* ставиться у відповідність фіксована кількість стовпчиків матриці *Gbin*, що визначається за процедурою ***BinDecParam***.
9.	***ACodDecimal***	Перетворення бінарних елементів матриці ***Gbin*** у дійсні десяткові числа. Результати, отримані за процедурою є оберненими до отриманих за процедурою ***ACodBinary***.
10.	***ADAPT***	Використовуючи масив значень функції пристосованості покоління особин, генерується массив *Adapt N* цілих невід'ємних чисел, сума яких дорівнює *N*. Через кожний елемент масиву з номером *j*, $(1 \le j \le N)$, вказується на кількість входжень «особини» з номером *j* в нове покоління. Обчислюється як сума попадань випадкового числа з рівномірним розподілом ймовірностей на відрізку [0..1] в інтервал $(\alpha_j, \alpha_{j+1}]$, де $0 = \alpha_1 < \alpha_2 < ....\alpha_N < \alpha_{N+1} = 1$.
11.	***Best*** і ***Worst***	Опрацювання (*M*+1)-го стовпчика матриці ***Gdec***, в якому містяться значення функції пристосованості *M* аргументів. За процедурою ***Best*** обчислюється номер рядка, в якому міститься максимальне значення цієї функції, за процедурою ***Worst*** – відповідно номер рядка, в якому міститься мінімальне значення цієї функції.
12.	***NewGeneration***	Використовуючи матрицю «попереднього покоління» $Gold(N, M+1)$ і вихідний масив *Adapt* процедури ***ADAPT***, формується нова матриця $GNew(N, M+1)$, в якій містяться «особини» нового покоління. До $GNew(N, M+1)$ записуються рядки матриці $Gold(N, M+1)$, кількість входжень кожного рядка визначається за відповідним значенням масиву *Adapt*.

Вказівки

1. Для генерації випадкових чисел із заданого інтервалу використовувати відповідні функції, наприклад, функцію ***Generate*** пакета ***Random Tools*** у системі комп'ютерної математики *Maple*.
2. Для обчислення кількості рядків і стовпчиків матриці використовувати відповідні функції, наприклад, функції ***RowDimension, ColumnDimension*** пакета ***LinearAlgebra*** у системі комп'ютерної математики Maple.
3. Для перетворення цілого десяткового числа у двійковий код використати відповідні функції, наприклад, функцію ***convert(xx,base,2)*** в системі комп'ютерної математики Maple.

Після створення, написання та тестування правильності кожної процедури формується основна програма як ітераційний процес, в тілі якого послідовно здійснюється виклик відповідних процедур та передавання параметрів.

Для тестування якості результатів, отриманих за генетичним алгоритмом, можна використовувати вже наведену вище функцію Розенброка , а також інші функції зі складною структурою екстремумів, наприклад функцію Растригіна [5] (рис. 3):

$$f(x_1, x_2, ...x_n) = A \cdot n + \sum_{i=1}^{n} [x_i^2 - A \cdot \cos(2\pi x_i)] \qquad A = 10,\ x_i \in [-5.12, 5.12]$$

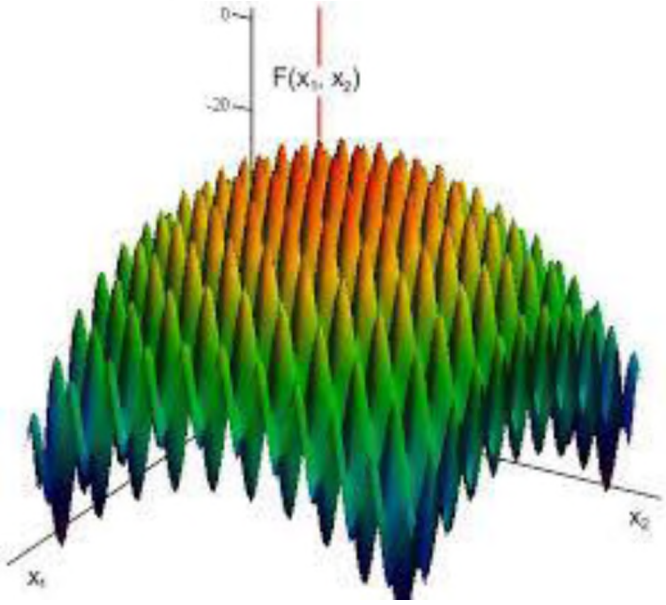

Рис.3

Для самостійного опанування студентам в якості проектів доцільно рекомендувати вивчити та опрацювати такі алгоритми агентного типу:

- метод бджолиного рою;
- метод колонії мурах;
- метод колонії бактерій.

Безумовно навчання еволюційних алгоритмів розширює кругозір студентів, сприяє кращому розумінню математичних та інформатичних аспектів сучасних постановок, методів комп'ютерного моделювання та розв'язування задач оптимізації. А це, в свою чергу, сприяє розвитку їх вмінь, навичок, творчих здібностей та привчає до самостійної роботи.

Список використаних джерел:

1. Эволюционные алгоритмы. ПостНаука. URL: https://postnauka.ru/animate/69879 (дата звернення: 20.01.2019).

