ΥΠΟΛΟΓΙΣΤΙΚΗ ΝΟΗΜΟΣΥΝΗ

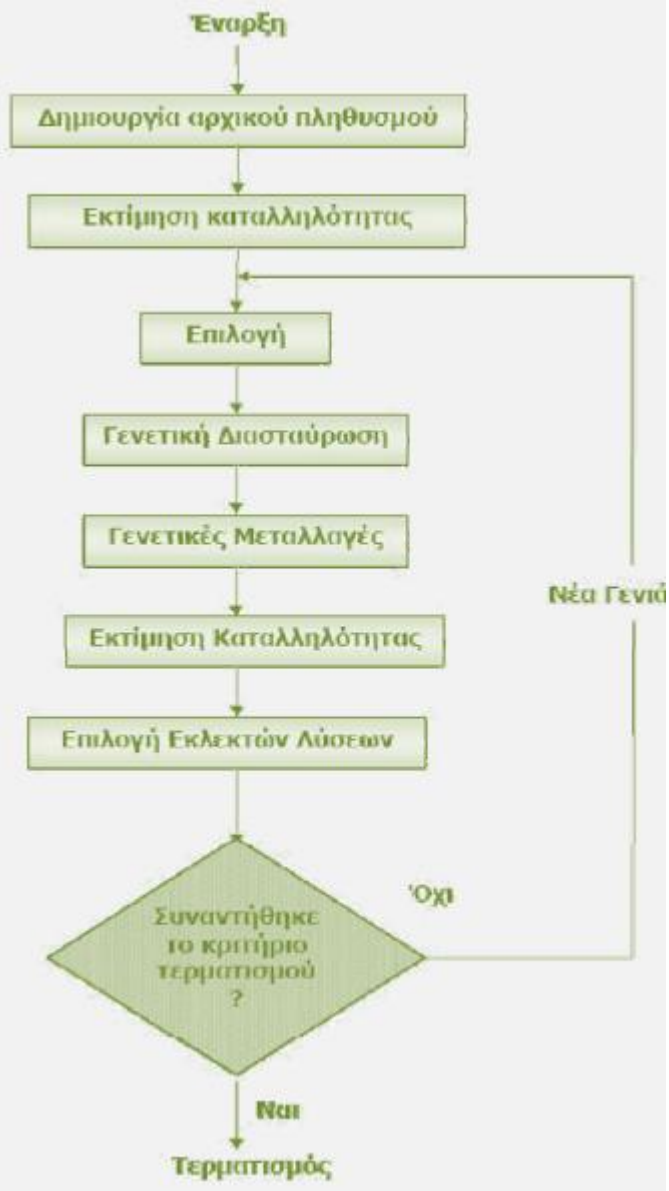
ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΣΤΟΥΣ ΓΕΝΕΤΙΚΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ

Θέμα: Το πρόβλημα του διαχωρισμού καρτών

Γενετικοί αλγόριθμοι

Οι γενετικοί αλγόριθμοι είναι αλγόριθμοι αναζήτησης εμπνευσμένοι από τη φυσική εξέλιξη (Θεωρία του Δαρβίνου). Χρησιμοποιούνται για την επίλυση προβλημάτων βελτιστοποίησης και αναζήτησης.

Ο αλγόριθμος αρχικά δημιουργεί έναν αρχικό τυχαίο πληθυσμό που αποτελείται από πιθανές λύσεις του προβλήματος. Στη συνέχεια, εκτελεί επαναλαμβανόμενα βήματα αναπαραγωγής, διασταύρωσης και μετάλλαξης για τη βελτίωση της ποιότητας του πληθυσμού και την εύρεση καλύτερων λύσεων.



Κατά την αναπαραγωγή, επιλέγονται οι καλύτερες λύσεις για να δημιουργήσουν νέες λύσεις μέσω διασταύρωσης. Στη συνέχεια, πραγματοποιείται μετάλλαξη στις νέες λύσεις για τη διατήρηση της ποικιλομορφίας του πληθυσμού.

Όλα τα παραπάνω στάδια συνεχίζονται μέχρι να βρεθεί μια αποδεκτή λύση ή μέχρι να επιτευχθεί μια προκαθορισμένη συνθήκη τερματισμού.

Στους γενετικούς αλγορίθμους, το χρωμόσωμα αναπαριστά μια πιθανή λύση στο πρόβλημα που θέλουμε να επιλύσουμε. Κάθε χρωμόσωμα αποτελείται από ένα σύνολο γονιδίων, τα οποία αντιστοιχούν στις παραμέτρους της λύσης που θέλουμε να βρούμε.

