Πολυτεχνείο Κρήτης

1η Προγραμματιστική Εργασία

Τεχνητή Νοημοσύνη (ΠΛΗ 417)

Νικολαίδης Δημήτρης **2015030100**
Νταουντάκης Σταύρος **2015030037**

Μέρος Α: Πειραματισμός με μεθόδους αναζήτησης

Αλγόριθμος BFS, DFS, A*

Η προσέγγιση τέτοιων προβλημάτων ξεκινάει με την δημιουργία ενός χώρου καταστάσεων μέσα στον οποίο μπορούν να κινηθούν οι αλγόριθμοι. Δημιουργήσαμε μια κλάση *State* η οποία διαθέτει όλες τις απαραίτητες παραμέτρους για την λειτουργία των αλγορίθμων (π.χ. Position, parentNode, τιμές f = g + h συναρτήσεων κλπ). Η κλάση διαθέτει επίσης απαραίτητες συναρτήσεις όπως successorStates() για την εύρεση των κόμβων-παιδιών της κλάσης, isLegal(), goalReached(), isVisited(), generateF() και άλλες τις οποίες χρειάζονται οι αλγόριθμοι για την λειτουργία τους. Πιό συγκεκριμένα :

- **BFS** : Το initialState (Θέση εκκίνησης) τοποθετείται σε μια άδεια ουρά. Έπειτα αφαιρείται απ'την ουρά και εξερευνούνται τα *successorStates()* του. Αυτά προστίθενται στην ουρά και όταν αυτών τα παιδιά εξερευνηθούν, αφαιρούνται από την ουρά. Κάθε state επισκέπτεται μία φορά (*isVisited()*) και η εκτέλεση σταματάει όταν φτάσουμε στον τερματισμό με επαναληπτική διαδικασία.

- **DFS** : Ο τρόπος λειτουργίας είναι παρόμοιος με της BFS με κύρια διαφορά ότι χρησιμοποιούμε Στοίβα αντί για ουρά ώστε όταν προστίθενται succesor States, να εξερευνούνται κατα βάθος.

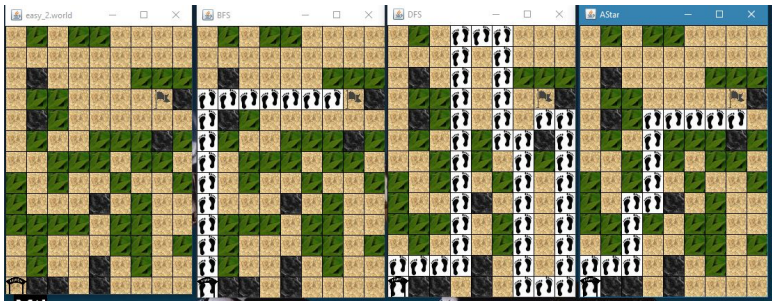
- **A*** : Αυτός ο αλγόριθμος ακολουθεί μονοπάτια με βάση το ποιό παιδί έχει το χαμηλότερο $f(n) = g(n) + h(n)$ όπου g(n) είναι το κόστος από την εκκίνηση εώς τον κόμβο n και h(n) είναι το κόστος από τον κόμβο n εώς τον τερματισμό. Για την συνάρτηση g(n) υπολογίζουμε τα κόστη με βάση το είδος του εδάφους σε κάθε node. Η ευριστική συνάρτηση h(n) μας δίνει ένα υποθετικό κόστος που στην περίπτωση μας ορίζεται ώς η ευκλείδεια απόσταση από τον κόμβο εώς τον τερματισμό. Με άλλα λόγια δημιουργούμε ένα νοητό ορθογώνιο τρίγωνο του οποίου η υποτείνουσα είναι μια ευθεία γραμμή από τον κόμβο εώς τον τερματισμό. Το μήκος της είναι η τιμή της h(n).

  Η εκτέλεση του A* έγινε ώς εξής : Δημιουργούμε δύο άδειες συνδεδεμένες λίστες, την openList και την closedList. Η πρώτη χρησιμοποιείται για την εισαγωγή τών State προς εξερεύνηση ενώ η δεύτερη για την εισαγωγή των φθηνότερων μονοπατιών. Το State εκκίνησης τοποθετείται στην open List, εξερευνούνται τα State-παιδιά του και αφαιρείται από την λίστα. Για κάθε παιδί ελέγχουμε δύο πράγματα. 1ον : Εάν υπάρχει κάποιο πιό φθηνό node που είναι στην ίδια θέση και βρίσκεται εντός στην openList και 2ον : Εάν υπάρχει πιο φθηνό node που είναι στην ίδια θέση και βρίσκεται εντός της closedList. Για να κατανοήσουμε πως μπορεί να συμβεί κάτι τέτοιο θα δώσουμε ένα απλό παράδειγμα. Έστω ότι κοιτάμε το node με θέση [3,4]. Αυτό είναι ένα παιδί που προήλθε από το πατέρα-node στην θέση [3,3]. Προηγουμένως, όμως, είχε

εξερευνηθεί ξανά το node [3,4] αλλα προήλθε απο το [2,4]. Εμείς θέλουμε να κρατήσουμε το πιό φθηνό από τα δύο μονοπάτια που καταλήγει στην θέση [3,4] και για αυτόν τον λόγο χρησιμοποιούμε τους δύο ελέγχους που προαναφέραμε. Εάν λοιπόν από τους ελέγχους προκύψει ότι το παιδί που κοιτάμε είναι πιο φθηνό από ήδη ερευνημένα nodes στην ίδια θέση τότε αυτό προστίθεται στην openList. Όταν ερευνηθούν και ελεγχθούν όλα τα παιδιά του πατέρα-node τότε αυτός προστίθεται στην closedList και προχωράμε επαναληπτικά σε επόμενες εξερευνήσεις των state που είναι στην openList. Η επαναληπτική διαδικασία σταματάει όταν φτάσουμε στο στόχο.

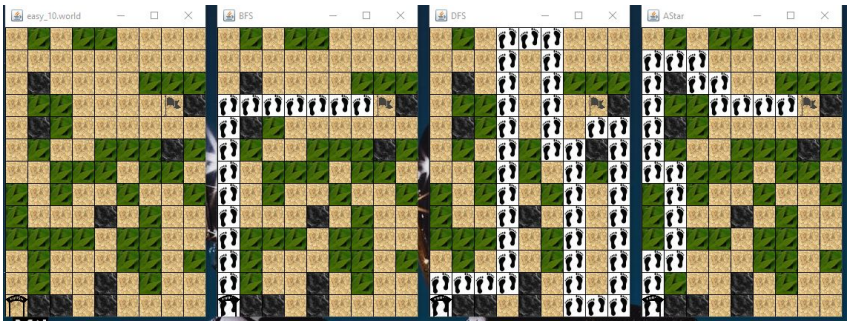
Πειραματικά αποτελέσματα

Κόσμος : *easy_2.world*



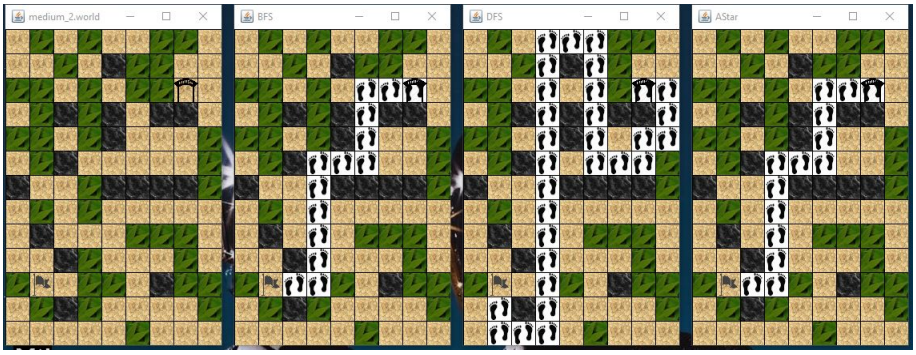
	BFS	DFS	A*
Κόστος αναζήτησης	109	71	483
Κόστος Λύσης	21	57	19

