ΑΤΕΙ ΣΕΡΡΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ:

Υλοποίηση ενός απλού Classifier System (σταθερού πλήθους κανόνων) για εφαρμογή στο πρόβλημα προσομοίωσης ενός ψηφιακού πολυπλέκτη κα ενός σύνθετου (μεταβλητού πλήθους κανόνων) για εφαρμογή σε απλά προβλήματα Classification

Φοιτητής: Ρόμπης Κων/νος
Επιβλέπων Καθηγητής: Δρ. Καζαρλής Σπυρίδων

Σέρρες 2006

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΕΧΟΜΕΝΑ....1

ΕΙΣΑΓΩΓΗ....3
GBML – Rule Based Machine Learning....3

CLASSIFIER SYSTEMS....5
Τι είναι ένα Classifier System και από τι αποτελείται....5
Κανόνες-Μηνύματα (Rule and Message System)....6
Σύστημα Καταμερισμού Βαθμών (Apportionment of credit system)....8
Γενετικός Αλγόριθμος (Genetic Algorithm)....10

ΓΕΝΕΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ....12
Γενικά....12
Λειτουργία Γενετικών Αλγορίθμων....13
ΤΕΧΝΙΚΗ ΠΕΡΙΓΡΑΦΗ....14
Αξιολόγηση του Πληθυσμού....16
Επιλογή Γονέων....16
Ανασυνδυασμός των Λύσεων-Διασταύρωση....17
Μετάλλαξη....20
Ελιτισμός....21
Άλλοι Γενετικοί Τελεστές....21
Αντικατάσταση Πληθυσμού....24
Τερματισμός του Γενετικού Αλγόριθμου....24
Ιδιότητες Γενετικών Αλγορίθμων....25
Πλεονεκτήματα-Μειονεκτήματα....26
Γιατί λειτουργούν οι Γενετικοί Αλγόριθμοι;....26
Υπόθεση Δομικών Στοιχείων (Building Block Hypothesis)....31

ΥΛΟΠΟΙΗΣΗ ΑΠΛΟΥ CLASSIFIER SYSTEM....32
ΣΚΟΠΟΣ....32
Δυαδικοί Πολυπλέκτες....32
Υλοποίηση Προγράμματος....34
ΟΘΟΝΕΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ....35
Εισαγωγική Οθόνη Επιλογής....35
Η Κύρια οθόνη του προγράμματος....36
Η Οθόνη Αρχικοποίησης Τιμών INITIAL....39
Η Οθόνη About....40
ΜΗΝΥΜΑΤΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ....40
ΕΛΛΕΙΠΗ ΑΡΧΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ....40
ΕΠΑΝΑΚΑΘΟΡΙΣΜΟΣ ΤΩΝ ΠΑΡΑΜΕΤΡΩΝ....41
Απόδοση του προγράμματος....41

ΥΛΟΠΟΙΗΣΗ ΣΥΝΘΕΤΟΥ CLASSIFIER SYSTEM....43
ΣΚΟΠΟΣ....43
ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ....43
ΟΘΟΝΕΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ....45
Η Κύρια οθόνη του προγράμματος....45

Η Οθόνη Αρχικοποίησης Τιμών INITIAL 48
Η Οθόνη Αρχικοποίησης Φόρων 49
Απόδοση του προγράμματος 50

Ο ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΤΟΥ SCS 52
ΤΕΧΝΙΚΗ ΠΕΡΙΓΡΑΦΗ 52
ΤΑ UNIT ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ 62
ΕΙΣΑΓΩΓΙΚΗ ΦΟΡΜΑ 62
ΦΟΡΜΑ ΤΟΥ SCS 63
ΦΟΡΜΑ ΑΡΧΙΚΟΠΟΙΗΣΗΣ ΤΙΜΩΝ INITIAL 69
ΦΟΡΜΑ ABOUT 70

