<u>דו"ח מעבדה 2 – אלגוריתמים גנטיים:</u>

<u>מגיש:</u> איתי גיא, ת.ז. - 305104184

<u>חלק א' – חיפוש לוקאלי:</u>

1. חישוב ממוצע ה-$fitness$ של האוכלוסייה בכל דור וסטיית התקן ממנו:

```cpp
static void calc_ga_stat(ga_vector &population)
{
    unsigned int sumFitness = 0;
    for (int i = 0; i < GA_POPSIZE; i++) // sum all
    {
        sumFitness += population[i].fitness;
    }
    double mean = sumFitness / GA_POPSIZE; // uniform mean
    double std = 0;
    for (int i = 0; i < GA_POPSIZE; i++)
    {
        double fitness = population[i].fitness;
        std += pow(fitness - mean, 2);
    }
    std = sqrt(std); // simple std as we recognize with
    for (int i = 0; i < GA_POPSIZE; i++)
    {
        population[i].mean = mean;
        population[i].std = std;
    }
}
```

דיווח ממוצע ה-$fitness$ של האוכלוסייה בכל דור וסטיית התקן ממנו:

```
Best: Hello Wprld! (1) -> [mean=11,std=1035.21] -> CPU Clock Ticks=592,Elapsed Run Time=0 -> Generation=34
Best: Hello World! (0) -> [mean=10,std=984.898] -> CPU Clock Ticks=607,Elapsed Run Time=1 -> Generation=35
End-up after: 623.00 ticks and 1.00 seconds
```

2. חישוב זמן ריצה עד להתכנסות לפי $clock\ ticks\ and\ elapsed\ time$ בכל דור:

```cpp
static inline void print_best(ga_vector &gav,int generation)
{
    string temp;
    for (int i = 0; i < GA_TARGET.size(); i++)
    {
        temp += gav[0].str[i];
    }
    cout << " Best: " << temp << " (" << gav[0].fitness << ")" << " -> [mean=" << gav[0].mean << ",std=" << gav[0].std << "]"
        << " -> " << "CPU Clock Ticks=" << gav[0].cpuTicks << ",Elapsed Run Time=" << gav[0].elapsedTime << " -> " << "Generation=" << generation << endl;
}
static inline void swap(ga_vector *&population, ga_vector *&buffer)
{
    ga_vector *temp = population;
    population = buffer;
    buffer = temp;
}
```

דיווח של זמן הריצה עד להתכנסות לפי *clock ticks and elasped time* בכל דור:

```
Best: Hello Wprld! (1) -> [mean=14,std=1014.18] -> CPU Clock Ticks=532,Elapsed Run Time=0 -> Generation=31
Best: Hello World! (0) -> [mean=12,std=956.333] -> CPU Clock Ticks=548,Elapsed Run Time=0 -> Generation=32
End-up after: 563.00 ticks and 0.00 seconds
```

3. היוריסטיקת "בול פגיעה" כחלק מהפונקציה *calc_fitness*:

```cpp
        else if (fitnessEstimation.compare("BACS") == 0)    // Bull And Cows
        {
            for (int i = 0; i < GA_POPSIZE; i++)
            {
                int fitness = 0;
                for (int j = 0; j < tsize; j++)
                {
                    if (population[i].str[j] != target[j])
                    {
                        fitness++; //caught different places to count the same as compliment
                    }
                }
                int localFitness = 0;
                for (int j = 0; j < tsize; j++)
                {
                    bool SEEN = false;
                    for (int k = 0; k < tsize; k++)
                    {
                        if (population[i].str[k] != target[k] && (j != k) && population[i].str[j] != target[k] && !SEEN)
                        {
                            localFitness++; // pay more for the difference places!
                            SEEN = true;
                        }
                    }
                }
                // update and assign values:
                population[i].caughtPlaces = tsize - fitness;
                population[i].fitness = fitness + localFitness;
                population[i].cpuTicks = ((double)(clock() - startClock));
                population[i].elapsedTime = time(&population[i].elapsedTime) - startTime;
            }
        }
        Population::calc_ga_stat(population);
    }
```

<u>השוואה בין היוריסטיקת "בול פגיעה" להיוריסטיקה המקורית:</u>

מקורית	**בול פגיעה**
פונקציית מרחק מנהטן משקלים מפולגים אחיד במחרוזת	פונקציה שמערבת בתוכה חוכמה בנוגע לתוכן הנתונים. מכיוון שאלו מחרוזת והיא רוצה למצוא התאמה היא ממשקלת טעויות והצלחות בהתאם **ולא** משקלים שמפולגים באופן אחיד כמו בפונקציה המקורית.
לינארית באורך הקלט	ריבועית באורך הקלט
דורשת הרבה דורות	דורשת מעט דורות

שימוש בהיוריסטיקת "בול פגיעה":

```
Best: Hello World! (0) -> [mean=15,std=52.6688] -> CPU Clock Ticks=403,Elapsed Run Time=0 -> Generation=15
End-up after: 424.00 ticks and 0.00 seconds
```