Κόσμος : *easy_10.world*



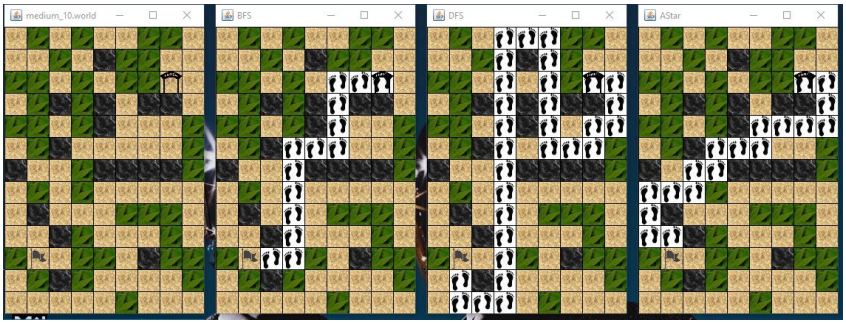
	BFS	DFS	A*
Κόστος αναζήτησης	109	71	549
Κόστος Λύσης	61	177	40

Κόσμος : *medium_2.world*



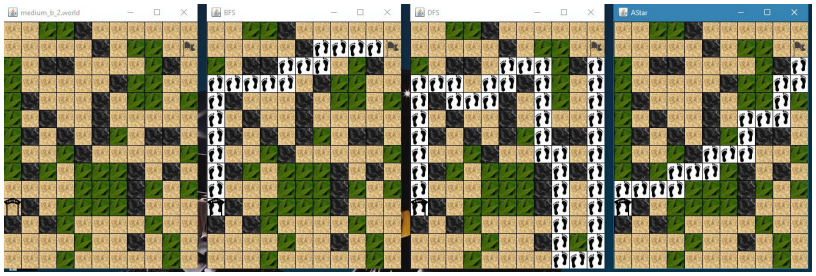
	BFS	DFS	A*
Κόστος αναζήτ.	94	35	184
Κόστος Λύσης	20	43	20

Κόσμος : *medium_10.world*



	BFS	DFS	A*
Κόστος αναζήτ.	94	35	114
Κόστος Λύσης	68	147	45

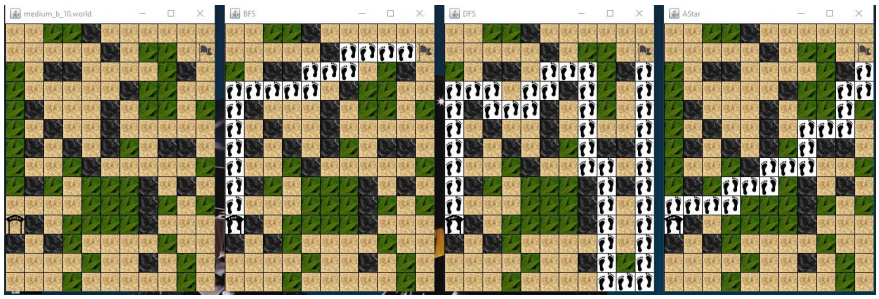
Κόσμος : *medium_b_2.world*



	BFS	DFS	A*
Κόστος αναζήτ.	122	67	246

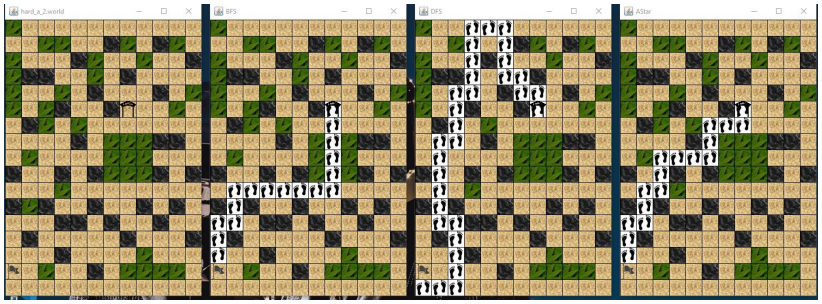
Κόστος Λύσης	27	57	21

Κόσμος : medium_b_*10.world*



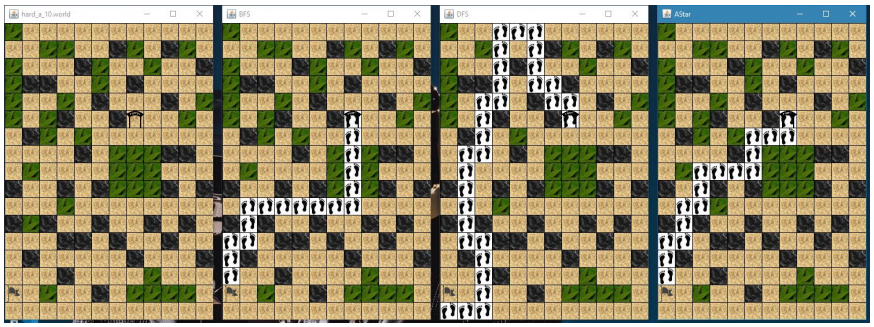
	BFS	DFS	A*
Κόστος αναζήτ.	122	67	184
Κόστος Λύσης	91	169	37

