Γενετικοί αλγόριθμοι Πρόβλημα Περιοδεύοντος Πωλητή (Traveling Salesman Problem)

Οι γενετικοί αλγόριθμοι μιμούνται την θεωρία της εξέλιξης των ειδών (γνωστή και από την βιολογία) με σκοπό να βρουν μία ικανοποιητικά καλή (όχι απαραιτήτως τη βέλτιστη) λύση σε κάποιο πρόβλημα. Οι μέθοδοι που χρησιμοποιούν είναι η *μετάλλαξη*, η *επιλογή* και η *διασταύρωση*. Οι τιμές των παραμέτρων του συστήματος κωδικοποιούνται σε binary μορφή.

Θεωρία εξελίξεως των ειδών

Όπως όλοι γνωρίζουμε, όλοι οι οργανισμοί αποτελούνται από κύτταρα, στα οποία υπάρχει ένας αριθμός χρωμοσωμάτων που καθορίζουν έναν οργανισμό. Τα χρωμοσώματα αυτά αποτελούνται από γονίδια, εκ των οποίων κάθε γονίδιο είναι υπεύθυνο για κάποιο χαρακτηριστικό του οργανισμού (ύψος, χρώμα μαλλιών κλπ). Όταν λοιπόν οι οργανισμοί αναπαράγονται, αρχικά έχουμε τη διαδικασία της *διασταύρωσης* (crossover) κατά την οποία προκύπτουν νέα χρωμοσώματα από τα γονίδια των γονιών. Σε κάθε νέο οργανισμό, όμως, παρατηρείται και η *μετάλλαξη* (mutation), δηλαδή η ελαφρά τροποποίηση των στοιχείων του DNA του. Ο νέος αυτός οργανισμός, λοιπόν, κρίνεται βιώσιμος ή όχι από το κατά πόσο θα καταφέρει να επιβιώσει, εξαρτώμενος τόσο από τα στοιχεία που πήρε από τους γονείς του, όσο και από την μετάλλαξη του.

Ο γενετικός αλγόριθμος δημιουργεί ένα πληθυσμό λύσεων από τα αρχικά δεδομένα, επιλέγοντας συνήθως τυχαίες τιμές, συγκρίνοντας τες με το επιθυμητό βέλτιστο αποτέλεσμα. Το πόσο κοντά είναι η κάθε λύση ως προς την επιθυμητή, υπολογίζεται με ένα μέτρο που ονομάζεται *Συνάρτηση Καταλληλότητας*.

Τυχαίες λύσεις που παράγονται ή οι λύσεις με την καλύτερη συνάρτηση καταλληλότητας (αυτό το φαινόμενο είναι ο *ελιτισμός* και θα το δούμε κατά κάποιο τρόπο στο παράδειγμα μας αργότερα), στην επόμενη γενιά, διασταυρώνονται πάλι και μεταλλάσσονται δημιουργώντας νέους πληθυσμούς λύσεων (παιδιά) που είναι ακόμα πιο κοντά στο επιθυμητό. Αυτό επαναλαμβάνεται όσες γενεές χρειαστεί, ούτως ώστε να πλησιάσουμε στο μέγιστο δυνατό το επιθυμητό αποτέλεσμα.

Παράδειγμα στη φύση

Ας δούμε πως λειτουργεί αυτή η μέθοδος στη φύση, εφόσον είπαμε ότι, αυτή η λογική είναι εμπνευσμένη από τη βιολογία. Τα ελάφια, για παράδειγμα, αποτελούν μία οικογένεια θηλαστικών μηρυκαστικών που έχουν ως απώτερο σκοπό να αναπαραχθούν, να εξελιχθούν και να αποφύγουν να γίνουν τροφή για τους σημαντικότερους εχθρούς τους (τα λιοντάρια, τους λύκους κλπ).

