Τεχνικές Βελτιστοποίησης - Project 2024

ΠΑΠΑΔΟΠΟΥΛΟΣ ΠΑΝΑΓΙΩΤΗΣ - 10697

Εισαγωγή:

Το συγκεκριμένο πρόβλημα μπορεί να διατυπωθεί μαθηματικά ως ένα πρόβλημα βελτιστοποίησης, όπου επιδιώκεται η ελαχιστοποίηση μιας συνάρτησης κόστους που περιγράφει τον συνολικό χρόνο διέλευσης των οχημάτων από το δίκτυο. Ωστόσο, λόγω της πολυπλοκότητας της ροής σε πραγματικά δίκτυα, η εξεύρεση της βέλτιστης λύσης με κλασικές μεθόδους είναι συχνά μη εφικτή.

Για την επίλυση του προβλήματος αυτού, χρησιμοποιήθηκε ένας γενετικός αλγόριθμος ($Genetic\ Algorithm - GA$), ο οποίος είναι μία στοχαστική μέθοδος βελτιστοποίησης εμπνευσμένη από τη φυσική επιλογή και την εξελικτική βιολογία. Οι γενετικοί αλγόριθμοι βασίζονται στη διατήρηση και βελτίωση ενός συνόλου πιθανών λύσεων (πληθυσμός), μέσω διαδικασιών επιλογής, αναπαραγωγής και μετάλλαξης.

Στην παρούσα μελέτη, ο γενετικός αλγόριθμος αναπτύχθηκε και υλοποιήθηκε στο $MATLAB$, με στόχο την εύρεση της βέλτιστης κατανομής της ροής των οχημάτων στο δίκτυο. Στο πλαίσιο της εργασίας, εξετάζονται διαφορετικά σενάρια συνολικής εισερχόμενης ροής, προκειμένου να αναλυθεί η επίδρασή τους στο αποτέλεσμα της βελτιστοποίησης.

Θέμα 1:

Η βελτιστοποίηση της ροής των οχημάτων σε ένα δίκτυο μπορεί να διατυπωθεί μαθηματικά ως ένα πρόβλημα ελαχιστοποίησης, όπου επιδιώκεται η βέλτιστη κατανομή των ροών στις ακμές του δικτύου, λαμβάνοντας υπόψη τους περιορισμούς ισορροπίας ροής και χωρητικότητας.

- **Μεταβλητές του προβλήματος**

Το πρόβλημα διατυπώνεται σε έναν κατευθυνόμενο γράφο $G = (N, E)$, όπου:

- N είναι το σύνολο των κόμβων (διασταυρώσεων ή σημείων του δικτύου).
- E είναι το σύνολο των ακμών (δρόμων ή διαδρομών που συνδέουν τους κόμβους).
- Κάθε ακμή $e \in E$ έχει μία μέγιστη χωρητικότητα C_e, που αντιπροσωπεύει τον μέγιστο αριθμό οχημάτων που μπορούν να διέλθουν.
- Η συνολική εισερχόμενη ροή στο δίκτυο είναι V, η οποία κατανέμεται στις πηγές ($source\ edges$) του γραφήματος.

Για κάθε ακμή e, ορίζουμε:

x_e = ροή οχημάτων στην ακμή e .

t_e = σταθερός χρόνος διέλευσης της ακμής e .

a_e = σταθερά που σχετίζεται με τον κυκλοφοριακό φόρτο στην ακμή e .

C_e = μέγιστη επιτρεπτή ροή στην ακμή e .

Το συνολικό κόστος μετακίνησης στο δίκτυο δίνεται από τη συνάρτηση:

$$T(x) = \sum_{e \in E} \left(t_e + \frac{a_e x_e}{1 - \frac{x_e}{C_e}} \right)$$

όπου ο δεύτερος όρος αναπαριστά την επίδραση της συμφόρησης στον χρόνο διέλευσης.