Ο ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΤΟΥ CCS 72
ΤΕΧΝΙΚΗ ΠΕΡΙΓΡΑΦΗ 72
ΤΑ UNIT ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ 76
ΦΟΡΜΑ ΤΟΥ SCS 76
ΦΟΡΜΑ ΑΡΧΙΚΟΠΟΙΗΣΗΣ ΤΙΜΩΝ INITIAL 81

ΕΙΣΑΓΩΓΗ

GBML – Rule Based Machine Learning

Τα GBML συστήματα των οποίων πατέρας θεωρείται ο Holland είναι συστήματα εκμάθησης κανόνων και κατηγοριοποίησης (Classifier Systems). Βασίζονται σε σύνολα κανόνων (rule-sets) που συνήθως βρίσκονται στην μορφή "*IF state THEN action*" και εκπαιδευόμενα μέσω γενετικών αλγορίθμων και εξελικτικών τεχνικών έχουν την ικανότητα αξιολόγησης και εξέλιξης της γνώσης που βρίσκεται μέσα σε αυτούς τους κανόνες και την περιγράφουν με τρόπο «πλήρη» και «περιεκτικό». Αναφέρονται σε προβλήματα για την λύση των οποίων απαιτείται λογισμικό που να έχει την ικανότητα προσαρμογής, δηλαδή να παρουσιάζει προσαρμοστικότητα. Προβλήματα των οποίων οι λύσεις έχουν άμεση συνάρτηση με τον περιβάλλοντα χώρο και τις ειδικές συνθήκες που επικρατούν σ' αυτόν. Μπορούν να προσομοιώσουν συστήματα εισόδου-εξόδου με την εκμάθηση των κανόνων και της εξέλιξης αυτών μέσα από έναν ΓΑ για την απόκριση της επιθυμητής συμπεριφοράς εισόδου-εξόδου. Θα πρέπει να αναφέρουμε επίσης ότι τα GBML συστήματα προσεγγίζουν τον χώρο της Τεχνητής Ευφυΐας και απομακρύνονται από τον χώρο της βελτιστοποίησης.

Όσον αφορά τις εξελικτικές τεχνικές, αφομοιώνονται μηχανισμοί και διαδικασίες που βασίζονται στην θεωρία που διατυπώθηκε από τον Δαρβίνο [1859] περί της εξέλιξης των ειδών. Οι τεχνικές αυτές εμπεριέχουν κάποιες αξιοσημείωτες ιδιότητες όπως παράλληλη επεξεργασία/υλοποίηση που παρέχει την δυνατότητα της ταυτόχρονης αξιολόγησης των λύσεων, προσαρμοστικότητα των προτεινόμενων λύσεων, διαδικασίες τροποποίησης των λύσεων και αλληλεπίδρασης μεταξύ τους και επιλογής των πιο βέλτιστων κάθε φορά λύσεων εντός μιας γενιάς ώστε να παραχθεί η επόμενη μέσα από χρήση γενετικών τελεστών.

Παρακάτω θα αναπτύξουμε τις αρχές που διέπουν τα *Classifier Systems* και θα τις εφαρμόσουμε σε μια υλοποίηση για την εύρεση του βέλτιστου σετ κανόνων για την περιγραφή ενός απλού ψηφιακού πολυπλέκτη 4 σε 1 και 8 σε 1.

Επίσης θα αναπτυχθεί και σύνθετο μοντέλο με δυνάμεις κανόνων και αλγόριθμο απονομής βαθμών (Bucket Brigade Algorithm) εφαρμόζοντας το σε απλό πρόβλημα

κατηγοριοποίησης (classification) αυξάνοντας την πολυπλοκότητα του και συνάμα αποτελώντας μια πιο επιθυμητή ρεαλιστική συμπεριφορά.

