

Artificial Intelligence Lab – Genetic Algorithms

Submission date :23/3/2021

Shadi Halloun – 313552309

Noor Khamaisi – 207076076

קישור ל GitHub שלנו , Source קוד וקובץ ה PDF מופיעים בצורה מסודרת.

<https://github.com/shadihalloun35/ArtificialIntelligenceLab.git>

Part A:

1- Average and standard deviation code :

```
void calc_fitness(ga_vector &population)
{
    string target = GA_TARGET;
    int tsize = target.size();
    unsigned int fitness;
    float average = 0;
    float deviation = 0;

    for (int i = 0; i < GA_POPSIZE; i++) {
        fitness = 0;
        for (int j = 0; j < tsize; j++) {
            fitness += abs(int(population[i].str[j] - target[j]));
        }

        population[i].fitness = fitness;
        average += fitness;
    }

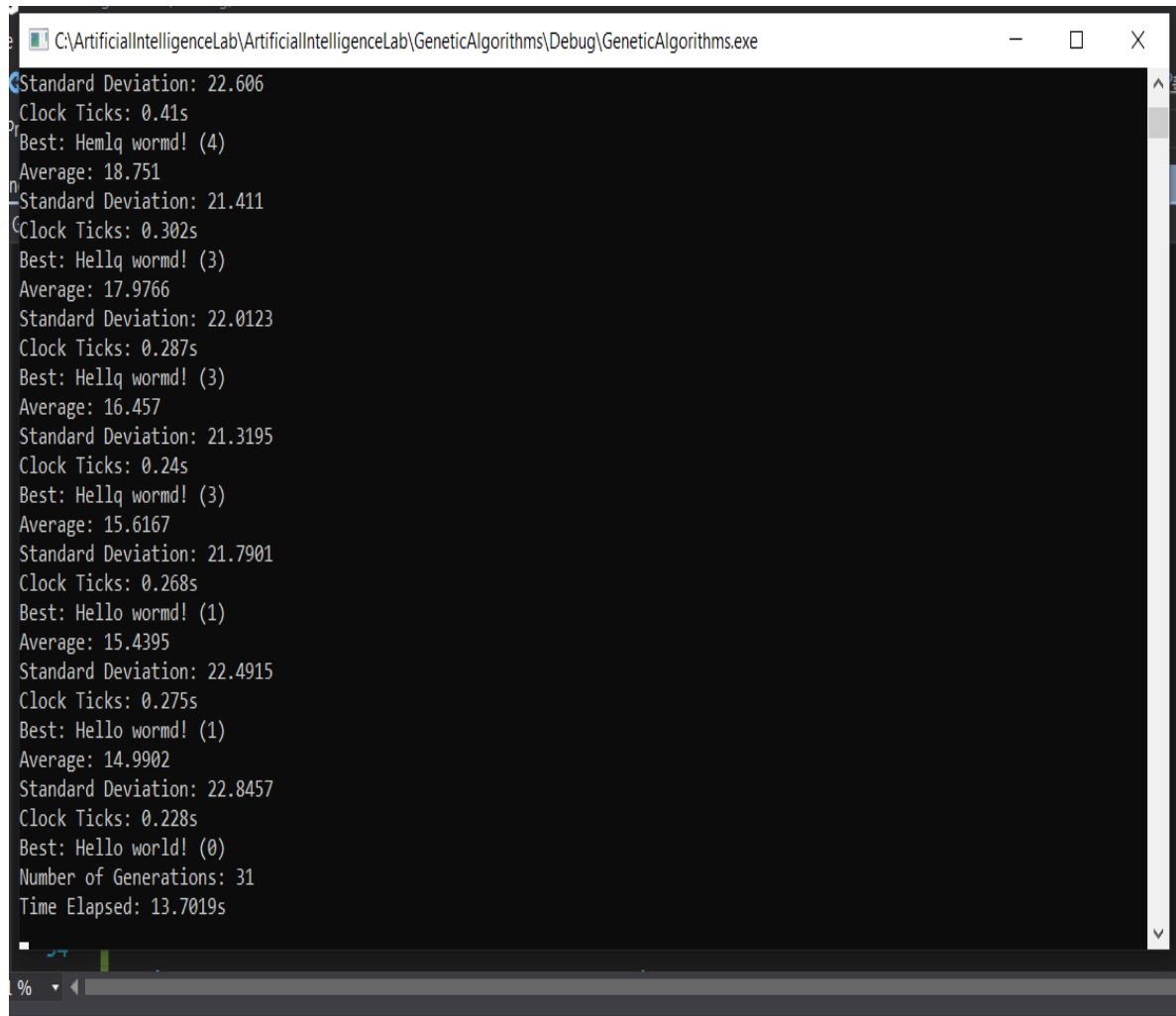
    average = average / GA_POPSIZE;           //calculating the average

    for (int i = 0; i < GA_POPSIZE; i++) {
        deviation += pow(population[i].fitness - average, 2);
    }

    deviation = sqrt(deviation / GA_POPSIZE);    //calculating the std deviation

    for (int i = 0; i < GA_POPSIZE; i++) {      //updating the average and the deviation for each citizen for the current generation
        population[i].average = average;
        population[i].deviation = deviation;
    }
}
```

Average and standard deviation results :



```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\GeneticAlgorithms\Debug\GeneticAlgorithms.exe
Standard Deviation: 22.606
Clock Ticks: 0.41s
Best: Hemlq wormd! (4)
Average: 18.751
Standard Deviation: 21.411
Clock Ticks: 0.302s
Best: Hellq wormd! (3)
Average: 17.9766
Standard Deviation: 22.0123
Clock Ticks: 0.287s
Best: Hellq wormd! (3)
Average: 16.457
Standard Deviation: 21.3195
Clock Ticks: 0.24s
Best: Hellq wormd! (3)
Average: 15.6167
Standard Deviation: 21.7901
Clock Ticks: 0.268s
Best: Hello wormd! (1)
Average: 15.4395
Standard Deviation: 22.4915
Clock Ticks: 0.275s
Best: Hello wormd! (1)
Average: 14.9902
Standard Deviation: 22.8457
Clock Ticks: 0.228s
Best: Hello world! (0)
Number of Generations: 31
Time Elapsed: 13.7019s
```

2- Clock ticks and time elapsed code :

```
int main()
{
    using clock = std::chrono::system_clock;
    using sec = std::chrono::duration<double>;
    const auto before = clock::now();
    int numOfGenerations = 0;
    srand(unsigned(time(NULL)));

    ga_vector pop_alpha, pop_beta;
    ga_vector *population, *buffer;

    init_population(pop_alpha, pop_beta);
    population = &pop_alpha;
    buffer = &pop_beta;

    for (int i = 0; i < GA_MAXITER; i++) {
        clock_t begin = std::clock();

        calc_fitness(*population);    // calculate fitness
        sort_by_fitness(*population); // sort them
        print_best(*population);      // print the best one

        if ((*population)[0].fitness == 0) break;

        mate(*population, *buffer);    // mate the population together
        swap(population, buffer);      // swap buffers

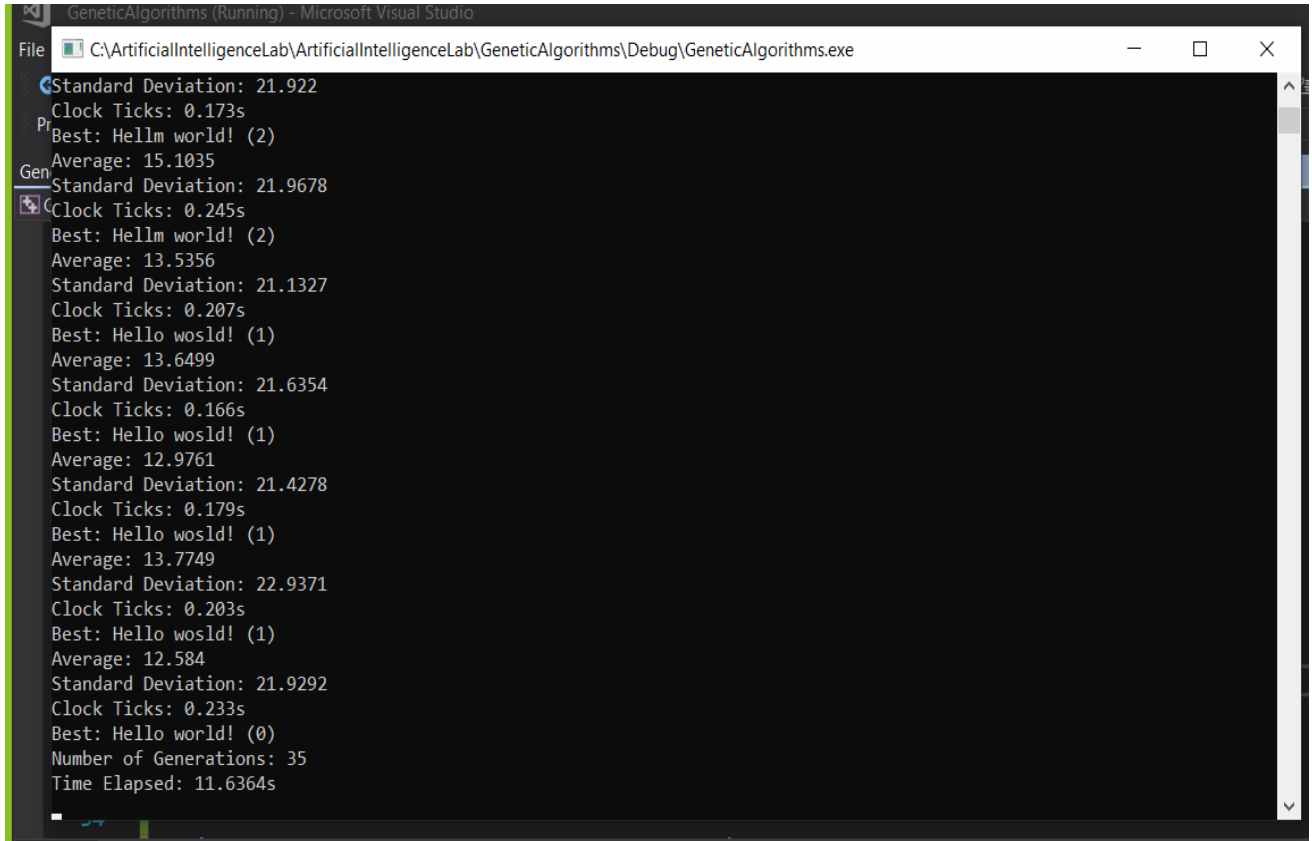
        clock_t end = std::clock();
        float time_spent = (float)(end - begin) / CLOCKS_PER_SEC;
        numOfGenerations += 1;
        std::cout << "Average: " << (*population)[0].average << std::endl;
        std::cout << "Standard Deviation: " << (*population)[0].deviation << std::endl;

        std::cout << "Clock Ticks: " << time_spent << "s" << std::endl;
    }

    std::cout << "Number of Generations: " << numOfGenerations << std::endl;

    const sec duration = clock::now() - before;
    std::cout << "Time Elapsed: " << duration.count() << "s" << std::endl;
}
```