• **Περιορισμοί του Προβλήματος**

- Ισορροπία ροής σε κάθε κόμβο (Διατήρηση Ροής)

Η συνολική ροή που εισέρχεται σε κάθε κόμβο πρέπει να ισούται με τη συνολική ροή που εξέρχεται, εξασφαλίζοντας ότι δεν δημιουργείται ή χάνεται ροή.

Για κάθε κόμβο $n \in N$, η συνθήκη ισορροπίας ροής είναι:

$$\sum_{e \in E_{in}(n)} x_e = \sum_{e \in E_{out}(n)} x_e$$

όπου:
• $E_{in}(n)$ είναι το σύνολο των ακμών που εισέρχονται στον κόμβο n.
• $E_{out}(n)$ είναι το σύνολο των ακμών που εξέρχονται από τον κόμβο n .

Οι πηγές του δικτύου ($source\ edges$) έχουν μόνο εξερχόμενη ροή, ενώ οι καταβόθρες ($sink\ edges$) έχουν μόνο εισερχόμενη ροή.

- Περιορισμοί Χωρητικότητας

Η ροή κάθε ακμής δεν μπορεί να ξεπερνά το όριο χωρητικότητας της:

$$0 \leq x_e \leq C_e, \quad \forall e \in E$$

- Περιορισμός Συνολικής Ροής

Η συνολική εισερχόμενη ροή πρέπει να αντιστοιχεί στο V:

$$\sum_{e \in E_{source}} x_e = V$$

• **Διατύπωση του Προβλήματος ως Πρόβλημα Βελτιστοποίησης**

Το πρόβλημα συνοψίζεται ως εξής:

- Στόχος (Αντικειμενική Συνάρτηση)

Ελαχιστοποίηση του συνολικού χρόνου μετακίνησης:

$$\min_x \sum_{e \in E} \left(t_e + \frac{a_e x_e}{1 - \frac{x_e}{C_e}} \right)$$

• Περιορισμοί

1. Διατήρηση ροής στους κόμβους:

$$\sum_{e \in E_{in}(n)} x_e = \sum_{e \in E_{out}(n)} x_e, \quad \forall n \in N$$

2. Ροές εντός χωρητικότητας:

$$0 \leq x_e \leq C_e, \quad \forall e \in E$$

3. Συνολική ροή ίση με V:

$$\sum_{e \in E_{source}} x_e = V$$

- **Συμπέρασμα**

Η μαθηματική διατύπωση αποτυπώνει το πρόβλημα ως ένα μη γραμμικό πρόβλημα βελτιστοποίησης, λόγω της μορφής της συνάρτησης κόστους και των περιορισμών χωρητικότητας. Λόγω της πολυπλοκότητας, η εύρεση της βέλτιστης λύσης με αναλυτικές μεθόδους είναι δύσκολη, γι' αυτό και επιλέγεται ένας γενετικός αλγόριθμος για τη βελτιστοποίηση.

Θέμα 2:

Η επίλυση του προβλήματος βελτιστοποίησης της ροής των οχημάτων στο δίκτυο πραγματοποιήθηκε με χρήση γενετικού αλγορίθμου ($Genetic\ Algorithm - GA$), ο οποίος προσομοιώνει τις διαδικασίες της φυσικής επιλογής και της εξέλιξης.

Ο γενετικός αλγόριθμος αναπτύχθηκε στο $MATLAB$ και βασίζεται στις βασικές διαδικασίες της αρχικοποίησης πληθυσμού, αξιολόγησης καταλληλότητας, επιλογής, διασταύρωσης ($crossover$), μετάλλαξης ($mutation$) και διόρθωσης πληθυσμού ($fixing$).

- **Δομή του Γενετικού Αλγορίθμου**

Ο γενετικός αλγόριθμος αποτελείται από τα εξής βασικά στάδια:

1. Αρχικοποίηση Πληθυσμού: Δημιουργία αρχικών λύσεων που σέβονται τους περιορισμούς χωρητικότητας και ισορροπίας ροής.

