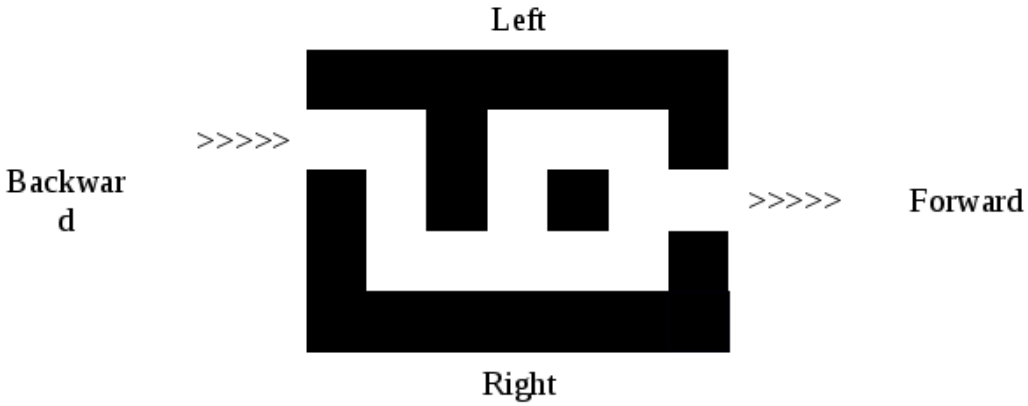
Πανεπιστήμιο Πατρών

Τμήμα Μηχ. Η/Υ & Πληροφορικής

Υπολογιστική Νοημοσύνη

Απαλλακτική Εργασία

Όνομα: Σπύρος Καφτάνης
ΑΜ: 5542
Εξάμηνο 8ο



1. Για να λύσουμε το συγκεκριμένο πρόβλημα με την βοήθεια των γενετικών αλγορίθμων, πρέπει αρχικά να ορίσουμε την αναπαράσταση των διαθέσιμων κινήσεων. Οι διαθέσιμες κινήσεις είναι οι:

- Μετακίνηση μπροστά κατά 1 κουτί
- Μετακίνηση πίσω κατά 1 κουτί
- Μετακίνηση δεξιά κατά 1 κουτί
- Μετακίνηση αριστερά κατά 1 κουτί

Μπορούμε να κωδικοποιήσουμε αυτές τις επιλογές με πολλούς τρόπους▪ εδώ θα προτιμήσουμε απλά να τους αριθμήσουμε με ακέραιους αριθμούς ξεκινώντας από το 1. Άρα έχουμε:

- 1 = Μετακίνηση μπροστά κατά 1 κουτί
- 2 = Μετακίνηση πίσω κατά 1 κουτί
- 3 = Μετακίνηση δεξιά κατά 1 κουτί
- 4 = Μετακίνηση αριστερά κατά 1 κουτί

με το πεδίο ορισμού να είναι προφανώς το D:{[1,4]}.

Μια άλλη επιλογή κωδικοποίησης θα ήταν η κωδικοποίηση με δυαδικά ψηφία, αλλά μιας και οι δυνατές κινήσεις είναι λίγες, δεν υπάρχει κάποιος ιδιαίτερος λόγος να γίνει αυτό.

Κατά την εκτέλεση του γενετικού αλγορίθμου δημιουργείται αρχικά ένας αρχικός πληθυσμός, ο οποίος περιέχει όσα άτομα έχουμε ορίσει (POPSIZE). Τα άτομα αυτά αντιπροσωπεύουν *κινήσεις* και αρχικοποιούνται τυχαία. Ένα μέλος του πληθυσμού που αρχικοποιείται τυχαία θα μπορούσε να είναι το:

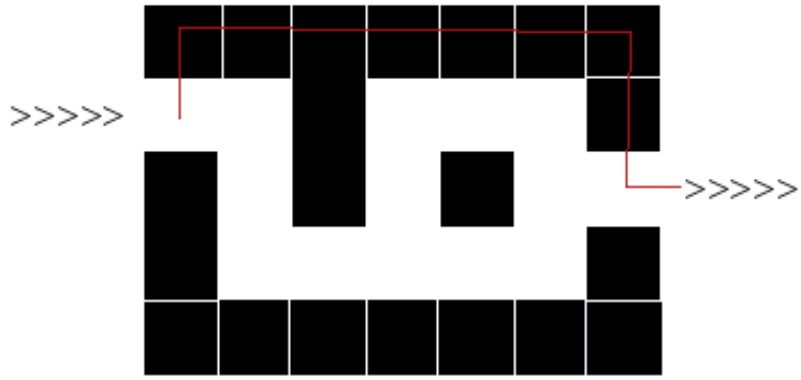
1,1,2,3,4,4,4,3....

Σύμφωνα με την κωδικοποίηση που έχουμε κάνει αυτό το μέλος δοκιμάζει να κάνει πρώτα κίνηση προς τα μπροστά δύο φορές, μετά κίνηση πίσω, μετά δεξιά κτλ. Το μέγιστο μέγεθος του γενότυπου ορίζεται στο πρόγραμμα (MOVE_LIMIT) από εμάς. Μπορούμε όμως να το κάνουμε να επιλέγεται τυχαία σε ένα διάστημα, όπως για παράδειγμα το [3,20].

Μια *συνάρτηση αξιολόγησης* θα μπορούσε να είναι η απόσταση Manhattan, η οποία χρησιμοποιείται σε πολλούς αλγορίθμους ευρετικού χαρακτήρα. Η απόσταση Manhattan δίνεται από τον τύπο:

|x1-x2|+|y1+y2|,
όπου (x1,x2) οι συντεταγμένες της τωρινής θέσης και (y1,y2)
οι συντεταγμένες του στόχου.

Πρακτικά είναι η κάθετη απόσταση μέχρι το στόχο, στην οποία αγνοούμε τα εμπόδια. Για παράδειγμα η απόσταση Manhattan από κουτί της έναρξης μέχρι το κουτί του τερματισμού είναι 9.



Εικόνα 1: Απόσταση Manhattan από την αρχή στο τέλος.

Μια άλλη δυνατή επιλογή θα ήταν η ευκλείδεια απόσταση. Εδώ θα προτιμήσουμε τη Manhattan, μιας και δίνει ακέραιες τιμές.

Κάθε φορά που ένα άτομο του πληθυσμού φτάνει στο σημείο της αξιολόγησης, θα εφαρμόζεται η συνάρτηση αξιολόγησης στο τελευταίο κουτί της διαδρομής. Αυτή η μετρική μας δείχνει δηλαδή το *πόσο κοντά φτάσαμε.* Το συγκεκριμένο πρόβλημα, πρόκειται για ένα πρόβλημα *ελαχιστοποίησης*, μιας η συνάρτηση αξιολόγησης δίνει μικρότερη τιμή, όσο πιο κοντά είμαστε στον στόχο και 0 όταν τον φτάσουμε.

Επιπλέον, για κάθε φορά που το ρομπότ πηγαίνει σε τοίχο, θα πρέπει να προσθέτουμε στη συνάρτηση αξιολόγησης κάποια ποινή (penalty). Η ποινή αυτή θα πρέπει να είναι ένας μεγάλος αριθμός, έτσι ώστε να μην προκύψει τελικά διαδρομή που να περιέχει τοίχους. Μια τιμή που μπορούμε να χρησιμοποιήσουμε για ποινή είναι το 100. Ποινή πρέπει να δίνεται κάθε φορά που το ρομπότ περνάει πάνω από τοίχο. Αυτό μπορεί να γίνει δημιουργώντας ένα σύστημα συντεταγμένων και ορίζοντας τις έγκυρες συντεταγμένες. Ή μη έγκυρες (τοίχοι και αυτές που βρίσκονται έξω από τον χάρτη) δίνουν ποινή.

