ДОДАТОК А

Графічний матеріал атестаційної роботи

ГЮИК.503100.003

(позначення документу)

Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник атестаційної роботи
проф. Безкоровайний В.В.

МЕТОДИ ВИЗНАЧЕННЯ ПІДМНОЖИН ЕФЕКТИВНИХ ВАРІАНТІВ У СИСТЕМАХ ПІДТРИМКИ ПРИЙНЯТТЯ ПРОЕКТНИХ РІШЕНЬ

Графічний матеріал

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮІК.503100.003-ЛЗ

РОЗРОБИВ:
ст. гр. СПРм-18-2
Вакуленко В.К.

ЗАТВЕРДЖЕНО
ГЮИК.503100.003-ЛЗ

МЕТОДИ ВИЗНАЧЕННЯ ПІДМНОЖИН ЕФЕКТИВНИХ ВАРІАНТІВ У СИСТЕМАХ ПІДТРИМКИ ПРИЙНЯТТЯ ПРОЕКТНИХ РІШЕНЬ

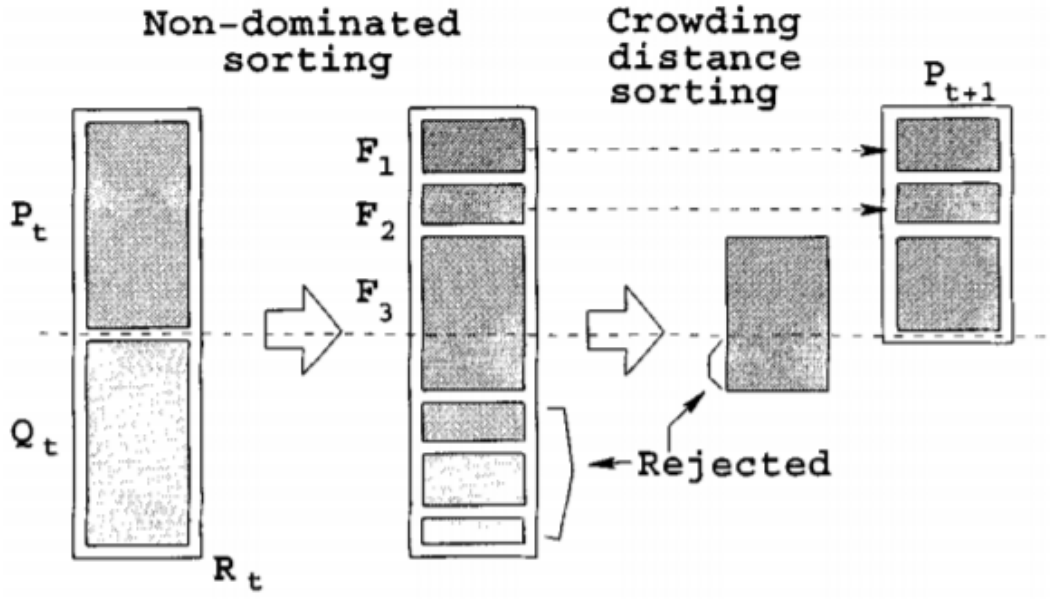
Графічний матеріал

ГЮИК.503100.003

АРКУШІВ 7

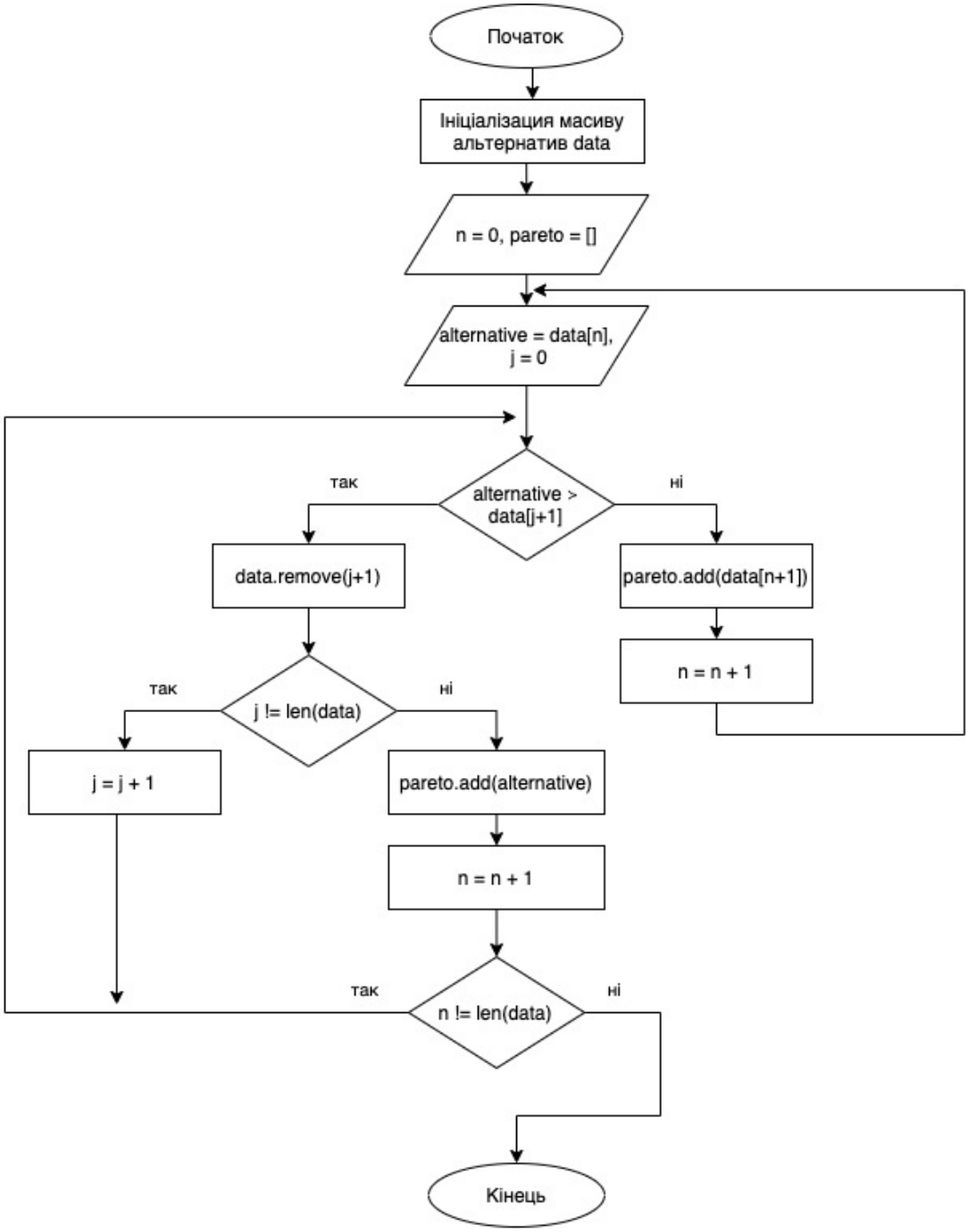
2020 р.