2. Υπολογισμός της Καταλληλότητας ($Fitness\ Function$): Αξιολόγηση κάθε λύσης με βάση τον συνολικό χρόνο μετακίνησης.

3. Επιλογή των Καλύτερων Ατόμων ($Selection$): Διατήρηση των δύο καλύτερων λύσεων.

4. Διασταύρωση ($Crossover$): Δημιουργία νέων λύσεων μέσω συνδυασμού γονιδίων από δύο γονείς.

5. Μετάλλαξη ($Mutation$): Τυχαία αλλαγή σε μία ακμή του δικτύου για 10% των απογόνων.

6. Διόρθωση Πληθυσμού ($Fix\ Population$): Εφαρμογή κανόνων για διατήρηση ισορροπίας ροής.

7. Επανάληψη μέχρι Σύγκλιση: Εκτέλεση του αλγορίθμου για συγκεκριμένο αριθμό γενεών ή μέχρι η βέλτιστη λύση να σταθεροποιηθεί.

- **Αρχικοποίηση Πληθυσμού**

Η σύνδεση των κόμβων μεταξύ τους αναπαρίσταται μέσω του πίνακα γειτνίασης των ακμών E, όπου:
- Κάθε γραμμή του E αντιστοιχεί σε μία ακμή του δικτύου.
- Κάθε στήλη αναπαριστά τη σύνδεση της ακμής με άλλες ακμές.
- $E(i, j) = 1$ αν η ακμή i τροφοδοτεί την ακμή j, διαφορετικά $E(i, j) = 0$.

Ο πίνακας αυτός χρησιμοποιείται σε όλα τα στάδια του αλγορίθμου για τον εντοπισμό των γονικών και αδελφικών ακμών, καθώς και για τη διατήρηση της ισορροπίας ροής.

Η αρχικοποίηση του πληθυσμού πραγματοποιείται με τυχαία κατανομή ροής στις πηγές, τηρώντας τον περιορισμό $\sum x_e = V$.

Για κάθε άτομο του πληθυσμού:
1. Οι πηγές ($source\ edges$) λαμβάνουν τυχαίες τιμές που αθροίζονται σε V.
2. Οι υπόλοιπες ακμές λαμβάνουν ροή βάσει των γονέων τους, διατηρώντας την ισορροπία ροής.
3. Οι χωρητικότητες C_e δεν παραβιάζονται.

```
function pop_matrix = initialize_population(pop_size, E, V, C, source_edges)
   num_edges = size(E, 1);
    pop_matrix = zeros(pop_size, num_edges);  % Initialize matrix with zeros
    num_source_edges = length(source_edges);
    for i = 1:pop_size
        correct_source_edges = false;
        while ~correct_source_edges
            % Assign values for all but one source edge
            for j = 1:num_source_edges-1
                % Calculate flow for assigned edges
                intake_flow_occupied = sum(pop_matrix(i, source_edges));
                % Calculate available flow for the rest of the edges
                available_flow = V - intake_flow_occupied;
                % Assign cars to the edge with the maximum being the available
                % flow or maximum allowed
                upper_limit = min(available_flow, C(j));
                pop_matrix(i, j) = rand * upper_limit;
            end
            % Assign value to last edge
            pop_matrix(i, length(source_edges)) = V - sum(pop_matrix(i, source_edges) );
            % Check if last edge is within acceptable range, else start
            % again from scratch
            if pop_matrix(i, length(source_edges)) <= C(num_source_edges)
                correct_source_edges = true;
                break;
            else
                pop_matrix(i,:) = zeros(1,num_edges);
            end
        end
        correct_edges = false;
        while ~correct_edges
            % Distribute the flow throughout the network
            for j = num_source_edges+1:num_edges
                % Parent edges are edges that a car is coming from
                parent_edges = find(E(:, j) == 1);
                % Sibling edges are edges that a car can choose to go instead
                % of the edge j being checked
                sibling_edges = find(any(E(parent_edges, :) == 1, 1));
                % Calculate the flow of the sibling edges
                flow_of_siblings = sum(pop_matrix(i, sibling_edges));
                % Calculate the flow of the parent edges
                flow_of_parents = sum(pop_matrix(i, parent_edges));
                % If this is the last sibling edge, equal the flow of the
                % parents. Else, assign random value between 0 and the
                % remaining flow of the intersection
                if j == max(sibling_edges)
```

```
                    pop_matrix(i, j) = flow_of_parents - flow_of_siblings;
                    % Check if last edge is within acceptable range, else start
                    % again from scratch
                    if pop_matrix(i, j) > C(j)
                        pop_matrix(i,num_source_edges+1:num_edges) = zeros(1,num_edges-
num_source_edges);
                        break;
                    elseif j == num_edges
                        correct_edges = true;
                        break;
                    end

                else
                    upper_limit = min(flow_of_parents-flow_of_siblings, C(j));
                    pop_matrix(i, j) = rand * upper_limit;
                end
            end
        end
    end
end
```