Κόσμος : *hard_a_2.world*



	BFS	DFS	A*
Κόστος αναζήτ.	123	33	92
Κόστος Λύσης	22	40	18

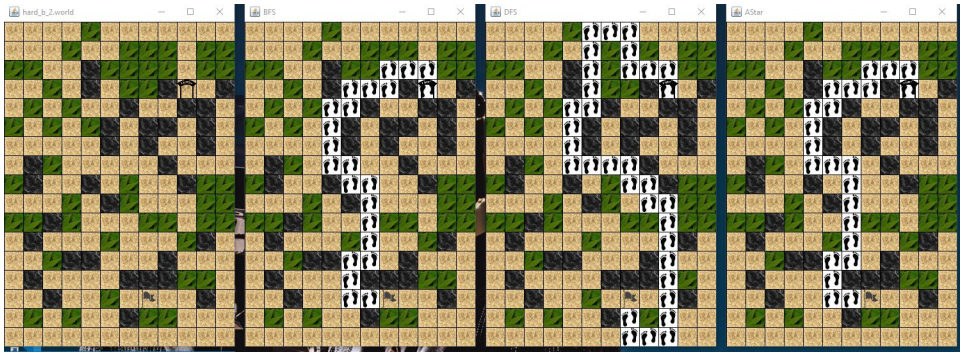
Κόσμος : *hard_a_10.world*



	BFS	DFS	A*
Κόστος	123	33	147

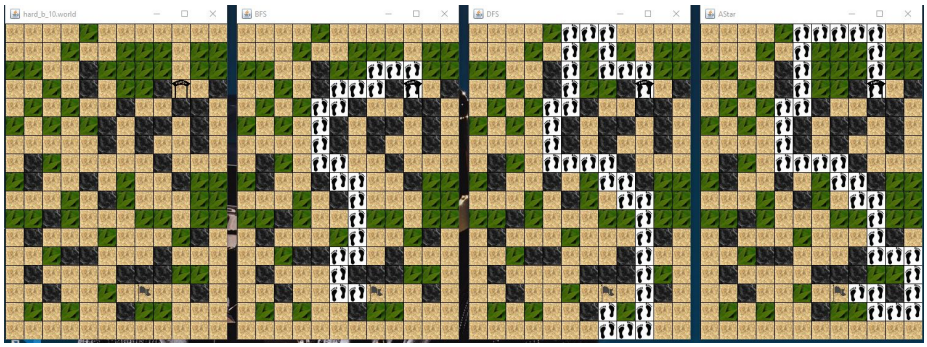
αναζήτησης			
Κόστος Λύσης	62	112	26

Κόσμος : *hard_b_2.world*



	BFS	DFS	A*
Κόστος αναζήτ.	164	60	601
Κόστος Λύσης	31	41	31

Κόσμος : *hard_b_10.world*

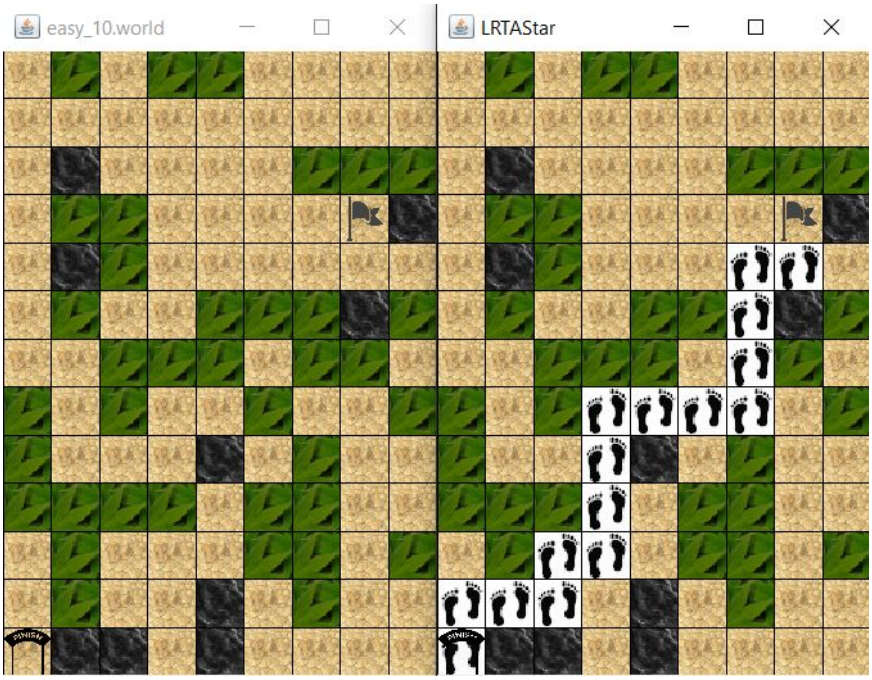


	BFS	DFS	A*
Κόστος αναζήτησης	164	60	352
Κόστος Λύσης	95	105	62

Αλγόριθμος LRTA*

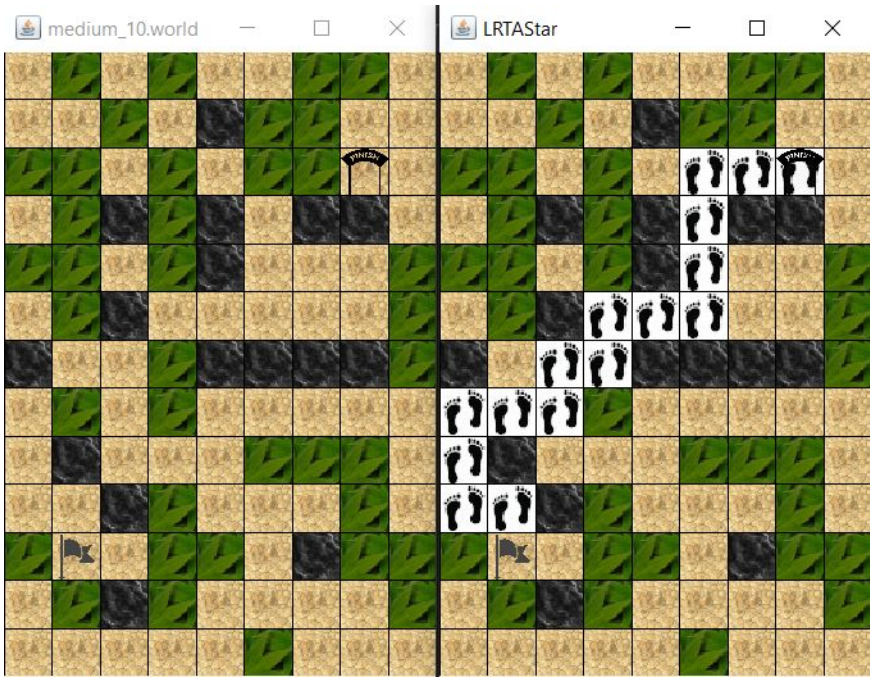
Ο αλγόριθμος LRTA* είναι ένας online αλγόριθμος αναζήτησης ο οποίος εξερευνά το state space ακολουθώντας διαδρομές με χαμηλό κόστος βήματος και ενημερόνοντας το κόστος διαδρομής τον προηγουμένων θέσεων του βάση της εξερευνησης του. Στο συγκεκριμένο πρόβλημα δεν είναι γνωστό το κόστος βήματος g() προτού γίνει εξερεύνηση του κελιού. Για αυτό τον λόγο γίνεται χρήση του ευρηστικού κόστους του κελιού (ευκλείδεια απόσταση από τον στόχο). Έτσι ο αλγόριθμος κατευθύνεται προς τον στόχο χωρίς όμως να ξέρει τα κόστοι των μονοπατιών που δεν ακολουθεί. Σε μια ιδανική περίπτωση θα βρει μονοπάτι στον στόχο επεκτείνοντας μικρό μικρό αριθμό καταστάσεων χωρίς όμως να είναι αυτο βέλτιστο μονοπάτι. Σε κακές περιπτώσεις ο αλγόριθμος βρίσκει τον στόχο αλλά μπαίνει σε πολλά αδιέξοδα και έτσι περνάει από περισσότερες καταστάσεις. Για περιπτώσεις τέτοιες είναι απαραίτητη η χρήση κάποιου αλγορίθμου για να κοπούν αυτές τα κομμάτια του μονοπατιού μπαίνουν σε αδιέξοδα. Επίσης σημαντική λεπτομέρεια είναι πως όταν εξετάζονται οι πιθανές επόμενες καταστάσεις βάση του ευρηστικού κόστους τους αυτό γίνεται με τυχαίο για να μην ακολουθείται πάντα η ίδια διαδρομή σε περίπτωση ισοπαλίας. Μία άλλη σημαντική προσθήκη στον αλγόριθμο θα ήταν η επανάληψη του μερικές φορές για την εύρεση τυχόν καλύτερου μονοπατιού.