СХЕМА РОБОТИ АЛГОРИТМУ NSGA 2



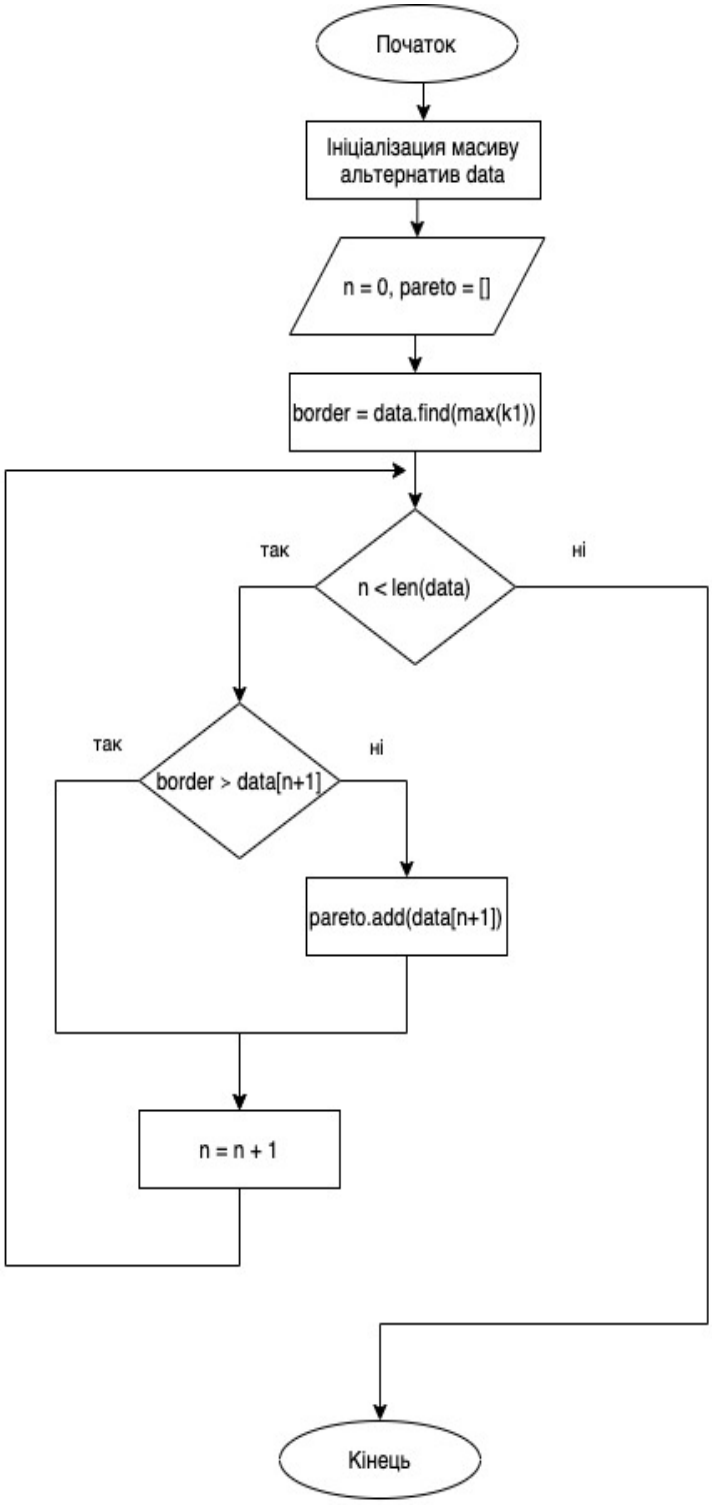
Розробив	*Вакуленко В.К.*			*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