- **Συνάρτηση Καταλληλότητας (*Fitness Function*)**

Η συνάρτηση καταλληλότητας υπολογίζει το συνολικό κόστος μετακίνησης:

$$T(x) = \sum_{e \in E} \left(t_e + \frac{a_e x_e}{1 - \frac{x_e}{C_e}} \right)$$

όπου t_e, a_e και C_e είναι γνωστές παράμετροι κάθε ακμής.

```
function fitness = evaluate_fitness(pop_matrix, t, a, C)
    % Evaluate fitness for each solution in the population
    fitness = sum(t + (a .* pop_matrix) ./ (1 - (pop_matrix ./ C)), 2);
end
```

- **Επιλογή των Καλύτερων Ατόμων (*Selection*)**

Η επιλογή βασίζεται στη διατήρηση των δύο καλύτερων λύσεων από κάθε γενιά:

```
[~, idx] = sort(fitness);
parent1 = pop_matrix(idx(1), :);
parent2 = pop_matrix(idx(2), :);
```

Οι δύο καλύτερες λύσεις διατηρούνται και παράγουν 98 νέους απογόνους.

- **Διασταύρωση (*Single − Point Crossover*)**

Για κάθε νέο άτομο, επιλέγεται ένα τυχαίο σημείο διασταύρωσης και οι γονείς ανταλλάσσουν τμήματα των γονιδίων τους:

```
function offspring = single_point_crossover(parent1, parent2)
    crossover_point = randi([1, length(parent1)-1]);  % Random crossover point
    offspring = [parent1(1:crossover_point), parent2(crossover_point+1:end)];
end
```

- **Μετάλλαξη (*Mutation*)**

Σε 10% των απογόνων, μία τυχαία ακμή αλλάζει την τιμή της:

```
function pop_matrix = apply_mutation(pop_matrix, mutation_rate, C, source_edges, V)
    num_individuals = size(pop_matrix, 1);  % Number of solutions in population
    num_edges = size(pop_matrix, 2);  % Number of edges

    num_mutations = round(num_individuals * mutation_rate);  % Number of individuals to mutate

    for i = 1:num_mutations
        individual_idx = randi(num_individuals);  % Randomly select an individual to mutate

        % Select a random edge, ensuring it's NOT a source edge
        edge_index = randi([1,11]);

        % Mutation value as a fraction of the edge capacity
        mutation_value = (rand - 0.5) * 0.2 * C(edge_index);  % +/- 10% of capacity

        % Apply mutation while respecting capacity limits
        pop_matrix(individual_idx, edge_index) = max(0, ...
            min(pop_matrix(individual_idx, edge_index) + mutation_value, C(edge_index)));
    end
end
```