שימוש בהיוריסטיקה המקורית:

```
Best: Hello World! (0) -> [mean=11,std=1015.48] -> CPU Clock Ticks=639,Elapsed Run Time=0 -> Generation=36
End-up after: 654.00 ticks and 0.00 seconds
```

- ההיוריסטיקה של "בול פגיעה" קשורה למוטציות במהלך האלגוריתם באופן שבו כאשר יתבצע שינוי מוטנטי חלק מאותם גנים מאוד יתקרבו ויקטינו את "קירבתם" לפתרון משמעותית (כי לאחר שינוי זה "בול פגיעה" תחושב שוב) ולכן זה יצור התכנסות מהירה יותר לפתרון.
- ההיוריסטיקה של "בול פגיעה" אכן משפרת את ההיוריסטיקה המקורית כפי שנאמר בטבלה הנ"ל מסיבת הבינה שמוכנסת אליה בהתאם לנתונים המעורבים.

4. תמיכה ב-SUS:

```cpp
static int SUS(ga_vector &population, ga_vector &buffer)
{
    ga_vector leftOver(GA_POPSIZE);
    double m = 0;
    int sum = population[0].fitness;
    for (int i = 0; i < population.size(); i++)
    {
        m += population[i].fitness;
    }
    m /= population.size(); // finding avg of population
    double r = (double)rand() / (RAND_MAX + 1);// pick random number between 0 to 1
    double delta = r * m;
    int j = 0, i = 0, k = 0;
    do {
        if (delta < sum) // for some delta we divide the population by sum - that is the point of this method [part of roullete]
        {
            buffer[i] = population[j];
            buffer[i].str = population[j].str;
            i++;
            delta += sum;
        }
        else
        {
            // keep into leftOver all individuals about sum less of equals than delta
            sum += population[j].fitness;
            leftOver[k] = population[j];
            leftOver[k].str = population[j].str;
            k++;
            j++;
        }
    } while (j < population.size());
    int startRest = i;
    for (k = i, j = 0; k < GA_POPSIZE - i; k++, j++) // coping leftOvers to the buffer's back
    {
        buffer[k] = leftOver[j];
        buffer[k].str = leftOver[j].str;
    }
    return startRest;
```

תמיכה בשיטת *Aging* נוספת:

Tournament:

```cpp
static int Tournament(ga_vector &population, ga_vector &buffer)
{
    int K = 2;  //rand() % GA_POPSIZE;
    vector<int> rest;
    ga_struct best;
    bool isBestNull = true;
    for (int i = 0; i < K; i++) // taking K best random indecies
    {
        int ind = rand() % population.size();
        if (isBestNull || (best.fitness > population[ind].fitness))
        {
            best = population[ind];
            isBestNull = false;
            rest.push_back(ind);
        }
    }
    // copy relevant object to buffer
    int j = 0, k = 0;
    for (int i = 0; i < GA_POPSIZE; i++)
    {
        if ((k < rest.size()) && (i == rest[k]))
        {
            buffer[j] = population[i];
            j++;
            k++;
        }
    }
    int restStart = j; // esize index for next mutation the rest population before next round
    k = 0;
    // copy the rest to buffer - the order between two groups is very important
    for (int i = 0; (k < rest.size()) && (j < GA_POPSIZE) && (i < GA_POPSIZE); i++)
    {
        if (i != rest[k])
        {
            buffer[j] = population[i];
            j++;
            k++;
        }
    }
    return restStart;
}
```

5. אסטרטגיית בחירה – "טורניר":

```cpp
static int Turnir(ga_vector &population, ga_vector &buffer)
{
    int K = 2;//rand() % GA_POPSIZE;
    vector<int> rest;
    int j = 0;
    for (int i = 0; i < K; i++)
    {
        int ipos = rand() % GA_POPSIZE;
        int jpos = rand() % GA_POPSIZE;
        bool SEEN = false;
        for (int h = 0; h < i; h++)
        {
            if (rest[h] == ipos || rest[h] == jpos)
            {
                // pick k random indivuduals and check there is no duplications
                h = i;
                SEEN = true;
            }
        }
        if (SEEN == true)
        {
            continue;
        }
        if (population[ipos].fitness < population[jpos].fitness)
        {
            // if one individual is bigger than the other he will win and pass to the next generation
            buffer[j] = population[ipos];
            rest.push_back(jpos);
        }
        else
        {
            buffer[j] = population[jpos];
            rest.push_back(ipos);
        }
        j++;
    }
    sort(rest.begin(), rest.end());
    int restStart = j;
    int k = 0;
    for (int i = 0, k = 0; i < GA_POPSIZE; i++) // put in the buffer's back
    {
        if (i == rest[k])
        {
            buffer[j] = population[i]; // rest[j]
            j++;
            k++;
        }
    }
    return restStart;
}
```