Κόσμος : *easy_10.world*



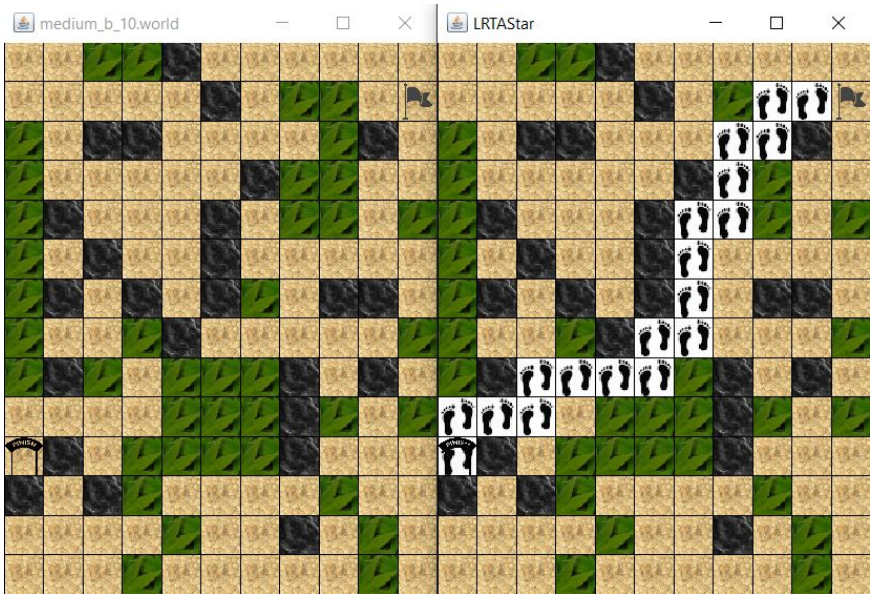
Κόστος αναζήτησης	16
Κόστος Λύσης	61

Κόσμος : *medium_10.world*



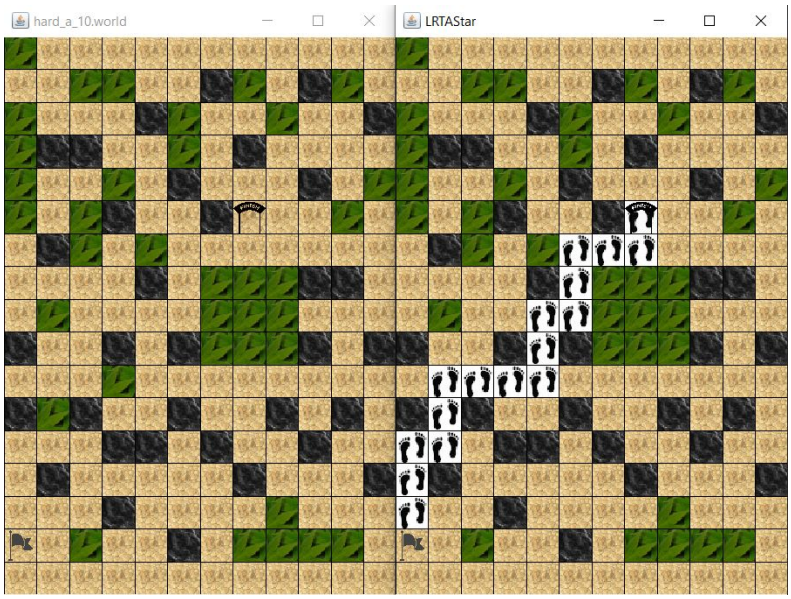
Κόστος αναζήτησης	16
Κόστος Λύσης	52

Κόσμος : *medium_b_10.world*



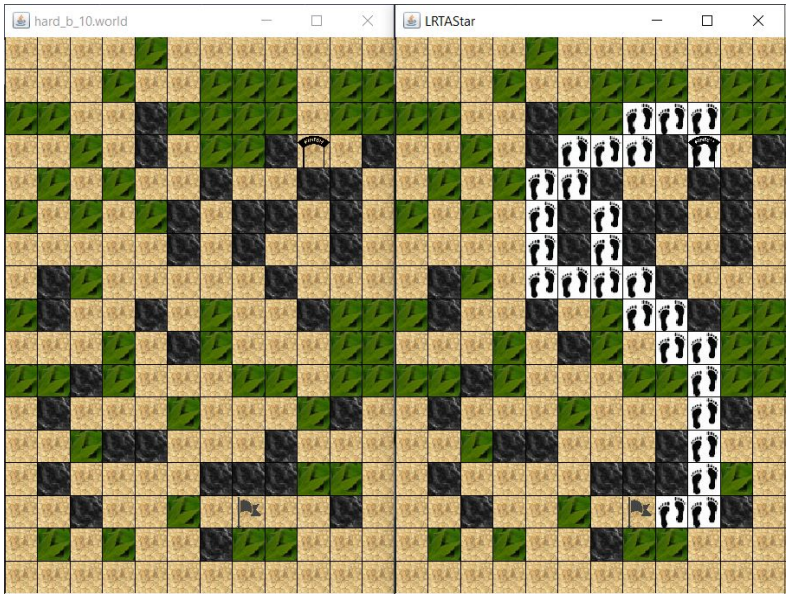
Κόστος αναζήτησης	19
Κόστος Λύσης	91

Κόσμος : *hard_a_10.world*



Κόστος αναζήτησης	17
Κόστος Λύσης	35

Κόσμος : *hard_b_10.world*



Κόστος αναζήτησης	29
Κόστος Λύσης	92

Μέρος Β: Υλοποίηση Γενετικού Αλγορίθμου για WHPP

Ο γενετικός αλγόριθμος βασίζεται στην εξέλιξη ενος πληθυσμού μέσω κάποιων επαναληπτικών διαδικασιών που παρομοιάζουν την *φυσική επιλογή*. Ακολουθώντας τον παρεχόμενο ψευδοκώδικα ενός γενετικού αλγορίθμου για το πρόβλημα WHPP στόχος είναι να διαμορφώσουμε ένα πρόγραμμα εργασίας για μια εταιρία έτσι ώστε να ικανοποιούνται πλήρως οι αυστηροί περιορισμοί και να ικανοποιούνται όσο το δυνατόν καλύτερα οι χαλαροί περιορισμοί. Παρακάτω θα εξηγήσουμε τον τρόπο που προσεγγίσαμε το πρόβλημα για τα διάφορα μέρη του αλγορίθμου.

Αρχικός Πληθυσμός

Ένας αρχικός πληθυσμός θεωρητικά είναι ένας εντελώς τυχάιος πληθυσμός. Στην πράξη όμως, πληθυσμοί με απόλυτη τυχαιότητα αδυνατούν να ικανοποιήσουν τα hard Constraints και ώς αποτέλεσμα είναι αδύνατον να εγκριθεί ένα τμήμα αυτού για την εκκίνηση του αλγορίθμου. Για αυτόν τον λόγο διαμορφώσαμε έναν αρχικό πληθυσμό με κάποια *μερική* τυχαιότητα. Φροντίζουμε δηλαδή να ικανοποιούνται τα hard Constraints των βαρδιών για κάθε μέρα αλλά με πλήρη τυχαιότητα για το σε ποιον υπάλληλο ανατίθεται η κάθε βάρδια.