Η συνάρτηση προσαρμοστικότητας (fitness function) είναι μια συνάρτηση αξιολόγησης που χρησιμοποιείται για να καθορίσει πόσο καλή είναι μια πιθανή λύση. Το fitness function είναι ουσιαστικά μια συνάρτηση που αντιστοιχεί σε κάθε χρωμόσωμα μια τιμή fitness, που δείχνει πόσο καλά προσαρμόζεται το χρωμόσωμα στο πρόβλημα που επιδιώκουμε να λύσουμε. Με τη χρήση του fitness function, ο

γενετικός αλγόριθμος επιδιώκει να βρει το χρωμόσωμα με την υψηλότερη τιμή fitness, καθώς αυτό υποδηλώνει μια καλύτερη λύση για το πρόβλημα βελτιστοποίησης.

Οι καλύτερες λύσεις επιλέγονται και αξιολογούνται με βάση κάποιο κατώφλι. ΑΝ είναι πολύ καλές και ξεπερνούν το κατώφλι, δε μπαίνουν στη διαδικασία να γίνουν γονείς, αλλά περνούν αυτούσιες στην επόμενη γενιά. Η διαδικασία αυτή ονομάζεται ελιτισμός.

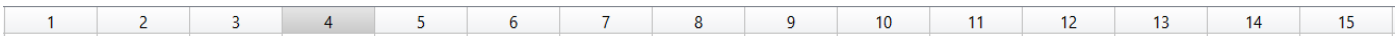
Το πρόβλημα του διαχωρισμού καρτών

Έχουμε 15 κάρτες αριθμημένες από το 1 μέχρι το 15. Θέλουμε να τις χωρίσουμε ισομερώς σε τρεις στοίβες, έτσι ώστε το άθροισμα των καρτών της πρώτης να είναι 49, το άθροισμα των καρτών της δεύτερης να είναι 33 και το γινόμενο της τελευταίας στοίβας να είναι 12600.

Να υλοποιηθεί πρόγραμμα σε MATLAB το οποίο να λύνει το παραπάνω πρόβλημα χρησιμοποιώντας γενετικούς αλγόριθμους.

Επίλυση του προβλήματος

1. Δημιουργία αρχικού πληθυσμού
   Οι 15 κάρτες που αντιστοιχούν σε κάποια πιθανή λύση του προβλήματος, αποτελούν ένα χρωμόσωμα. Παρακάτω φαίνεται μια πιθανή λύση, δηλαδή ένα χρωμόσωμα.



   Κατά την αρχικοποίηση του πληθυσμού επιλέξαμε να δημιουργούνται 60 χρωμοσώματα, με την παράμετρο **'PopulationSize'**.

```
pop_size = 60;
```

   Η συνάρτηση που χρησιμοποιήθηκε για την αρχικοποίηση του πληθυσμού φαίνεται παρακάτω. Με τη συνάρτηση randperm εξασφαλίζεται ότι σε κάθε 15άδα καρτών, ο κάθε αριθμός θα εμφανίζεται μία μόνο φορά.

```
function pop =
cards_problem_create_permutations(NVARS,FitnessFcn,options)
totalPopulationSize = sum(options.PopulationSize);
n = NVARS;
pop = cell(totalPopulationSize,1);
for i = 1:totalPopulationSize
    pop{i} = randperm(n);
end
```

2. Εκτίμηση καταλληλότητας (fitness function)
   Για την αξιολόγηση των λύσεων χρησιμοποιήθηκε η παρακάτω fitness function. Η συνάρτηση αυτή χωρίζει τις κάρτες σε τρεις στοίβες των πέντε καρτών και υπολογίζει το άθροισμα των δύο πρώτων στοιβών και το γινόμενο της τρίτης στήλης. Στη συνέχεια, υπολογίζουμε πόσο απέχουν οι τιμές που έχουμε βρει από τις επιθυμητές τιμές που μας δίνει στο πρόβλημα και προσθέτουμε τις διαφορές, ώστε να βρούμε την τιμή του fitness function.

```
function scores = cards_problem_fitness(x)
    target = [49, 33, 12600];
    pop_size = size(x,1);
    scores = zeros(size(x,1),1);

    for i = 1:pop_size
        perm = x{i};
        stack1 = perm(1:5);
        stack2 = perm(6:10);
        stack3 = perm(11:15);

        sum1 = sum(stack1);
        sum2 = sum(stack2);
        prod3 = prod(stack3);

        total_sum = abs([sum1, sum2, prod3] - target);
        scores(i) = sum(total_sum);
    end
end
```