Αν υποθέσουμε, λοιπόν, ότι σε μία γενιά ελαφιών ένα μέρος τους φαγωθεί-νικηθεί από τους εχθρούς, αυτό αυτομάτως σημαίνει ότι, όσα επιβιώσουν θα είναι τα ‘αρτιότερα’ και ‘δυνατότερα’ κατά πλειοψηφία ή έστω ως γενικό σύνολο. Τα εναπομείναντα λοιπόν ελάφια, στην επόμενη γενιά, θα διασταυρωθούν και θα δημιουργήσουν δυνατότερους απογόνους σε μία γενιά συνολικά ‘δυνατότερη’ (πιο βιώσιμη) από την προηγούμενη. Δηλαδή, η νέα γενιά θα είναι πιο κοντά προς την «ιδανική γενιά» που δεν θα μπορεί να φαγωθεί από τα λιοντάρια (κάτι παρόμοιο με την συνάρτηση καταλληλότητας των Γ.Α). Με παρόμοιο τρόπο θα λειτουργήσει και η επόμενη, μεθεπόμενη κοκ γενιά ελαφιών, κατά τις οποίες ένα ποσοστό δεν θα επιβιώνει, αλλά όσα επιβιώνουν θα δημιουργούν ‘αρτιότερους’ συνολικά απογόνους. Στον κάθε απόγονο, βέβαια, όπως αναφέραμε προηγουμένως, παρατηρείται και η μετάλλαξη, η οποία μπορεί να λειτουργήσει είτε αρνητικά-είτε θετικά στο θέμα της βιωσιμότητας του οργανισμού.

Μόλις παρατηρήσαμε στη φύση την μετάλλαξη και την διασταύρωση που χρησιμοποιούν και οι Γ.Α. Ενδιαφέρον και κορυφαίο παράδειγμα για την εξελικτική λειτουργία των ειδών είναι, επίσης, οι κατσαρίδες και το πώς και πόσο γρήγορα εξελίσσονται και μεταλλάσσονται, ούτως ώστε να γίνονται ανθεκτικές στα εντομοκτόνα φάρμακα εξόντωσης που παρασκευάζει συνεχώς ο άνθρωπος για αυτές (ως γνωστόν, ένα παλιό εντομοκτόνο δεν καταφέρνει τίποτα σε μία κατσαρίδα, για αυτό και τα εντομοκτόνα συνεχώς αναπτύσσονται).

Παράδειγμα στην πληροφορική

Πριν προχωρήσουμε στην ανάλυση της παρούσας άσκησης, καλό θα ήταν να αναφέρουμε επιγραμματικά ένα απλό παράδειγμα γενετικών αλγορίθμων στην πληροφορική.

Ας υποθέσουμε ότι, έχουμε τέσσερα τυχαία χρώματα (έστω τα red, blue, green, yellow) εκ των οποίων επιθυμούμε μέσω Γ.Α να παράγουμε το ροζ (pink). Όπως εισαγωγικά είπαμε, οι τιμές των παραμέτρων κωδικοποιούνται σε δυαδική μορφή, οπότε:

Έχουμε:
RED: 111111110000000000000000
BLUE: 000000000000000011111111
GREEN: 000000001111111100000000
YELLOW: 111111111111111100000000

Θέλουμε:

PINK: 111111110110100110110100

Κατά την επόμενη γενιά, τα 4 αυτά χρώματα θα διασταυρωθούν με τυχαίους τρόπους και θα μεταλλάσσονται, ούτως ώστε να παράγουν γονίδια-παιδιά χρώματα.

Από τα νέα χρώματα, ο αλγόριθμος θα κρατήσει τα πιο κοντινά προς το ροζ (είναι σημαντικό σε κάθε βήμα να γνωρίζουμε την συνάρτηση καταλληλότητας) και θα απομακρύνει τα 'ασθενέστερα', ούτως ώστε να διασταυρώσει τα 'ισχυρότερα' για την δημιουργία της επόμενης γενιάς χρωμάτων κοκ μέχρι να προσεγγίσουμε το ροζ στον μέγιστο βαθμό σε κάποια νιοστή γενιά χρωμάτων.

Σημείωση: Σε κάθε γενιά ο αλγόριθμος, εφόσον επιλέγει τα δυνατότερα χρώματα, απομακρύνει τα πιο αδύναμα που δεν καταφέρουν να επιβιώσουν.

