1. МЕТОД РОЗВ’ЯЗАННЯ ТА ОПИС АЛГОРИТМУ

3.1 **Опис алгоритмів розв’язку задачі**

3.1.1 **Алгоритм RWGA**

Для роботи генетичного алгоритму спочатку необхідно створити початкову популяцію індивідів розміром *N*.

Далі у генетичному алгоритмі RWGA для кожної цільової функції *fk(x)* необхідно визначити випадкову вагу критерію, що визначається за формулою

,



де *rj* - випадкові числа з відрізка [0,1];

*q* - число цільових функцій.

Далі вираховується сумісна цільова функція, яка складається з кожної цільової функції *fi(x)* за формулою

,



де *wi* - це значення ваги, отримані на попередньому кроці алгоритму.

Визначивши значення фітнес-функції для кожної особини у популяції, необхідно провести відбір найкращих особин для подальшого схрещення та отримання нової популяції нащадків.

Для селекції найкращих особин використовується процедура турнірного відбору. Для цього з популяції, яка складається із *N* особин, вибираються випадковим чином дві особини, які порівнюються за значенням цільових функцій, і найкраща особина (переможець турніру) записується в проміжний масив. Ця операція повторюється *N* раз. Особини в отриманому проміжному масиві потім використовуються для схрещування.

Далі для отримання нових особин популяції, застосовується механізм лінійного схрещування.

Значення кожної змінної для кожної особини-нащадка знаходиться за формулою

*zi = r∙xi + (1 - r) ∙yi* ,

де *r* - випадкове число на інтервалі [0,1];

*xi* та *yi* - значення змінних першої та другої батьківської особини, отриманих за допомогою турнірного відбору.

Перевага цього методу у тому, що для схрещування йому необхідні лише фенотип даної особини, тобто чисельні значення кожної змінної для даного рішення. В той час як інші методи схрещування використовують генотип особини, закодований у бінарній формі, що суттєво збільшує вимоги до обчислювальної техніки, потребує більших витрат пам’яті та збільшує час на пошук оптимального рішення [31].

Кожна пара батьківських особин дає два нащадка. Провівши процедуру схрещування *N* разів, отримуємо популяцію нащадків.

На наступному кроці виконується генетичний оператор мутації. З деякою зазначеною вірогідністю значення хромосом особини змінюється на деяке випадкове число

*zi = zi + r* ,

де *zi* - значення хромосоми особини;

*r -* випадкове число.

Мутація необхідна для підтримання різноманітності, і попереджає втрати, які могли б відбутися внаслідок виключення якого-небудь значимого гена в результаті схрещування [1-4].

Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

Далі виконується перевірка умови зупинки роботи алгоритму. Якщо умова виконується, то з усієї популяції обирається особина з найкращим значенням цільової функції, яка і є оптимальним рішенням задачі. Якщо ні, то алгоритм повертається до другого кроку.

Блок-схему генетичного алгоритму RWGA зображено на рисунку 3.1.

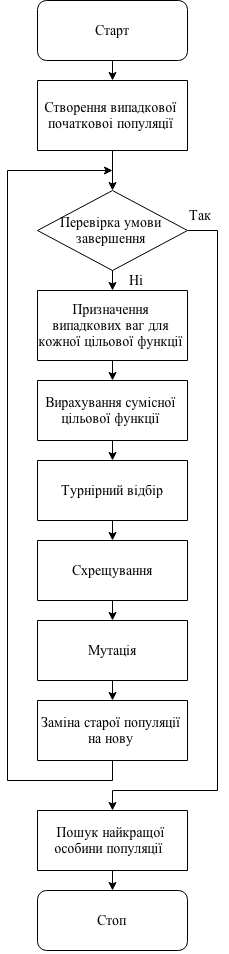


Рисунок 3.1 – Генетичний алгоритм RWGA

3.1.2 **Алгоритм VEGA**

Алгоритм багатокритеріальної оптимізації VEGA відрізняється тим, що селекція у цьому методі відбувається для кожного з *К* критеріїв окремо, тим самим проміжна популяція заповнюється рівними порціями індивідів, відібраних по кожному з критеріїв [36].