- **Διόρθωση Πληθυσμού (*Fix Population*)**

Μετά από κάθε μετάλλαξη, η ροή διορθώνεται ώστε να διατηρείται η ισορροπία στους κόμβους:

```
function pop_matrix = fix_population(pop_matrix, source_edges, V, C, E)
    num_source_edges = length(source_edges);
    for i = 3:size(pop_matrix, 1)
        % Fix source edge flows
        total_flow = sum(pop_matrix(i, source_edges));
        deficit = V - total_flow;
        correction = round(deficit/num_source_edges);
        if deficit > 0
            for j=source_edges
                [~, min_index] = min(pop_matrix(i, source_edges));
                if j ~= source_edges(num_source_edges)
                    pop_matrix(i,min_index) = pop_matrix(i,min_index) + correction;
                else
                    total_flow = sum(pop_matrix(i, source_edges));
                    % disp(['Total flow for adjustment: ', num2str(V-total_flow)])
                    pop_matrix(i,min_index) = pop_matrix(i,min_index) + V-total_flow;
                end
            end
        end
       if deficit < 0
           for j=source_edges
                [~, max_index] = max(pop_matrix(i, source_edges));
                if j ~= source_edges(num_source_edges)
                    pop_matrix(i,max_index) = pop_matrix(i,max_index) + correction;
                else
                    total_flow = sum(pop_matrix(i, source_edges));
                   pop_matrix(i,max_index) = pop_matrix(i,max_index) + V-total_flow;
                end
            end
        end
        for j=num_source_edges+1:length(C)
           parent_edges = find(E(:, j) == 1);
            sibling_edges = find(any(E(parent_edges, :) == 1, 1));
            flow_of_siblings = sum(pop_matrix(i, sibling_edges));
            flow_of_parents = sum(pop_matrix(i, parent_edges));
            if abs(flow_of_parents - flow_of_siblings) > 1e-6
               deficit = flow_of_parents - flow_of_siblings;
                num_siblings_edges = length(sibling_edges);
                if deficit > 0
                    for k=sibling_edges
```

```
                        [~, min_relative_index] = min(pop_matrix(i, sibling_edges));
                        min_index = sibling_edges(min_relative_index);
                        pop_matrix(i,min_index) = pop_matrix(i,min_index) + deficit/
num_siblings_edges;
                    end
                end
                if deficit < 0
                    for k=sibling_edges
                        [~, max_relative_index] = max(pop_matrix(i, sibling_edges));
                        max_index = sibling_edges(max_relative_index);
                        pop_matrix(i,max_index) = pop_matrix(i,max_index) + deficit/
num_siblings_edges;
                    end
                end
            end
        end
        % Ensure no edge exceeds capacity
        pop_matrix(i, :) = min(pop_matrix(i, :), C);
    end
end
```