Επόμενη γενιά:

Διασταύρωση RED με BLUE (πχ τα 12 πρώτα ψηφία του RED και τα 12 τελευταία του BLUE):

111111110000000011111111

Το νέο χρώμα που προκύπτει, με κάποια μικρή μετάλλαξη:

111111110000000011111110

Διασταύρωση RED με GREEN (πχ τα 12 πρώτα ψηφία του red και τα 12 τελευταία του green):

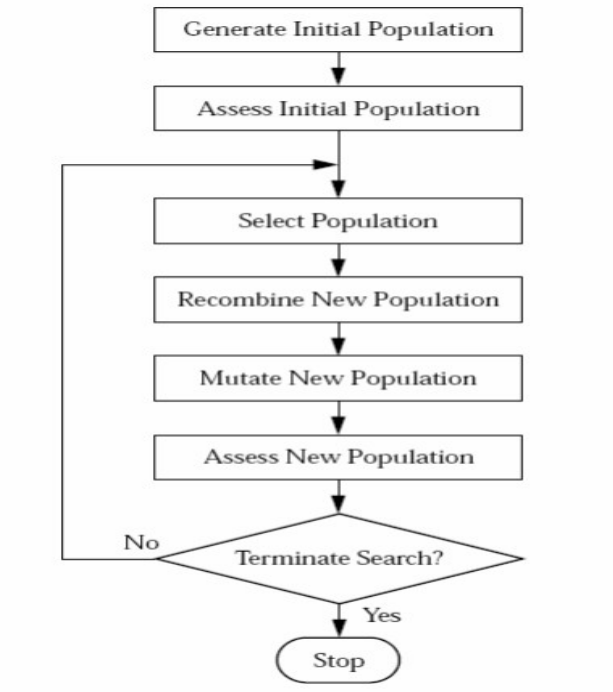
111111110000111100000000

Το νέο χρώμα που προκύπτει, με κάποια μικρή μετάλλαξη:

111111110000111100001000

Κοκ.

Παρατηρούμε ότι, παράγονται νέα χρώματα μέσω των διασταυρώσεων των αρχικών χρωμάτων. Έστω ότι, έχουμε ορίσει στον αλγόριθμο να παράγει με αυτόν τον τρόπο 20 τυχαία νέα χρώματα. Ο αλγόριθμος θα απομακρύνει τα 16 ασθενέστερα και θα κρατήσει τα 4 ισχυρότερα ώστε, στην επόμενη γενιά να τα επαναδιασταυρώσει και να παράγει 20 γενικά πλησιέστερα προς το ζητούμενο ροζ. Ο αλγόριθμος θα επαναλαμβάνεται μέχρι να φτάσουμε σε ένα ικανοποιητικό αποτέλεσμα.



Εικόνα 16. Κλασική δομή ενός απλού γενετικού αλγορίθμου

4.2 Πρόβλημα περιοδεύοντος πωλητή

Αν υποθέσουμε ότι έχουμε 50 πόλεις, οι πιθανές διαδρομές σε περίπτωση χρήσης κλασικού αλγορίθμου θα ήταν περίπου 50! (δηλαδή 3.04140932e64) υπολογισμοί για να βρεθεί η ελάχιστη διαδρομή, ενώ μέσω χρήσης γενετικών εξελικτικών αλγορίθμων θα χρειαστούμε μόλις 3486 επαναλήψεις. Οι εξελικτικοί αλγόριθμοι (evolutionary algorithms) δεν επιχειρεί να εντοπίσει τη καλύτερη λύση, αλλά ξεκινώντας από ένα αρχικό πληθυσμό λύσεων, παράγει νέες γενιές λύσεων, καλύτερες από τις προηγούμενες.

Σε γενικές γραμμές και με απλά λόγια οι γενετικοί αλγόριθμοι είναι χρήσιμοι όταν:

- Γνωρίζουμε τι έχουμε
- Γνωρίζουμε που θέλουμε να φτάσουμε
- Δεν γνωρίζουμε (ή γνωρίζουμε λίγα για) όλα τα ενδιάμεσα

Προβλήματα που δεν λύνονται (ή είναι πολύ δύσκολο και χρονοβόρο να λυθούν) με κλασικούς αλγόριθμους είναι τα **μη πολυωνυμικά προβλήματα**, δηλαδή τα προβλήματα όπου είναι δυνατή η αναζήτηση μιας προσεγγιστικής λύσης που δεν είναι απαραίτητα η βέλτιστη σε πολυωνυμικό χρόνο. Αυτά τα προβλήματα έχουν συνήθως έναν απλό, αλλά πολύ αργό αλγόριθμο O(2^n), του οποίου η χρήση είναι **απαγορευτική** όταν το εύρος λύσεων είναι μεγάλο. Το παρόν πρόβλημα του περιοδεύοντος πωλητή είναι ένα χαρακτηριστικό μη πολυωνυμικό πρόβλημα.