Τέλος, εάν έχουμε επιλέξει μεταβλητό MOVE_LIMIT, πρέπει να επιβραβεύουμε τις διαδρομές που έγιναν σε λιγότερα βήματα. Αυτό μπορεί να γίνει αφαιρώντας την μέγιστη δυνατή διαδρομή που έχουμε ορίσει (20) μείον το πλήθος της διαδρομής. Έτσι όσο μεγαλύτερη διαδρομή έχει γίνει τόσο μικρότερος θα είναι ο αριθμός που θα αφαιρείται από τη συνάρτηση αξιολόγησης. Υπενθυμίζουμε ότι το πρόβλημα είναι πρόβλημα ελαχιστοποίησης, οπότε όσο μικρότερη η συνάρτηση αξιολόγησης, τόσο καλύτερα. Ακόμα η αφαίρεση αυτή θα πρέπει να γίνεται μόνο όταν βρισκόμαστε σε μία διαδρομή που οδηγεί στο τέρμα γιατί, διαφορετικά μια διαδρομή που με που έκανε μόλις 3 βήματα θα μπορούσε να χαρακτηριστεί καλύτερη από μία που έφτανε στο τέρμα.

Συνεπώς η συνάρτηση αξιολόγησης είναι:

fitness =$|x_{n1}-x_{n2}|+|y_1+y_2|$,
όπου (x_{n1},x_{n2}) οι συντεταγμένες που καταλήγει η διαδρομή.

Αν η διαδρομή περάσει από απαγορευμένη περιοχή:
fitness=fitness+100

Αν fitness=0
fitness=fitness-(20-πλήθος_βημάτων)

Η ελάχιστη τιμή που μπορεί να φτάσει η συνάρτηση στον συγκεκριμένο χάρτη είναι -11 και δίνει την καλύτερη διαδρομή.

Κατά την διαδικασία της *επιλογής* υπολογίζεται η συνάρτηση αξιολόγησης για κάθε χρωμόσωμα. Στη συνέχεια κατασκευάζεται μια ρουλέτα (roulette wheel) και υπολογίζεται η πιθανότητα επιλογής κάθε χρωμοσώματος που είναι ίση με τη τιμή της συνάρτησης του κάθε μέλους δια το άθροισμα των τιμών αξιολόγησης όλων. Έπειτα υπολογίζονται οι αθροιστικές πιθανότητες και παράγονται POP_SIZE τυχαίοι αριθμοί στο διάστημα [0,1]. Για κάθε αριθμό, αν είναι μικρότερος από το q_1 επιλέγουμε το πρώτο χρωμόσωμα v_1 να περάσει, αλλιώς επιλέγουμε το v_i $2 \leq i \leq POPSIZE$ έτσι ώστε $q(i-1) < r \leq q(i)$. Κάποια μέλη μπορούν να περάσουν και δύο φορές στον νέο πληθυσμό.

Κατά τη διαδικασία της *διασταύρωσης* αρχικά επιλέγουμε από πριν μία πιθανότητα διασταύρωσης p_c, που θεωρητικά μας δείχνει το ποσοστό των μελών του πληθυσμού που θα διασταυρωθούν. Παράγονται τυχαία POP_SIZE αριθμοί στο [0,1], ένας για κάθε χρωμόσωμα. Αν ο αριθμός αυτός είναι μικρότερος του p_c, τότε το συγκεκριμένο χρωμόσωμα επιλέγεται για διασταύρωση. Στη συνέχεια τα χρωμοσώματα ζευγαρώνονται ανά δύο (αν το πλήθος τους είναι περιττό επιλέγεται τυχαία και κάποιο άλλο). Επιλέγεται τυχαία μία θέση pos, που είναι το σημείο διασταύρωσης. Τα ψηφία που βρίσκονται μετά τη θέση pos στο ένα μέλος αντικαθίστανται με τα αντίστοιχα του άλλου και ανάποδα. Αν πχ είχαμε τις διαδρομές:

1,1,2,|2,3,4 και 2,2,4,|1,1,1

μετά τη διασταύρωση στο σημείο που βρίσκεται το pipe θα είχαμε:

1,1,2,1,1,1 και 2,2,4,2,3,4.