Спочатку необхідно ініціалізувати початкову популяцію розміром *N* з випадкових особин.

Для кожного з *K* критеріїв створюється підпопуляція розміром *N/K*, де *N* – розмір всієї популяції. Для кожної підпопуляції вираховується окреме значення фітнес-функції *Fk(x)* по цільовій функції *fk(x).*

Турнірний відбір у методі VEGA також виконується окремо для кожної з підпопуляцій. У кожній окремій підпопуляції розміром *N/K* особин, вибираються випадковим чином дві особини, які порівнюються за значенням цільових функцій, які записуються у загальну проміжну популяцію. Це дозволяє отримати проміжну популяцію, для якої записані особини, найкращі за кожним критерієм оптимізації.

Схрещування та мутація у алгоритмі VEGA відбуваються так само, як вони описані у пункті 3.1.1.

Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

По завершенню цих кроків відбувається перевірка умови зупинки роботи алгоритму. Якщо умова виконується, то у кожній підпопуляції знаходяться найкращі значення для кожної цільової функції. Якщо ні, то алгоритм повертається до другого кроку.

Блок схему генетичного алгоритму VEGA зображено на рисунку 3.2.

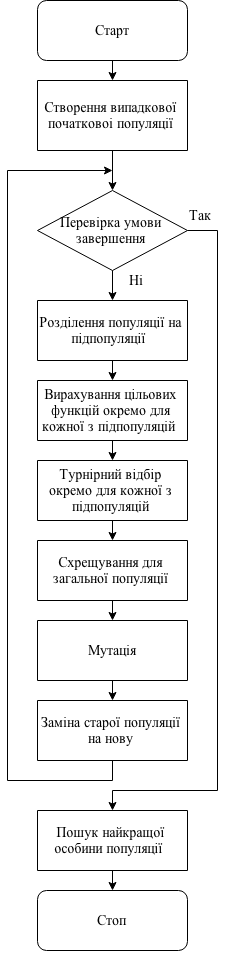


Рисунок 3.2 – Генетичний алгоритм VEGA

3.1.3 **Алгоритм MOGA**

Метод MOGA використовує засновану на Парето-домінуванні процедуру ранжирування індивідів за допомогою рангів Голдберга [32].

На початку створюється початкова популяція розміром N, заповнена випадковими індивідами.

Далі для кожної особини у популяції необхідно визначити її ранг. Ранжування за Голдбергом виконується таким чином:

1. з популяції обираються два індивіди - *i* та *j*;
2. для обраних індивідів порівняти один з одним значення фітнес-функцій по кожній функції *fk(x)*;
3. якщо індивід *i* кращий за індивід *j* хоч би по одному з критеріїв оптимізації, та не гірший за рештою критеріїв, тоді *j* - домінуєме рішення відносно *i*;
4. відповідно, якщо індивід *j* кращий за індивід *i* хоч би по одному з критеріїв оптимізації, але не гірший за рештою критеріїв, тоді *i* - домінуєме рішення відносно *j*;
5. ранг домінованої особини збільшується на одиницю.

У решті, провівши процедуру ранжування з усіма особинами популяції, кожному індивіду буде присвоєний ранг, що дорівнює кількості особин, якими домінується цей індивід.

Далі для популяції проводиться селекція найкращих особин за допомогою турнірного відбору. Але під час турніру у пари індивідів порівнюються не значення їх фітнес-функцій, а отримані на попередньому кроці ранги. До схрещування допускається та перемагає турнір та особина, у якої менше значення рангу.

Операції схрещування та мутації для відібраних особин у алгоритмі MOGA відбуваються так само, як вони були описані у пункті (3.1.1).

Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

Далі виконується перевірка умови зупинки роботи алгоритму. Якщо умова виконується, то з усієї популяції обирається особина з найкращим значенням цільової функції, яка і є оптимальним рішенням задачі. Якщо ні, то алгоритм повертається до другого кроку.

Блок схему генетичного алгоритму MOGA зображено на рисунку 3.3.

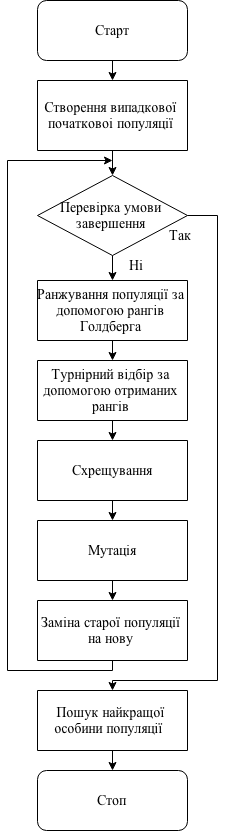


Рисунок 3.3 – Генетичний алгоритм MOGA

3.1.4 **Алгоритм NPGA**

У генетичному алгоритмі NPGA закладено механізм підтримки різноманітності, що запобігає передчасному сходженню рішення [35, 37].

Спочатку необхідно ініціалізувати початкову популяцію розміром *N* з випадкових особин.

Далі алгоритм NPGA використовує оригінальний механізм селекції найкращих особин, що забезпечує різноманітність популяції:

1. з популяції розміром *N* обирається випадкова порівняльна множина розміром *M*;
2. з популяції обираються два випадкових індивіда *i* та *j*;
3. якщо індивід *i* хоч би за одною цільовою функцією кращий за усі індивіди порівняльної множини, та не гірший за усіма іншими цільовими функціями, тоді індивід *i* - недомінуємий порівняльною множиною;
4. відповідно, якщо індивід *j* хоч би за одною цільовою функцією кращий за усі індивіди порівняльної множини, та не гірший за усіма іншими цільовими функціями, тоді індивід *j* - недомінуємий порівняльною множиною;
5. якщо індивід *i* - недомінуємий , а індивід *j* - домінуємий, тоді індивід *i* перемагає турнір і допускається до схрещування;
6. інакше, коли індивід *j* - недомінуємий , а індивід *i* - домінуємий, тоді індивід *j* перемагає турнір.

Якщо виявилось так, що жодна з цих умов не була виконана, і обидві особини виявилися домінованими чи недомінованими порівняльною множиною, тоді переможцем турніру обирається та особина, у радіусі певної ніші *σshare* якої знаходиться менша кількість сусідніх особин. Це дозволяє підтримувати різноманітність популяції, та запобігати накопичуванню рішень у одній області.

Схрещування та мутація у алгоритмі NPGA відбуваються так само, як вони описані у пункті 3.1.1.

Далі виконується перевірка умови зупинки роботи алгоритму. Якщо умова виконується, то з усієї популяції обирається особина з найкращим значенням цільової функції, яка і є оптимальним рішенням задачі. Якщо ні, то алгоритм повертається до другого кроку.

Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

Блок схему генетичного алгоритму NPGA зображено на рисунку 3.4.

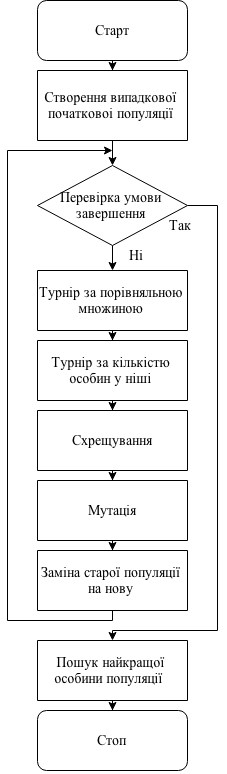


Рисунок 3.4 – Генетичний алгоритм NPGA

3.2 **Опис розробленого програмного забезпечення**

3.2.1 **Загальні відомості**

Назва розробленого додатку – «MultiobjectiveGA». Розроблене програмне забезпечення призначене для проведення багатокритеріальної оптимізації за допомогою генетичних алгоритмів.