Το πρόβλημα του περιοδεύοντος πωλητή

Ένας πωλητής πρέπει να ταξιδέψει σε κάποιες πόλεις, οι οποίες συνδέονται μεταξύ τους και των οποίων οι αποστάσεις μεταξύ τους είναι γνωστές, με τον πιο σύντομο δυνατό τρόπο και περνώντας μόνον μία φορά από κάθε πόλη, επιστρέφοντας στο τέλος στη βάση από όπου ξεκίνησε.

4.2.2 Κώδικας Matlab

```
function varargout = tsp_ga(xy,dmat,popSize,numIter,showProg,showResult)

//Αρχικοποίηση μεταβλητών και έλεγχοι
nargs = 6;
for k = nargin:nargs-1
switch k
case 0
    xy = 10*rand(50,2);
case 1
    N = size(xy,1);
    a = meshgrid(1:N);
    dmat = reshape(sqrt(sum((xy(a,:)-xy(a',:)).^2,2)),N,N);
case 2
    popSize = 100;
case 3
    numIter = 1e4;
case 4
    showProg = 1;
case 5
    showResult = 1;
otherwise
end
end

//Έλεγχος τιμών εισόδου
[N,dims] = size(xy);
[nr,nc] = size(dmat);
if N ~= nr || N ~= nc
error('Invalid XY or DMAT inputs!')
end
n = N;
```

```
//Έλεγχοι
popSize = 4*ceil(popSize/4);
numIter = max(1,round(real(numIter(1))));
showProg = logical(showProg(1));
showResult = logical(showResult(1));

//Αρχικοποίηση πληθυσμού
pop = zeros(popSize,n);
pop(1,:) = (1:n);
for k = 2:popSize
pop(k,:) = randperm(n);
end

//Τρέχει ο Γενετικός αλγόριθμος
globalMin = Inf;
totalDist = zeros(1,popSize);
distHistory = zeros(1,numIter);
tmpPop = zeros(4,n);
newPop = zeros(popSize,n);
if showProg
      pfig = figure('Name','TSP_GA | Current Best Solution','Numbertitle','off');
end
for iter = 1:numIter

//Υπολογισμός συνολικής απόστασης
for p = 1:popSize
      d = dmat(pop(p,n),pop(p,1)); % Closed Path
      for k = 2:n
      d = d + dmat(pop(p,k-1),pop(p,k));
end
      totalDist(p) = d;
end
```

```
// Βρίσκει τη βέλτιστη διαδρομή από το πληθυσμό
[minDist,index] = min(totalDist);
distHistory(iter) = minDist;
if minDist < globalMin
        globalMin = minDist;
        optRoute = pop(index,:);
if showProg

//Σχεδίαση της καλύτερης διαδρομής

figure(pfig);
rte = optRoute([1:n 1]);
if dims > 2, plot3(xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
       else plot(xy(rte,1),xy(rte,2),'r.-'); end
        title(sprintf('Total Distance = %1.4f, Iteration = %d',minDist,iter));
end
end

//Χειριστές των Γενετικών αλγορίθμων

randomOrder = randperm(popSize);
for p = 4:4:popSize
rtes = pop(randomOrder(p-3:p),:);
dists = totalDist(randomOrder(p-3:p));
[ignore,idx] = min(dists); %#ok
bestOf4Route = rtes(idx,:);
routeInsertionPoints = sort(ceil(n*rand(1,2)));
I = routeInsertionPoints(1);
J = routeInsertionPoints(2);
for k = 1:4 % Mutate the Best to get Three New Routes
tmpPop(k,:) = bestOf4Route;
switch k
        case 2 //Flip
        tmpPop(k,I:J) = tmpPop(k,J:-1:I);
        case 3 //Swap//ανταλλαγή
        tmpPop(k,[I J]) = tmpPop(k,[J I]);
        case 4 //Slide//στρέφει τα σημεία προς τα δεξιά
```

```
        tmpPop(k,I:J) = tmpPop(k,[I+1:J I]);
otherwise % Do Nothing
          end
    end
    newPop(p-3:p,:) = tmpPop;
  end
  pop = newPop;
end

if showResult


//Εμφάνιση τελικών διαγραμμάτων των Γ.Α
figure('Name','TSP_GA | Results','Numbertitle','off');
subplot(2,2,1);
pclr = ~get(0,'DefaultAxesColor');
if dims > 2, plot3(xy(:,1),xy(:,2),xy(:,3),'.','Color',pclr);
else plot(xy(:,1),xy(:,2),'.','Color',pclr); end
title('City Locations');
subplot(2,2,2);
imagesc(dmat(optRoute,optRoute));
title('Distance Matrix');
subplot(2,2,3);
rte = optRoute([1:n 1]);
if dims > 2, plot3(xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
else plot(xy(rte,1),xy(rte,2),'r.-'); end
title(sprintf('Total Distance = %1.4f',minDist));
subplot(2,2,4);
plot(distHistory,'b','LineWidth',2);
title('Best Solution History');
set(gca,'XLim',[0 numIter+1],'YLim',[0 1.1*max([1 distHistory])]);
end

// Επιστροφή αποτελεσμάτων
if nargout
varargout{1} = optRoute;
varargout{2} = minDist;
end
```