6. אסטרטגיית שחלוף נוספת - "$twoPoints$":

```
static void twoPoints(ga_struct &populationi1, ga_struct &populationi2, ga_struct &bufferi, int spos1, int spos2)
{
    bufferi.str = populationi1.str.substr(0, spos1) +
        populationi2.str.substr(spos1, spos2 - spos1) +
        populationi1.str.substr(spos2, GA_TARGET.size() - spos2);
}
```

אסטרטגיית שחלוף נוספת:

```
static void uniform(ga_struct &populationi1, ga_struct &populationi2, ga_struct &bufferi)
{
    for(int i = 0;i < GA_TARGET.size();i++){
        int gen_to_select = rand() % 2;
        if(gen_to_select == 0){
            bufferi.str[i] = populationi1.str[i];
        } else {
            bufferi.str[i] = populationi2.str[i];
        }
    }
}
```

אסטרטגיית מוטציה נוספת:

```
static void swap(ga_struct &member)
{
    // swapping between two indecies - very simple method:
    int tsize = GA_TARGET.size();
    int ipos = rand() % tsize;
    int jpos = rand() % tsize;

    char temp = member.str[ipos];
    member.str[ipos] = member.str[jpos];
    member.str[jpos] = temp;
}
```

7. <u>בדיקת רגישות:</u>

- נסמן – (1) = בעיית המחרוזת עם ההיוריסטיקה המקורית ו-(2) = בעיית המחרוזת עם היוריסטיקת "בול פגיעה".

א.

גודל אוכלוסייה	**(1)**	**(2)**
2048	29 דורות,0 שניות,638 טיקים	14 דורות,1 שניות,449 טיקים
5500	28 דורות,2 שניות,1679 טיקים	14 דורות,1 שניות,1191 טיקים
10000	25 דורות,3 שניות,2763 טיקים	12 דורות,2 שניות,1859 טיקים
20000	22 דורות,5 שניות,5056 טיקים	11 דורות,4 שניות,3577 טיקים

- ניתן להבחין שגודל האוכלוסייה משפיע באופן דרמתי על כמות הטיקים של שעון המחשב ועל זמן החישוב. יתרה מזאת ניתן לראות באופן מובהק שמספר הדורות באופן ממוצע נשאר זהה – רגישות לגודל האוכלוסייה בפרמטר הזמן.

ב.

הסתברות למוטציות	(1)	(2)
10%	43 דורות,1 שניות,968 טיקים	14 דורות,1 שניות,421 טיקים
25%	36 דורות,1 שניות,807 טיקים	14 דורות,1 שניות,422 טיקים
55%	38 דורות,1 שניות,854 טיקים	16 דורות,1 שניות,503 טיקים
90%	98 דורות,2 שניות,2309 טיקים	17 דורות,0 שניות,533 טיקים

- ניתן להבחין בשינוי מספר הדורות של אלגוריתם (1) בפונקציה של הסתברות למוטציות לעומת אלגוריתם (2) שהוא יותר יציב. בפרמטרי הזמן בממוצע אין שינוי – רגישות להסתברות למוטציות בפרמטר של מספר דורות עבור אלגוריתם (1).

ג.

פרופורציית האוכלוסייה האליטיסטית	(1)	(2)
10%	28 דורות,1 שניות,611 טיקים	14 דורות,1 שניות,445 טיקים
30%	34 דורות,1 שניות,725 טיקים	16 דורות,1 שניות,491 טיקים
60%	46 דורות,1 שניות,973 טיקים	22 דורות,0 שניות,667 טיקים
90%	278 דורות,6 שניות,5699 טיקים	56 דורות,1 שניות,1644טיקים

- ניתן להבחין שהפרמטרים ששני האלגוריתמים גדלים כפונקציה של פרופורציית האוכלוסייה האליטיסטית פרט לפרמטר הזמן שהוא באופן ממוצע יציב אצל אלגוריתם (2) – רגישות לפרופורציית האוכלוסייה האליטיסטית פרט לזמן באלגוריתם (2).

ד.

אסטרטגיית הבחירה	(1)	(2)
elitism	35 דורות,1 שניות,765 טיקים	15 דורות,1 שניות,441 טיקים
Turnir	37 דורות,0 שניות,860 טיקים	14 דורות,1 שניות,438 טיקים
SUS	67 דורות,2 שניות,1570 טיקים	13 דורות,0 שניות,414 טיקים
Tournament	49 דורות,1 שניות,1085 טיקים	15 דורות,1 שניות,486 טיקים

- ניתן להבחין שאלגוריתם 1 יותר מושפע משינוי שיטת הבחירה מאשר אלגוריתם (2) שיציב עם שונות נמוכה – רגישות לאסטרטגיית הבחירה באלגוריתם (1).

ה.

אסטרטגיית השחלוף	(1)	(2)
onePoint	53 דורות,1 שניות,906 טיקים	15 דורות,0 שניות,402 טיקים
twoPoints	26 דורות,1 שניות,568 טיקים	15 דורות,0 שניות,457 טיקים
uniform	96 דורות,2 שניות,1562 טיקים	14 דורות,0 שניות,379 טיקים

- כמו באסטרטגיית הבחירה הנ"ל גם כאן ניתן להבחין ברגישות אלגוריתם (1) לשינוי שיטת השחלוף בניגוד ליציבות של אלגוריתם (2).
- איכות הפתרון לפי כל פרמטר הוא מצוין, **כלומר ברוב המוחלט** של המקרים האלגוריתם התכנס.