Розроблене програмне забезпечення виконує усі етапи роботи генетичного алгоритму: ініціалізацію ГА, оцінку пристосованості особин, селекцію найкращих особин популяції для схрещення, застосування генетичних операторів, формування нової популяції та вибір найкращого результату.

У програмному забезпеченні реалізовані генетичні алгоритми багатокритеріальної оптимізації - RWGA, VEGA, MOGA та NPGA.

Програма орієнтована на розгортання на веб-сервері для подальшого використання за допомогою мережі "Інтернет". Програмне забезпечення написане з використанням HTML, CSS, JavaScript. Середовище розробки – блокнот Notepad++, панель розробника браузера Google Chrome.

Програма використовує допоміжні бібліотеки HighCharts та jQuery для побудови графіків.

* + 1. **Функціональне призначення**

Як показали дослідження, в субмікро- та нанокристалічних металах фізико-механічні властивості суттєво перевищують відповідні показники у крупнокристалічних металах. Межа плинності збільшується до 1,5 разу, а мікротвердість – до 2 разів. В даний час розроблені ефективні технології інтенсивної пластичної деформації (кручення під високим тиском, рівноканальне кутове пресування, всебічна ковка) для створення металів з субмікро- та нанокристалічною структурою. Під час інтенсивної пластичної деформації відбувається перетворення мікроструктури металу у субмікро- та нанокристалічний стан, супроводжуваний змінами кристалічної решітки та утворенням нервіноважного стану зерна.

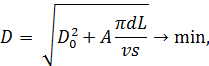
Заготовки отримують методами інтенсивної пластичної деформації в умовах високого тиску та умовно низьких температур. У об’ємних заготівках формується субмікро- та нанокристалічна структура, що складається з зерен, розділених переважно нерівноважними високоенергетичними границями. При впливі високих температур, стають більш інтенсивними процеси рекристалізації, бо структура субмікро- та нанокристалічних металів є термічно нестабільною.

При виготовленні виробів, як правило, застосовують механічну обробку. Механічна обробка супроводжується значними пластичними деформаціями та високою температурою різання. Відзначені умови призводять до повернення структури металу до крупнокристалічного стану та втрати вихідних високих фізико-механічних властивостей.

Програма призначена для багатокритеріальної оптимізації функцій продуктивності



та розміру зерна металу



за допомогою генетичного алгоритму.

Програма виконує усі етапи роботи генетичного алгоритму:

* генерацію початкової популяції генетичного алгоритму відповідного розміру;
* оцінку пристосованості особин популяції за допомогою фітнес-функцій;
* вибір найкращих особин для селекції та створення нової популяції;
* переведення генів особин у бінарну форму для подальшого схрещення;
* схрещення найкращих особин популяції та створення нової популяції;
* переведення генів особин нової популяції у десятичну форму для подальшої перевірки пристосованості популяції.

Зважаючи на те, що поставлена задача передбачає багатокритеріальну оптимізацію параметрів по декільком функціям, для цього були використані генетичні алгоритми багатокритеріальної оптимізації. Генетичні алгоритми добре зарекомендували себе в якості методик пошуку у багатьох областях практично при повній відсутності інформації про властивості цільової функцій і обмежень. У програмі були використані чотири методи багатокритеріальної оптимізації генетичними алгоритмами, що реалізують різні схеми призначення придатності та селекції:

1. RWGA – Random Weights Genetic Algorithm;
2. VEGA – Vector Evaluated Genetic Algorithm;
3. MOGA – Multiobjective Genetic Algorithm;
4. NPGA – Niched Pareto Genetic Algorithm.

Результатом роботи програми є вивід на екран результатів обчислення - мінімальних значень цільових функцій, та значення усіх змінних у точках мінімуму. Також програма будує графік, на якому виводиться значення цільових функцій на кожній ітерації роботи алгоритму.