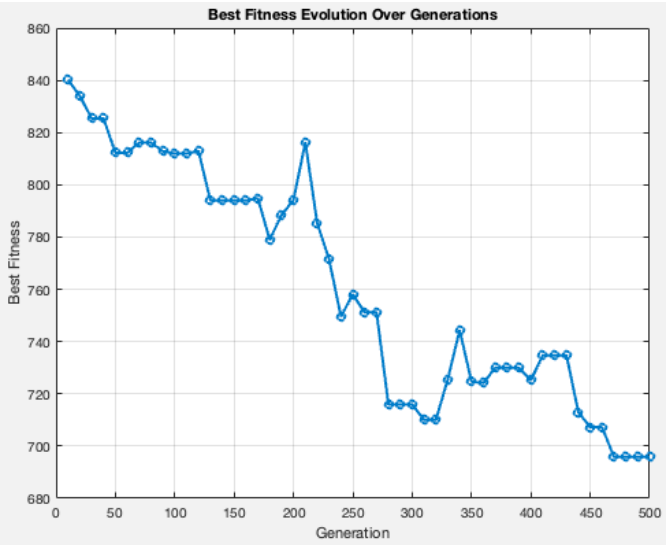
- **Επανάληψη μέχρι Σύγκλιση**

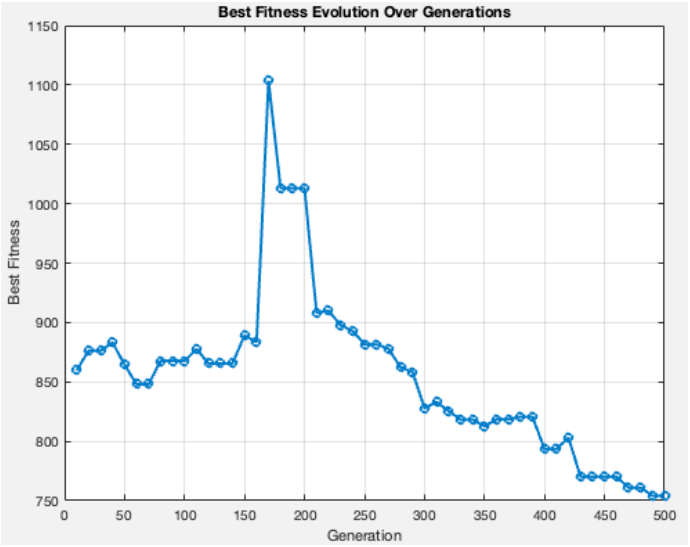
Ο αλγόριθμος εκτελείται για συγκεκριμένο αριθμό γενεών, και η βέλτιστη λύση εκτυπώνεται κάθε 10 γενιές:

```
if mod(generation, 10) == 0
        best_fitness = min(fitness);  % Best fitness value in the population
        disp(['Generation ', num2str(generation), ': Best Fitness = ', num2str(best_fitness)]);
    end
```

- **Συμπέρασμα και Αποτελέσματα**

Ο γενετικός αλγόριθμος βελτιστοποιεί την κατανομή της ροής στο δίκτυο, ελαχιστοποιώντας τον συνολικό χρόνο μετακίνησης, ενώ διατηρείται η ισορροπία ροής και η τήρηση των περιορισμών χωρητικότητας. Ακολουθούν τα αποτελέσματα για διάφορες περιστάσεις του αλγορίθμου.