CLASSIFIER SYSTEMS

Τι είναι ένα Classifier System και από τι αποτελείται

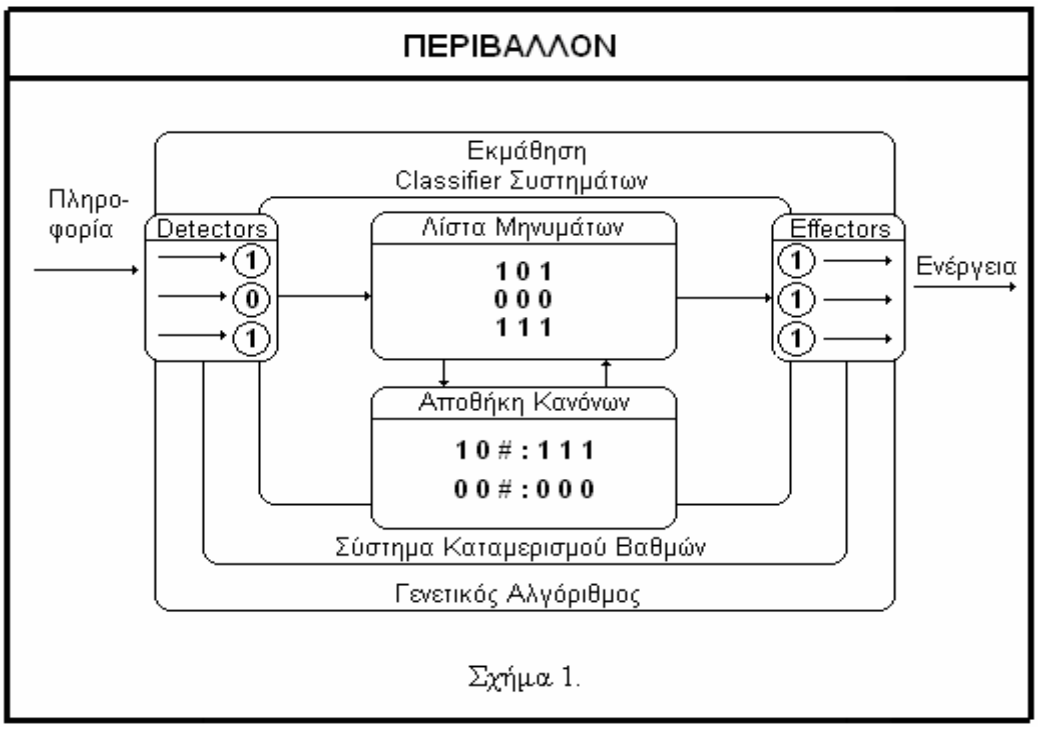
Ένα Classifier System είναι ένα σύστημα εκμάθησης και κατηγοριοποίησης συντακτικά απλών αλφαριθμητικών κανόνων (rules ή classifiers) με γενικό μηχανισμό για την παράλληλη επεξεργασία των κανόνων, για την αξιολόγηση της αποδοτικότητας των κανόνων και την προσαρμοστική παραγωγή νέων πιο βέλτιστων.

Έτσι ένα Classifier System αποτελείται από τρία κύρια μέρη:

1. Κανόνες-Μηνύματα (Rule and Message System)
2. Σύστημα καταμερισμού Βαθμών (Apportionment of credit system)
3. Γενετικό αλγόριθμο (Genetic Algorithm)

Πολλά παραδοσιακά συστήματα Τεχνητής Ευφυΐας χρησιμοποιούν σειριακή ενεργοποίηση των κανόνων. Αυτή η από κανόνα σε κανόνα διαδικασία δεν βοηθάει στην αύξηση της αποδοτικότητας και πολλές διαφορές μεταξύ τέτοιων συστημάτων αφορούν στην στρατηγική της επιλογή ενός μόνο "βέλτιστου" απλού ενεργοποιημένου κανόνα. Τα Classifier συστήματα υπερπηδούν την κατάσταση αυτή με την δυνατότητα της παράλληλης ενεργοποίησης πολλών κανόνων ταυτόχρονα κατά την διάρκεια ενός κύκλου.