Hard/Soft Constraints

Με τις συναρτήσεις *checkHardConstraints()* και *penaltyFunction()* ελέγχουμε την ικανοποίηση των hardConstraint επιστρέφοντας True ή False για κάθε "χρωμόσωμα" του πληθυσμού και το κόστος (penalty) του κάθε χρωμοσώματος που ικανοποιεί τα hard Constraints αντίστοιχα. Η *penaltyFunction()* εσωτερικά της απλά αυξάνει το penalty για κάθε *soft* παράβαση σύμφωνα με τον δοσμένο πίνακα της εκφώνησης. Επιπλέον, κάποια από τα soft constraints ίσως παραβιάζονται παραπάνω από μία φορά (π.χ. "Βάρδια-Εργασία-Βάρδια") και έχουμε φροντίσει να προστίθενται τα penalty για όσες φορές παραβιάζονται.

Selection

Με την συνάρτηση selectWorthyChromosomes() υλοποιείται η επιλογή των χρωμοσωμάτων που θα προωθηθούν στη συνέχεια. Ο αλγόριθμος επιλογής που υλοποιείται είναι συνδυασμός ελιτισμού και Roulette Wheel. Αρχικά δηλαδη τα δύο καλύτερα χρωμοσώματα περνάνε αυτόματα στην επόμενη φάση και αφαιρούνται από τον πληθυσμό. Από τα απομένοντα χρωμοσώματα δημιουργείται ένα roulette wheel και επιλέγονται άλλα population-2 χρωμοσώματα , όπου population το μέγεθος του πληθυσμού που καλείται η συνάρτηση, βάση του πόσο μεγάλο είναι το penalty τους με μεγαλύτερη πιθανότητα σε αυτά με τον μικρότερο πληθυσμό. Στην υλοποίηση του αλγορίθμου η πιθανότητας επιλογής στοιχείων δεν φάνηκε αναγκαία καθώς το τμήμα roulette wheel εισάγει την τυχαία επιλογή χρωμοσωμάτων εξ' ορισμού.

Crossover

Το δεύτερο στάδιο του γενετικού αλγορίθμου είναι η Διασταύρωση (Crossover). Υλοποιήσαμε δύο μεθόδους Διασταύρωσης : την *Uniform* και την *Two Point.* Η *Uniform Crossover* παίρνει δύο χρωμοσώματα-γονείς και με ίση πιθανότητα αποφασίζει αν θα ανταλλάξει η όχι ένα στοιχείο του πίνακα (*gene*). Η *Two Point* μέθοδος επιλέγει τυχαία δύο εύρη τιμών για τις γραμμές οι οποίες θα ανταλλαχθούν μεταξύ των δύο γονέων-χρωμοσωμάτων. Για διευκρίνιση, εάν π.χ επιλεχθούν τα σημεία 8, 23, θα γίνουν εναλλαγές γραμμών για το εύρος [1,8] και [23,30]. Ο κάθε αλγόριθμος εφαρμόζεται στον πληθυσμό ανά δυάδες με μία πιθανότητα pcross.

Mutation

Στο στάδιο του mutation υλοποιήθηκαν δύο αλγόριθμοι για την μετάλλαξη των χρωμοσωμάτων. Ο αλγόριθμος swap κατά τον οποίο για κάθε ημέρα με πιθανότητα 0.33 επιλέγουμε δύο τυχαίους υπαλλήλους και αλλάζουμε τις βάρδιες μεταξύ τους. Για τον αλγόριθμο bit-flip η μετάλλαξη έγινε επιλέγοντας για κάθε ημέρα με πιθανότητα 0.33 ένα υπάλληλο και του δίνουμε μια τυχαία τιμή βάρδιας η άδεια από τις επιτρεπτές. Η συνάρτηση μετάλλαξης προκαλεί μετάλλαξη σε ένα χρωμόσωμα με πιθανότητα pmut.

Termination Criteria

Τα κριτήρια τερματισμού που αποφασίσαμε είναι τα εξής :

- Το μέσο πέναλτι παραμένει σταθερό για διαδοχικές γενιές. Συγκεκριμένα, εάν δούμε ότι για 4 γενιές το penalty παραμένει σχετικά σταθερό τότε κρίνουμε ότι ο αλγόριθμος δεν βελτιώνει το πρόβλημα και άρα τερματίζει.

- Ο πληθυσμός έχει μειωθεί πολύ. Εάν ο πληθυσμός είναι πολύ μικρός τότε παύει να λειτουργεί αποδοτικά ο γενετικός αλγόριθμος επομένως τερματίζει.

- Φτάσαμε στις μέγιστες επαναλήψεις. Αυτός ο αριθμός αποφασίζεται από τον χρήστη κατα την εκτέλεση του προγράμματος.

Set 1 : pop = 5000, itermax = 20, pcross = 0.8, pmut = 0.2
Set 2 : pop = 2000, itermax = 15, pcross = 0.4 , pmut = 0.1

	Uniform Set 1	TwoPoint Set 1
Swap Set 1	*55989.6*	*55467.4*
Bit Flit Set 1	*56024.6*	*55804.6*

	Uniform Set 2	TwoPoint Set 2
Swap Set 2	*57278.4*	*56361.2*
Bit flit Set 2	*57259.2*	*55646.4*

Συμπεράσματα

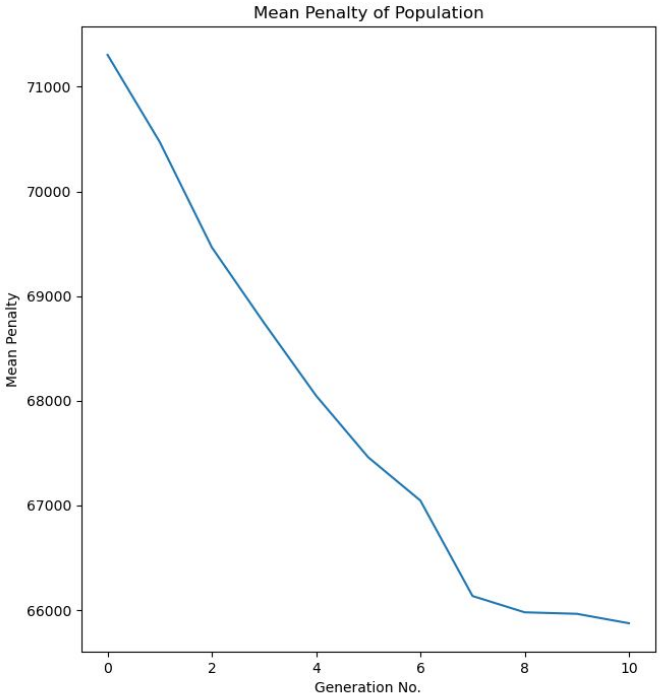
Μετά από πειραματισμούς με διάφορους συνδυασμούς μεθόδων Crossing και Mutation καθώς και με τις πιθανότητες εκτέλεσης τους παρατηρούμε ότι ο γενετικός αλγόριθμος δεν μπορεί να εγγυηθεί δραματική βελτίωση των αποτελεσμάτων. Παρ' όλο που η μέση τιμή των πέναλτυ για κάθε γενιά πληθυσμού φθίνει, δεν μειώνεται αρκετά ώστε να καταλήξουμε σε ένα ικανοποιητικό πρόγραμμα ανάθεσης βαρδιών.