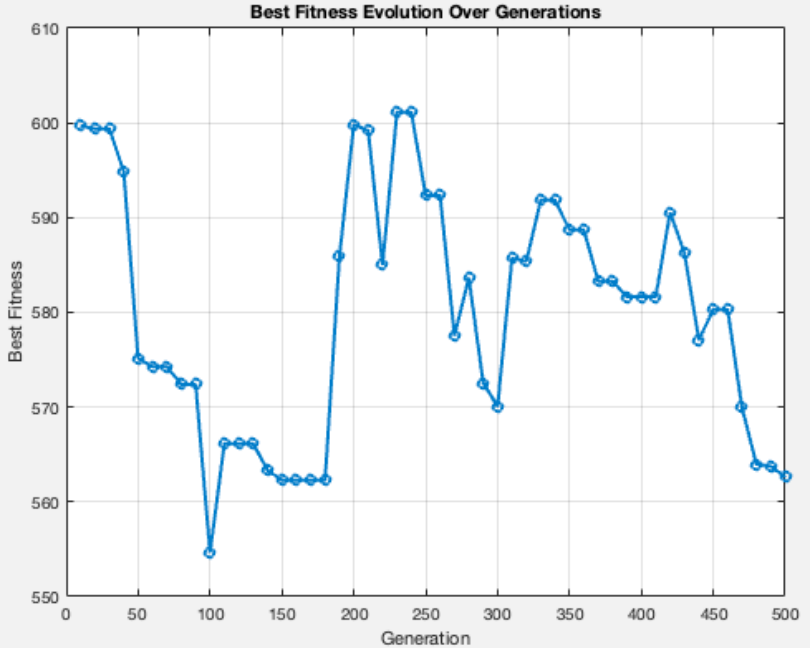




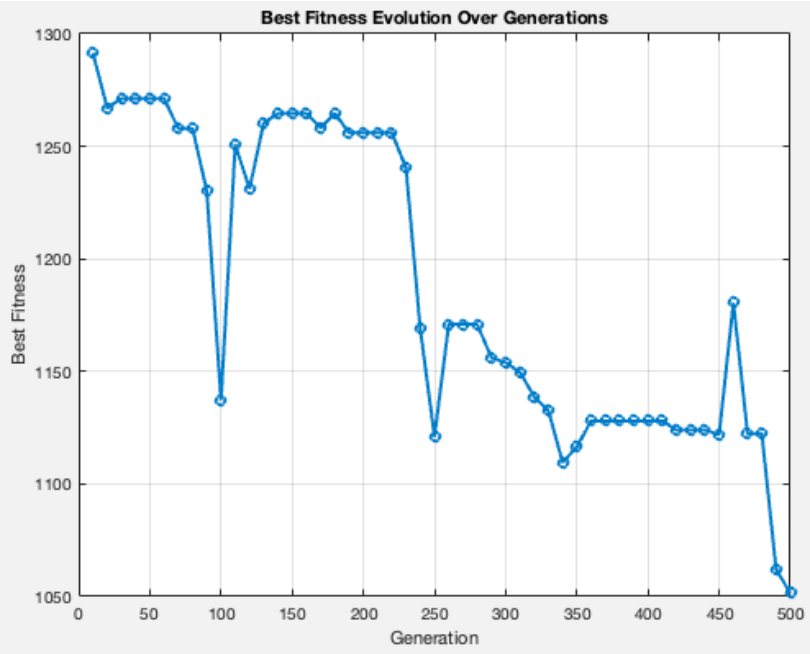
Θέμα 3:

Σε αυτή την ενότητα, εξετάζεται η επίδραση της συνολικής εισερχόμενης ροής V στο αποτέλεσμα της βελτιστοποίησης. Ο στόχος είναι να αναλυθεί η συμπεριφορά του γενετικού αλγορίθμου και η προσαρμοστικότητα του σε διαφορετικά σενάρια φόρτου κυκλοφορίας.

Για τιμές $V = 0.85V$, δηλαδή $V = 85$:



Για τιμές $V = 1.15V$, δηλαδή $V = 115$:



- **Ανάλυση Διαγραμμάτων**

- Το πρώτο διάγραμμα ξεκινά με $fitness \approx 1300$ και καταλήγει κοντά στο 1050.
- Το δεύτερο διάγραμμα ξεκινά με $fitness \approx 600$ και καταλήγει κοντά στο 550.
- Και στις δύο περιπτώσεις, το τελικό σημείο σύγκλισης είναι σημαντικά χαμηλότερο από το αρχικό, που σημαίνει ότι ο αλγόριθμος πέτυχε βελτίωση στη λύση.

- **Πιθανές Βελτιώσεις**

- Οι αιφνίδιες αυξήσεις στη $fitness$ υποδηλώνουν υπερβολική επίδραση της μετάλλαξης, κάτι που μπορεί να διορθωθεί με μείωση του $mutation\ rate$.
- Η διασταύρωση φαίνεται να έχει μεγάλο αντίκτυπο, και ίσως μια διαφορετική στρατηγική (π.χ. $uniform\ crossover$) να βελτιώσει τα αποτελέσματα.
- Για ακόμα ταχύτερη σύγκλιση, μπορεί να δοκιμαστεί $elitism$, ώστε να μην χάνονται οι πολύ καλές λύσεις σε κάθε γενιά.

- **Συμπεράσματα**

- Ο γενετικός αλγόριθμος συγκλίνει προοδευτικά, αποφεύγοντας παγίδευση σε τοπικά ελάχιστα.
- Οι έντονες διακυμάνσεις δείχνουν εξερεύνηση νέων λύσεων, μερικές από τις οποίες είναι λιγότερο αποδοτικές.
- Η τελική λύση είναι αισθητά καλύτερη από την αρχική, αποδεικνύοντας την επιτυχία του αλγορίθμου.
- Υπάρχουν περιθώρια βελτίωσης στη μετάλλαξη και τη διασταύρωση για πιο σταθερή σύγκλιση.