Στο σχήμα 1 παρουσιάζεται η όλη διαδικασία που λαμβάνει χώρα σε ένα σύστημα εκμάθησης κανόνων - Classifier System.



Σχήμα 1.

Κανόνες-Μηνύματα (Rule and Message System)

Το σύστημα των κανόνων και μηνυμάτων είναι ένα σύστημα που προσομοιώνει ένα σύστημα εισόδου-εξόδου. Οι κανόνες αυτοί είναι της μορφής:

Εάν <συνθήκη> τότε <ενέργεια>

Αυτό σημαίνει πως όταν επαληθευτεί η συνθήκη (condition) τότε παράγεται η αντίστοιχη ενέργεια (action). Μια πιο προσδιοριστική μορφή ενός Classifier είναι η εξής:

< *classifier*> ::= < *συνθήκη* > : <*μήνυμα*>

όπου το *μήνυμα* (action) είναι μια συμβολοσειρά σταθερού μήκους και κωδικοποιείται από το αλφάβητο $\{0,1\}^L$, δηλαδή μια αλληλουχία L χαρακτήρων από 0 και 1.

Η συνθήκη (condition) ενός κανόνα είναι και αυτή σταθερού μήκους μόνο που εδώ πρέπει να προστεθεί στο παραπάνω αλφάβητο ο χαρακτήρας "#" (wildcard character ή don't care symbol) που λειτουργεί είτε με την τιμή 0 είτε με την τιμή 1. Παράδειγμα ενός classifier με 4 bits για condition και action είναι το:

##00 **:** 0001

που ενεργοποιείται από το *μήνυμα* 1100 αλλά όχι από το 1110 και παράγει το μήνυμα 0001.

Αυτό που συμβαίνει σε ένα σύστημα κανόνων και μηνυμάτων είναι πως η πληροφορία που απορρέει από το περιβάλλον λαμβάνεται διαμέσου των *detectors*, που είναι τα αισθητήρια όργανα – τα αυτιά και τα μάτια ενός Classifier συστήματος – και την κωδικοποιούν σε ένα μήνυμα πεπερασμένου μεγέθους. Τα μηνύματα αυτά αποστέλλονται σε μια πεπερασμένη λίστα μηνυμάτων (message list) και μπορούν να διεγείρουν έναν ή περισσότερους κανόνες. Με την διέγερση των κανόνων παράγονται μηνύματα που και αυτά μπορούν να ενεργοποιήσουν άλλους κανόνες μέχρι αυτή η διαδικασία να σταματήσει με την απόκριση του συστήματος μέσω των *effectors*.

Με τα όσα ειπώθηκαν μέχρι τώρα δημιουργούνται κάποιες απορίες όπως όταν η λίστα μηνυμάτων είναι ανεπαρκούς μεγέθους για να αποθηκεύσει όλους τους κανόνες που "ταιριάξανε" με τα μηνύματα, πως καθορίζεται ποιοι από αυτούς θα ενεργοποιηθούν; Η απάντηση για το ερώτημα αυτό όπως και άλλες προσομοιάζουσες συνδέονται με το σύστημα καταμερισμού βαθμών (Apportionment of credit system) η οποία εφαρμόζει μία «οικονομία» ανταγωνιστικού είδους, που προκρίνει τους καλύτερους και η οποία θα αναπτυχθεί αμέσως μετά.

Ακολουθεί ένα παράδειγμα για να κατανοηθεί καλύτερα το σύστημα κανόνων και μηνυμάτων. Έστω το ακόλουθο σύστημα κανόνων και μηνυμάτων:

Classifier

1)	01## : 0000
2)	00#0 : 1100
3)	11## : 1000
4)	##00 : 0001

Αρχικά εμφανίζεται σαν είσοδος στην λίστα μηνυμάτων το μήνυμα 0111 (περιβάλλον). Ενεργοποιείται ο κανόνας 1 και παράγεται το μήνυμα 0000. Αυτό με την σειρά του ενεργοποιεί ταυτόχρονα τους κανόνες 2 και 4 και παράγονται τα μηνύματα 1100 και 0001 αντίστοιχα. Το 0001 δεν ενεργοποιεί κανέναν κανόνα. Το 1100 ενεργοποιεί τους κανόνες 3 και 4 και παράγονται τα μηνύματα 1000 και 0001 αντίστοιχα. Το 1000 ενεργοποιεί τον κανόνα 4 που παράγει το μήνυμα 0001 και η διαδικασία σταματά αφού το τελευταίο μήνυμα δεν ενεργοποιεί κανέναν κανόνα.

