**מבוא לבינה מלאכותית**

**מעבדה 3**

**סטודנט 1 : שם: אניס סרוג'י**

**ת.ז : 206052201**

**סטודנט2 : שם: עומר חטיב**

**ת.ז : 211706445**

1+2) We implemented a genetic algorithm to solve the bin packing problem. We used the previous code from lab2 with the same mating function. But for our fitness function we chose the following equation.

Where **F** - fitness of the solution, **n** - number of bins,  - fill of the **i-**th bin, **c** - capacity of the bin, **k** - constant greater than 1.

The function takes in consideration how much space is left (wasted) in each bin and gives a big penalty for over filled bins (The penalty is controlled by variable k; higher k means bigger penalty).

\*The algorithm needs to run enough time to find true solution (not optimal) as we place the items randomly inside the bins at the beginning.

We chose a population of size 2048 with K equal 2.

Results:

|  |  |  |
| --- | --- | --- |
| Test number | Time | Number of bins |
| 1 | 30 sec | 61 |
| 2 | 83 sec | 123 |
| 3 | 120 sec (limit) | 261 |
| 4 | 120 sec (limit) | 467 |

3) Each one of our gens is represented by a vector which maps each item i to it’s bin. So, our levenshtein distance would be calculated the following way:

for each item k in gens g1, g2. If the k is in a bin of different size in g1 than g2, we add a penalty of one.

\*\*Note: We can’t add penalty if the items are not in the same bin number as the number of bins are randomizes but they might contain identical items.

4)a+b)

We divided the population to niches (eight niches) and implemented fitness sharing algorithm with Probabilistic Crowding. In the pairing phase we took a random two gens and checked their levenshtein distance from each other, and according to that we decided to pair or not.

5+6) We added functions to calculate the average fitness of the population and the average levenshtein distance.

The values of those functions will control a variable called Mutation\_Factor which is responsible for increasing/ decreasing the mutation while mating (Triggered hyper-mutation).

While also those values responsible for adding a new random gens to the population (Random Immigrants).

7) We implemented all of the code required to NSGA-2 algorithm.

The next section was written by my partner , hence it’s in Hebrew and not English.

**חלק ב**

1. אחרי הרצת הסימולציה על 100 דורות , קיבלנו את התוצאות הבאות :
2. רואים מהתוצאות שאחוז הביטים הנכונים עלה ל 35% בממוצע , ואחוז הביטים הלא נכונים ירד ל 21% בממוצע , ואילו אחוז הביטים הנלמדים נשאר כמעט 0% לכל האיטרציות.

רואים מהגרף שאפקט בולדווין נמצא כאן אך לא בעוצמה שרצינו. כלומר למרות אחוז הלימה הקטן, אחוז הביטים הנכונים עלה ב 10% והלא נכונים ירד ב 4%. לכן כן הלמידה משנה את מרחב החיפוש וכן נותנת דרכי אבולוציה יותר טובים מאשר ללא למידה

אך לא רואים קפיצה חזקה מאוד כמו שראינו בהרצאות.