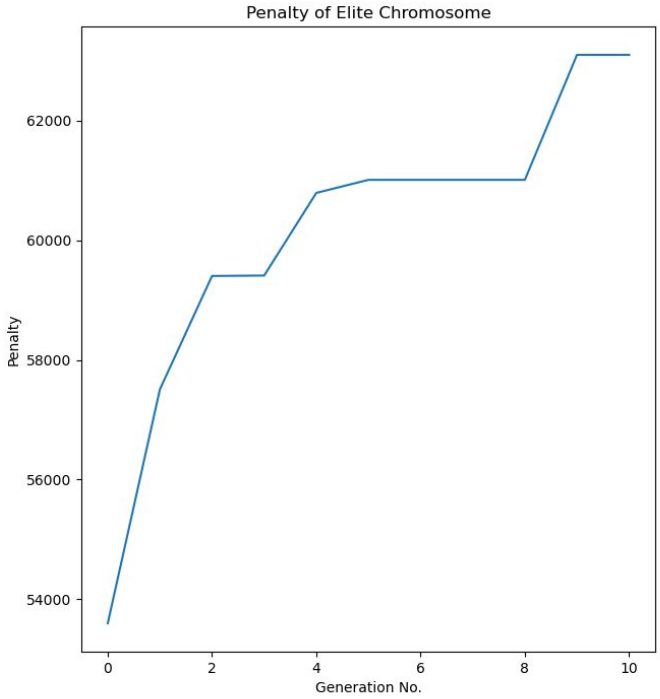
Σαν πρώτη σκέψη, θα μπορούσαμε να "κατηγορήσουμε" το μέγεθος του αρχικού πληθυσμού, ότι δηλαδή δεν δίνονται περιθώρια στον αλγόριθμο να βελτιωθεί διότι ο πληθυσμός δεν είναι αρκετά μεγάλος. Παρ' όλα αυτά, με πειραματισμούς που κάναμε και με πληθυσμούς άνω των 20,000 είδαμε ότι δεν έχουμε μεγάλη βελτίωση στις τελευταίες γενιές και για αυτό καταλήξαμε να πειραματιζόμαστε με πιό μικρούς πληθυσμούς.

Παρατηρήσαμε επιπλέον ότι ο διαφορετικοί συνδυασμοί μεθόδων έχουν διαφορετικές καμπύλες βελτίωσης και σίγουρα μπορούμε να ξεχωρίσουμε ποιό κατάλληλους συνδυασμούς, όπως π.χ. *Two-Point Crossover* και *Swap Mutation.* Ταυτόχρονα διακρίναμε ότι η bit-flip mutation λόγω της μεγάλης τυχαιότητας της παράγει χρωμοσώματα που κατά μεγάλη πιθανότητα θα αποτύχουν τον έλεγχο των hard constraints με αποτέλεσμα αυτού να μειώνεται πιο γρήγορα ο πληθυσμός. Έτσι δίνονται λιγότερες πιθανότητες στον αλγόριθμο να βρεί καλύτερα αποτελέσματα και δεν εισάγει χρήσιμες μεταλλάξεις στον πληθυσμό.

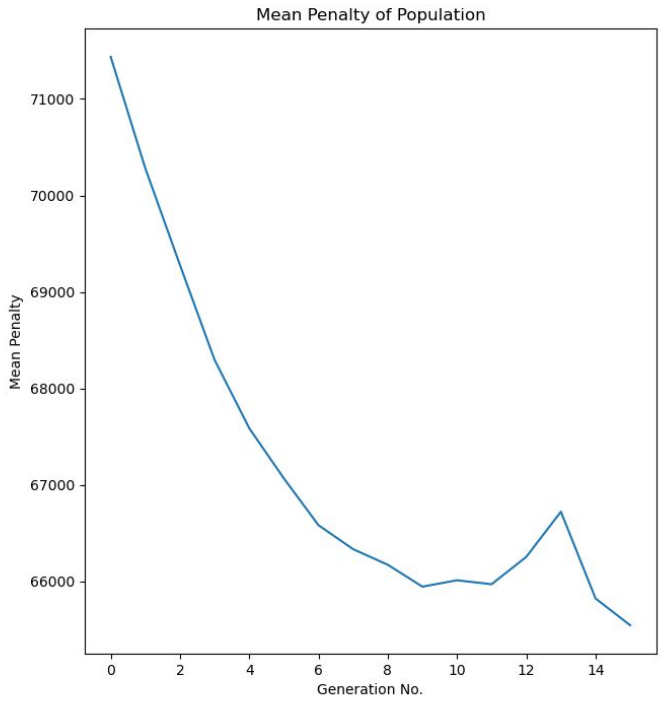
Παρ' όλα αυτά κανένας απο τους διαφορετικούς συνδυασμούς δείχνει να παράγει κάποιο "χρωμόσωμα" με βαθμολογία καλύτερη από το εύρος των ~55,000. Αυτό μπορεί να οφείλεται είτε σε πολύ αυστηρά termination criteria η στην μέθοδο επιλογής τον χρωμοσωμάτων. Γενικά πιστεύουμε ότι θα είχε ενδιαφέρον η υλοποίηση και εξέταση ενός άλλου αλγορίθμου επιλογής για το συγκεκριμένο πρόβλημα.

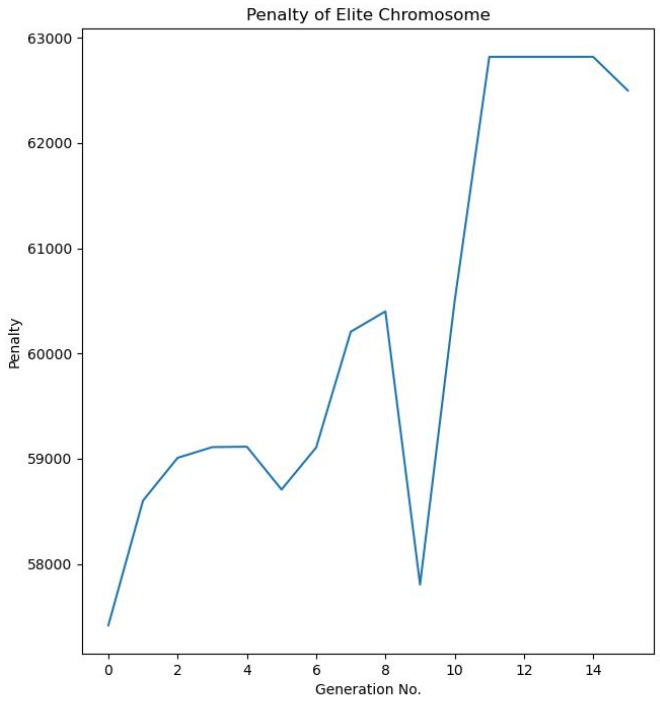
Results with pop=5000 for pcorss=0.8, corss_type=two-point, pmut=0.2, mut_type=bit-flip