חלק ב' – חיפוש עם אילוצים:

8. ייצוג מתאים לגן באורך N:

```
struct stateBoard {
    vector<int> _board;
    unsigned int _fitness;
};
```

- וקטור באורך N כאשר העמודה ה-i מכיל את השורה שבה המלכה ממוקמת.

אופרטורים נוספים למוטציה:

:simple reverse

```
static void simpleReverse(state &member)
{
    // simple reverse random chunck
    int tsize = GA_NQUE_SIZE;
    int ipos = rand() % tsize;
    int jpos = rand() % tsize;

    int low = 0, high = 0;
    if (ipos < jpos)
    {
        low = ipos;
        high = jpos;
    }
    else
    {
        low = jpos;
        high = ipos;
    }
    vector<int> s(GA_NQUE_SIZE);
    for (int i = 0; i < low; i++)
    {
        s.push_back(member._board[i]);
    }
    vector<int> toRev;
    for (int i = low; i <= high; i++)
    {
        toRev.push_back(member._board[i]);
    }
    reverse(toRev.begin(), toRev.end());
    for (int i = 0; i < toRev.size(); i++)
    {
        s.push_back(toRev[i]);
    }
    for (int i = high + 1; i < GA_NQUE_SIZE; i++)
    {
        s.push_back(member._board[i]);
    }
     for(int i = 0;i < GA_NQUE_SIZE;i++){
         member._board[i] = s[i];
     }
}
```

:insertion

```
static void insertion(state &member)
{
    // insert new random letter to random place
    int tsize = GA_NQUE_SIZE;
    int ipos = rand() % tsize;
    int nextPlace = rand() % tsize;

    vector<int> temp(GA_NQUE_SIZE);
    for(int i = 0;i < GA_NQUE_SIZE;i++){
        if(i != ipos){
            temp.push_back(member._board[i]);
        } else if(i == nextPlace){
            temp.push_back(member._board[i]);
            temp.push_back(member._board[ipos]);
        }
    }
    for(int i = 0;i < GA_NQUE_SIZE;i++){
        member._board[i] = temp[i];
    }
}
```

<u>אופרטורים נוספים לשחלוף:</u>

:*PMX operator*

```cpp
static void PMX(state &populationi1, state &populationi2, state &bufferi)
{
    //initial buffer for result usage
    for(int i = 0;i < GA_NQUE_SIZE;i++){
        bufferi._board[i] = -1;
    }
    //picking randomly block chunck
    int ipos = rand() % GA_NQUE_SIZE;
    int jpos = rand() % GA_NQUE_SIZE;
    int low = 0,high = 0;
    if(ipos <= jpos){
        low = ipos;
        high = jpos;
    } else {
        low = jpos;
        high = ipos;
    }
    //directly trasform the p1 segment to the children keeping the alels order
    for(int i = low;i <= high;i++){
        bufferi._board[i] = populationi1._board[i];
    }
    vector<int> p2_src; // values from p1
    vector<int> p2_dest; // values from p2
    for(int i = low;i <= high;i++){
        bool IS_THERE = false;
        for(int j = low;j <= high;j++){
            if(populationi2._board[i] == populationi1._board[j]){
                IS_THERE = true;
                break;
            }
        }
        //building the mapping between values in p2 to values same location in p1 where these values are no in child already
        if(!IS_THERE){
            p2_src.push_back(populationi2._board[i]);
            p2_dest.push_back(populationi1._board[i]);
        }
    }
    for(int i = 0;i < p2_dest.size();i++){
        for(int j = 0;j < GA_NQUE_SIZE;j++){
            // find mapped value in the p2 array
            if(populationi2._board[j] == p2_dest[i]){
                //check if this place is empty
                if(bufferi._board[j] == -1){
                    bufferi._board[j] = p2_src[i];
                }
                //else {
                    // because of duplication is allowd better is to live this case because of Inf loop and bad performance we could achieve.
                    // this is working well without it!
                //}
            }
        }
    }
    for(int i = 0;i < GA_NQUE_SIZE;i++){
        if(bufferi._board[i] == -1){
            bufferi._board[i] = populationi2._board[i];
        }
    }
}
```

:*OX operator*

```
static void OX(state &populationi1, state &populationi2, state &bufferi)
{
    //initial buffer for result usage
    for(int i = 0;i < GA_NQUE_SIZE;i++){
        bufferi._board[i] = -1;
    }
    // pick chunck of alels from populationi1
    int ipos = rand() % GA_NQUE_SIZE;
    int jpos = rand() % GA_NQUE_SIZE;
    int low = 0,high = 0;
    if(ipos <= jpos){
        low = ipos;
        high = jpos;
    } else {
        low = jpos;
        high = ipos;
    }
    //directly trasform the p1 segment to the children keeping the alels order
    for(int i = low;i <= high;i++){
        bufferi._board[i] = populationi1._board[i];
    }
    //picking each value in index i to be in buffer index i -> this operation is handling duplications otherwise there is no any coverage.
    for(int i = 0;i < low;i++){
        bufferi._board[i] = populationi2._board[i];
    }
    for(int i = (high + 1);i < GA_NQUE_SIZE;i++){
        bufferi._board[i] = populationi2._board[i];
    }
}
```