Στη *μετάλλαξη* τέλος, με μία πιθανότητα, που συνήθως είναι μικρή ,αλλάξει τυχαία κάποιο ψηφίο. Εδώ, θα μπορούσαμε εκεί που επιλέχθηκε να γίνει η μετάλλαξη να αλλάξαμε το ψηφίο τυχαία σε κάποιο άλλο από τα τρία διαθέσιμα.

2. Στον γενετικό προγραμματισμό, με τη χρήση των γενετικών τελεστών δημιουργείται ένας αλγόριθμος ο οποίος λύνει το πρόβλημα. Για να φτάσουμε εκεί, πρέπει να ορίσουμε με σαφήνεια το πρόβλημα, δίνοντας την αναπαράσταση, τα τερματικά, τις συναρτήσεις και τη συνάρτηση αξιολόγησης.

Πριν όμως πρέπει να κάνουμε κάποιες υποθέσεις.
Υποθέτουμε λοιπόν ότι το ρομπότ δεν "κοιτάει" προς τα κάπου, δηλαδή δεν χρειάζεται να στρίψει πριν προχωρήσει. Μπορεί να πάει Forward, Right, Left και Backward με τις φορές αυτές να είναι πάντα όπως φαίνονται στο εξώφυλλο. Επίσης το ρομπότ μπορεί να ελέγχει γύρω του προς τις τέσσερις κατευθύνσεις για το αν υπάρχουν εμπόδια, όταν χρειάζεται.

Μπορούμε τώρα να ορίσουμε την αναπαράσταση ξεκινώντας από τις συναρτήσεις.

Συναρτήσεις:

- if_wall_forward: Εάν στο forward κουτί, υπάρχει τοίχος, τότε εκτελείται το πρώτο παιδί (αριστερά στο δέντρο), αλλιώς το δεύτερο.
- if_wall_right: Αντίστοιχα για εμπόδιο δεξιά.
- if_wall_left: Αντίστοιχα για εμπόδιο αριστερά.
- Progn-2: Πρώτα εκτελείται το αριστερό παιδί και μετά το δεξί. (Μπορεί να μη χρησιμοποιηθεί).

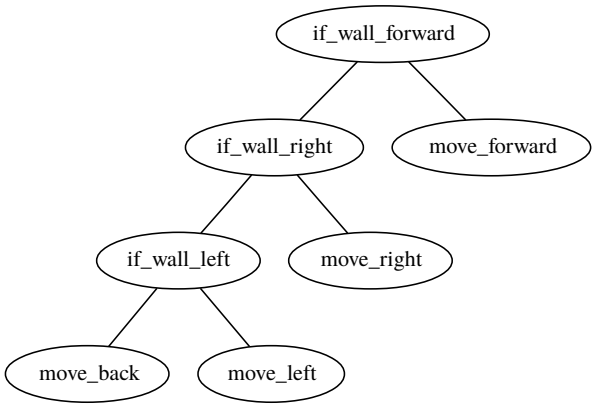
Τερματικά:

- Move forward 1 box
- Move right 1 box
- Move left 1 box
- Move backward 1 box

Ένας αλγόριθμος-λύση που θα μπορούσε να προκύψει από τη διαδικασία είναι ο παρακάτω:

```
if_wall_forward {
  if_wall_right {
    if_wall_left
      Move_Backwards()
    else
      Move_Left()
  }
  else
     Move_Right()
}
else
   Move_forward()
```

Η λύση αυτή είναι ενδεικτική και δημιουργήθηκε πριν τη δημιουργία του κώδικα. Παρ' όλα αυτά είναι μια λύση που ίσως τελικά προκύψει, μιας και βρίσκει την έξοδο από τη συντομότερη διαδρομή. Το δυαδικό δέντρο το οποίο θα αναπαριστούσε αυτόν τον αλγόριθμο φαίνεται στην εικόνα 2.