Clock ticks and time elapsed results :



```
GeneticAlgorithms (Running) - Microsoft Visual Studio
File C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\GeneticAlgorithms\Debug\GeneticAlgorithms.exe
Standard Deviation: 21.922
Clock Ticks: 0.173s
Pr Best: Hellm world! (2)
Average: 15.1035
Gen Standard Deviation: 21.9678
Clock Ticks: 0.245s
Best: Hellm world! (2)
Average: 13.5356
Standard Deviation: 21.1327
Clock Ticks: 0.207s
Best: Hello wosld! (1)
Average: 13.6499
Standard Deviation: 21.6354
Clock Ticks: 0.166s
Best: Hello wosld! (1)
Average: 12.9761
Standard Deviation: 21.4278
Clock Ticks: 0.179s
Best: Hello wosld! (1)
Average: 13.7749
Standard Deviation: 22.9371
Clock Ticks: 0.203s
Best: Hello wosld! (1)
Average: 12.584
Standard Deviation: 21.9292
Clock Ticks: 0.233s
Best: Hello world! (0)
Number of Generations: 35
Time Elapsed: 11.6364s
```

3- Single Point , Two Point and Uniform crossover implementation

```
case 1:                // Single Point Operator

for (int i = esize; i < GA_POPSIZE; i++) {
    i1 = rand() % (GA_POPSIZE / 2);
    i2 = rand() % (GA_POPSIZE / 2);
    spos = rand() % tsize;

    buffer[i].str = population[i1].str.substr(0, spos) +
        population[i2].str.substr(spos, tsize - spos);

    if (rand() < GA_MUTATION) mutate(buffer[i]);
}

break;
```

```
case 2:                // Two Point Operator

for (int i = esize; i < GA_POPSIZE; i++) {
    i1 = rand() % (GA_POPSIZE / 2);
    i2 = rand() % (GA_POPSIZE / 2);
    spos = rand() % tsize;
    spos2 = rand() % tsize;

    buffer[i].str = population[i1].str.substr(0, std::min(spos, spos2)) +
        population[i2].str.substr(std::min(spos, spos2), std::max(spos, spos2) - std::min(spos, spos2)) +
        population[i1].str.substr(std::max(spos, spos2), tsize - std::max(spos, spos2));

    if (rand() < GA_MUTATION) mutate(buffer[i]);
}

break;
```

```
case 3:                // Uniform Point Operator

for (int i = esize; i < GA_POPSIZE; i++) {
    i1 = rand() % (GA_POPSIZE / 2);
    i2 = rand() % (GA_POPSIZE / 2);
    string myStr;
    myStr.erase();
    for (int j = 0; j < tsize; j++) {
        spos = rand() % 2;

        if (spos == 0)
        {
            myStr += population[i1].str.substr(j, 1);
        }
        if (spos == 1)
        {
            myStr += population[i2].str.substr(j, 1);
        }
    }
    buffer[i].str = myStr;
    if (rand() < GA_MUTATION) mutate(buffer[i]);
}

break;
```

4- Adding the bull's eye heuristic:

```
void BullsEye_calc_fitness(ga_vector &population)
{
    string target = GA_TARGET;
    int tsize = target.size();
    unsigned int fitness;
    float average = 0;
    float deviation = 0;

    for (int i = 0; i < GA_POPSIZE; i++) {
        fitness = tsize * 10;

        for (int j = 0; j < tsize; j++) {
            if (population[i].str[j] == target[j]) {
                fitness -= 10;
            }

            else {
                for (int k = 0; k < tsize; k++) {
                    if (population[i].str[j] == target[k]) {
                        fitness -= 1;
                        break;
                    }
                }
            }
        }
        population[i].fitness = fitness;
        average += fitness;
    }

    average = average / GA_POPSIZE;          //calculating the average

    for (int i = 0; i < GA_POPSIZE; i++) {
        deviation += pow(population[i].fitness - average, 2);
    }

    deviation = sqrt(deviation / GA_POPSIZE); //calculating the std deviation

    for (int i = 0; i < GA_POPSIZE; i++) {    //updating the average and the deviation for each citizen for the current generation
        population[i].average = average;
        population[i].deviation = deviation;
    }
}
```

out

5- בהסתמכות על התוצאות שאנחנו מקבלים אנו רואים שההירסטטיקה "בול פגיעה" היא יותר יעילה מההירסטטיקה הקודמת (גם מבחינת זמן ריצה "elapsed time" וגם בכמות הפעמים של התכנסות הפתרון ל-GLOBAL OPTIMA, אנו חושבים שההבדל בתוצאות נבע מזה שההירסטטיקה הזאת נותנת "בונוס" ומעדיפה גנים שיש בהם אות הנמצא באותו מקום לאות במילת היעד. עוד דוגמאות בהמשך.

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\GeneticAlgorithms\Debug\GeneticAlgorithms.exe
Standard Deviation: 9.17857
Clock Ticks: 0.293s
Best: Hewlo o`ldM (38)
Average: 66.2427
Standard Deviation: 9.26057
Clock Ticks: 0.277s
Best: HeQlo wAHld! (29)
Average: 59.7378
Standard Deviation: 9.39691
Clock Ticks: 0.328s
Best: HeQlo wCrld! (20)
Average: 52.7632
Standard Deviation: 9.35132
Clock Ticks: 0.241s
Best: HeQlo world! (10)
Average: 45.7178
Standard Deviation: 9.09523
Clock Ticks: 0.212s
Best: HeQlo world! (10)
Average: 39.0293
Standard Deviation: 9.61147
Clock Ticks: 0.195s
Best: Hwllo world! (9)
Average: 32.6143
Standard Deviation: 9.31653
Clock Ticks: 0.187s
Best: Hello world! (0)
Number of Generations: 15
Time Elapsed: 6.72314s
```


6- החלק האחראי על ה – EXPLOITATION באלגוריתם היא פונקציה ה elitism, שבה שמרנו את 10% מהגינים הכי טובים (ה FITNESS שלהם הכי קטן) לדור הבא וזה EXPLOITATION כי אנחנו מעבירים את ה 10% הכי טוב שהגענו אליו עד עכשיו לדור הבא.

```
void elitism(ga_vector &population,
ga_vector &buffer, int esize)
{
    for (int i = 0; i < esize; i++) {
        buffer[i].str = population[i].str;
        buffer[i].fitness = population[i].fitness;
    }
}
```

החלק האחראי על ה – EXPLORATION היא פונקציה ה mate בחלק של Reproduction Operators (one point ,two point, uniform cross over) - וגם פונקציה ה mutate כי כאן אנחנו "מערבבים" גינים ומקבלים גינים חדשים שלא היו לנו בדור הקודם.