Розроблений додаток складається з головної сторінки index.html, файлів rwga.js, vega.js, moga.js, npga.js, styles.css, а також допоміжних файлів highcharts.js та jquery.js.

Для завантаження програми користувач повинен відкрити сторінку index.htmlу своєму web-браузері. На цій сторінці реалізований інтерфейс програми. Користувач бачить обчислювальні функції, а також поля вводу, куди необхідно ввести параметри генетичного алгоритму - кількість ітерацій алгоритму, розмір популяції та вірогідність мутації. Також за допомогою радіо-кнопок користувач повинен обрати необхідний алгоритм багатокритеріальної оптимізації.

Після того, як користувач запустить обчислення, натиснувши на кнопку "Оптимізувати", буде викликана функція Optimize(). Програма за допомогою методу getElementById() зчитає введені користувачем початкові дані.

Залежно від того, яку опцію обирає користувач, програма виконає багатокритеріальну оптимізацію за алгоритмами RWGA, VEGA, MOGA або NPGA, кожен з яких описаний у відповідному файлі.

По завершенню обчислень, програма побудує графіки, а також виведе на екран отримані результати оптимізації.

У файлі rwga.js знаходиться функція rwga(), що реалізує алгоритм RWGA. Цей метод є розвитком класичних методів бакатокритеріальної оптимізації, де нова загальна цільова функція будується з окремих цільових функцій у вигляді їх суми, де для кожної цільової функції визначається окрема вага. Такий підхід був уперше використаний для отримання змінного напрямку пошуку фронту Парето. При фіксованих вагах в даному підході генетичний алгоритм відображає тенденцію постійного напряму пошуку, в той час як використання випадкових ваг відображає тенденцію змінного напрямку пошуку, більш пристосованою для пошуку фронту рішень.

У файлі vega.js знаходиться функція vega(), у якій був реалізований генетичний алгоритм VEGA. Цей метод належить до категорії селекції по перемикаючим цільовим функціям. Це означає, що селекція проводиться за придатністю індивідів для кожного з *K* критеріїв окремо. Проміжна популяція заповнюється рівними порціями індивідів, відібраних по кожному з окремих критеріїв - для кожного з *K* критеріїв створюється підпопуляція розміром *N / K*, де *N* - розмір всієї популяції. У ці підпопуляціі особини відбираються за допомогою турнірної селекції щодо придатності за кожним критерієм окремо. Далі здійснюються схрещування і мутація згідно загальній схемі генетичного алгоритму для усієї загальної популяції, об'єднаної з усіх відібраних особин кожної підпопуляції.

У файлі moga.js знаходиться функція moga(), що реалізує алгоритм МОGA. Метод МОGA використовує процедуру ранжирування індивідів, де ранг кожного індивіда визначається числом домінуючих його індивідів, яка називається рангами Голдберга. Придатність індивіда призначається не для кожного критерію окремо, а для індивіда в цілому і визначається не значеннями цільових функцій, а рангом кожного індивіда в популяції, який заснований на понятті Парето-домінування. Основна його ідея полягає у визначенні поняття не домінування для окремих рішень оптимізаційної задачі. Рішення *x1* домінує інше рішення *x2*, якщо одночасно виконуються дві наступні умови. Рішення *x1* повинне бути не гірше рішення *x2* по будь-якому з розглянутих в задачі критеріїв. У той же час рішення *x1* суворо краще рішення *x2* принаймні по одному з критеріїв. Якщо не існує жодного рішення, що задовольняє перерахованим вище умовам, то *x2* є не домінуючим або Парето-оптимальним рішенням багатокритеріальної задачі.