<u>הרצת האלגוריתם לגדלי לוח שונים:</u>

:$n = 8$

```
8 Queens solution using Genetic algorithms:
Best Position: <1,5,7,0,6,3,2,7> (fitness=2,generation=1)
Best Position: <5,1,4,7,0,3,1,6> (fitness=1,generation=2)
Best Position: <0,6,1,7,5,3,2,4> (fitness=1,generation=3)
Best Position: <5,1,4,7,0,3,1,6> (fitness=1,generation=4)
Best Position: <7,4,1,5,2,6,0,3> (fitness=1,generation=5)
Best Position: <4,4,7,0,3,6,2,5> (fitness=1,generation=6)
Best Position: <2,4,7,3,0,6,1,5> (fitness=0,generation=7)

Solution visualization:
------------------------
O  O  O  O  X  O  O  O
O  O  O  O  O  O  X  O
X  O  O  O  O  O  O  O
O  O  O  X  O  O  O  O
O  X  O  O  O  O  O  O
O  O  O  O  O  O  O  X
O  O  O  O  O  X  O  O
O  O  X  O  O  O  O  O

End-up after: 179.00 ticks and 0.00 seconds
```

$n = 15$:

```
15 Queens solution using Genetic algorithms:
Best Position: <4,2,10,6,1,3,12,3,0,11,7,2,7,13,8> (fitness=5,generation=1)
Best Position: <4,2,10,6,1,3,12,3,0,11,7,2,7,13,8> (fitness=5,generation=2)
Best Position: <9,2,0,13,11,8,12,3,3,14,4,10,5,1,7> (fitness=4,generation=3)
Best Position: <9,2,0,13,11,8,12,3,3,14,4,10,5,1,7> (fitness=4,generation=4)
Best Position: <4,14,11,3,10,6,1,9,2,0,7,1,7,11,13> (fitness=4,generation=5)
Best Position: <12,5,11,8,10,4,14,0,11,6,1,7,10,13,3> (fitness=3,generation=6)
Best Position: <12,5,11,8,10,4,14,0,11,6,1,7,10,13,3> (fitness=3,generation=7)
Best Position: <3,1,11,14,9,6,12,0,0,13,5,8,2,4,10> (fitness=3,generation=8)
Best Position: <12,5,11,8,10,4,14,0,11,6,1,7,10,13,3> (fitness=3,generation=9)
Best Position: <3,1,11,14,9,6,12,0,0,13,5,8,2,4,10> (fitness=3,generation=10)
Best Position: <12,5,11,8,10,4,14,0,11,6,1,7,10,13,3> (fitness=3,generation=11)
Best Position: <3,1,11,14,9,6,12,0,0,13,5,8,2,4,10> (fitness=3,generation=12)
Best Position: <4,1,14,0,6,7,12,10,13,2,9,3,8,11,1> (fitness=2,generation=13)
Best Position: <13,4,5,8,0,12,4,2,14,11,9,1,3,10,7> (fitness=2,generation=14)
Best Position: <4,1,14,0,6,7,12,10,13,2,9,3,8,11,1> (fitness=2,generation=15)
Best Position: <13,4,5,8,0,12,4,2,14,11,9,1,3,10,7> (fitness=2,generation=16)
Best Position: <4,1,14,0,6,7,12,10,13,2,9,3,8,11,1> (fitness=2,generation=17)
Best Position: <13,4,5,8,0,12,4,2,14,11,9,1,3,10,7> (fitness=2,generation=18)
Best Position: <4,1,14,0,6,7,12,10,13,2,9,3,8,11,1> (fitness=2,generation=19)
Best Position: <13,4,5,8,0,12,4,2,14,11,9,1,3,10,7> (fitness=2,generation=20)
Best Position: <4,1,14,0,6,7,12,10,13,2,9,3,8,11,1> (fitness=2,generation=21)
Best Position: <13,4,5,8,0,12,4,2,14,11,9,1,3,10,7> (fitness=2,generation=22)
Best Position: <3,5,10,14,11,6,12,2,0,8,12,9,1,13,7> (fitness=2,generation=23)
Best Position: <4,1,14,0,6,7,12,10,13,2,9,3,8,11,1> (fitness=2,generation=24)
Best Position: <13,4,5,8,0,12,4,2,14,11,9,1,3,10,7> (fitness=2,generation=25)
Best Position: <3,5,10,14,11,6,12,2,0,8,12,9,1,13,7> (fitness=2,generation=26)
Best Position: <4,1,14,0,6,7,12,10,13,2,9,3,8,11,1> (fitness=2,generation=27)
Best Position: <13,4,5,8,0,12,4,2,14,11,9,1,3,10,7> (fitness=2,generation=28)
Best Position: <13,4,5,11,6,14,12,2,0,8,10,1,3,9,7> (fitness=1,generation=29)
Best Position: <13,4,5,11,6,14,12,2,0,8,10,1,3,9,7> (fitness=1,generation=30)
Best Position: <13,4,5,11,6,14,12,2,0,8,10,1,3,9,7> (fitness=1,generation=31)
Best Position: <13,4,5,11,6,14,12,2,0,8,10,1,3,9,7> (fitness=1,generation=32)
Best Position: <13,4,5,11,6,14,12,2,0,8,10,1,3,9,7> (fitness=1,generation=33)
Best Position: <1,11,4,12,7,0,8,3,13,9,14,2,5,10,6> (fitness=1,generation=34)
Best Position: <13,4,5,11,6,14,12,2,0,8,10,1,3,9,7> (fitness=1,generation=35)
Best Position: <1,11,4,12,7,0,8,3,13,9,14,2,5,10,6> (fitness=1,generation=36)
Best Position: <1,11,4,12,7,0,8,3,13,9,14,2,5,10,6> (fitness=1,generation=37)
Best Position: <4,1,12,5,9,13,3,8,14,7,2,0,11,6,10> (fitness=0,generation=38)
```

```
Solution visualization:
-----------------------
O  O  O  O  O  O  O  O  O  O  O  X  O  O  O
O  X  O  O  O  O  O  O  O  O  O  O  O  O  O
O  O  O  O  O  O  O  O  O  O  X  O  O  O  O
O  O  O  O  O  O  X  O  O  O  O  O  O  O  O
X  O  O  O  O  O  O  O  O  O  O  O  O  O  O
O  O  O  X  O  O  O  O  O  O  O  O  O  O  O
O  O  O  O  O  O  O  O  O  O  O  O  O  X  O
O  O  O  O  O  O  O  O  O  X  O  O  O  O  O
O  O  O  O  O  O  O  X  O  O  O  O  O  O  O
O  O  O  O  X  O  O  O  O  O  O  O  O  O  O
O  O  O  O  O  O  O  O  O  O  O  O  O  O  X
O  O  O  O  O  O  O  O  O  O  O  O  X  O  O
O  O  X  O  O  O  O  O  O  O  O  O  O  O  O
O  O  O  O  O  X  O  O  O  O  O  O  O  O  O
O  O  O  O  O  O  O  O  X  O  O  O  O  O  O

End-up after: 1203.00 ticks and 1.00 seconds
```