4.2.2 Επεξήγηση Κώδικα

Ο παρών κώδικας δεν ακολουθεί την αυστηρή δομή και φιλοσοφία των γενετικών αλγορίθμων (επιλογή, διασταύρωση, μετάλλαξη), διότι δεν χρησιμοποιεί πουθενά την μέθοδο crossover (διασταύρωση), αλλά παρόλα αυτά παραμένει Γ.Α και δίνει μία πολύ καλή προσεγγιστικά και γρήγορη λύση στο πρόβλημα.

Ο λόγος που δεν χρησιμοποιείται η crossover είναι διότι, οι τελεστές θα λειτουργούσαν εν τέλει καταστροφικά κάνοντας υπερβολικά μεγάλες αλλαγές στις ανα γενιά διαδρομές. Αντ' αυτού ο δημιουργός προτιμάει 3 διαφορετικές μεθόδους μετάλλαξης.

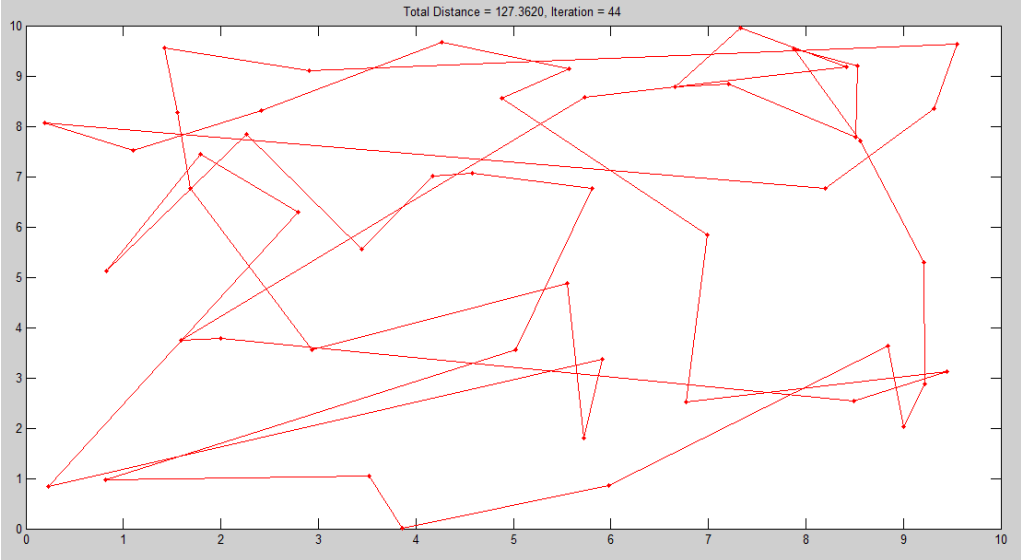
Συγκεκριμένα, η λογική του κώδικα, καθώς και οι mutation τελεστές που χρησιμοποιούνται έχουν ως εξής: (βλ. // Χειριστές των Γενετικών αλγορίθμων στον παρόντα κώδικα).

Στο πρώτο for του εν λόγω σημείου ομαδοποιούνται αρχικά ομάδες των 4 διαδρομών, εκ των οποίων κρατάει την καλύτερη διαδρομή από τις τέσσερις αυτές και την περνάει στην νέα γενιά. Στο δεύτερο for πραγματοποιούνται τρεις μεταλλάξεις *(Flip, Swap, Slide)* σε αυτήν την 'καλύτερη διαδρομή' με αποτέλεσμα να δημιουργηθούν 3 ακόμη μεταλλαγμένες διαδρομές οι οποίες (μαζί με την κανονική 'καλύτερη') περνάνε επίσης στην επόμενη γενιά.