У файлі npga.js знаходиться функція npga(), у якій був реалізований генетичний алгоритм багатокритеріальної оптимізації NPGA. Цей метод принципово відрізняється від усіх попередніх, тому що в ньому закладено механізм підтримки різноманітності індивідів. Цей метод являє собою комбінацію турнірної селекції та концепції домінування по Парето. Етап призначення придатності замінюється модифікованою схемою поділу придатності з використанням поняття ніші, яка визначається для індивідів у просторі цільових функцій і забезпечує можливість підтримання різноманітності, дозволяючи отримати необхідну множину Парето. Алгоритм використовує оригінальний механізм селекції найкращих особин, що забезпечує різноманітність популяції. Обрана пара особин перевіряється на недомінованість випадковою порівняльною множиною. Недомінована особина допускається до селекції. Якщо виявилось так, що обидві особини виявилися домінованими чи недомінованими порівняльною множиною, тоді переможцем турніру обирається та особина, у радіусі певної ніші *σshare* якої знаходиться менша кількість сусідніх особин. Це дозволяє підтримувати різноманітність популяції, та запобігати накопичуванню рішень у одній області.

Функціональні обмеження – додаток повинен бути розгорнутий на сервері. Функціональні обмеження – для роботи з додатком необхідна ПЕОМ з доступом до сіті «Інтернет» та будь-яким сучасним браузером.

* + 1. **Опис логічної структури**

Додаток складається з головної сторінки index.html, файлів rwga.js, vega.js, moga.js, npga.js, styles.css, а також допоміжних файлів highcharts.js та jquery.js.

Для початку роботи користувач програми повинен відкрити сторінку index.htmlу своєму web-браузері. На цій сторінці реалізований інтерфейс програми. Користувач бачить обчислювальні функції, а також поля вводу, куди необхідно ввести параметри генетичного алгоритму - кількість ітерацій алгоритму, розмір популяції та вірогідність мутації. Також за допомогою радіо-кнопок користувач повинен обрати необхідний алгоритм багатокритеріальної оптимізації.

Ідентифікатори елементів інтерфейсу, призначених для вводу початкових даних для роботи програми, та змінні, які вони зберігають, наведені у таблиці 3.1.

Таблиця 3.1 – Зчитувана інформація

|  |  |
| --- | --- |
| Поле | Опис |
| populationSize | Містить значення розміру популяції, яка використовується генетичним алгоритмом. Розмір популяції за замовчуванням - 100 |

Закінчення таблиці 3.1

|  |  |
| --- | --- |
| Поле | Опис |
| iterationLimit | Містить значення кількості ітерацій, після виконання яких генетичний алгоритм завершує свою роботу. Ліміт ітерацій за замовчуванням - сто кроків |
| mutationProbability | Містить значення вірогідності мутації, що застосовується у генетичному операторі мутації для кожного за представлених алгоритмів |
| algo1 | Перша радіо-кнопка, при обиранні якої програма здійснює багатокритеріальну генетичну оптимізацію за допомогою алгоритму RWGA. Обрана за замовченням. |
| algo2 | Друга радіо-кнопка, при обиранні якої програма здійснює багатокритеріальну генетичну оптимізацію за допомогою алгоритму VEGA. |
| algo3 | Третя радіо-кнопка, при обиранні якої програма здійснює багатокритеріальну генетичну оптимізацію за допомогою алгоритму MOGA. |
| algo4 | Четверта радіо-кнопка, при обиранні якої програма здійснює багатокритеріальну генетичну оптимізацію за допомогою алгоритму NPGA. |

Після того, як користувач запустить обчислення, натиснувши на кнопку "Оптимізувати", буде викликана функція Optimize(). Програма за допомогою методу getElementById() зчитає введені користувачем початкові дані.

Залежно від обраного методу оптимізації програма виконає багатокритеріальну оптимізацію за алгоритмами RWGA, VEGA, MOGA або NPGA, кожен з яких описаний у відповідному файлі, визиваючи необхідну функцію - відповідно rwga(), vega(), moga() або npga().