АЛГОРИТМ ПОПАРНИХ ПОРІВНЯНЬ



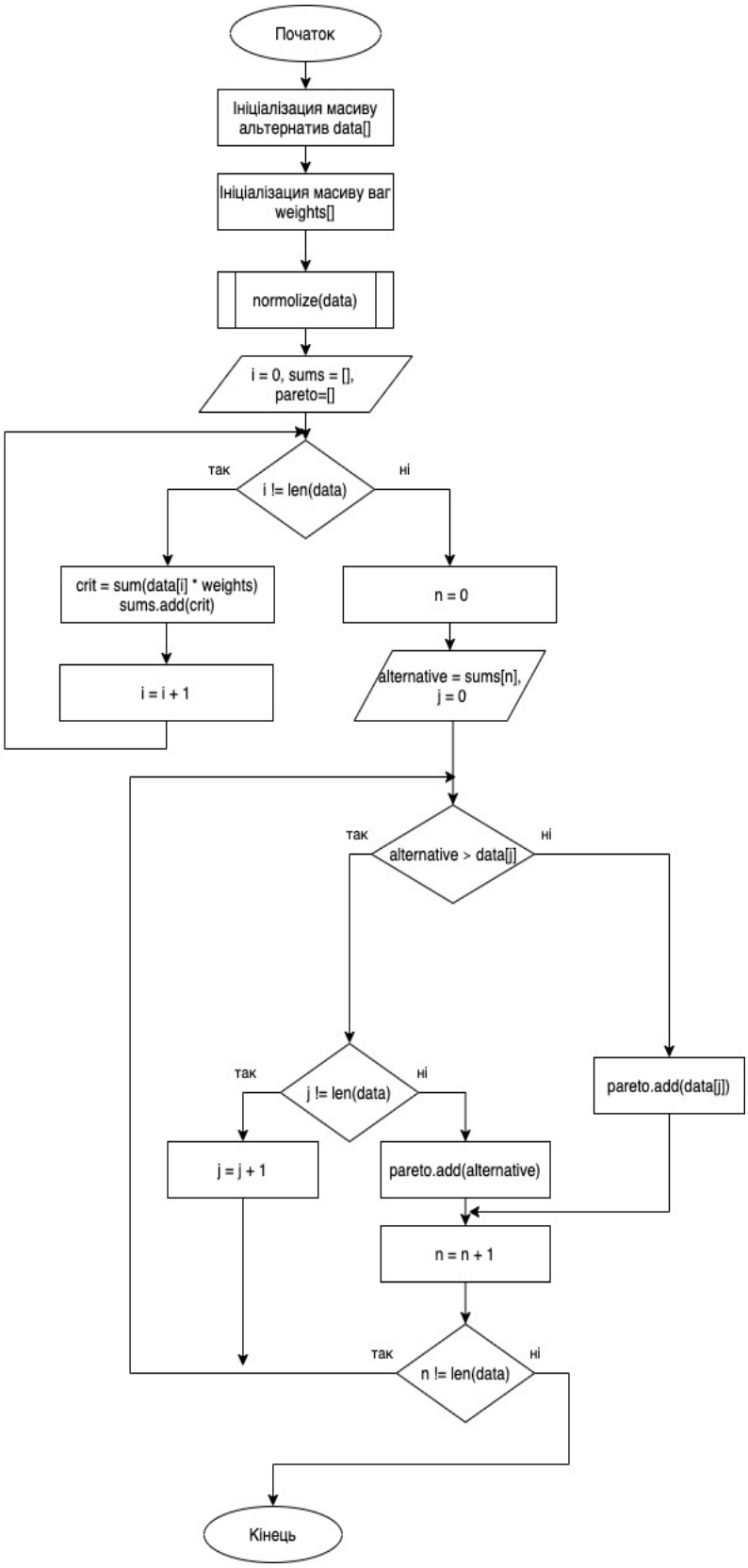
Розробив	*Вакуленко В.К.*			*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

АЛГОРИТМ ВИДІЛЕННЯ НАБЛИЖЕНОЇ МНОЖИНИ КОМПРОМІСІВ



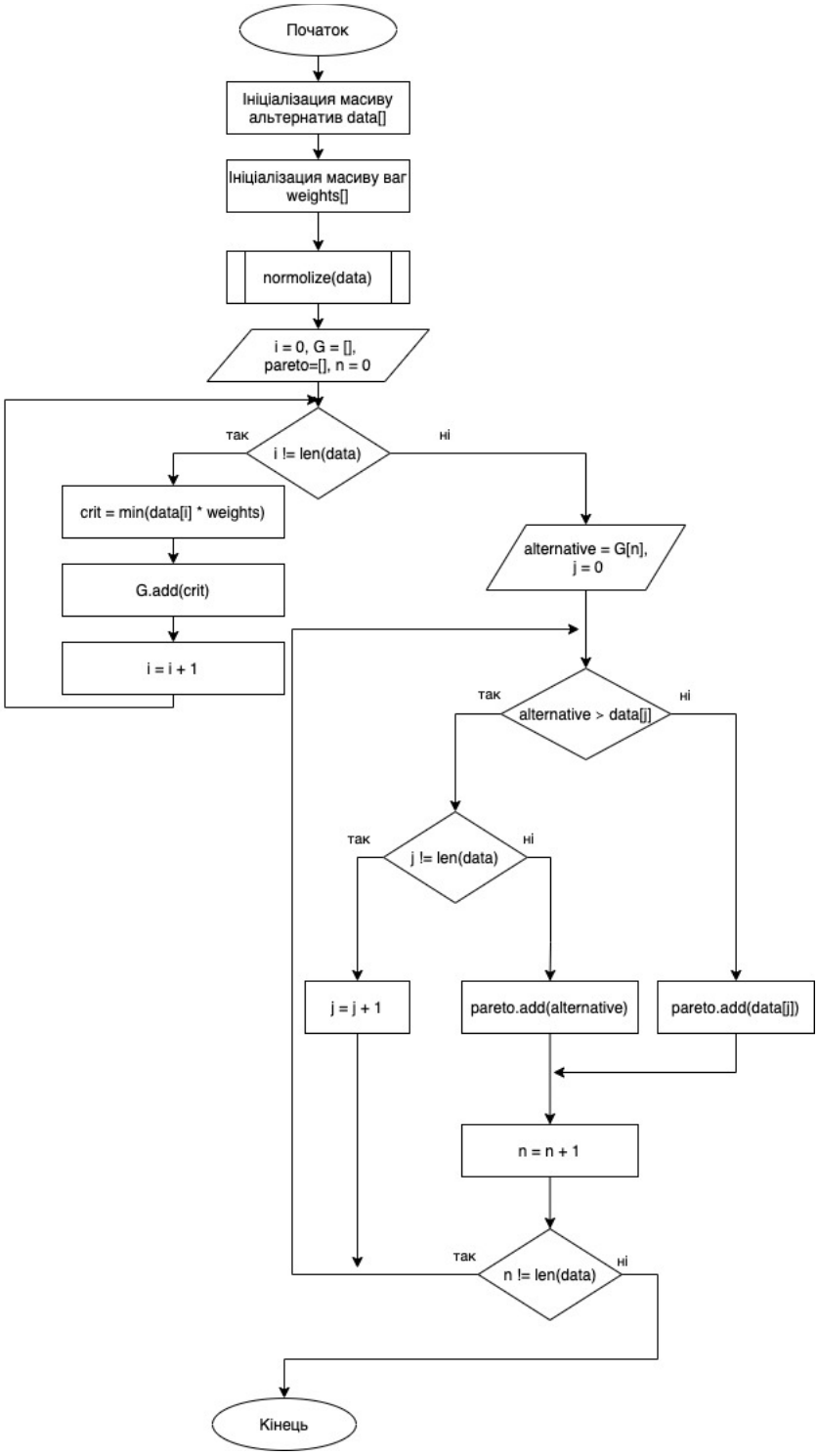
Розробив	*Вакуленко В.К.*			*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