   Όσο μικρότερη είναι η τιμή της συνάρτησης προσαρμοστικότητας τόσο καλύτερη είναι η λύση που έχουμε βρει. Η ιδανική λύση είναι αυτή για την οποία το fitness function έχει μηδενική τιμή, αυτό σημαίνει ότι τα δύο αθροίσματα και το γινόμενο στις τρεις στοίβες μας έχουν ακριβώς τις τιμές που δίνονται στην εκφώνηση του προβλήματος.

3. Γενετική διασταύρωση (crossover)
   Η διαδικασία με την οποία δημιουργούνται νέοι απόγονοι καθορίζεται από τη συνάρτηση διασταύρωσης. Στη συνάρτηση αυτή επιλέγονται δύο γονείς, οι οποίοι κληρονομούν την πληροφορία στο παιδί. Τα παιδιά που δημιουργούνται, επιλέξαμε να είναι τα μισά από τους γονείς.
   Στη λύση που αντιστοιχεί στο παιδί, έχουν επιλεχθεί δύο αριθμοί και το διάνυσμα που βρίσκεται ανάμεσα από τους δύο αυτούς αριθμούς αντιστρέφεται.

   Για παράδειγμα αν έχουμε τον παρακάτω γονέα.

Το παιδί που θα προκύψει από τη διαδικασία της διασταύρωσης μπορεί να είναι το παρακάτω.


```
function xoverKids  =
cards_problem_crossover_permutation(parents,options,NVARS, ...
    FitnessFcn,thisScore,thisPopulation)
nKids = length(parents)/2;
xoverKids = cell(nKids,1); % Normally zeros(nKids,NVARS);
index = 1;

for i=1:nKids
    % here is where the special knowledge that the population is
a cell
    % array is used. Normally, this would be
thisPopulation(parents(index),:);
    parent = thisPopulation{parents(index)};
    index = index + 2;

    % Flip a section of parent1.
    p1 = ceil((length(parent) -1) * rand);
    p2 = p1 + ceil((length(parent) - p1- 1) * rand);
    child = parent;
    child(p1:p2) = fliplr(child(p1:p2));
    xoverKids{i} = child; % Normally, xoverKids(i,:);
end
```

4. Γενετικές μεταλλάξεις (Mutations)
   Κατά τη διαδικασία της μετάλλαξης, ένα γονίδιο μεταβάλλεται με τυχαίο τρόπο σε ένα ή περισσότερα χρωμοσώματα. Με τον τρόπο αυτό είναι πιθανό να καταλήξουμε σε καλύτερη λύση, αν όμως γίνονται πάρα πολλές μεταλλάξεις οι λύσεις μας θα εμπεριέχουν σε μεγαλύτερο βαθμό τον παράγοντα της τύχης, κάτι που δεν είναι επιθυμητό.
   Ο τρόπος με τον οποίο γίνονται οι μεταλλάξεις είναι ότι επιλέγονται δύο σημεία σε κάποιο γονέα. Οι αριθμοί που βρίσκονται στα σημεία αυτά αλλάζουν θέση μεταξύ τους και στη συνέχεια το μεταλλαγμένο γονίδιο κληρονομείται στο παιδί.

   Για παράδειγμα αν έχουμε τον παρακάτω γονέα.

   Το παιδί που θα προκύψει από τη διαδικασία της μετάλλαξης μπορεί να είναι το παρακάτω.

```
function mutationChildren =
cards_problem_mutate_permutation(parents ,options,NVARS, ...
    FitnessFcn, state, thisScore,thisPopulation,mutationRate)

mutationChildren = cell(length(parents),1);% Normally
zeros(length(parents),NVARS);
for i=1:length(parents)
    parent = thisPopulation{parents(i)}; % Normally
thisPopulation(parents(i),:)
    p = ceil(length(parent) * rand(1,2));
    child = parent;
    child(p(1)) = parent(p(2));
    child(p(2)) = parent(p(1));
    mutationChildren{i} = child; % Normally mutationChildren(i,:)
end
```

5. Αποτελέσματα και διαγράμματα

   Ο κώδικας που χρησιμοποιήσαμε καλεί τις συναρτήσεις που αναφέραμε παραπάνω και τυπώνει τα περιεχόμενα κάθε στοίβας, τα αθροίσματα των δύο πρώτων στοιβών και το γινόμενο της τρίτης και την τιμή του fitness function.