Εικόνα 2: Δυαδικό δέντρο αναπαράστασης αλγορίθμου

Συνάρτηση Αξιολόγησης:

Θέλουμε το πρόγραμμα να αξιολογεί τον παραγόμενο αλγόριθμο λαμβάνοντας υπ' όψιν του το πόσο κοντά στην έξοδο έφτασε και το πόσο μεγάλο ήταν το δέντρο. Προφανώς όσο πιο κοντά στο στόχο έφτασε τόσο το καλύτερο και όσο πιο μικρό ήταν το δέντρο τόσο καλύτερη είναι η λύση. Επιπλέον θέλουμε να δίνουμε μια αρκετά μεγάλη ποινή για τις διαδρομές που βγαίνουν εκτός ορίων(τοίχοι ή έξω από το χάρτη). Μια τέτοια ποινή θα μπορούσε να είναι 100 πόντοι για κάθε στοιχείο που είναι εκτός. Μια τέτοια συνάρτηση θα μπορούσε να είναι η παρακάτω:

Fitness=-distance-100*(moves_out_of_boarders)-length

όπου length εννοούμε το μέγεθος του αλγορίθμου και distance την απόσταση *ευκλείδεια απόσταση* από το στόχο. Η απόσταση είναι 0 εάν έχουμε φτάσει στον στόχο, διαφορετικά μπορεί να είναι αρκετά μεγάλη σαν αριθμός μιας και μιλάμε για ευκλείδεια απόσταση. Η αύξηση οποιασδήποτε εκ των τριών τιμών (distance, moves_out_of_boarders, lenght), χειροτερεύει τη Fitness, γι αυτό το λόγο υπάρχει το μείον μπροστά από αυτές τις τιμές. Το πρόβλημα παραμένει πρόβλημα *βελτιστοποίησης*, απλά δουλεύει με αρνητικούς αριθμούς. Στη περίπτωση όπου βρεθεί ο σωστός αλγόριθμος, δηλαδή στη καλύτερη περίπτωση, το Fitness είναι -9, δηλαδή όσο το μήκος του αλγορίθμου (length) μιας και το distance και το moves_out_of_boarders πρέπει να είναι 0. Η χειρότερη περίπτωση είναι ένας μεγάλος αρνητικός αριθμός αν λάβουμε υπ' όψιν μας μετράμε ευκλείδεια απόσταση.
Αν θέλαμε να δουλεύαμε με *θετικούς αριθμούς* θα μπορούσαμε να προσθέσουμε έναν μεγάλο θετικό (πχ 5000) στην παραπάνω έκφραση.