АЛГОРИТМ МЕТОДУ КАРЛІНА



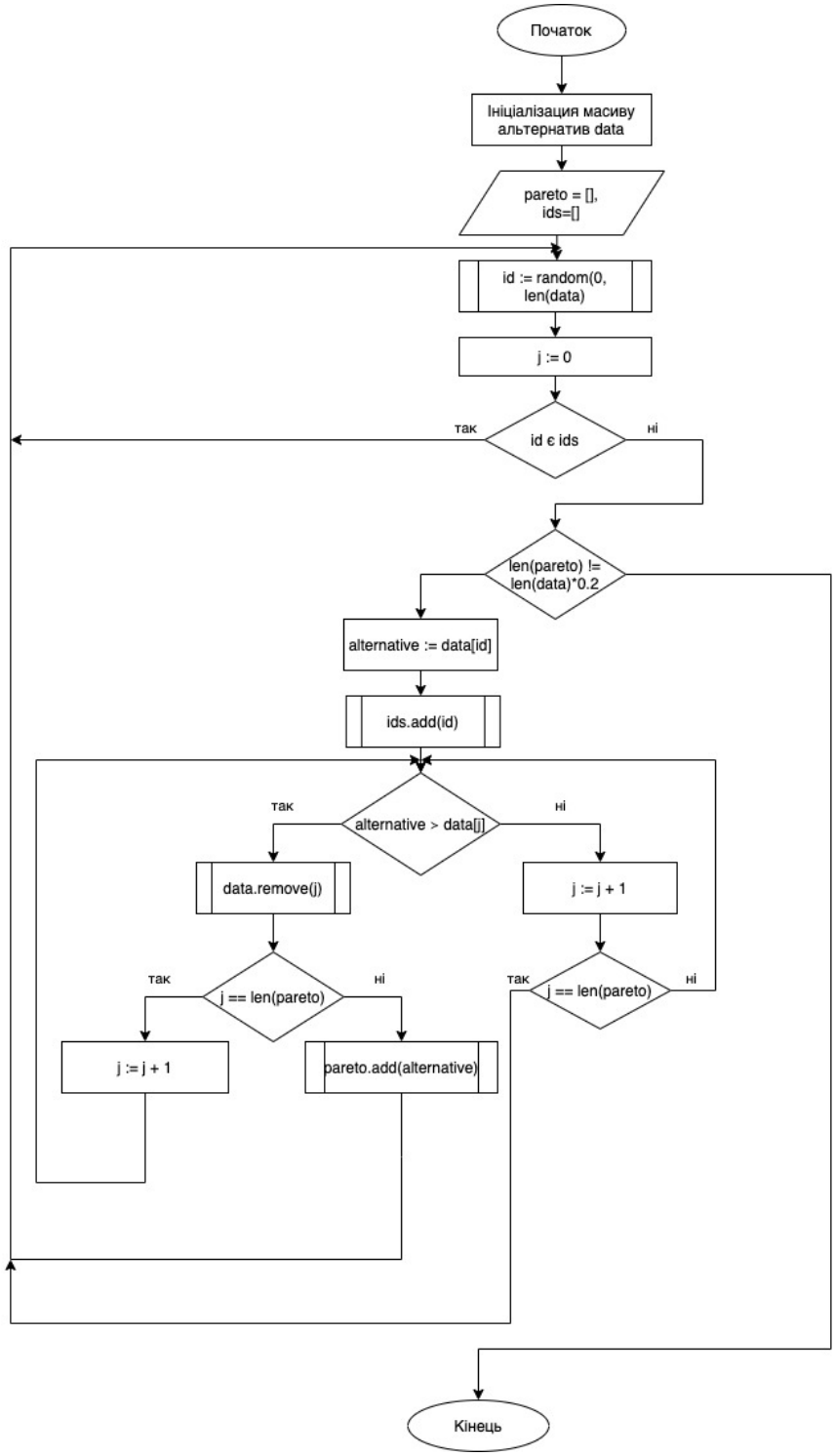
Розробив	*Вакуленко В.К.*			*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