:$n = 25$

```
25 Queens solution using Genetic algorithms:
Best Position: <17,12,14,11,3,23,6,16,3,10,21,16,9,19,6,22,20,8,22,24,1,12,2,0,7> (fitness=12,generation=1)
Best Position: <13,23,13,0,8,22,17,7,21,19,15,9,10,20,4,12,18,11,24,0,16,22,13,3,16> (fitness=11,generation=2)
Best Position: <0,12,22,11,3,23,6,16,3,10,21,16,14,2,6,22,20,13,9,15,18,4,19,17,7> (fitness=10,generation=3)
Best Position: <0,12,22,11,3,23,6,16,3,10,21,16,14,2,6,22,20,13,9,15,18,4,19,17,7> (fitness=10,generation=4)
Best Position: <17,8,1,4,11,7,16,22,19,2,14,20,6,5,0,13,7,18,7,19,23,15,10,3,23> (fitness=9,generation=5)
Best Position: <17,8,1,4,11,7,16,22,19,2,14,20,6,5,0,13,7,18,7,19,23,15,10,3,23> (fitness=9,generation=6)
Best Position: <17,8,1,4,11,7,16,22,19,2,14,20,6,5,0,13,7,18,7,19,23,15,10,3,23> (fitness=9,generation=7)
Best Position: <10,5,9,19,7,12,6,20,3,21,15,5,14,24,15,23,22,4,6,1,16,2,13,18,8> (fitness=8,generation=8)
Best Position: <14,22,3,9,16,10,5,20,0,21,7,20,23,8,17,23,6,2,18,21,4,1,10,10,13> (fitness=8,generation=9)
Best Position: <10,5,9,19,7,12,6,20,3,21,15,5,14,24,15,23,22,4,6,1,16,2,13,18,8> (fitness=8,generation=10)
Best Position: <14,22,3,9,16,10,5,20,0,21,7,20,23,8,17,23,6,2,18,21,4,1,10,10,13> (fitness=8,generation=11)
Best Position: <10,5,9,19,7,12,6,20,3,21,15,5,14,24,15,23,22,4,6,1,16,2,13,18,8> (fitness=8,generation=12)
Best Position: <6,15,0,10,19,5,11,17,3,22,16,14,24,8,4,23,9,21,2,20,7,1,18,20,20> (fitness=7,generation=13)
Best Position: <15,9,21,14,7,22,0,6,13,16,2,24,12,3,22,4,23,0,20,10,21,1,8,10,19> (fitness=7,generation=14)
Best Position: <6,15,0,10,19,5,11,17,3,22,16,14,24,8,4,23,9,21,2,20,7,1,18,20,20> (fitness=7,generation=15)
Best Position: <15,9,21,14,7,22,0,6,13,16,2,24,12,3,22,4,23,0,20,10,21,1,8,10,19> (fitness=7,generation=16)
Best Position: <6,15,0,10,19,5,11,17,3,22,16,14,24,8,4,23,9,21,2,20,7,1,18,20,20> (fitness=7,generation=17)
Best Position: <7,18,13,2,22,20,8,10,21,14,9,17,12,23,12,19,5,24,1,3,11,17,15,4,16> (fitness=7,generation=18)
```