2. Колективний інтелект. Вікіпедія URL: https://uk.wikipedia.org/wiki/Колективний_інтелект (дата звернення: 17.02.2019).

3. Kennedy J., Eberhart R. (1995). Particle Swarm Optimization. Proceedings of IEEE Intern3ational Conference on Neural Networks IV. p. 1942–1948.

4. Shi Y., Eberhart R.C. (1998). A modified particle swarm optimizer. Proceedings of IEEE International Conference on Evolutionary Computation. c. 69–73.

5. Пантелеев А.В., Метлицкая Д.В., Алешина Е.А. Методы глобальной оптимизации. Метаэвристические стратегии и алгоритмы. Москва, 2013. 244 с.

6. Holland John H. Adaptation in natural and artificial systems: an introductory analysis with application to biology, control and artificial intelligence. USA, 1975. 211 p.

7. Курейчик В.М. Генетические алгоритмы и их применение. Таганрог, 2002. 242 с.
8. Авторский сайт Ю. Цоя URL: http://www.qai.narod.ru/ (дата звернення: 11.02.2019).
9. Кузьміна Н.М. Основи теорії і методів оптимізації: програма навчальної дисципліни для підготовки студентів спеціальностей 7.04030201, 8.04030201 «Інформатика*» Інституту інформатики НПУ імені М.П. Драгоманова. Київ, 2014. 28 с.

References:

1. Evolyutsionnyye algoritmy. PostNauka. URL: https://postnauka.ru/animate/69879 (data zvernennya: 20.01.2019).
2. Kolektyvnyj intelekt. Vikipediya URL: https://uk.wikipedia.org/wiki/Kolektyvnyj_intelekt (data zvernennya: 17.02.2019).
3. Kennedy J., Eberhart R. (1995). Particle Swarm Optimization. Proceedings of IEEE Intern3ational Conference on Neural Networks IV. p. 1942–1948.
4. Shi Y., Eberhart R.C. (1998). A modified particle swarm optimizer. Proceedings of IEEE International Conference on Evolutionary Computation. s. 69–73.
5. Panteleev A.V., Metlyckaya D.V., Aleshyna E.A. Metodы hlobal"noj optymyzacyy. Metaэvrystycheskye stratehyy y alhorytmы. Moskva, 2013. 244 s.
6. Holland John H. Adaptation in natural and artificial systems: an introductory analysis with application to biology, control and artificial intelligence. USA, 1975. 211 p.
7. Kurejchyk V.M. Henetycheskye alhorytmы y yx prymenenye. Tahanroh, 2002. 242 s.
8. Avtorskyj sajt Yu. Coya URL: http://www.qai.narod.ru/ (data zvernennya: 11.02.2019).
9. Kuz"mina N.M. Osnovy teoriyi i metodiv optymizaciyi: prohrama navchal"noyi dyscypliny dlya pidhotovky studentiv special"nostej 7.04030201, 8.04030201 «Informatyka*» Instytutu informatyky NPU imeni M.P. Drahomanova. Kyyiv, 2014. 28 s.

Teaching informatics students' evolutionary algorithms

Kuzmina N., Kuzmin A.

Abstract. The article discusses the basics of evolutionary algorithms concepts and the methodological aspects of teaching them to students specialized in informatics. Examples of software implementation of the Swarm intelligence algorithm are given, and the modular structure of the genetic algorithm is described. The effectiveness of the algorithms is tested on classical test functions, such as Rosenbrock and Rastrigin functions.

Keywords: evolutionary algorithms, Swarm intelligence algorithms, genetic algorithms, fitness function, selection, crossing, mutation.

DOI 10.31392/NPU-nc.series 2.2019.21(28).04
УДК 378.011.3-051:004

Т.В. Підгорна
доктор педагогічних наук, доцент,
Національний педагогічний університет імені М.П.Драгоманова

ПЕДАГОГІЧНА ІНФОРМАТИКА ЯК СКЛАДОВА ПІДГОТОВКИ МАЙБУТНІХ ВЧИТЕЛІВ ДО РОБОТИ В УМОВАХ ІНФОРМАТИЗОВАНОГО НАВЧАЛЬНОГО ПРОЦЕСУ

Анотація. В статті обґрунтовано необхідність вивчення педагогічної інформатики майбутніми вчителями всіх спеціальностей. В межах педагогічної інформатики майбутні вчителі опановують навчальний матеріал, що пов'язаний із організацією та здійсненням навчання і виховання гармонійно розвинених членів інформаційного суспільства в умовах інформатизованого навчального процесу з врахуванням переваг і проблем, що виникають в таких умовах.

Ключові слова: педагогічна інформатика, педагогічно виважене використання інформаційно-комунікаційних технологій, інформаційна безпека дітей.

Сучасні люди активно використовують в своїй професійній діяльності і повсякденному житті інформаційно-комунікаційні технології (ІКТ) та Інтернет-ресурси, які весь час розвиваються. Виняток не становить і педагогічний процес.

На основі аналізу отриманих результатів значної кількості досліджень щодо використання інформаційно-комунікаційних технологій в навчальному процесі можна зробити висновок про позитивні наслідки такого використання ІКТ. Серед вітчизняних науковців які займалися даною