АЛГОРИТМ МЕТОДУ ЗГОРТКИ ГЕРМЕЙЄРА



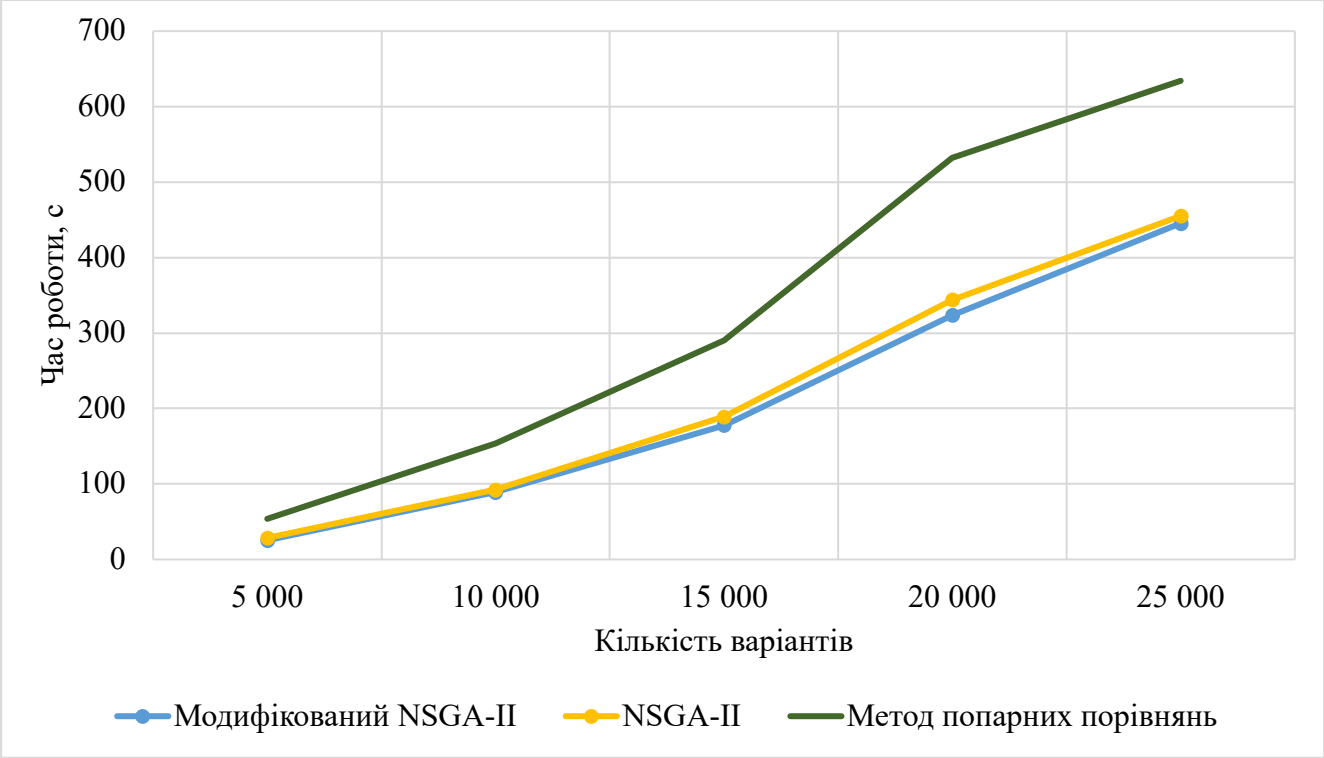
Розробив	*Вакуленко В.К.*			*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

АЛГОРИТМ ВИПАДКОВОГО ПОШУКУ



Розробив	*Вакуленко В.К.*			*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ПОРІВЯННЯ МЕТОДІВ NSGA-II, ЙОГО МОДИФИКАЦИЇ ТА ПОПАРНИХ ПОРІВНЯНЬ



Розробив	*Вакуленко В.К.*			*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	
Перевірив	*Безкоровайний В.В.*				
Н. контроль	*Безкоровайний В.В.*			*СПРм-18-2*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ДОДАТОК Б

Текст програми

ГЮИК.503100.003 – 01 12 01

(позначення документу)

Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
керівник атестаційної роботи
проф. Безкоровайний В.В.

МЕТОДИ ВИЗНАЧЕННЯ ПІДМНОЖИН ЕФЕКТИВНИХ ВАРІАНТІВ У СИСТЕМАХ ПІДТРИМКИ ПРИЙНЯТТЯ ПРОЕКТНИХ РІШЕНЬ

MOOPTIMIZE

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК.503100.003-01 12 01-ЛЗ

РОЗРОБИВ:
ст. гр. СПРм-18-2
Вакуленко В.К.

2020 р

ЗАТВЕРДЖЕНО
ГЮИК.503100.003- 01 12 01-ЛЗ

МЕТОДИ ВИЗНАЧЕННЯ ПІДМНОЖИН ЕФЕКТИВНИХ ВАРІАНТІВ У СИСТЕМАХ ПІДТРИМКИ ПРИЙНЯТТЯ ПРОЕКТНИХ РІШЕНЬ

MOOPTIMIZE

Текст програми

ГЮИК.503100.003-01 12 01

АРКУШІВ 9

2020 р.