Функція rwga(), що реалізує алгоритм RWGA, знаходиться у файлі rwga.js. У цьому методі оптимізції загальна цільова функція будується з окремих цільових функцій у вигляді їх суми, де для кожної цільової функції визначається окрема вага. При фіксованих вагах в даному підході генетичний алгоритм відображає тенденцію постійного напряму пошуку, в той час як використання випадкових ваг відображає тенденцію змінного напрямку пошуку, більш пристосованою для пошуку фронту рішень.

Завантаживши введені користувачем початкові дані, алгоритм створює початкову популяцію індивідів. Розмір популяції зберігається у змінній length, значення якої береться зі введених користувачем параметрів алгоритму. Популяція є набором масивів: inds[], indv[], indl[], indd[], indd0[], де кожний елемент масиву зберігає значення змінних для кожного індивідууму популяції (фенотип особини).

Далі програма ініціює цикл, кількість ітерацій у якому дорівнює змінній limit, яка була задана користувачем.

Потім визначаються випадкові ваги для кожного з критеріїв оптимізації, сума яких дорівнює одиниці.

Для кожної особини додаток вираховує загальну цільову функцію яка складається з кожної цільової функції *fi(x)* за формулою, зазначеною у 3.1.1.

Для селекції найкращих особин програма використовує турнірний відбір, без з пари випадково обраних індивідуумів обирається той, у якого менше значення фітнес-функції.

Далі для отримання нових особин популяції, застосовується механізм лінійного схрещування.

Значення кожної змінної для кожної особини-нащадка знаходиться за допомогою лінійного кросинговеру, де використовуються значення змінних першої та другої батьківської особини, отриманих за допомогою турнірного відбору.

Кожна пара батьківських особин дає два нащадка. Провівши процедуру схрещування *N* разів, отримуємо популяцію нащадків.

На наступному кроці виконується генетичний оператор мутації. З деякою зазначеною вірогідністю значення хромосом особини змінюється на деяке невелике випадкове число. Мутація допомагає підтримувати різноманітність популяції.

Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

Алгоритм завершує свою роботу тоді, коли буде вичерпано ліміт ітерацій limit у циклу.

Функція vega(), що знаходиться у файлі vega.js, реалізує генетичний алгоритм VEGA. Селекція проводиться за придатністю особин окремо для кожного з критеріїв. Проміжна популяція заповнюється рівними порціями індивідів, відібраних по кожному з окремих критеріїв - для кожного з критеріїв створюється підпопуляція розміром *N / K*, де *N* - розмір всієї популяції. Відбір особин відбувається за допомогою турнірної селекції щодо придатності за кожним критерієм окремо.

Отримавши зчитані з полів вводу початкові дані, алгоритм створює початкову популяцію індивідів. Розмір популяції зберігається у змінній length, значення якої береться зі введених користувачем параметрів алгоритму. Популяція є набором масивів: inds[], indv[], indl[], indd[], indd0[], де кожний елемент масиву зберігає значення змінних для кожного індивідууму популяції.

Далі програма ініціює цикл, кількість ітерацій у якому дорівнює змінній limit, яка була задана користувачем. Для кожного з *K* критеріїв створюється підпопуляція розміром *N/K*, де *N* – розмір всієї популяції. Для кожної підпопуляції вираховується окреме значення фітнес-функції *Fk(x)* по цільовій функції *fk(x).*

Турнірний відбір виконується окремо для кожної з підпопуляцій - обидві особини для порівняння на кожному кроці турніру беруться з однієї підпопуляції.

Для отримання нових особин популяції, застосовується процедури схрещування та мутації. Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

Алгоритм завершує свою роботу тоді, коли буде вичерпано ліміт ітерацій limit у циклу.

Функція moga(), що знаходиться файлі moga.js, реалізує алгоритм МОGA. Цей алгоритм використовує процедуру ранжирування індивідів, де ранг кожного індивіда визначається числом домінуючих його індивідів, яка називається рангами Голдберга.