   Παρακάτω μπορούμε να δούμε το αποτέλεσμα του προγράμματος, το οποίο έχει βρει τη βέλτιστη λύση. Έχει επιλύσει δηλαδή το πρόβλημα του διαχωρισμού καρτών.

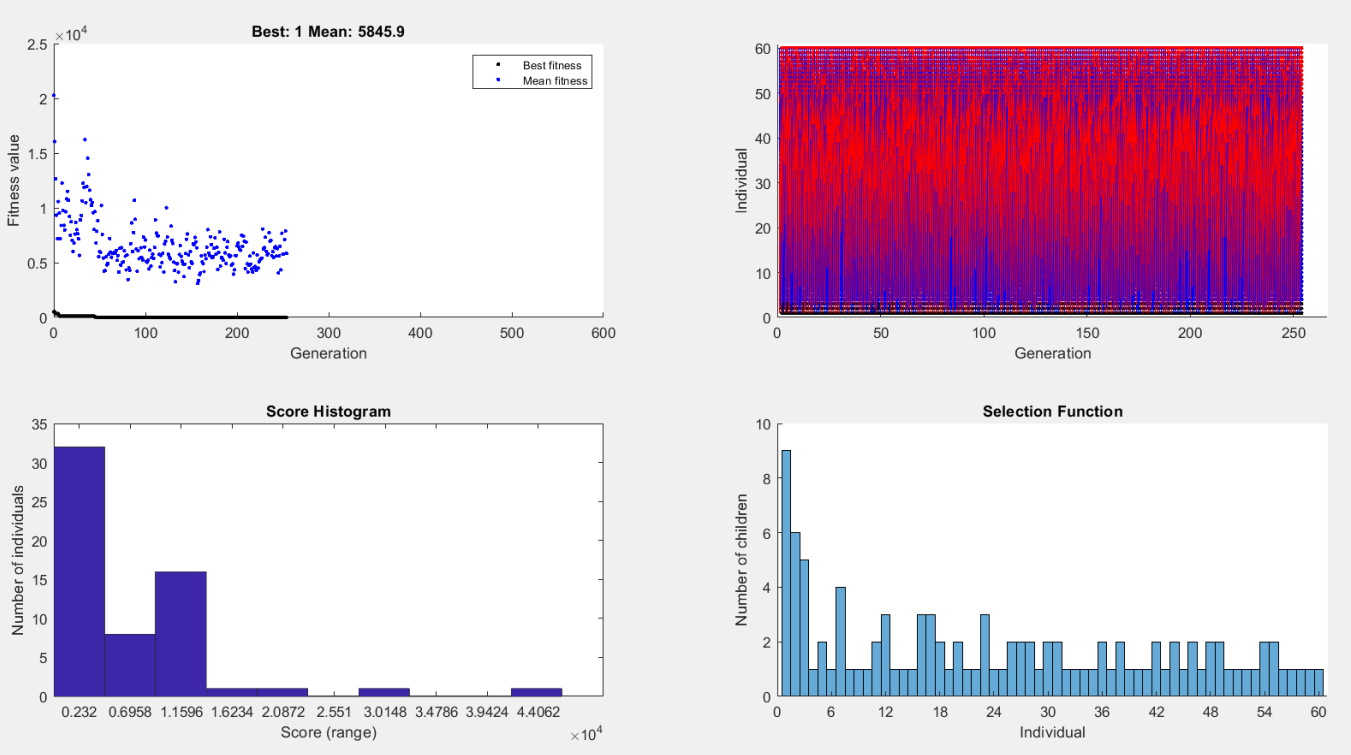
```
Stack 1: 11   1  13  15   9
Sum of stack 1: 49
Stack 2: 4  12   8   7   2
Sum of stack 2: 33
Stack 3: 5  10  14   3   6
Product of stack 3: 12600
Fitness score: 0.00
```

   Αν τρέξουμε το πρόγραμμα πολλές φορές θα δούμε ότι δε βρίσκει πάντα τη βέλτιστη λύση, κάποιες φορές βρίσκει απλά κάποια καλή λύση. Γενικά οι τιμές του fitness function κυμαίνονται από 0 έως 2.