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 738,
   "metadata": {},
   "outputs": [],
   "source": [
    "import numpy as np\n",
    "import time\n",
    "from decimal import Decimal\n",
    "import struct\n",
    "import math\n",
    "import matplotlib.pyplot as plt\n",
    "from pymoo.algorithms.nsga2 import calc_crowding_distance\n",
    "from scipy import spatial"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {},
   "outputs": [],
   "source": [
    "lines = []\n",
    "data = []\n",
    "with open('test_data.txt', 'r') as f:\n",
    "    lines = f.readlines()[8:]\n",
    "for l in lines:    \n",
    "    new = l.replace(' \\\n', '').replace(',', '.').split(' ')\n",
    "    d = [*map(lambda x: float(x), new)]\n",
    "    data.append(d)\n",
    "data = np.array(data)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {},
   "outputs": [],
   "source": [
    "row_sums = data.sum(axis=1)\n",
    "new_matrix = data / row_sums[:, np.newaxis]"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1279,
   "metadata": {},
   "outputs": [],
   "source": [
```

```
"def hipow(minX, maxX):\n",
"    ''' Return the highest power of 2 within n. '''\n",
"    tmp = (maxX - minX) * np.power(10, 2)\n",
"    exp = 0\n",
"    while 2**exp <= tmp:\n",
"        exp += 1\n",
"    return [exp, tmp]\n",
"\n",
"def encode(value, minLimit, maxLimit):\n",
"    hiPowFirst = hipow(minLimit,maxLimit)\n",
"    firstHiPow = hiPowFirst[0]\n",
"    highValue = hiPowFirst[1]\n",
"    ratio = float(format((highValue / np.power(2, firstHiPow)), '.5f'))\n",
"    binValue = bin(math.floor(value / ratio * np.power(10,2)))\n",
"    binValue = ('00000000'+ binValue[2:])[-firstHiPow:]\n",
"    return binValue\n",
"\n",
"def decode(binary, minLimit, maxLimit):\n",
"    hiPowFirst = hipow(minLimit,maxLimit)\n",
"    firstHiPow = hiPowFirst[0]\n",
"    highValue = hiPowFirst[1]\n",
"    ratio = float(format((highValue / np.power(2, firstHiPow)), '.5f'))\n",
"    decimalNum = int(binary,2)\n",
"    converted = (decimalNum * ratio)/ np.power(10,2)\n",
"    return round(float((format(converted, '.5f'))), 3)\n",
"\n",
"def getChromosome(values, population_size):\n",
"    chromosomes = ["  for x in range(population_size)]\n",
"    for i in range(len(values)):\n",
"        objectives = values[i]\n",
"        for j in range(len(objectives[0])):\n",
"            limits = objectives[1]\n",
"            chromosomes[j] = chromosomes[j] + str(encode(objectives[0][j], limits[0], limits[1]))\n",
"    return chromosomes\n",
"\n",
"def calculate_crowding(scores):\n",
"    \n",
"    size = len(scores[:, 0])\n",
"    num_scores = len(scores[0, :])\n",
"\n",
"    # create crowding matrix of population (row) and score (column)\n",
"    crowding_matrix = np.zeros((size, num_scores))\n",
"\n",
"    # normalise scores (ptp is max-min)\n",
"    normed_scores = (scores - scores.min(0)) / scores.ptp(0)\n",
"\n",
"    # calculate crowding distance for each score in turn\n",
"    for col in range(num_scores):\n",
"        crowding = np.zeros(size)\n",
"\n",
"        # end points have maximum crowding\n",
```

```
"        crowding[0] = 1\n",
"        crowding[size - 1] = 1\n",
"\n",
"        # Sort each score (to calculate crowding between adjacent scores)\n",
"        sorted_scores = np.sort(normed_scores[:, col])\n",
"\n",
"        sorted_scores_index = np.argsort(\n",
"            scores[:, col])\n",
"\n",
"        # Calculate crowding distance for each individual\n",
"        crowding[1:size - 1] = \\\n",
"            (sorted_scores[2:size] -\n",
"             sorted_scores[0:size - 2])\n",
"\n",
"        # resort to orginal order (two steps)\n",
"        re_sort_order = np.argsort(sorted_scores_index)\n",
"        sorted_crowding = crowding[re_sort_order]\n",
"\n",
"        # Record crowding distances\n",
"        crowding_matrix[:, col] = sorted_crowding\n",
"\n",
"    # Sum crowding distances of each score\n",
"    crowding_distances = np.sum(crowding_matrix, axis=1)\n",
"\n",
"    return crowding_distances\n",
"\n",
"def fast_non_dominated_sort(parent, children):\n",
"    S=[[] for i in range(0,len(parent))]\n",
"    front = [[]]\n",
"    n=[0 for i in range(0,len(parent))]\n",
"    rank = [0 for i in range(0, len(parent))]\n",
"\n",
"    for p in range(0,len(parent)):\n",
"        S[p]=[]\n",
"        n[p]=0\n",
"        for q in range(0, len(parent)):\n",
"            if (parent[p] > parent[q] and children[p] > children[q]) or (parent[p] >= parent[q] and
children[p] > children[q]) or (parent[p] > parent[q] and children[p] >= children[q]):\n",
"                if q not in S[p]:\n",
"                    S[p].append(q)\n",
"            elif (parent[q] > parent[p] and children[q] > children[p]) or (parent[q] >= parent[p] and
children[q] > children[p]) or (parent[q] > parent[p] and children[q] >= children[p]):\n",
"                n[p] = n[p] + 1\n",
"        if n[p]==0:\n",
"            rank[p] = 0\n",
"            if p not in front[0]:\n",
"                front[0].append(p)\n",
"\n",
"    i = 0\n",
"    while(front[i] != []):\n",
"        Q=[]\n",
```