…

```
Best Position: <6,13,15,10,7,2,20,23,0,11,21,12,16,9,24,4,17,22,3,18,14,19,5,1,8> (fitness=1,generation=120)
Best Position: <6,13,15,10,7,2,20,23,0,11,21,12,16,9,24,4,17,22,3,18,14,19,5,1,8> (fitness=1,generation=121)
Best Position: <6,13,15,2,7,10,20,23,0,16,14,1,21,9,24,4,17,11,3,12,22,19,5,18,8> (fitness=1,generation=122)
Best Position: <6,13,15,10,7,2,20,23,0,11,21,12,16,9,24,4,17,22,3,18,14,19,5,1,8> (fitness=1,generation=123)
Best Position: <6,13,15,10,7,2,20,23,0,11,21,12,16,9,24,4,17,22,3,18,14,19,5,1,8> (fitness=1,generation=124)
Best Position: <6,13,15,10,7,2,20,23,0,11,21,12,16,9,24,4,17,22,3,18,14,19,5,1,8> (fitness=1,generation=125)
Best Position: <6,13,15,21,7,2,20,23,0,16,10,1,11,9,24,4,17,22,3,12,14,19,5,18,8> (fitness=0,generation=126)

Solution visualization:
-----------------------
0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0
X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X
0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0
0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0
0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0
0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0
0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  X  0  0  0  0  0  0  0  0  0  0

End-up after: 5913.00 ticks and 5.00 seconds
```

9. השוואה בין ביצועי האלגוריתם הגנטי לבין $Minimal\ conflicts$:

פרמטר	GA	MC
זמן	סביר	מהיר
מקום	צריכה גבוהה	לינארי באורך הקלט
יכולת לפתור לוחות גדולים **ללא** מצב התחלה טוב	טובה מאוד	גרועה
יכולת לפתור לוחות גדולים **עם** מצב התחלה טוב	מצוינת	מצוינת
איטרציות **ללא** מצב התחלה טוב	סבירות	גבוהות
איטרציות עם מצב התחלה טוב	מעטות	מעטות

- ההכלאה בין האלגוריתמים הינה אפשרית באופן שבו $minimal\ conflicts$ יכול להוות מוטציה לצורך שיפור הגן לקראת הדור הבא וע"י כך לשפר את הדור הבא בצורה חכמה יותר ולשפר את מהירות ההתכנסות בצורה משמעותית.

10. הדגמה ושימוש באלגוריתם הגנטי בכדי לפתור את "בעיית השק":

פריט יראה מהצורה הבאה:

```
struct item {
    vector<pair<pair<unsigned int, unsigned int>,unsigned int>> _products; // one vector usage because of conceptualy understanding the problem
    unsigned int _weights; // constraint
    unsigned int _fitness; // profit
};

typedef vector<item> ga_couples;
```

- ה-$typedef$ נועד לצורך שימוש במערך של פריטים באורך נוח במהלך התוכנית.

באופן דומה ל-$framework$ הנ"ל נשתמש:

1. שיטת בחירה – $elitism$
2. שיטת שחלוף – $onePoint$
3. שיטת מוטציה - $flip\ one\ random\ bit$

פתרון בעיה 1:

```
Computing problem number 1 solution...
Best Solution: <1,1,0,1,0,0,1,0,0,0> (Weight = 161,Profit = 284,Generation = 1)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 2)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 3)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 4)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 5)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 6)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 7)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 8)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 9)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 10)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 11)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 12)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 13)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 14)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 15)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 16)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 17)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 18)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 19)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 20)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 21)
Best Solution: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 22)


An Optimal Packing: <1,1,1,1,0,1,0,0,0,0> (Weight = 165,Profit = 309,Generation = 22)
End-up after: 123.00 ticks and 0.00 seconds
```

פתרון בעיה 2:

```
Computing problem number 2 solution...
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 1)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 2)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 3)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 4)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 5)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 6)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 7)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 8)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 9)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 10)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 11)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 12)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 13)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 14)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 15)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 16)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 17)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 18)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 19)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 20)
Best Solution: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 21)


An Optimal Packing: <0,1,1,1,0> (Weight = 26,Profit = 51,Generation = 21)
End-up after: 97.00 ticks and 0.00 seconds
```