```
case 1: // Single Point Operator
    for (int i = esize; i < GA_POPSIZE; i++) {
        i1 = rand() % (GA_POPSIZE / 2);
        i2 = rand() % (GA_POPSIZE / 2);
        spos = rand() % tsize;

        buffer[i].str = population[i1].str.substr(0, spos) +
            population[i2].str.substr(spos, tsize - spos);

        if (rand() < GA_MUTATION) mutate(buffer[i]);
    }

    break;
```

```
void mutate(ga_struct &member)
{
    int tsize = GA_TARGET.size();
    int ipos = rand() % tsize;
    int delta = (rand() % 90) + 32;

    member.str[ipos] = ((member.str[ipos] + delta) % 122);
}
```

7- PSO Algorithm:

```
GeneticAlgorithms (Global Scope) PSO()

void PSO()
{
    init_pso();           // for initlizing the global , local and velocity randomly

    int tsize = GA_TARGET.size();

    Particle global_particle;           // for calculating the fitness of the global particle

    for (int k = 0; k < GA_MAXITER; k++)
    {
        if (global_particle.calc_fitness_particle(globalBest) == 0) break;           //our termination criterion

        for (int i = 0; i < GA_POPSIZE; i++) {

            string myVelocity;
            string myStr;

            myVelocity.erase();
            myStr.erase();

            for (int j = 0; j < tsize; j++) {           // implementation the algorithm
                                                         // like described in the class

                double r1 = (double)rand() / (RAND_MAX);
                double r2 = (double)rand() / (RAND_MAX);
                myVelocity += W * particle_vector[i].get_velocity()[j]
                    + C1 * r1 * (particle_vector[i].get_localBest()[j] - particle_vector[i].get_str()[j])
                    + C2 * r2 * (globalBest[j] - particle_vector[i].get_str()[j]);

                myStr += particle_vector[i].get_str()[j] + myVelocity[j];
            }

            particle_vector[i].set_velocity(myVelocity);           // updating the velocity and the fitness
            particle_vector[i].set_str(myStr);
            particle_vector[i].set_fitness(particle_vector[i].calc_fitness_particle(myStr));

            if (particle_vector[i].get_fitness()           // updating the local and global best
                < particle_vector[i].calc_fitness_particle(particle_vector[i].get_localBest()))
            {
                particle_vector[i].set_localBest(particle_vector[i].get_str());

                if (particle_vector[i].calc_fitness_particle(particle_vector[i].get_localBest())
                    < particle_vector[i].calc_fitness_particle(globalBest))
                {
                    globalBest = particle_vector[i].get_localBest();
                }
            }
        }

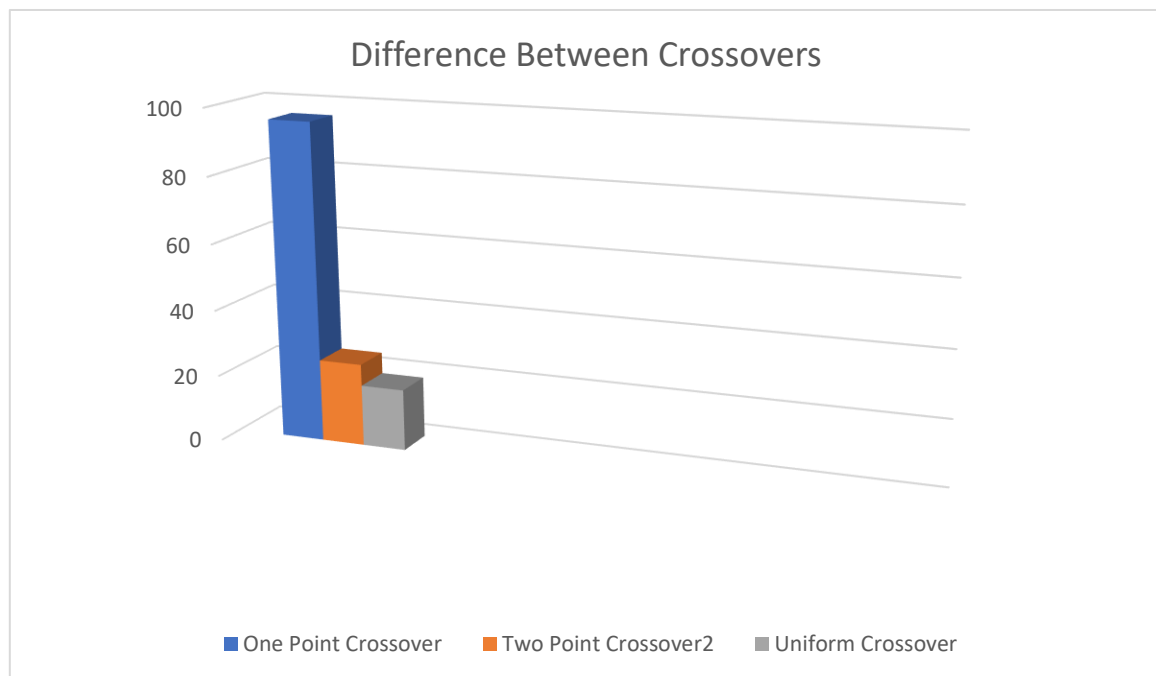
        cout << "Best: " << globalBest << " (" << global_particle.calc_fitness_particle(globalBest) << ")" << endl;
    }
}
```

- **הערה :** בחלק מההרצות שעשינו קיבלנו שהפתרון נקלע ללוקאל אופטימא ויש הרצות שלוקחות יותר זמן יחסית מהרצות אחרות ואני חושב שזה בגלל שהאלגוריתמים הגינתיים הם הסתברותיים.

בטבלה הבאה השתמשתי בשיטת השחלוף "ONEP Crossover" וההיורסטיקה המקורית (הנתונה מתחילת התרגיל).

אלגוריתם	גודל האוכלוסיה	היוריסטקה	ה-Fitness בסוף	זמן
Genetic algorithm	1000	המרחקים	נקלע ללוקאל אופטימא ב-FITNESS 1	עצרנו אותו אחרי 50 שניות
Genetic algorithm	1000	בול פגיעה	התכנס לגלובאל אופטימא - 0	2.47643s
PSO	1000	המרחקים	0	0.5s
PSO	1000	בול פגיעה	0	3.17s
Genetic algorithm	1500	המרחקים	0	2.093s
Genetic algorithm	1500	בול פגיעה	0	1.98s
PSO	1500	המרחקים	0	0.3s
PSO	1500	בול פגיעה	0	2.24s
Genetic algorithm	2048	המרחקים	0	1.59s
Genetic algorithm	2048	בול פגיעה	0	1.34s
PSO	2048	המרחקים	0	0.63311s
PSO	2048	בול פגיעה	0	2.12s

בתרשים למטה מופיע את הקשר וההשפעה של השיטות השונות של השיחלוף לבין מספר הדורות



Part B:

1-

RWS :

```
int* RWS(ga_vector &population, int *points, int* newFitness)
{
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);
    int numOfParents = 2 * (GA_POPSIZE - esize);
    int *parents = new int[numOfParents];           // the selected parents
    int* sumFitness = new int[GA_POPSIZE];         // sum of the fitness till index i
    sumFitness[0] = newFitness[0];
    for (int i = 1; i < GA_POPSIZE; i++) {
        sumFitness[i] = sumFitness[i - 1] + newFitness[i];
    }
    for (int i = 0; i < numOfParents; i++) {        // Roulette Wheel Selection
        int j = 0;
        while (1)
        {
            if (sumFitness[j] >= *(points + i))
                break;
            j++;
        }
        if (j > GA_POPSIZE)
        {
            j = GA_POPSIZE;
            break;
        }
        if (j > GA_POPSIZE)
            j = GA_POPSIZE;
        *(parents + i) = j;
    }
    return parents;
}
```

Scaling:

```
void Scaling(ga_vector &population)
{
    unsigned int a = population[0].fitness, b = population[0].fitness;           // our constants

    for (int i = 0; i < GA_POPSIZE; i++) {
        a = min(population[i].fitness, a);
        b = max(population[i].fitness, b);
    }

    for (int i = 0; i < GA_POPSIZE; i++) {
        population[i].fitness = static_cast<unsigned int>(0.2 * population[i].fitness + 10); // linear transformation
    }
}
```

SUS :

```
int* SUS(ga_vector &population, long totalFitness, int* newFitness)
{
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);
    int numOfParents = 2 * (GA_POPSIZE - esize);
    int *parents = new int[numOfParents];           // the selected parents
    int distance = totalFitness / numOfParents;     // distance between the pointers
    int start = rand() % (distance + 1);           // random number between 0 and distance
    int *points = new int[numOfParents];           // list of (sorted) random numbers from 0 to the total fitness

    for (int i = 0; i < numOfParents; i++) {        // points is a (sorted) list of random numbers
        *(points + i) = start + (i * distance);    // from 0 to total fitness with constant steps.
    }

    return RWS(population, points, newFitness);
}
```

Tournament:

```
int PlayTournament(ga_vector &population, int* players) {  
    int winner = players[0];  
    for (int i = 0; i < K; i++) {  
        if (population[players[i]].fitness < population[winner].fitness)  
            winner = players[i];  
    }  
    return winner;  
}  
  
int* Tournament(ga_vector &population)  
{  
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);  
    int numOfParents = 2 * (GA_POPSIZE - esize);  
    int *parents = new int[numOfParents];           // the selected parents  
    int* players = new int[K];                     // K players in the tournament  
  
    for (int i = 0; i < numOfParents; i++) {  
        for (int j = 0; j < K; j++) {  
            players[j] = rand() % GA_POPSIZE;  
        }  
        parents[i] = PlayTournament(population, players);  
    }  
    return parents;  
}
```

2- Aging:

```
int* Aging(ga_vector &population)
{
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);
    int tsize = GA_TARGET.size();
    int numOfParents = 2 * (GA_POPSIZE - esize);
    int *parents = new int[numOfParents];           // the selected parents

    for (int i = esize; i < GA_POPSIZE; i++) {
        if (population[i].age > MAX_AGE) {
            ga_struct citizen;
            citizen.fitness = 0;
            citizen.age = 0;                        //reset age
            citizen.str.erase();

            for (int j = 0; j < tsize; j++)
                citizen.str += (rand() % 90) + 32;
            population[i] = citizen;
        }
    }

    for (int i = 0; i < numOfParents; i++) {
        while (1)
        {
            parents[i] = rand() % GA_POPSIZE;

            if (population[parents[i]].age > 0)
                break;
        }
    }

    return parents;
}
```


3-8Queens problem:

שינינו את ה `ga_struct` כך שבמקום ה `string`, יש לנו מערך באורך מספר המלכות כך שמהווים פרמוטציה כלשהו כלומר אנחנו מוודאים שלא יהיו מלכות באותה עמודה או באותה שורה ואז רק צריך לטפל במקרים שהם באלכסון.