```
"        for p in front[i]:\n",
"            for q in S[p]:\n",
"                n[q] =n[q] - 1\n",
"                if( n[q]==0):\n",
"                    rank[q]=i+1\n",
"                    if q not in Q:\n",
"                        Q.append(q)\n",
"        i = i+1\n",
"        front.append(Q)\n",
"\n",
"    del front[len(front)-1]\n",
"    return front\n",
"\n",
"def cross(population, size, point):     \n",
"    new_pop = [] \n",
"    for _ in range(size): \n",
"        p = population[np.random.randint(0, len(population))] \n",
"        m = population[np.random.randint(0, len(population))] \n",
"        entity = p[:point] + m[point:]\n",
"        new_pop.append(entity) \n",
"      \n",
"    return new_pop\n",
"\n",
"def function1(arr):\n",
"    return np.power(arr[0], 2) + np.power(arr[1], 2)\n",
"\n",
"def function2(arr):\n",
"    return np.power(arr[0], 2) + np.power(arr[1], 2)\n",
"\n",
"def calculateFitnessFunction(nums):\n",
"    func1_values = map(lambda f1: function1(f1), nums)\n",
"    func2_values = map(lambda f2: function2(f2), nums)\n",
"    #print(list(func1_values))\n",
"    return [list(func1_values), list(func2_values)]\n",
"\n",
"def mutation(chromosome, probability):\n",
"    prob = np.random.random()\n",
"    if(prob <= probability):\n",
"        point = np.random.randint(0, len(chromosome))\n",
"        chrm = list(chromosome)\n",
"        chrm[point] = str(1 - int(chrm[point]))\n",
"        chromosome = ".join(chrm)\n",
"    return chromosome\n",
"\n",
"def compareMethods(initial, pairSolutions, gaSolutions):\n",
"    count = 0\n",
"    similar_sol = []\n",
"    for sgi in range(len(gaSolutions)):\n",
"        if pairSolutions.count(gaSolutions[sgi]) > 0:\n",
"            count = count + 1\n",
"            similar_sol.append(initial[gaSolutions[sgi]])\n",
```

```
 "            print('Find solution ', gaSolutions[sgi], ': ', initial[gaSolutions[sgi]])\n",
 "    print('Total similar solutions from both methods', count)\n",
 "    print('Solutions: ')\n",
 "    print(similar_sol)\n",
 "    \n",
 "def getSolutionsGA(parent, child, solutionCount, popSize):\n",
 "    variants = parent + child\n",
 "    indexes = new_population[:solutionCount]\n",
 "    solution_ga = []\n",
 "    for vi in range(len(variants)):\n",
 "        if indexes.count(vi) > 0 and vi <= popSize:\n",
 "            solution_ga.append(vi)\n",
 "            #print('Variant ', vi + 1, ': ', variants[vi])\n",
 "    return solution_ga\n",
 "\n",
 "def identify_pareto(scores):\n",
 "    population_size = scores.shape[0]\n",
 "    population_ids = np.arange(population_size)\n",
 "    pareto_front = np.ones(population_size, dtype=bool)\n",
 "    for i in range(population_size):\n",
 "        for j in range(population_size):\n",
 "            if all(scores[j] >= scores[i]) and any(scores[j] > scores[i]):\n",
 "                pareto_front[i] = 0\n",
 "                break\n",
 "    return scores[pareto_front], population_ids[pareto_front]\n",
 "\n",
 "def drawPlot(dataXY):\n",
 "    xs = list(map(lambda x: x[0], dataXY))\n",
 "    ys = list(map(lambda y: y[1], dataXY))\n",
 "    plt.figure(figsize=(15,15))\n",
 "    plt.scatter(xs, ys)\n",
 "    for x,y in zip(xs,ys):\n",
 "\n",
 "        label = \"({:.2f};{:.2f})\".format(x,y)\n",
 "\n",
 "        plt.annotate(label, # this is the text\n",
 "                     (x,y), # this is the point to label\n",
 "                     textcoords=\"offset points\", # how to position the text\n",
 "                     xytext=(0,20), # distance from text to points (x,y)\n",
 "                     ha='center')\n",
 "\n",
 "    plt.show()"
 ]
},
{
 "cell_type": "code",
 "execution_count": 1264,
 "metadata": {},
 "outputs": [],
 "source": [
  "population_size = 6\n",
```

```
   "crossover_probablity = 0.25\n",
   "mutation_probability = 0.15\n",
   "initialValues = [[[10.00, 4.99, 6.09, 6.91, 5.21, 7.90], [0,10]],
[[19.61, 5.10, 0.79, 10.62, 18.87, 8.98], [0, 20]]]"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 1265,
  "metadata": {},
  "outputs": [],
  "source": [
   "lines = []\n",
   "test_val = []\n",
   "with open('test_data.txt', 'r') as f:\n",
   "    lines = f.readlines()[8:]\n",
   "for l in lines:    \n",
   "    new = l.replace(' \\\n', '').replace(',', '.').split(' ')\n",
   "    d = [*map(lambda x: round(float(x), 3), new)]\n",
   "    test_val.append(d)\n",
   "ntest_val = list(map(lambda v: [np.array(v), [0, 1]], list(zip(*test_val))))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 1266,
  "metadata": {},
  "outputs": [],
  "source": [
   "population_size = len(ntest_val[0][0])\n",
   "initialValues = ntest_val"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "scrolled": true
  },
  "outputs": [],
  "source": []
 },
 {
  "cell_type": "code",
  "execution_count": 1267,
  "metadata": {},
  "outputs": [],
  "source": [
 },
 {
  "cell_type": "code",
```