Σύστημα Καταμερισμού Βαθμών (Apportionment of credit system)

Καθώς το μήνυμα ταιριάζει με πολλούς κανόνες είναι απαραίτητο να δούμε ποιοι από αυτούς θα ενεργοποιηθούν καθώς η λίστα μηνυμάτων είναι πεπερασμένη. Έτσι κάθε κανόνας έχει ένα βαθμό «δύναμης» S (Strength) μέσω του οποίου καθορίζεται ποιος κανόνας θα ενεργοποιηθεί.

Για την δυναμική αξιολόγηση των κανόνων έχουν προταθεί πολλοί τρόποι. Εμείς θα χρησιμοποιήσουμε τον πιο διαδεδομένο, αυτόν που ο Holland ονόμασε "***bucket brigade algorithm***".

Σύμφωνα με τον αλγόριθμο αυτόν όταν ένας κανόνας ταιριάξει με ένα μήνυμα δεν ενεργοποιείται κατευθείαν αλλά γίνεται υποψήφιος προς ενεργοποίηση. Για να επιλεγεί ένας κανόνας ώστε να ενεργοποιηθεί προσφέρει ένα ποσό ανάλογο της σχετικής του δύναμης B (Bid). Το ποσό αυτό, αποτελεί «πληρωμή» P (payment) και μοιράζεται σε όλους τους κανόνες που παρήγαγαν τα μηνύματα τους και ενεργοποίησαν τον κανόνα. Οι κανόνες λοιπόν συμμετέχουν σε μια «οικονομία της πληροφορίας» όπου η δυνατότητα του να *δίνεις* και να *λαμβάνεις* πληροφορία αγοράζεται και πουλιέται. Αυτό γίνεται με 2 μηχανισμούς που αποτελούν τα μέρη του Bucket Brigade:

1. Την «*Δημοπρασία*» (Auction), κάθε κανόνας συναγωνίζεται με τους άλλους για να ενεργοποιηθεί.
2. και την «*Πληρωμή Επιταγών*», κάθε κανόνας προσφέρει σε αυτούς που τον ενεργοποίησαν

Προκύπτουν λοιπόν κάποια συμπεράσματα. Κανόνες με μεγάλη δύναμη έχουν μεγαλύτερη πιθανότητα να ενεργοποιηθούν πράγμα που δείχνει την σπουδαιότητα του κάθε κανόνα. Ακόμη το μοίρασμα του πόσου που προσφέρεται από τους κανόνες συμβάλλει στο να δημιουργούνται ομάδες κανόνων που δείχνουν την αναζητούμενη συμπεριφορά όπως συμβαίνει και με το "fitness sharing", μόνο που εδώ η μέθοδος αυτή μετακινεί άτομα από περιοχές με ανεξέλεγκτα συσσωρευμένα άτομα-λύσεις (κανόνες) σε περιοχές υπο-βέλτιστων λύσεων όπου δημιουργούν υπο-πληθυσμούς.

Για να κατανοηθούν ακόμη περισσότερο τα όσα αναφέρθηκαν θα ξαναειπωθεί το παράδειγμα που αναφέρθηκε στην προηγούμενη παράγραφο εφαρμόζοντας σ' αυτό τον BB (Bucket Brigade).