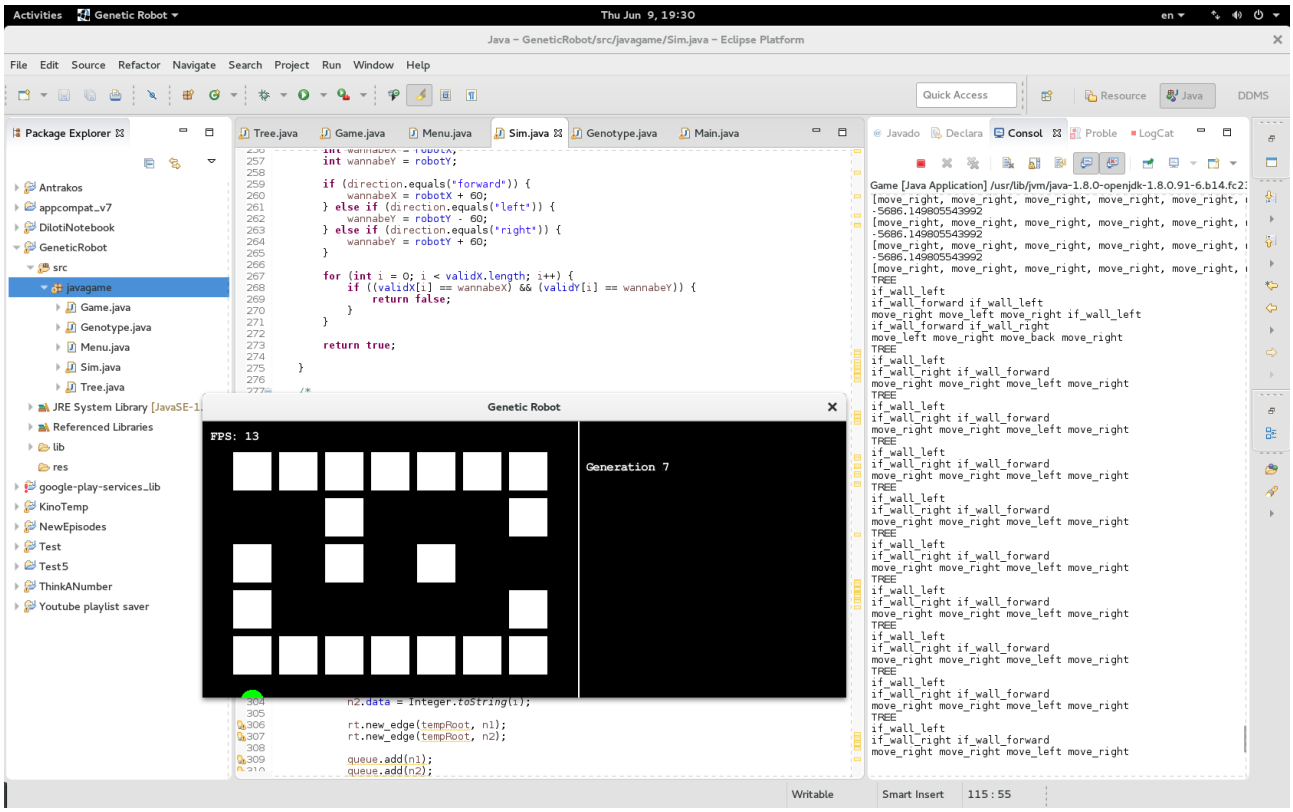
Τα δέντρα που δημιουργούνται περιορίζονται σε δέντρα που δίνουν αλγορίθμους μέχρι και 10 τερματικών.

Ο τελεστής της επιλογής λειτουργεί όπως και στους γενετικούς αλγορίθμους, με τη διαφορά ότι εδώ αξιολογούνται αλγόριθμοι. Μετά τη δημιουργία του τυχαίου binary tree, χρησιμοποιώντας τη συνάρτηση getAlgorithm, εξάγουμε τα βήματα (moves) που θα γίνουν καθώς εκτελείται ο αλγόριθμος. Τα βήματα αυτά, σε συνδυασμό με το μέγεθος του δέντρου, οδηγούν στην αξιολόγηση.

Στη διασταύρωση επιλέγεται δύο τυχαία υπόδεντρα από δύο τυχαία δέντρα του πληθυσμού και αντιμετατίθενται.

Στη μετάλλαξη ένα υπόδεντρο ενός τυχαίου δέντρου αντικαθίσταται από ένα άλλο τυχαίο δέντρο, ενώ τέλος στην αναπαραγωγή το καλύτερο γονίδιο του πληθυσμού αντικαθιστά στο χειρότερο.

Η υλοποίηση έγινε σε Java με τη βιβλιοθήκη γραφικών slick2D.
Πατώντας στον κύκλο στο αρχικού μενού η διαδικασία λαμβάνει χώρα και δημιουργείται μία λίστα η οποία περιέχει το καλύτερο γενότυπο από κάθε γενιά. Αυτό εμφανίζεται στην οθόνη του χάρτη, ενώ το δέντρο που εκτελείται εκείνη τη στιγμή (δηλαδή ο αλγόριθμος) φαίνεται στην κονσόλα.