```
struct ga_struct
{
    int *board = new int[N];           // the chess board size N
    unsigned int fitness;              // its fitness;
    unsigned int age;
};
```

```
void init_population(ga_vector &population,
    ga_vector &buffer)
{
    int tsize = N;
    for (int i = 0; i < GA_POPSIZE; i++) {
        ga_struct citizen;
        citizen.fitness = 0;
        citizen.age = 1;

        for (int j = 0; j < N; j++)
            citizen.board[j] = j;

        random_shuffle(citizen.board, citizen.board + N);

        population.push_back(citizen);
    }

    buffer.resize(GA_POPSIZE);
}
```

4-

Mutation:

we chose to implement the exchange (swap) mutation and the insertion mutation.

```
int *swapIndecies(int * arr, int ipos1, int ipos2)
{
    int temp = *(arr + ipos1);
    *(arr + ipos1) = *(arr + ipos2);
    *(arr + ipos2) = temp;

    return arr;
}

void exchangeMutation(ga_struct &member)
{
    int ipos1 = rand() % N;
    int ipos2 = rand() % N;

    member.board = swapIndecies(member.board, ipos1, ipos2);
}
```

```
void insertionMutation(ga_struct &member)
{
    int ipos1 = rand() % N;
    int temp = ipos1;
    int ipos2 = rand() % N;

    ipos1 = min(ipos1, ipos2), ipos2 = max(temp, ipos2);

    int * myArr = new int[N];

    for (int i = 0; i < ipos1; i++)
    {
        *(myArr + i) = member.board[i];
    }

    temp = *(member.board + ipos1);

    for (int i = ipos1; i < ipos2; i++)
    {
        *(myArr + i) = member.board[i + 1];
    }

    *(myArr + ipos2) = temp;

    for (int i = ipos2 + 1; i < N; i++)
    {
        *(myArr + i) = member.board[i];
    }
}
```

Crossover:

We chose to implement the PMX crossover and the OX crossover.

```
[ ]
void PMX(ga_vector &population, ga_struct &member1, ga_struct &member2, int spos, int i1, int i2)
{
    for (int i = 0; i < N; i++) {                // producing the children as is their parents
        member1.board[i] = population[i1].board[i];
        member2.board[i] = population[i2].board[i];
    }

    int temp1 = member1.board[spos], temp2 = member2.board[spos];

    for (int i = 0; i < N; i++)                  // crossover
    {
        if (member1.board[i] == temp2)
            member1.board[i] = temp1;

        if (member2.board[i] == temp1)
            member2.board[i] = temp2;
    }
    member1.board[spos] = temp2;
    member2.board[spos] = temp1;
}

void OX(ga_vector &population, ga_struct &member1, int i1, int i2)
```

```

void OX(ga_vector &population, ga_struct &member1, int i1, int i2)

{
    int *tempArray = new int[N];
    for (int i = 0; i < N; i++)
    {
        tempArray[i] = i;
    }

    random_shuffle(tempArray, tempArray + N);
    sort(tempArray + N / 2, tempArray + N);

    int pointer = N / 2;
    bool flag;

    for (int i = 0; i < N; i++)
    {
        member1.board[i] = population[i1].board[i];
    }

    for (int i = 0; i < N; i++)
    {
        flag = false;

        for (int j = 0; j < N / 2; j++)
        {
            if (population[i2].board[i] == population[i1].board[tempArray[j]])
            {
                flag = true;
                break;
            }
        }

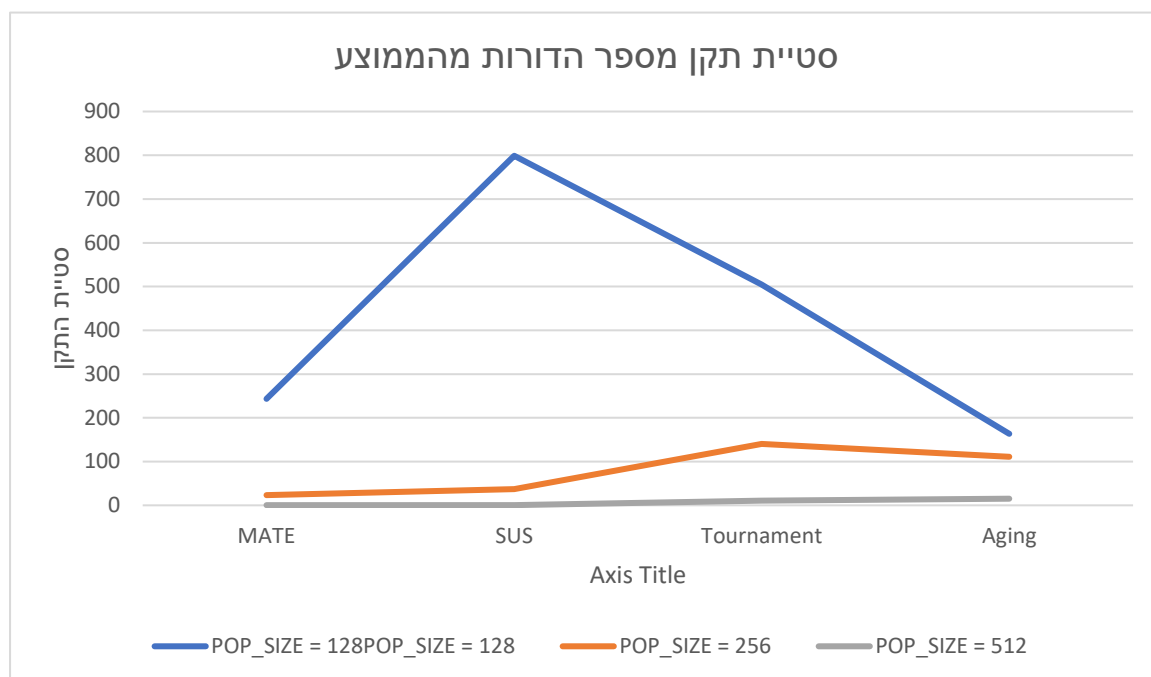
        if (flag == true) continue;