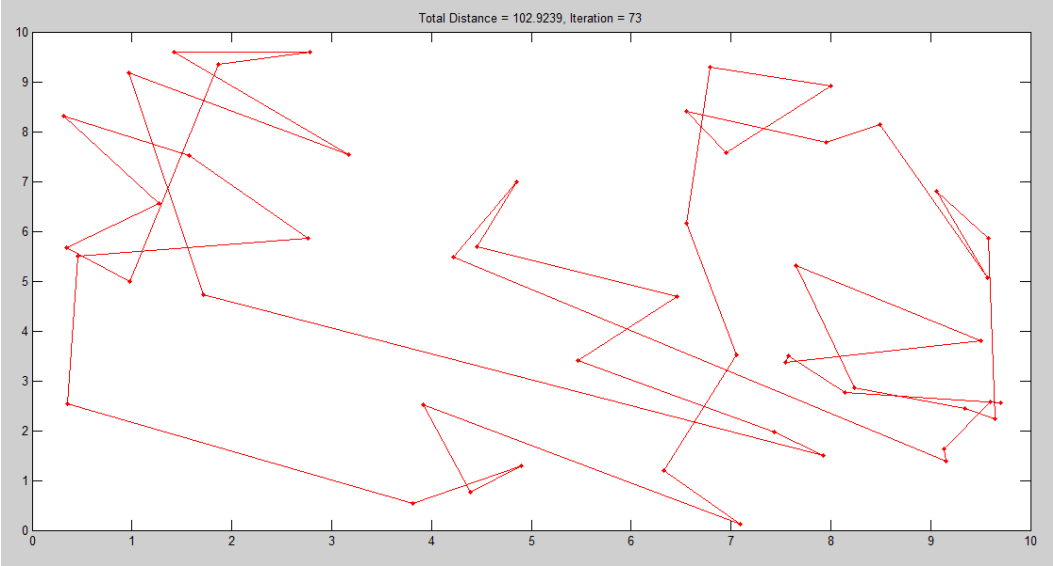
Έτσι, στην επόμενη γενιά, θα έχουμε νέες ομάδες των τεσσάρων διαδρομών, εκ των οποίων πάλι θα επιλεχθεί η καλύτερη και θα επαναμεταλλαχθεί κοκ. Η διαδικασία θα επαναλαμβάνεται μέχρι να επιτευχθεί ένα ικανοποιητικό αποτέλεσμα, το οποίο δεν θα μικραίνει ιδιαίτερα ή θα παραμένει σταθερό μετά από κάποια επανάληψη και έπειτα. Αξιοσημείωτο είναι το φαινόμενο του ελιτισμού που παρατηρείται στον κώδικα μας, εφόσον επιλεκτικά διατηρούνται προς αναπαραγωγή οι δυνατότεροι απόγονοι και ο αλγόριθμος δεν αφήνεται πλήρως στην τυχαιότητα.

<u>Αποτελέσματα κώδικα</u>

Στην παρούσα εκτέλεση του κώδικα έχουμε 50 πόλεις *(n=50)* και πληθυσμούς μεγέθους 100. Στις πρώτες 2 εικόνες παρατηρούμε τα αποτελέσματα που εξάγονται κατά την πορεία των επαναλήψεων του κώδικα. Συγκεκριμένα, απεικονίζονται οι πορείες κατά την 44η και την 73η επανάληψη.