Αρχικά η δύναμη του κάθε κανόνα θα έχει την τιμή 200. Ακόμη το ποσό που θα προσφέρεται από τους κανόνες θα είναι ανάλογο της δύναμης τους και θα καθορίζεται από έναν συντελεστή C_{bid} με τιμή 0.1 . Άρα η αρχική προσφορά κάθε κανόνα θα είναι Bid = Strength * C_{bid} , Bid=20. Θα πρέπει επίσης, να ορίσουμε και ένας είδος φορολογίας ώστε να μην αυξάνεται υπερβολικά η δύναμη των πιο συχνά εμφανιζόμενων κανόνων. Έτσι κάθε ν κύκλους εφαρμόζονται φόροι Τ (Taxes) στον ΒΒ.

Κι εδώ το μήνυμα που έρχεται από το περιβάλλον και λαμβάνεται από τα *detectors* είναι το 0111 (Τα παραδείγματα αυτά υπάρχουν αυτού καθ΄ αυτού και στις σημειώσεις "Εξελικτική Υπολογιστική" του τμήματος Πληροφορικής).

		t=0				t=1				t=2			
α/α	Κανόνες	Δυν.	Μην.	Ταιρ.	Προσφ.	Δυν.	Μην.	Ταιρ.	Προσφ.	Δυν.	Μην.	Ταιρ.	Προσφ.
1	01## : 0000	200		Περ.	20	180	0000			220			
2	00#0 : 1100	200				200		1	20	180	1100		
3	11## : 1000	200				200				200		2	20
4	##00 : 0001	200				200		1	20	180	0001	2	18
	Περιβάλλον		0111			20				20			

Πίνακας 1

Όπως φαίνεται και στον Πίνακα 1,στον χρόνο 0, το μήνυμα από το περιβάλλον 0111 διεγείρει τον κανόνα 1 που παράγει το μήνυμα 0000 και προσφέρει 20 μονάδες από την δύναμη του στο περιβάλλον επειδή τον ενεργοποίησε. Το 0000 ταιριάζει με τους κανόνες 2 και 4 που παράγουν τα μηνύματα 1100 και 0001 αντίστοιχα. Από τους 2 και 4 προσφέρονται 20 μονάδες από τον καθένα στον κανόνα 1. Η προσφορά είναι ίδια και από τους δύο κανόνες και παράγονται τα μηνύματα τους. Το μήνυμα 0001 του κανόνα 4 δεν ταιριάζει με κανένα κανόνα . Το 1100 ταιριάζει με τους 3 και 4 που προσφέρουν 20 και 18 μονάδες αντίστοιχα. Ο Οι μονάδες αυτές προσφέρονται στον 2

		t=3				t=4				t=5			
α/α	Κανόνες	Δυν.	Μην.	Ταιρ.	Προσφ.	Δυν.	Μην.	Ταιρ.	Προσφ.	Δυν.	Αμοιβή περιβάλλοντος		
1	01## : 0000	220				220				220			
2	00#0 : 1100	218				218				218			
3	11## : 1000	180	1000			196				196			
4	##00 : 0001	162	0001	3	16	146	0001			196	50		
	Περιβάλλον	20				20				20			

Πίνακας 2

Παράγονται τα μήνυμα 1000 και 0001. Το 1000 ταιριάζει με τον κανόνα 4 και που προσφέρει 16 μονάδες στον 3. Ο 4 ενεργοποιείται και ο 3 λαμβάνει τις 16 μονάδες αλλά το μήνυμα που παράγεται δεν ταιριάζει με κανέναν κανόνα και η διαδικασία σταματά και το αποτέλεσμα "εκκρίνεται" δια μέσου των "effectors". Τέλος η τελική προσφορά είναι η ανταμοιβή από το περιβάλλον και δίνεται στον τελευταίο ενεργοποιημένο κανόνα, τον κανόνα 4 (βλέπε και πίνακα 2).