        member1.board[tempArray[pointer]] = population[i2].board[i];
        pointer++;
    }
}

```

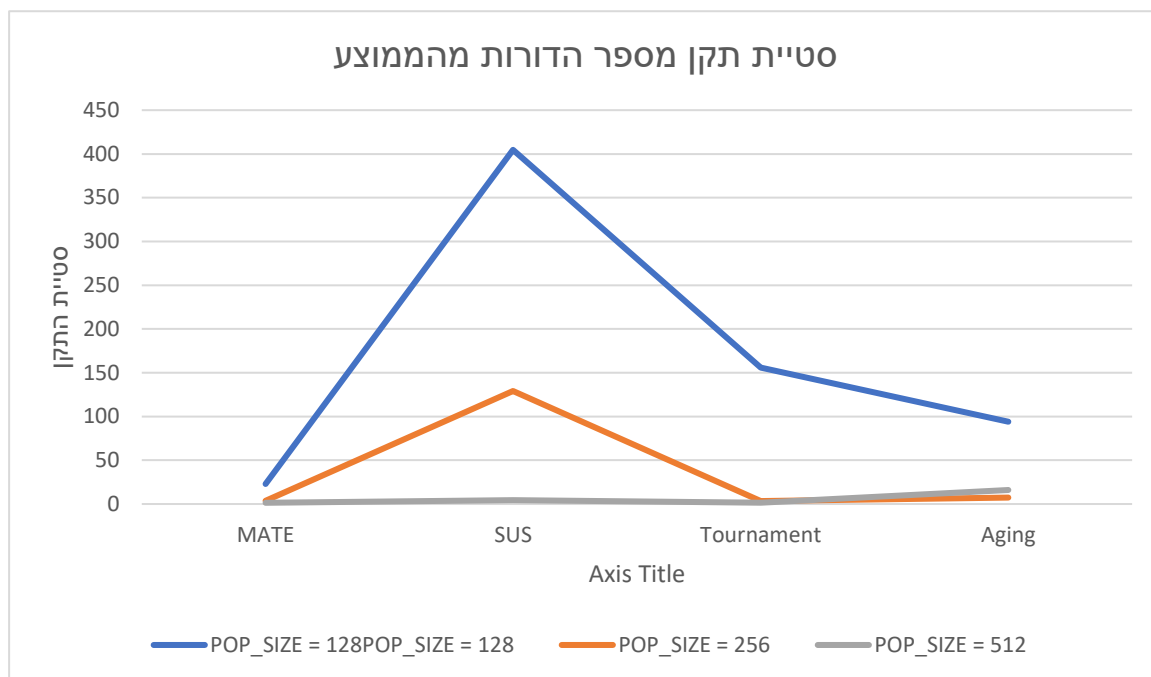
בטבלה הבאה השתמשתי בשיטת השחלוף "Uniform Crossover" וההיורסטיקה המקורית (הנתונה מתחילת התרגיל).

גודל האוכלוסיה	iterations	בול פגיעה מילת היעד: "Hello world!"							
		Mate		SUS		Tournament (k=5)		Aging	
		דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)
128	1	78	1.025	30	0.46	319	4.85	509	8.99
	2	75	0.984	1185	14.22	152	2.38	549	9.52
	3	146	1.77	1052	13.00	46	0.79	375	6.58
	4	79	1.07	1883	22.13	33	0.49	824	14.53
	5	635	7.60	37	0.62	1237	18.26	585	10.21
	ממוצע	202.6	2.48	837.4	10.08	357.4	5.35	568.4	9.96
	סטיית תקן	243.54	2.87	798.85	9.39	504.86	7.41	163.48	2.89
256	1	11	0.33	14	0.42	44	0.88	113	3.09
	2	65	1.33	101	2.05	340	6.68	77	2.31
	3	12	0.41	24	0.63	19	0.45	207	5.25
	4	16	0.44	11	0.35	11	0.32	181	4.67
	5	13	0.39	32	0.82	36	0.79	363	9.01
	ממוצע	23.4	0.58	36.4	0.85	90	1.82	188.2	4.86
	סטיית תקן	23.33	0.42	37.05	0.69	140.36	2.72	110.65	2.60
512	1	11	0.66	11	0.64	8	0.42	46	2.48
	2	11	0.61	12	0.66	32	1.23	67	3.32
	3	12	0.64	13	0.70	9	0.52	32	1.90
	4	11	0.64	12	0.70	22	0.91	33	2.31
	5	11	0.62	11	0.61	8	0.46	32	1.84
	ממוצע	11.2	0.63	11.8	0.66	15.8	0.70	42	2.37
	סטיית תקן	0.44	0.01	0.83	0.03	10.82	0.35	15.18	0.59



בטבלה הבאה השתמשנו בשיטת השחלוף "Partially Matched Crossover" ו-"Exchange (Swap) Mutation"

גודל האוכלוסיה	iterations	N Queens N=15							
		Mate		SUS		Tournament (k=5)		Aging	
		דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)
128	1	18	0.42	81	1.90	14	0.35	17	0.42
	2	24	0.59	937	21.51	46	1.09	51	1.20
	3	68	1.54	30	0.72	341	7.85	241	5.57
	4	23	0.53	12	0.29	14	0.33	30	0.72
	5	9	209	14	329	267	6.18	31	0.70
	ממוצע	28.4	42.41	214.8	70.68	136.4	3.15	74	1.72
	סטיית תקן	22.91	93.12	404.68	144.67	155.76	3.58	94.14	2.16
256	1	17	0.71	9	0.39	10	0.46	35	1.56
	2	16	0.68	127	5.05	15	0.63	34	1.40
	3	10	0.44	12	0.52	14	0.62	27	1.10
	4	18	0.74	12	0.88	9	0.39	45	1.82
	5	19	0.78	306	11.87	7	0.32	43	1.78
	ממוצע	16	0.67	93.2	3.74	11	0.48	36.8	1.53
	סטיית תקן	3.53	0.13	129.13	4.93	3.39	0.13	7.29	0.29
512	1	12	0.95	22	1.71	9	0.75	30	2.34
	2	10	0.80	21	1.69	7	0.60	24	1.91
	3	10	0.79	20	1.55	8	0.65	57	4.20
	4	10	0.81	13	1.03	5	0.44	24	1.80
	5	13	1.00	13	1.04	6	0.50	15	1.17
	ממוצע	11	0.86	17.8	1.40	7	0.58	30	2.284
	סטיית תקן	1.41	0.09	4.43	0.34	1.58	0.12	16.01	1.14



אפשר להסיק שעבור אוכלוסיה (Population) קטנה יחסית, סטיית התקן גבוהה מאוד.

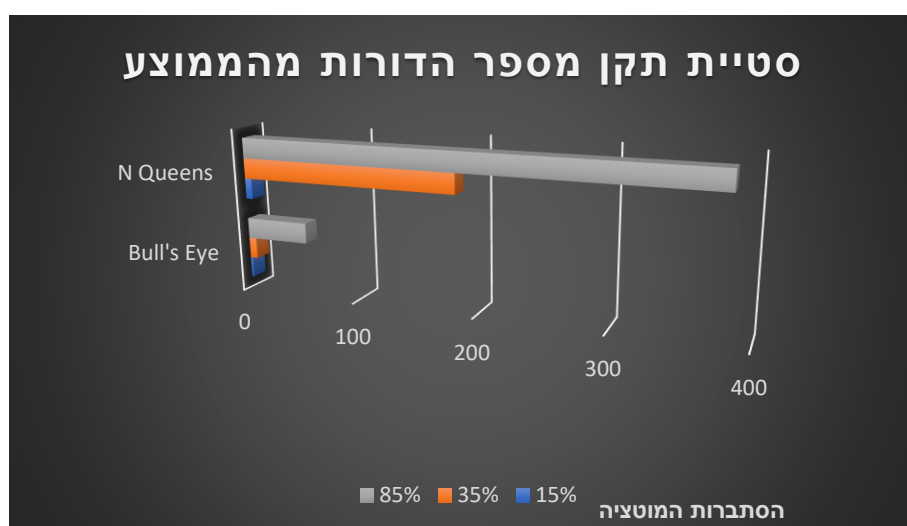
- רגישות הפתרון להסתברות המוטציה: (POP_SIZE=256)

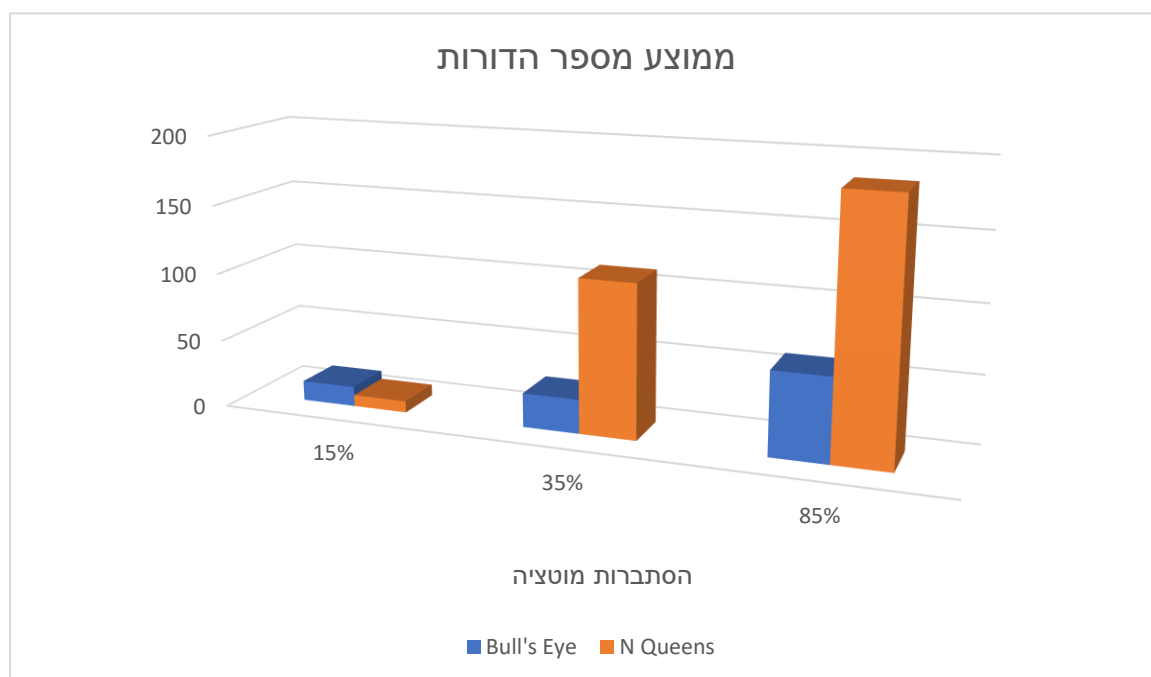
בעיית "בול פגיעה": בטבלה הבאה השתמשתי בשיטת הבחירה הנתונה "MATE", בשיטת השחלוף "Uniform Crossover" וההיורסטיקה המקורית (הנתונה מתחילת התרגיל).

בעיית "N המלכות": השתמשתי בשיטת הבחירה MATE, בשיטת השחלוף "Partially Matched Crossover" ו-"Exchange (Swap) Mutation".

Mutation Rate	iteration	"בול פגיעה"		"N המלכות"	
		דורות	זמן (שניות)	דורות	זמן (שניות)
15%	1	15	0.44	5	0.22
	2	15	0.45	3	0.15
	3	17	0.50	9	0.37
	4	13	0.45	18	0.71
	5	13	0.39	7	0.29
	מוצע	14.6	0.44	8.4	0.34
	סטיית התקן	1.67	0.03	5.81	0.21
	1	24	0.63	22	0.88

35%	2	27	0.68	12	0.49
	3	24	0.63	435	16.13
	4	16	0.46	15	0.60
	5	34	0.78	76	2.86
	ממוצע	25	0.63	112	4.19
	סטיית התקן	6.48	0.11	182.43	6.74