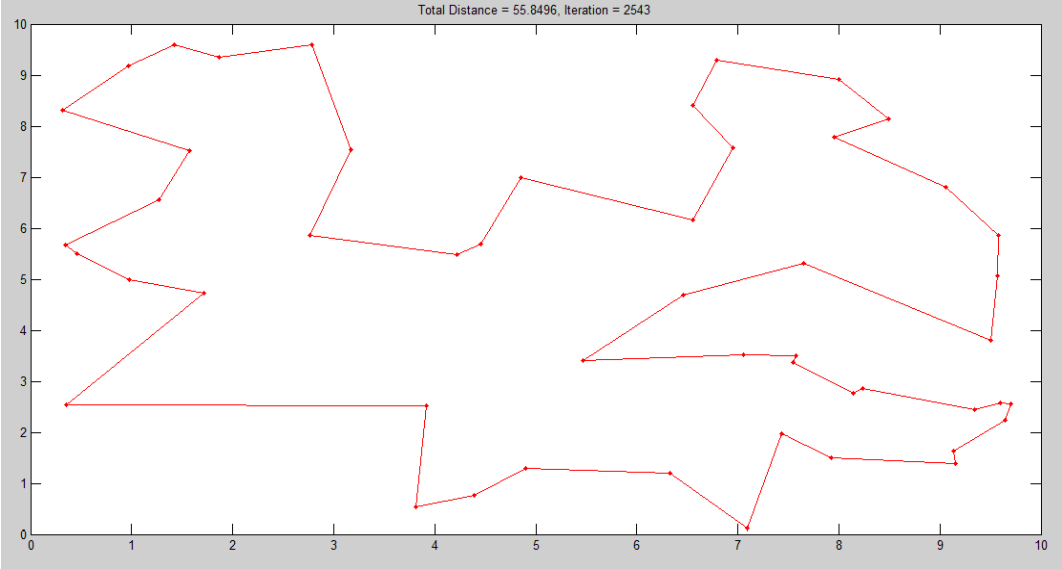


Εικόνα 17. Αποτελέσματα κατά την 44η επανάληψη.

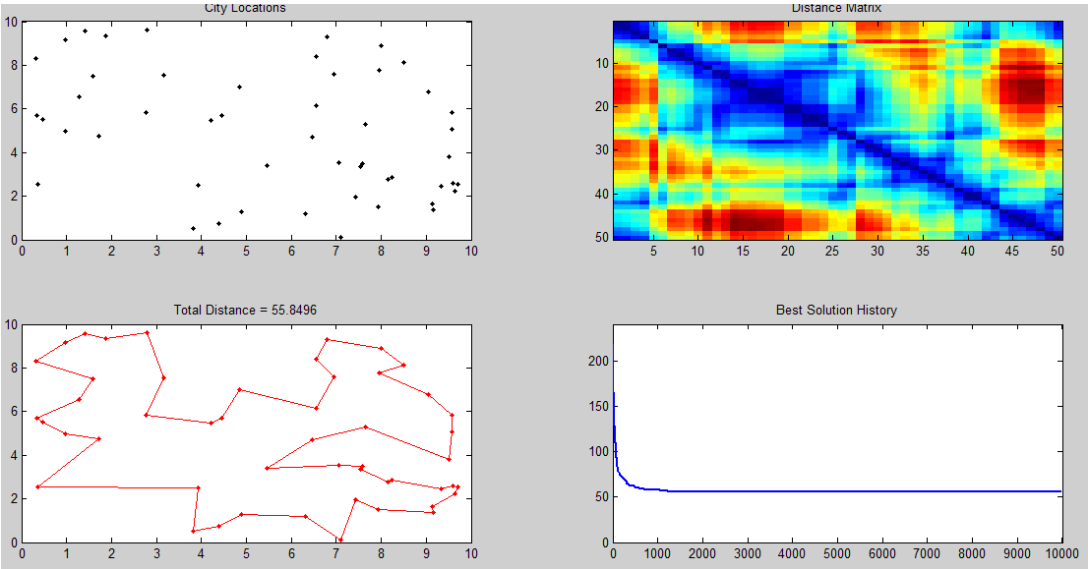


Εικόνα 18. Αποτελέσματα κατά την 73η επανάληψη.

Στις παρακάτω εικόνες παρουσιάζονται τόσο τα αποτελέσματα της τελευταίας επανάληψης, όσο και οι τελικοί γράφοι που δημιουργούνται με τις τοποθεσίες των πόλεων, τον πίνακα αποστάσεων, τη τελική βέλτιστη διαδρομή, καθώς και το ιστορικό καλύτερων λύσεων ανά επανάληψη.



Εικόνα 19. Αποτελέσματα κατά τη 2543η επανάληψη



Εικόνα 20. Τελικά διαγράμματα: 1ο Τελικές θέσεις πόλεων (City locations), 2ο Βέλτιστη διαδρομή (Distance Matrix), 3ο Συνολική διαδρομή (Total Distance), 4ο Ιστορικό καλύτερων λύσεων (Best Solution History).