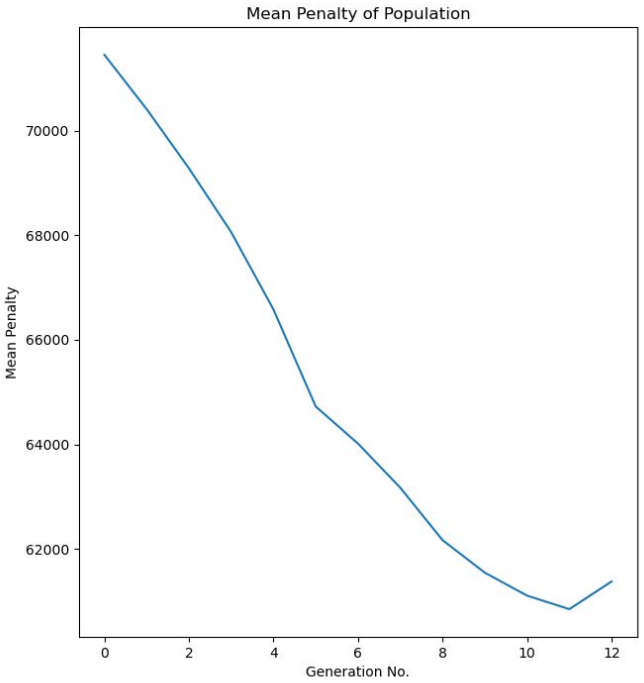


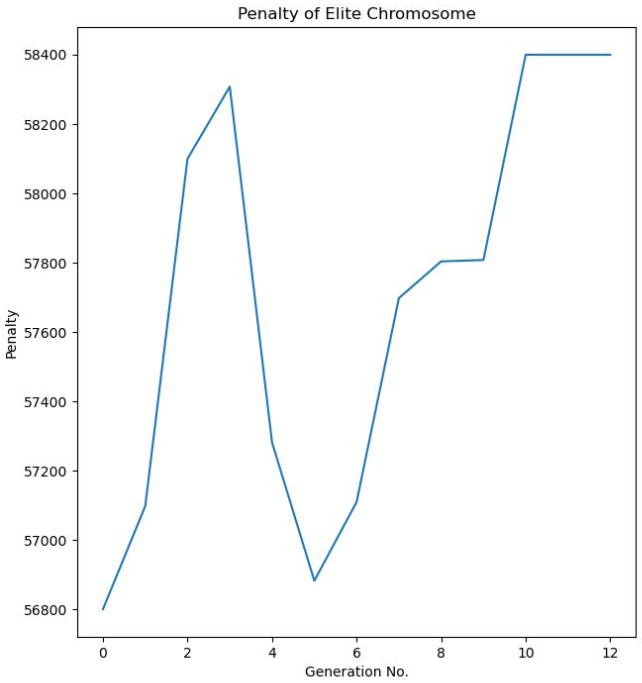
Results with pop=5000 for pcorss=0.8, corss_type=two-point, pmut=0.2, mut_type=swap



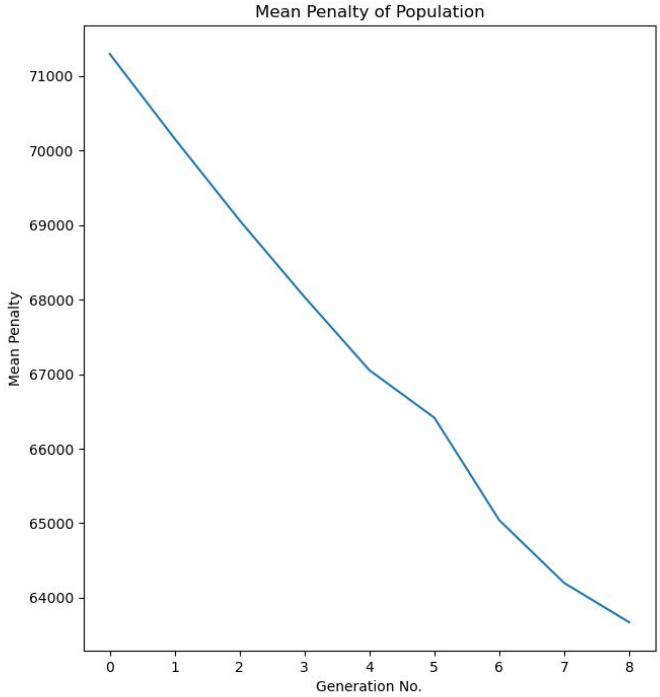


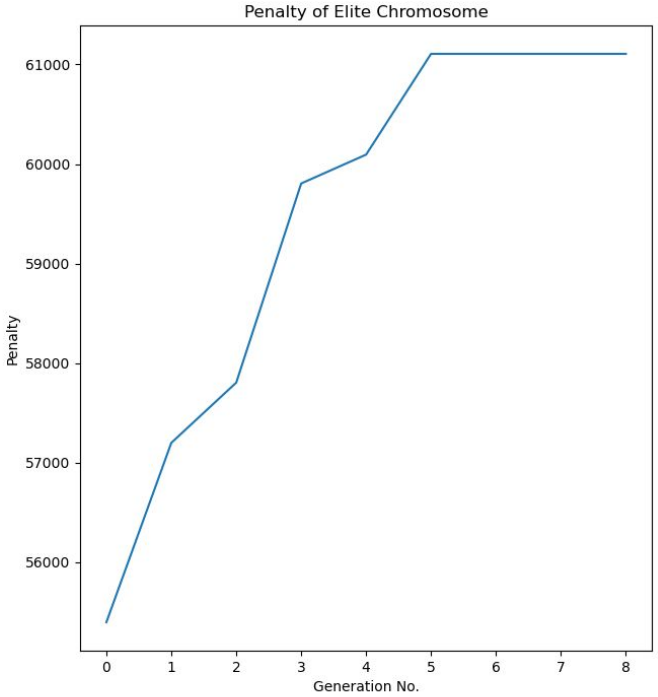
Results with pop=5000 for pcorss=0.8, corss_type=uniform, pmut=0.2, mut_type=swap



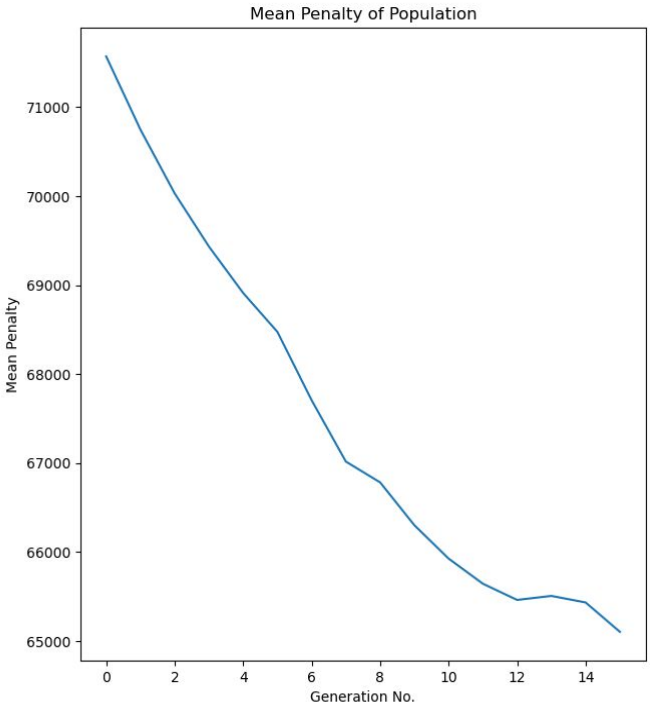


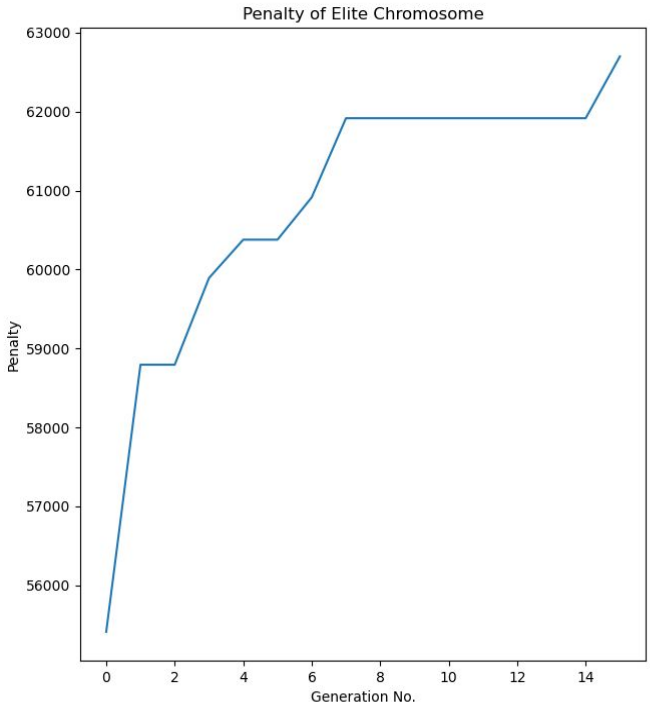
Results with pop=5000 for pcorss=0.8, corss_type=uniform, pmut=0.2, mut_type=bit-flip