85%	1	94	1.81	871	32.53
	2	13	0.38	10	0.43
	3	54	1.08	17	0.68
	4	11	0.34	23	0.88
	5	135	2.46	8	0.34
	ממוצע	61.4	1.21	185.8	6.97
	סטיית התקן	53.42	0.91	383.08	14.28





רגישות הפתרון לפרופורצית האוכלוסיה האליטיסטית

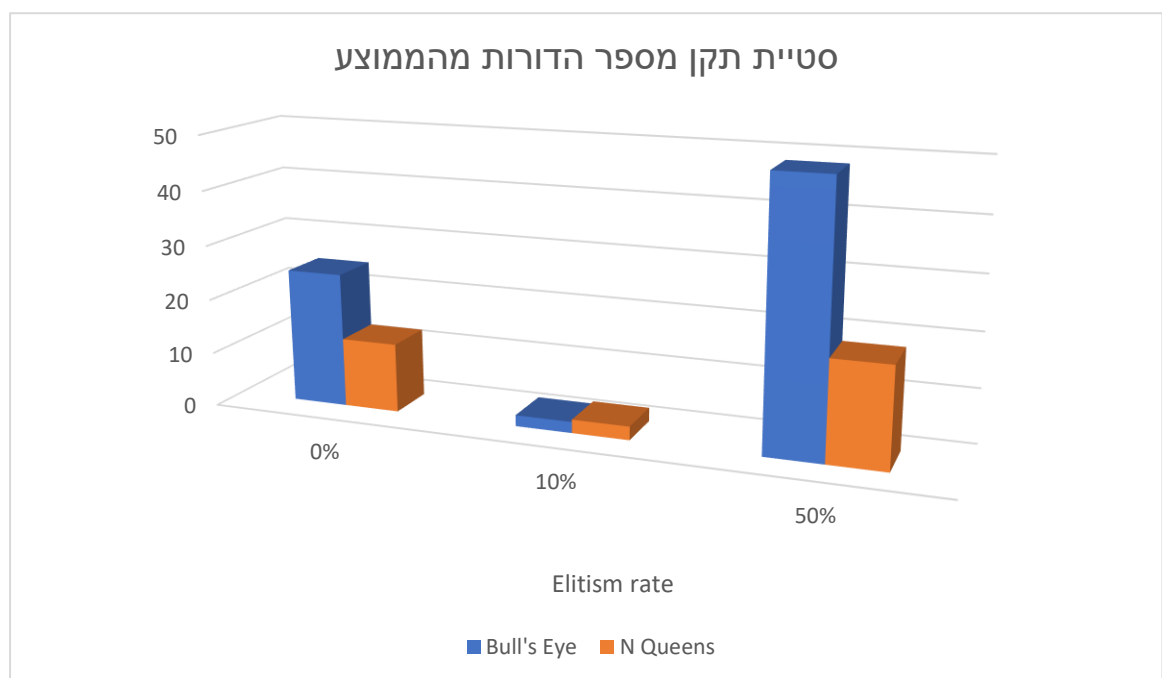
(POP_SIZE=256)

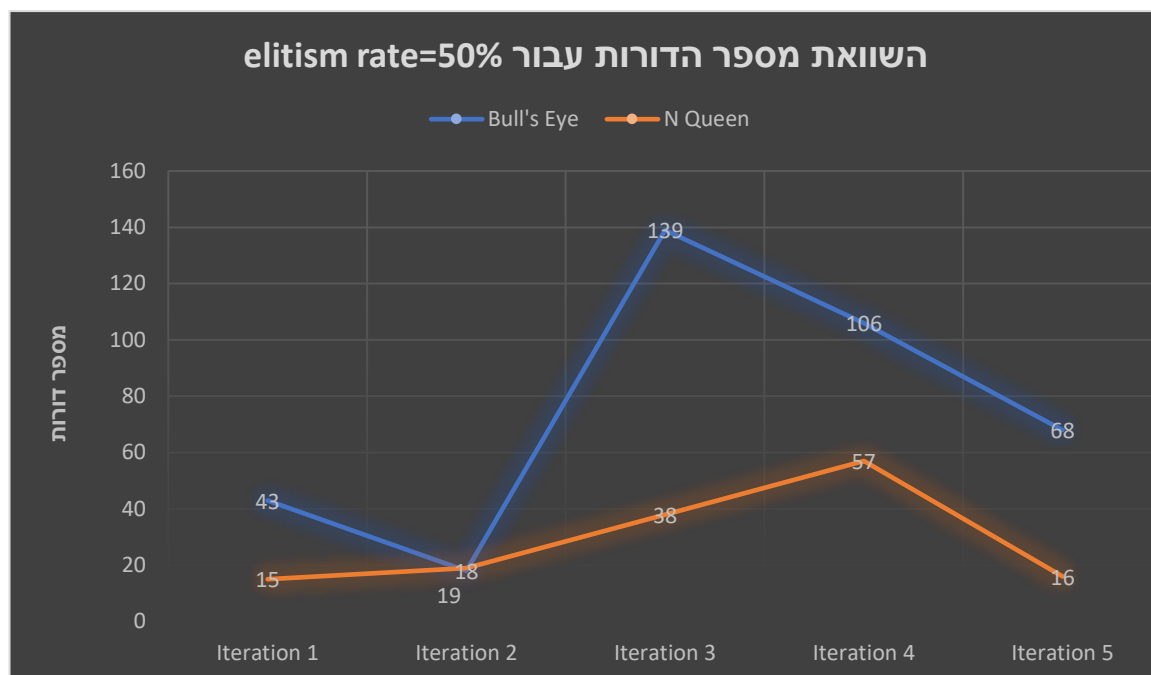
בעיית "בול פגיעה": בטבלה הבאה השתמשתי בשיטת הבחירה הנתונה "MATE", בשיטת השחלוף "Uniform Crossover" וההיורסטיקה המקורית (הנתונה מתחילת התרגיל).

בעיית "N המלכות": השתמשתי בשיטת הבחירה MATE, בשיטת השחלוף "Partially Matched Crossover" ו-"Exchange (Swap) Mutation".

Elitism rate	iteration	"בול פגיעה"		"N המלכות"	
		דורות	זמן (שניות)	דורות	זמן (שניות)
0%	1	34	0.91	22	0.87
	2	20	0.61	17	0.69
	3	19	0.56	48	1.86
	4	30	0.81	36	1.49

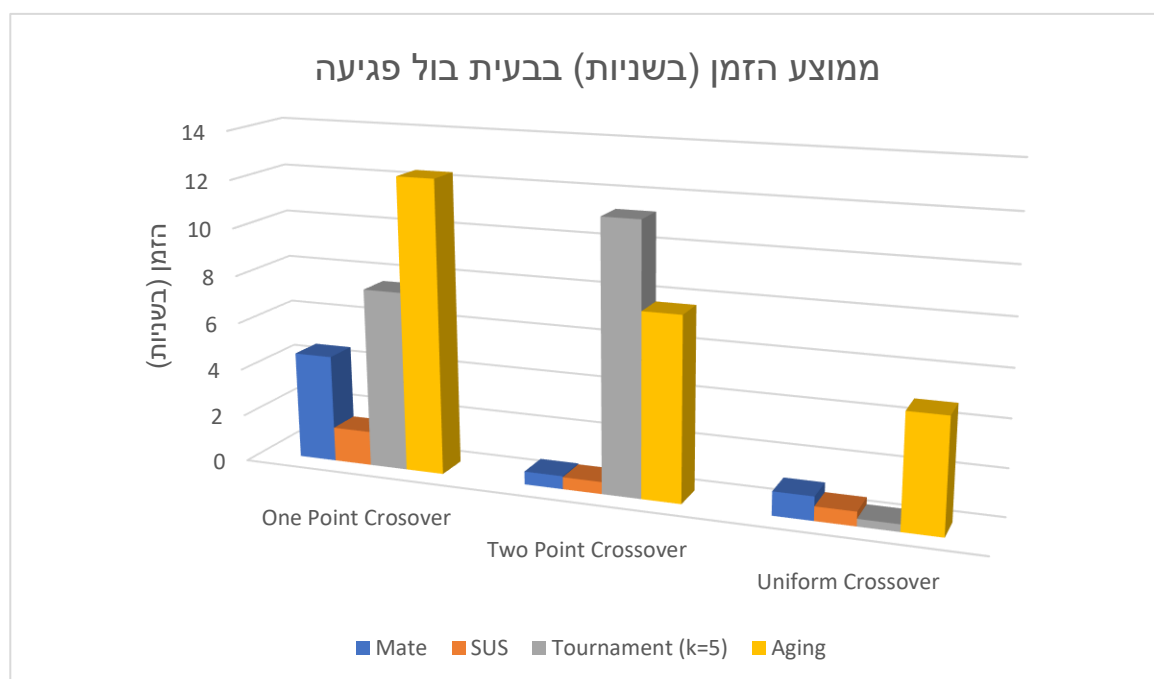
	5	79	1.93	23	0.92
	ממוצע	36.4	0.96	29.2	1.16
	סטיית התקן	24.66	0.55	12.63	0.49
10%	1	18	0.51	9	0.40
	2	14	0.42	13	0.53
	3	14	0.43	15	0.63
	4	17	0.49	15	0.61
	5	18	0.52	14	0.58
	ממוצע	16.2	0.47	13.2	0.55
	סטיית התקן	2.04	0.04	2.48	0.09
50%	1	43	0.89	15	0.61
	2	18	0.46	19	0.74
	3	139	2.40	38	1.44
	4	106	1.92	57	2.13
	5	68	1.30	16	0.63
	ממוצע	74.8	1.39	29	1.10
	סטיית התקן	48.42	0.77	18.23	0.66



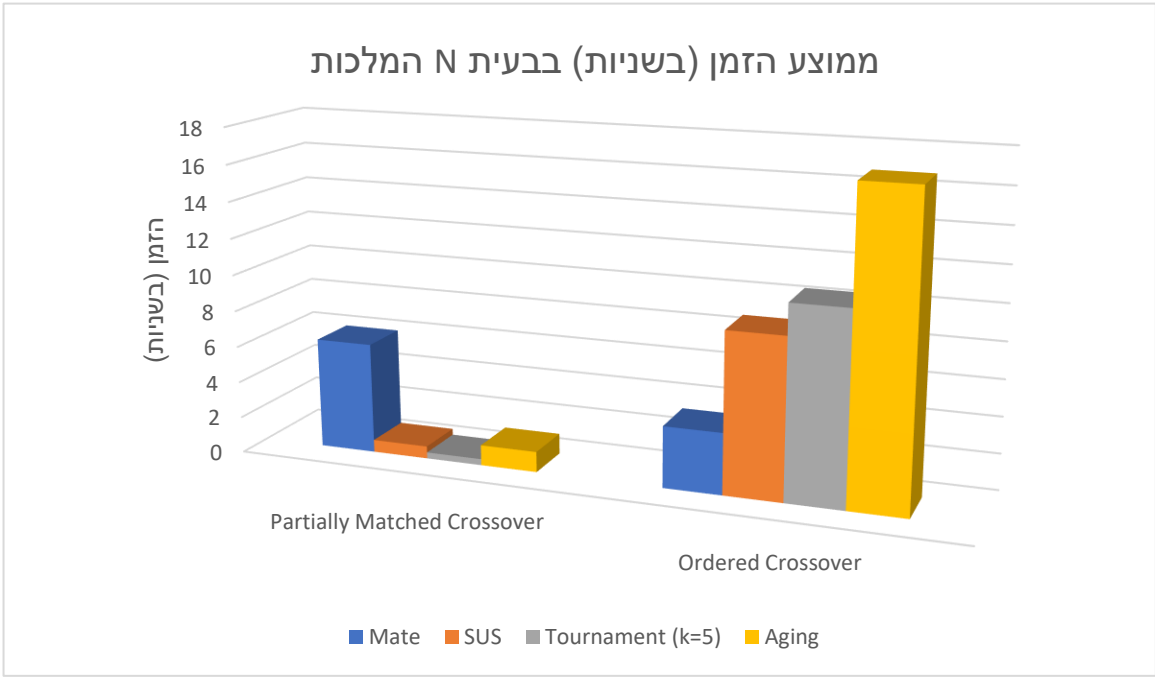


- רגישות הפתרון עבור אסטרטגיית הבחירה + שרידות + אסטרטגית השיחלוק:

שיטת שיחלוק	iterations	בול פגיעה מילת היעד: "Hello world!"							
		Mate		SUS		Tournament (k=5)		Aging	
		דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)
One Point Crossover	1	42	0.74	60	0.79	3915	27.73	155	2.95
	2	3047	20.01	31	0.48	992	8.31	383	5.90
	3	144	1.18	33	0.54	80	0.90	1647	23.28
	4	28	0.41	32	0.47	52	0.60	1790	25.25
	5	22	0.37	664	4.97	23	0.35	272	4.38
	ממוצע	656.6	4.54	164	1.45	1012.4	7.57	849.4	12.35
	סטיית תקן	1337.19	8.65	279.77	1.97	1673.03	11.74	799.06	10.94
Two Point Crossover	1	22	0.41	39	0.67	56	0.76	1783	26.98
	2	18	0.36	19	0.39	36	0.53	92	1.83
	3	49	0.65	33	0.57	4146	37.75	151	2.95
	4	40	0.55	24	0.46	12	0.26	121	2.45
	5	62	0.81	31	0.49	1862	17.21	246	4.33
	ממוצע	38.2	0.55	29.2	0.51	1222.4	11.30	478.6	7.70
	סטיית תקן	18.41	0.18	7.82	0.10	1815.87	16.45	731.47	10.81
Uniform Crossover	1	24	0.50	96	1.57	29	0.54	245	5.14
	2	157	2.25	13	0.35	10	0.27	427	8.88
	3	67	1.05	14	0.37	9	0.25	209	4.32
	4	22	0.47	14	0.38	8	0.24	32	0.88
	5	46	0.76	14	0.37	10	0.28	234	4.73
	ממוצע	63.2	1.00	30.2	0.60	13.2	0.31	229.4	4.79
	סטיית תקן	55.54	0.73	36.78	0.53	8.87	0.12	140.26	2.84



שיטת השיחולוף	iterations	N Queens N=15							
		Mate		SUS		Tournament (k=5)		Aging	
		דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)	דורות	זמן(שניות)
Partially Matched Crossover	1	12	0.47	12	0.48	5	0.22	15	0.55
	2	775	28.79	9	0.36	13	0.48	14	0.51
	3	6	0.28	24	0.85	15	0.56	11	0.41
	4	12	0.48	8	0.31	4	0.17	100	3.44
	5	15	0.60	41	1.41	8	0.31	22	0.77
	ממוצע	164	6.12	18.8	0.68	9	0.34	32.4	1.13
	סטיית תקן	341.57	12.67	13.95	0.45	4.84	0.16	38.00	1.29
Ordered Crossover	1	63	2.35	142	5.25	30	1.18	43	1.60
	2	261	9.33	144	5.40	24	0.91	184	6.60
	3	5	0.22	445	15.94	110	4.13	886	31.42
	4	68	2.53	412	15.11	17	0.65	361	12.93
	5	69	2.52	66	2.37	1231	45.72	870	31.41
	ממוצע	93.2	3.38	241.8	8.81	282.4	10.51	468.8	16.79
	סטיית תקן	97.55	3.46	173.70	6.25	531.61	19.72	390.20	13.93



7- האלגוריתם של minimal conflict:

```
void minimalConflict(ga_struct &game)
{
    int* randQueens = new int[N];
    int queen = (rand() % N);
    int previousFitness = game.fitness;
    int previousQueen = game.board[queen];

    for (int i = 0; i < N; i++) {
        randQueens[i] = i;
    }
    random_shuffle(randQueens, randQueens + N);    //moving the queens

    game.board[queen] = randQueens[0];
    game_calc_fitness(game);

    int currentFitness = game.fitness;
    int chosenQueen = randQueens[0];

    for (int i = 1; i < N; i++)    // making a move
    {
        game.board[queen] = randQueens[i];
        game_calc_fitness(game);

        if (game.fitness < currentFitness)    // good move
        {
            currentFitness = game.fitness;
            chosenQueen = randQueens[i];
        }
    }

    if (currentFitness >= previousFitness)    // bad move
    {
        game.board[queen] = previousQueen;
        game_calc_fitness(game);
    }

    game.board[queen] = chosenQueen;
    game_calc_fitness(game);    // updating the fitness
}

int main()
Output
```

הפונקציה הזו מנסה לתקן את הלוח ע"י הזזת מלכה בעמודה שלה למקום בו פחות איומים משאר המלכות.

```
void game_calc_fitness(ga_struct &game)
{
    unsigned int fitness;
    fitness = 0;
    for (int j = 0; j < N - 1; j++) {
        for (int k = j + 1; k < N; k++)
        {
            if (game.board[j] == game.board[k])
                fitness++;

            if ((k - j) == abs(game.board[j] - game.board[k]))
                fitness += 1;
        }
    }
    game.fitness = fitness;
}
```

מסעיפים קודמים שיטת הבחירה "Tournament" $K=5$, ושיטת השיחלוף "Partially Matched Crossover" ($POPSIZE = 512$) נותנים תוצאה בצורה הכי מהירה ומספר הדורות כדי להגיע לפתרון קטן יחסית. לכן אשתמש בו כדי להשוות אותו עם ה-MINIMAL CONFLICT.

מספר המלכות (גודל הלוח)	iteration	Genetic Algorithm	Minimal Conflicts
		זמן (מילי שניה)	זמן (מילי שניה)
N=10	1	103	2
	2	169	2
	3	67	8
	4	98	10
	5	100	4
	ממוצע	107.4	5.2
	סטיית התקן	37.38	3.63
N=20	1	4285	18
	2	16268	10
	3	1984	13
	4	3422	13
	5	1707	22
	ממוצע	5533.2	15.2
	סטיית התקן	6092.77	4.76
N=30	1	7752	64
	2	176803	72
	3	10390	24
	4	35733	97
	5	9090	114
	ממוצע	47953.6	74.2
	סטיית התקן	72953.89	34.39

אפשר לראות בקלות שהאלגוריתם Minimal Conflict יעיל יותר ומהיר יותר.

אפשר לראות גם כמה זמן שאר השיטות לוקחות עבור $N=15$ (עמוד 28) והאלגוריתם Minimal Conflict פותר אפילו בעיות קשות יותר בזמן הרבה מאוד פחות.

ניתן להכליא בין שני האלגוריתמים, אפשר פשוט לייצר גנים בדור הבא ע"י הפעלת הפונקציה MinimalConflict אחד מהגנים שהיה בדור הקודם (נבחר לפי שיטת בחירה כלשהי כמו SUS).. ומנסה לייצר לוח עם פחות איומים על מלכה שנבחרה באקראי.

אפשר גם לחשוב על האלגוריתם Minimal Conflict כאלגוריתם גינטי בעל POPULATION בגודל 1 ו-Mutation rate = 100% ופונקצית המוטציה היא MinimalConflict .

דוגמאות הרצה לאלגוריתם Minimal Conflict

For N = 100:

```
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\GeneticAlgorithms\Release\GeneticAlgorithms.exe
89 84 30 43 33 95 87 42 63 65 32 72 39 1 83 13 79 9 97 12 24 52 8 55 34 45 17 78 41 38 26 50 85 5 68 77 25 22 37 40 61 80
94 76 81 69 64 44 28 23 57 14 2 58 49 96 10 47 4 67 98 16 6 0 75 21 91 70 20 86 93 73 36 11 88 90 46 19 53 3 92 48 51 99
15 62 27 7 35 54 71 18 74 66 60 29 56 59 82 31 Number of Generations: 455
Time Elapsed: 1.23701s
```

For N = 200:

```
(Global Scope)
// for elapsed time
C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\GeneticAlgorithms\Release\GeneticAlgorithms.exe
50 53 15 188 153 178 13 91 159 9 121 84 4 52 46 117 73 83 79 199 170 145 98 28 82 11 60 179 195 174 2 132 48 1 168 12 69 149
181 154 105 44 162 190 27 189 68 129 35 180 61 119 99 146 186 67 182 87 18 43 124 161 47 109 191 144 114 104 0 155 173 10 45 1
28 123 102 25 33 59 37 41 3 30 76 112 39 158 138 192 187 56 71 81 26 5 183 115 150 80 36 75 198 110 107 127 130 135 6 152 108
54 7 38 23 55 160 126 176 20 89 72 66 93 32 122 22 8 118 85 Number of Generations: 1262
Time Elapsed: 18.3738s
```

8-Knapsack problem:

שינינו את ה `ga_struct` כך שיש לנו מערך באורך מספר הITEMS כך ש1 אומר הITEM נמצא ב SACK ו 0 אחרת, בנוסף הוספנו שדה של WEIGHT של ה SACK שהוא סכום כל משקלי הITEM הנמצאים בתוך ה SACK ואנחנו רוצים שהPROFIT יהיה גדול ככל האפשר בתנאי שמשקל ה SACK לא יהיה יותר גבוה מסף מסוים (נתון)

```
struct ga_struct
{
    int *sack = new int[N];           // the number of items N
    unsigned int fitness;             // its fitness;
    int weight;                       // the weight of the sack
    unsigned int age;
};
```

חישבנו ה FITNESS כך שהוא שווה לה PROFITS של ה ITEM כלומר אם יש לנו 1 בתא J במערך ה SACK אז לוקחים ה PROFIT שלו מהמערך של ה PROFITS (נתון) באותו מקום J, שינינו גם את פונקצית ה SORTFITNESS כך שעכשיו FITNESS גבוה יותר אומר פתרון יותר טוב. (רוצים FITNESS גבוה) גם צריך לוודא שאף פעם לא עוברים את הסף של המשקל לכן יש פונקציה REMOVE_ITEMS אחראית על זה.

```
template <class S>
bool fitness_sort(S x, S y)
{
    return (x.fitness > y.fitness);
}
```

```

void calc_fitness(ga_vector &population)
{
    unsigned int fitness;
    float average = 0;
    float deviation = 0;

    for (int i = 0; i < GA_POPSIZE; i++) {
        fitness = 0;
        for (int j = 0; j < N ; j++) {
            if (population[i].sack[j] == 1)           // higher fitness means better
                fitness += profits[j];
        }

        population[i].fitness = fitness;
        average += fitness;
    }

    average = average / GA_POPSIZE;           //calculating the average
    averages.push_back(average);

    for (int i = 0; i < GA_POPSIZE; i++) {
        deviation += pow(population[i].fitness - average, 2);
    }

    deviation = sqrt(deviation / GA_POPSIZE);       //calculating the std deviation
    deviations.push_back(deviation);
}

```

Output

```

}

void remove_items(ga_struct &citizen) {
    int index = 0;
    int* randomStuff = new int[N];

    for (int i = 0; i < N; i++) {           // to choose a random stuff
        randomStuff[i] = i;
    }

    random_shuffle(randomStuff, randomStuff + N);
    while (1) {
        if (citizen.weight <= MAX_WEIGHT)
            break;

        while (index < N) {
            if (citizen.sack[randomStuff[index]] == 1)
            {
                citizen.weight = (citizen.weight - weights[randomStuff[index]]);
                citizen.sack[randomStuff[index]] = 0;
                index++;
                break;
            }
            index++;
        }
    }
}

```

tpout

צריך להוסיף שבאתחול של ה SACK , לכל איבר N מתוך מספר ה ITEMS מגרילים מספר רנדומלי שיהיה 0 או 1 כלומר $RAND(2\%)$, ואז בודקים אם עברנו את הסף , אם כן קוראים לפונקציה שהזכרתי למעלה.

```
void init_population(ga_vector &population,
                    ga_vector &buffer)
{
    for (int i = 0; i < GA_POPSIZE; i++) {
        ga_struct citizen;
        citizen.fitness = 0;
        citizen.weight = 0;
        citizen.age = 1;

        for (int j = 0; j < N; j++)
        {
            citizen.sack[j] = rand() % 2;    //0-1 problem, either it contains the item or not

            if (citizen.sack[j] == 1)
                citizen.weight += weights[j];
        }

        if (citizen.weight > MAX_WEIGHT)
        {
            remove_items(citizen);
        }

        population.push_back(citizen);
    }

    buffer.resize(GA_POPSIZE);
}
```

- 8 הבעיות נמצאות בקובץ ה CPP בהערות עם הפתרון שיצא לנו, הנה הרצה לבעיה ה – 8

דוגמא להרצת 8 PROBLEM

```
Select C:\ArtificialIntelligenceLab\ArtificialIntelligenceLab\GeneticAlgorithms\Release\GeneticAlgorithms.exe
Average: 1.33444e+07
Standard Deviation: 488583
Clock Ticks: 0.02s
Best: 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 fitness: (13549094) Weight: (6402560)
Average: 1.33191e+07
Standard Deviation: 509498
Clock Ticks: 0.012s
Best: 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 fitness: (13549094) Weight: (6402560)
Average: 1.33293e+07
Standard Deviation: 515643
Clock Ticks: 0.01s
Best: 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 fitness: (13549094) Weight: (6402560)
Average: 1.33326e+07
Standard Deviation: 486558
Clock Ticks: 0.01s
Best: 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 fitness: (13549094) Weight: (6402560)
Average: 1.33409e+07
Standard Deviation: 475332
Clock Ticks: 0.013s
Best: 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 fitness: (13549094) Weight: (6402560)
Average: 1.33175e+07
Standard Deviation: 519129
Clock Ticks: 0.016s
Best: 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 fitness: (13549094) Weight: (6402560)
Average: 1.33477e+07
Standard Deviation: 484703
Clock Ticks: 0.009s
Number of Generations: 1024
Time Elapsed: 29.0364s
```

```
35 #define MAX_WEIGHT 165 //weight
36 #define N 10 //number of items
37
38 int weights[] = { 23,31,29,44,53,38,63,85,89,82 }; //array that stores the weight of each item
39 int profits[] = { 92,57,49,68,60,43,67,84,87,72 }; //array that stores the profit of each item
40
41 //Best : {1,1,1,1,0,1,0,0,0,0}
42
43 /*2-
44 Number of Items: 5
45 Weight: 26*/
46 #define MAX_WEIGHT 26 //weight
47 #define N 5 //number of items
48 int weights[] = {12,7,11,8,9}; //array that stores the weight of each item
49 int profits[] = {24,13,23,15,16}; //array that stores the profit of each item
50
51 //Best :{0,1,1,1,0}
52
53 /*3-
54 Number of Items: 6
55 Weight: 190*/
56 #define MAX_WEIGHT 190 //weight
57 #define N 6 //number of items
58 int weights[] = {56,59,80,64,75,17}; //array that stores the weight of each item
59 int profits[] = {50,50,64,46,50,5}; //array that stores the profit of each item
60
61 //Best :{1,1,0,0,1,0}
62
63 /*4-
64 Number of Items: 7
65 Weight: 50*/
66 #define MAX_WEIGHT 50 //weight
67 #define N 7 //number of items
68 int weights[] = {31,10,20,19,4,3,6}; //array that stores the weight of each item
69 int profits[] = {70,20,39,37,7,5,10}; //array that stores the profit of each item
70
71 //Best :{1,0,0,1,0,0,0}
72
73 /*5-
74 Number of Items: 8
75 Weight: 104*/
76 #define MAX_WEIGHT 104 //weight
77 #define N 8 //number of items
78 int weights[] = {25,35,45,5,25,3,2,2}; //array that stores the weight of each item
79 int profits[] = {350,400,450,20,70,8,5,5}; //array that stores the profit of each item
80
81
82
83
84
```

(POPSIZE=1024, שיטת בחירה MATE, Uniform Crossover, מספר איטירציות 1024)

השוואה ובדיקת אם הגענו לפתרון האופטמלי:

מספר בעיה	נסיון	פתרון אופטמלי?
1	1	YES
	2	YES
	3	YES
2	1	YES
	2	YES
	3	YES
3	1	YES
	2	YES
	3	YES
4	1	YES
	2	YES
	3	YES
5	1	YES
	2	YES
	3	YES
6	1	YES
	2	YES
	3	YES
7	1	YES
	2	YES
	3	YES
8	1	YES
	2	YES
	3	YES