   Με την παράμετρο 'Generations', καθορίζουμε ότι ο γενετικός αλγόριθμος θα τρέξει το μέγιστο για 1000 γενιές. Με το 'PopulationSize', ορίζουμε τον αριθμό των πιθανών λύσεων (γονέων) που θέλουμε να δημιουργηθούν, εμείς το ορίσαμε 60 στο συγκεκριμένο πρόβλημα. Με την παράμετρο 'StallGenLimit', ορίζουμε ότι ο αλγόριθμος θα τερματιστεί αν δεν υπάρξει βελτίωση του καλύτερου fitness score για 200 γενιές.

```
n = 15;
pop_size = 60;

type cards_problem_create_permutations.m
type cards_problem_crossover_permutation.m
type cards_problem_mutate_permutation.m
type cards_problem_fitness.m

FitnessFcn = @(x) cards_problem_fitness(x);

options = gaoptimset('PopulationType', 'custom', 'PopInitRange',
...
                     [1;n]);
options =
gaoptimset(options,'CreationFcn',@cards_problem_create_permutatio
ns, ...

'CrossoverFcn',@cards_problem_crossover_permutation, ...

'MutationFcn',@cards_problem_mutate_permutation, ...

'Generations',1000,'PopulationSize',pop_size, ...

'StallGenLimit',200,'Vectorized','on');
numberOfVariables = n;
[x, fval, reason, output] = ga(FitnessFcn, numberOfVariables,
options)

% Display results
stack1 = x{1}(1:5);
stack2 = x{1}(6:10);
stack3 = x{1}(11:15);

fprintf('Stack 1: %s\n', num2str(stack1));
fprintf('Sum of stack 1: %d\n', sum(stack1));
fprintf('Stack 2: %s\n', num2str(stack2));
fprintf('Sum of stack 2: %d\n', sum(stack2));
fprintf('Stack 3: %s\n', num2str(stack3));
fprintf('Product of stack 3: %d\n', prod(stack3));

fprintf('Fitness score: %.2f\n', fval); % We want to make this as
smaller as possible
% fprintf('Check: %d\n', abs(49-sum(stack1))+abs(33-
sum(stack2))+abs(12600-prod(stack3)));
```

Παρακάτω μπορούμε να δούμε ορισμένα διαγράμματα που κάναμε με τη βοήθεια του optimtool.



Από το πρώτο διάγραμμα μπορούμε να δούμε ότι ο αλγόριθμος τρέχει μόνο για περίπου 250 γενιές και βρίσκει τη βέλτιστη λύση, σύμφωνα με τα κριτήρια που θέσαμε, οπότε δεν υπάρχει λόγος να συνεχίσει να τρέχει. Επίσης, παρατηρούμε ότι πάρα πολλές φορές το fitness function λαμβάνει την τιμή 0 που είναι και η βέλτιστη στο πρόβλημά μας. Επίσης, οι μέσες τιμές της συνάρτησης προσαρμοστικότητας φαίνεται να έχουν μια πτώση όσο περνάμε από γενιά σε γενιά.

Επιπλέον, παρατηρούμε ότι στο κάτω αριστερά διάγραμμα οι περισσότερες λύσεις φτάνουν να έχουν πολύ μικρή τιμή στο fitness function, τιμές μικρότερες από 1. Στο κάτω δεξιά διάγραμμα, βλέπουμε ότι οι πρώτες 3 λύσεις παράγουν τα περισσότερα παιδιά.

Στο πάνω δεξιά διάγραμμα, φαίνεται η εξέλιξη των γενεών για κάθε χρωμόσωμα. Μπορούμε να διακρίνουμε πως εξελίσσεται η κάθε λύση, όσο περνάνε οι γενιές.
Το μπλε χρώμα μας δείχνει τις διασταυρώσεις, το κόκκινο είναι οι μεταλλάξεις και το μαύρο είναι ο ελιτισμός.

Συμπεράσματα

Απ’ όλα τα παραπάνω καταλήγουμε στο συμπέρασμα ότι για το συγκεκριμένο πρόβλημα η χρήση των γενετικών αλγορίθμων μπορεί να προσφέρει μία πολύ καλή ή και τη βέλτιστη λύση αν οι παράμετροι οριστούν με κατάλληλο τρόπο.

Βιβλιογραφία

Διαφάνειες μαθήματος και εργαστηρίου, ‘’Υπολογιστική Νοημοσύνη’’.