פתרון בעיה 3:

```
Computing problem number 3 solution...
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 1)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 2)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 3)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 4)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 5)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 6)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 7)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 8)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 9)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 10)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 11)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 12)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 13)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 14)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 15)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 16)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 17)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 18)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 19)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 20)
Best Solution: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 21)


An Optimal Packing: <1,1,0,0,1,0> (Weight = 190,Profit = 150,Generation = 21)
End-up after: 98.00 ticks and 0.00 seconds
```

פתרון בעיה 4:

```
Computing problem number 4 solution...
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 1)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 2)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 3)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 4)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 5)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 6)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 7)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 8)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 9)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 10)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 11)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 12)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 13)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 14)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 15)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 16)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 17)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 18)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 19)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 20)
Best Solution: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 21)


An Optimal Packing: <1,0,0,1,0,0,0> (Weight = 50,Profit = 107,Generation = 21)
End-up after: 116.00 ticks and 0.00 seconds
```

פתרון בעיה 5:

```
Computing problem number 5 solution...
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 1)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 2)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 3)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 4)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 5)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 6)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 7)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 8)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 9)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 10)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 11)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 12)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 13)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 14)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 15)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 16)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 17)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 18)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 19)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 20)
Best Solution: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 21)


An Optimal Packing: <1,0,1,1,1,0,1,1> (Weight = 104,Profit = 900,Generation = 21)
End-up after: 109.00 ticks and 1.00 seconds
```

פתרון בעיה 6:

```
Computing problem number 6 solution...
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 1)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 2)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 3)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 4)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 5)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 6)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 7)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 8)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 9)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 10)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 11)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 12)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 13)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 14)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 15)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 16)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 17)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 18)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 19)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 20)
Best Solution: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 21)


An Optimal Packing: <0,1,0,1,0,0,1> (Weight = 169,Profit = 1735,Generation = 21)
End-up after: 109.00 ticks and 1.00 seconds
```

<u>פתרון בעיה 7:</u>

```
Computing problem number 7 solution...
Best Solution: <1,1,0,0,0,1,1,1,1,0,0,0,0,1,1> (Weight = 750,Profit = 1455,Generation = 1)
Best Solution: <1,1,0,0,0,1,1,1,1,0,0,0,0,1,1> (Weight = 750,Profit = 1455,Generation = 2)
Best Solution: <1,1,0,0,0,1,1,1,1,0,0,0,0,1,1> (Weight = 750,Profit = 1455,Generation = 3)
Best Solution: <0,1,1,1,0,0,1,1,1,0,0,0,0,1,1> (Weight = 750,Profit = 1456,Generation = 4)
Best Solution: <0,1,1,1,0,0,1,1,1,0,0,0,0,1,1> (Weight = 750,Profit = 1456,Generation = 5)
Best Solution: <0,1,1,1,0,0,1,1,1,0,0,0,0,1,1> (Weight = 750,Profit = 1456,Generation = 6)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 7)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 8)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 9)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 10)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 11)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 12)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 13)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 14)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 15)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 16)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 17)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 18)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 19)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 20)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 21)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 22)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 23)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 24)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 25)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 26)
Best Solution: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 27)


An Optimal Packing: <1,0,1,0,1,0,1,1,1,0,0,0,0,1,1> (Weight = 749,Profit = 1458,Generation = 27)
End-up after: 202.00 ticks and 0.00 seconds
```

<u>פתרון בעיה 8:</u>

```
Computing problem number 8 solution...
Best Solution: <1,1,0,0,1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,0,1,1,0,1> (Weight = 6390158,Profit = 13316207,Generation = 1)
Best Solution: <1,1,0,0,1,0,0,0,0,1,1,0,1,0,1,1,0,0,0,0,1,1,0,1> (Weight = 6390158,Profit = 13316207,Generation = 2)
Best Solution: <0,1,0,1,0,0,0,0,1,1,1,0,1,0,0,1,0,0,0,0,1,0,1,0> (Weight = 6393588,Profit = 13316780,Generation = 3)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 4)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 5)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 6)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 7)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 8)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 9)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 10)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 11)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 12)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 13)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 14)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 15)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 16)
```

```
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 17)
Best Solution: <1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0> (Weight = 6398128,Profit = 13518963,Generation = 18)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 19)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 20)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 21)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 22)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 23)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 24)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 25)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 26)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 27)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 28)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 29)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 30)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 31)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 32)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 33)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 34)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 35)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 36)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 37)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 38)
Best Solution: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 39)


An Optimal Packing: <1,1,0,1,1,1,0,0,0,1,1,0,1,0,0,1,0,0,0,0,0,1,1,1> (Weight = 6402560,Profit = 13549094,Generation = 39)
End-up after: 477.00 ticks and 0.00 seconds
```

- כפי שניתן לראות הנ"ל הבעיות נפתרו מאוד מהר הן כפרמטר של מספר דורות עד להתכנסות והן כפרמטר של זמן.

תודה רבה על זמן הקריאה