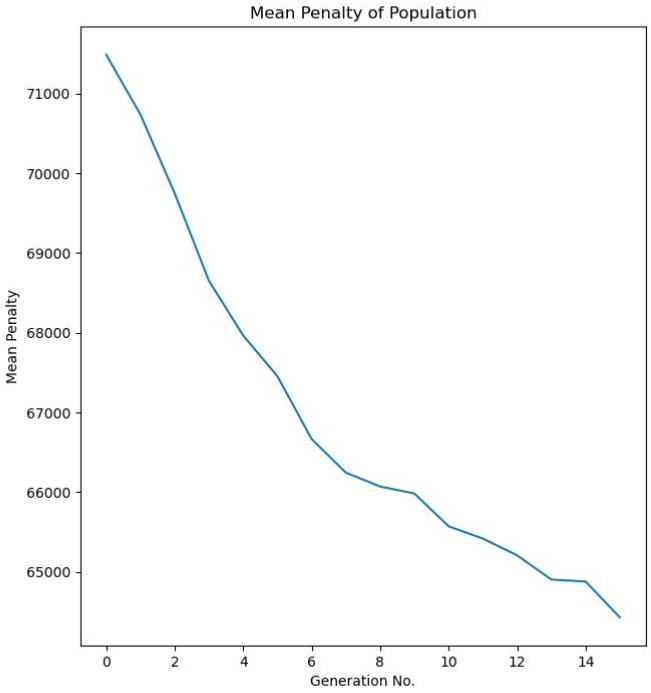


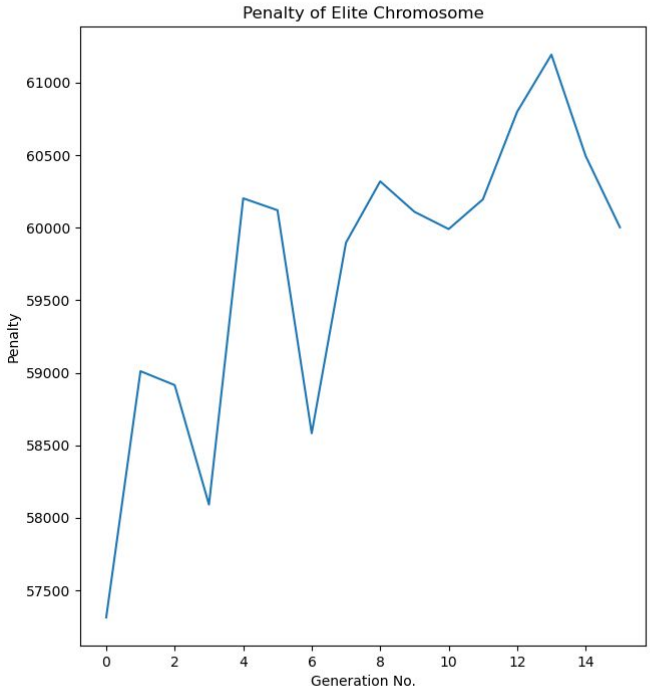
Results with pop=2000 for pcorss=0.4, corss_type=two-point, pmut=0.1, mut_type=bit-flip



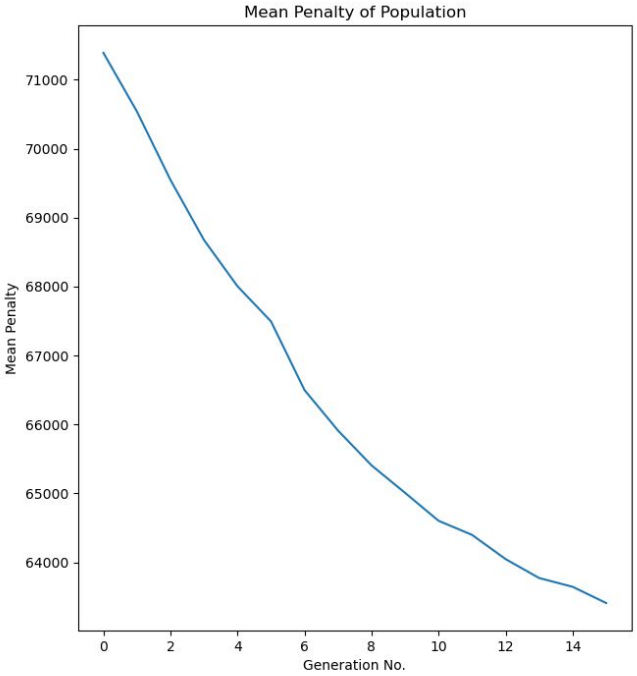


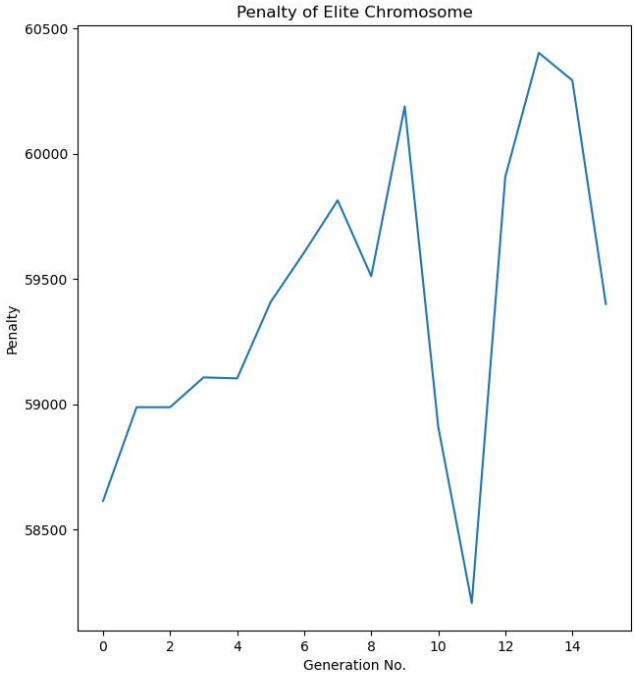
Results with pop=2000 for pcorss=0.4, corss_type=uniform, pmut=0.1, mut_type=swap





Results with pop=2000 for pcorss=0.4, corss_type=two-point, pmut=0.1, mut_type=swap





Results with pop=2000 for pcorss=0.4, corss_type=uniform, pmut=0.1, mut_type=bit-flip