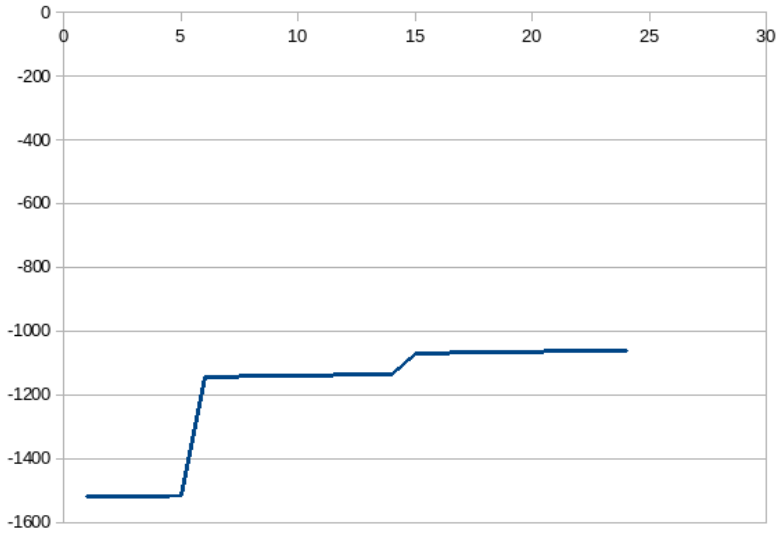


Εικόνα 3: Παράδειγμα Εκτέλεσης.

Στην κλάση Tree υλοποιείται το δέντρο και οι κόμβοι και υπάρχουν διάφοροι μέθοδοι που χρειάζονται για αυτό. Η κλάση Genotype περιέχει όλα τα χαρακτηριστικά που έχει ένα γενότυπο (δέντρο, Fitness κτλ), όπως φυσικά και τη συνάρτηση που υπολογίζει το Fitness.

Στην συνάρτηση init της Sim, αρχικοποιείται ένας τυχαίος πληθυσμός. Έπειτα στη render (τη πρώτη φορά που εκτελείται), εκτελούνται οι τελεστές και ο πληθυσμός εξελίσσεται. Στην update, εφόσον ελέγχουμε ότι έχουν ολοκληρωθεί όλες οι γενιές απλά γίνεται οπτικοποίηση των αποτελεσμάτων.

Στο παρακάτω διάγραμμα φαίνεται η Fitness του καλύτερου γενότυπου σε συνάρτηση με τις γενιές. Λόγο ενός bug (βλ. σημείωση), ποτέ δε φτάνει στο τέλειο.



Εικόνα 4: Διάγραμμα γενιών-τιμής συνάρτησης αξιολόγησης καλύτερου γενότυπου

Σημείωση:
Λόγω κάποιου bug (πιθανόν στη μετάφραση των δέντρων σε αλγόριθμο) το πρόγραμμα τελικά δε καταφέρνει να βρει τη βέλτιστη λύση αν και κάποιες φορές φτάνει αρκετά κοντά. Έγιναν δοκιμές και με πολύ περισσότερες γενιές, αλλά δεν εντοπίστηκε η λεπτομέρεια που το χαλάει. Για την αποφυγή πολύ μεγάλων δέντρων τώρα, ορίστηκε κάτι σαν νέος τελεστής ο οποίος επιλέγει ένα τυχαίο γενότυπο και το αρχικοποιεί ξανά σαν μικρό, πράγμα που βελτίωσε την επίδοση μιας και τα πολύ μεγάλα δέντρα οδηγούσαν σε χειρότερες επιδόσεις. Τα αρχεία του προγράμματος βρίσκονται στον φάκελλο GeneticRobot. Το IDE που χρησιμοποιήθηκε ήταν το Eclipse Marc 2.0.