Зчитавши з полів вводу початкові дані, алгоритм створює початкову популяцію індивідів. Розмір популяції зберігається у змінній length. Популяція є набором масивів: inds[], indv[], indl[], indd[], indd0[], де кожний елемент масиву зберігає значення змінних для кожної особини.

Далі для кожного індивідууму у популяції визначаються його ранги:

1. з популяції обираються два індивіди - *i* та *j*;
2. для обраних індивідів порівняти один з одним значення фітнес-функцій по кожній функції *fk(x)*;
3. якщо індивід *i* кращий за індивід *j* хоч би по одному з критеріїв оптимізації, та не гірший за рештою критеріїв, тоді *j* - домінуєме рішення відносно *i*;
4. відповідно, якщо індивід *j* кращий за індивід *i* хоч би по одному з критеріїв оптимізації, але не гірший за рештою критеріїв, тоді *i* - домінуєме рішення відносно *j*;
5. ранг домінованої особини збільшується на одиницю.

Таким чином, кожній особині буде присвоєний ранг, що дорівнює кількості особин, якими домінується цей індивід.

Далі програма проводить селекцію найкращих особин за допомогою турнірного відбору. Пари індивідів порівнюються не значення їх фітнес-функцій, а отримані на попередньому кроці ранги.

Для отримання нових особин популяції, застосовуються генетичні оператори схрещування та мутації. Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

Коли буде вичерпано ліміт ітерацій limit у циклу, алгоритм завершить свою роботу.

У файлі npga.js знаходиться функція npga(), у якій був реалізований генетичний алгоритм багатокритеріальної оптимізації NPGA. У цьому генетичному алгоритмі закладено механізм підтримки різноманітності, що запобігає передчасному сходженню рішення.

Отримавши вхідні дані, алгоритм створює початкову популяцію індивідів. Розмір популяції зберігається у змінній length, значення якої береться зі введених користувачем параметрів алгоритму. Популяція є набором масивів: inds[], indv[], indl[], indd[], indd0[], де кожний елемент масиву зберігає значення змінних для кожного індивідууму популяції.

Потім ініціюється цикл, кількість ітерацій у якому дорівнює змінній limit, яка була зчитана з відповідного поля вводу.

Далі алгоритм використовує оригінальний механізм відбору індивідів, що забезпечує різноманітність популяції:

1. з популяції розміром *N* обирається випадкова порівняльна множина розміром *M*;
2. з популяції обираються два випадкових індивіда *i* та *j*;
3. якщо індивід *i* хоч би за одною цільовою функцією кращий за усі індивіди порівняльної множини, та не гірший за усіма іншими цільовими функціями, тоді індивід *i* - недомінуємий порівняльною множиною;
4. відповідно, якщо індивід *j* хоч би за одною цільовою функцією кращий за усі індивіди порівняльної множини, та не гірший за усіма іншими цільовими функціями, тоді індивід *j* - недомінуємий порівняльною множиною;
5. якщо індивід *i* - недомінуємий , а індивід *j* - домінуємий, тоді індивід *i* перемагає турнір і допускається до схрещування;
6. інакше, коли індивід *j* - недомінуємий , а індивід *i* - домінуємий, тоді індивід *j* перемагає турнір.

Якщо виявилось так, що жодна з цих умов не була виконана, і обидві особини виявилися домінованими чи недомінованими порівняльною множиною, тоді переможцем турніру обирається та особина, у радіусі певної ніші *σshare* якої знаходиться менша кількість сусідніх особин. Це дозволяє підтримувати різноманітність популяції, та запобігати накопичуванню рішень у одній області.

Для отримання нових особин популяції, застосовується процедури схрещування та мутації. Після використання генетичних операторів, батьківська популяція змінюється на популяцію нащадків.

Алгоритм завершує свою роботу тоді, коли буде вичерпано ліміт ітерацій limit у циклу.

Після завершення роботи будь-якого з обраних генетичних алгоритмів, у файлі index.html програма виведе на екран побудовані графіки та результати обчислювання, тобто мінімальні значення цільових функцій, та відповідні значення змінних.