```
 "execution_count": 1268,
 "metadata": {},
 "outputs": [
  {
   "data": {
    "text/plain": [
     "16"
    ]
   },
   "execution_count": 1268,
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "population_size"
 ]
},
{
 "cell_type": "code",
 "execution_count": 1269,
 "metadata": {},
 "outputs": [],
 "source": [
  "maxGen = 5"
 ]
},
{
 "cell_type": "code",
 "execution_count": 1282,
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "-- Interation 1 --\n",
    "Fronts:\n",
    "[[9], [20], [6], [27], [29], [24], [1], [22], [18], [16], [19],
[10], [3], [2], [15], [17], [11], [12], [14], [7], [23], [4], [28], [21], [13], [0], [26], [5], [31], [8], [30],
[25]]\n",
    "-- Interation 2 --\n",
    "Fronts:\n",
    "[[22], [17], [28], [21], [19, 25, 27], [20], [26], [23], [31], [30], [29], [18], [16], [24], [12], [11], [1],
[6], [8], [2], [7], [0], [13], [3], [5], [4], [9], [15], [10], [14]]\n",
    "Fronts:\n",
    "[[27, 28], [29], [23], [16],
[19], [24], [18], [21], [31], [20], [25], [30], [22], [17], [26], [5], [11],
 [7], [8], [13], [0], [6], [15], [9], [12], [3], [10], [2], [14], [1], [4]]\n",
    "Fronts:\n",
    "[[22], [23], [30], [25],
```

[21], [29], [18], [24], [17], [20, 26, 28], [27], [19], [31], [16], [11], [12], [10], [15], [7], [13], [0], [6], [3], [2], [8], [9], [1], [4], [14], [5]]\n",
"[[6.80122000e-01 6.80122000e-01]\n",
"-- Interation 5 --\n",
"Fronts:\n",
"[[20], [22], [26], [18], [29], [25], [31], [17], [16], [21], [30], [28], [24], [23], [19], [27], [3], [2], [9], [10], [6], [0], [12], [7], [1], [5], [4], [14], [11], [15], [8], [13]]\n",
"[[[0.159205, 0.159205]]]\n"
]
}
],
"source": [
"population = initialValues\n",
"count = 0\n",
"while count < maxGen:\n",
" initialPopulation = getChromosome(population, population_size)\n",
" #print(initialPopulation)\n",
" #point = math.floor(len(initialPopulation[0]) * crossover_probablity)\n",
" point = 4\n",
" child_population = list(map(\n",
"   lambda individual: mutation(individual, mutation_probability), cross(initialPopulation, population_size, point)))\n",
" decoded = [[] for x in range(len(child_population))]\n",
" for i in range(len(child_population)):\n",
"   individuals = child_population[i]\n",
"   #decode_point = hipow(0, 1)[0]\n",
"   decode_point = 7\n",
"   size = decode_point\n",
"   #fdecode=[individuals[y-size:y] for y in range(size, len(individuals)+size,size\n",
"   fdecode=[individuals[:7], individuals[7:]]\n",
"   dec = []\n",
"   for fdi in range(len(fdecode)):\n",
"   dec.append(decode(fdecode[fdi], 0, 1))\n",
"   decoded[i] = dec\n",
" #print(decoded)\n",
" calculation = [[] for x in range(population_size)]\n",
" #print(population)\n",
" for m in range(len(population)):\n",
"   data = population[m]\n",
"   for k in range(population_size):\n",
"   calculation[k].append(data[0][k])\n",
" #print(calculation)\n",
" child_values = calculateFitnessFunction(decoded)\n",
" #print(decoded)\n",
" combined_child = np.vstack((child_values[0], child_values[1])).T\n",
" #print(combined_child)\n",
" parent_values = calculateFitnessFunction(calculation)\n",
" combined_parent = np.vstack((parent_values[0], parent_values[1])).T\n",
" #print(combined_parent)\n",
" Rt = np.append(combined_parent, combined_child, axis = 0)\n",

```
   "    #print(Rt)\n",
   "    fronts = fast_non_dominated_sort(np.append(parent_values[0], child_values[0]),
np.append(parent_values[1], child_values[1]))\n",
   "    #fast_non_dominated_sort(parent_values, child_values)\n",
   "    #print(fronts)\n",
   "    solutions = []\n",
   "    for f in range(len(fronts)):\n",
   "        front = fronts[f]\n",
   "        solution = []\n",
   "        for ki in range(len(front)):\n",
   "            idx = front[ki]\n",
   "            solution.append(list(Rt[idx]))\n",
   "        solutions.append(solution)\n",
   "\n",
"metadata": {
 "kernelspec": {
  "display_name": "Python 3",
  "language": "python",
  "name": "python3"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.7.6"
 }
},
"nbformat": 4,
"nbformat_minor": 4
}
```

ДОДАТОК В

Сертифікат учасника конференції

ДОДАТОК Г

Відомість атестаційної роботи

ГЮИК.503100.003

(позначення документу)

№	Позначення	Найменування	Доп. відомості
		Текстові документи	
1	***ГЮИК.503100.003 ПЗ***	*Пояснювальна записка*	*77 с.*
2	***ГЮИК.503100.003- 01 12 01***	*Текст програми*	*9 с.*
		Графічні документи	
3		*Схема роботи алгоритму*	
		NSGA 2	*1 аркуш*
4		*Алгоритм попарних*	
		порівнянь	*1 аркуш*
5		*Алгоритм виділення*	
		наближеної області	
		компромісів	*1 аркуш*
6		*Алгоритм методу*	
		Карліна	*1 аркуш*
7		*Алгоритм методу*	
		згортки Гермейєра	*1 аркуш*
8		*Алгоритм випадкового*	
		пошуку	*1 аркуш*
9		*Порівняння методів*	
		NSGA-II, його модифікації	
		та попарних порівнянь	*1 аркуш*

					ГЮИК.503100.003 ДЗ		
Змін.	*Арк.*	*№ документа*	*Підпис*	*Дата*			
Розробив		*Вакуленко В.К.*		20.05.20	*Методи визначення підмножин ефективних варіантів у системах підтримки прийняття проектних рішень*	Аркуш	Аркушів
Перевірив		*Безкоровайний В.В.*		20.05.20		*1*	*1*
Н. контроль		*Безкоровайний В.В.*		20.05.20		*ХНУРЕ Кафедра СТ*	
Затвердив		*Гребеннік І.В.*					