Παρόλα αυτά η εκμάθηση ενός Classifier δεν σταματά εδώ. Μέχρι τώρα είδαμε πως δημιουργούνται οι κανόνες και πως καθορίζονται δυναμικά οι δυνάμεις τους. Πως μπορούμε να εισάγουμε όμως καινούργιους κανόνες ώστε να καλυτερέψουμε το σύστημα μας. Την δουλειά αυτή επιτυγχάνει μέσω γενετικής εξέλιξης, η χρήση Γενετικού Αλγόριθμου (ΓΑ).

Γενετικός Αλγόριθμος (Genetic Algorithm)

Την αναζήτηση και εύρεση βέλτιστων λύσεων (κανόνων) καθιστά δυνατή η εισαγωγή γενετικού αλγόριθμου στο σύστημα. Μέσω της τριμερούς διαδικασίας *αναπαραγωγή, διασταύρωση, μετάλλαξη* (reproduction, crossover, mutation), οι "εξελιγμένοι" κανόνες τοποθετούνται στον πληθυσμό, έτοιμοι να επανακαθορίσουν τις δυνάμεις τους μέσα από τον μηχανισμό της *δημοπρασίας* της *πληρωμής* και της *ενίσχυσης* (auction, payment, reinforcement). Ιδιαίτερη προσοχή πρέπει να δοθεί στην επιλογή του κανόνα ο οποίος θα αντικατασταθεί μέσα στον πληθυσμό. Στην εκμάθηση συστημάτων ενδιαφερόμαστε να διατηρήσουμε μια πιο υψηλού επιπέδου ελεγχόμενη εκτέλεση της αντικατάστασης του πληθυσμού. Για αυτό στα συστήματα χρησιμοποιούμε:

- Αναπαραγωγή Σταθερής Κατάστασης (Steady-State Replacement)
- ή Αντικατάσταση ενός μέρους του πληθυσμού κατά ένα μικρό ποσοστό (Partial Replacement).

Στο σημείο αυτό να αναφέρουμε πως υπάρχει μια παράμετρος στους ΓΑ που καλείται *Generation Gap* (G) και χρησιμοποιείται για να ελέγχει την υπερθετικότητα του πληθυσμού.

Ο Γενετικός αλγόριθμος λαμβάνει χώρα κάθε μία περίοδο $T_{ΓΑ}$ που ορίζεται ως ένας κύκλος μεταξύ κανόνων και μηνυμάτων και εκλαμβάνει τους κανόνες ως τον τρέχοντα πληθυσμό. Η περίοδος αυτή συμπεριφέρεται είτε ντετερμινιστικά –ο ΓΑ καλείται κάθε $T_{ΓΑ}$ κύκλους– , είτε στοχαστικά/τυχαία –ο ΓΑ καλείται πιθανοκρατικά κάθε μέσο όρο περιόδων $T_{ΓΑ}$–.

Η επιλογή των γονέων γίνεται βάση της δύναμης του κάθε κανόνα, όπως αναφέρθηκε στον BB, και τέλος παράγεται ο απόγονος μέσω των γενετικών τελεστών. Θα πρέπει να σημειωθεί ότι για τον τελεστή «Μετάλλαξης» λαμβάνεται υπόψη το τριαδικό αλφάβητο για την συνθήκη (condition) και το δυαδικό για το μήνυμα (action).

Είναι ιδιαίτερα χρήσιμο οι γενετικοί αλγόριθμοι να αναλυθούν με εκτενή τρόπο, γι' αυτό και αφιερώνεται ένα ολόκληρο κεφάλαιο για σχετικά με το πώς λειτουργούν, τις ιδιότητες τους και γιατί δουλεύουν τόσο αποτελεσματικά.

ΓΕΝΕΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ

Γενικά

Η ανάγκη μοντελοποίησης ενός φαινομένου είναι σημαντική γιατί βοηθάει να μελετηθεί ένα σύστημα διεξοδικότερα και να προσομοιώσει τη δομή και τη λειτουργία του. Μέσα από ένα μοντέλο ο ερευνητής μπορεί να προσομοιώσει με μαθηματικές σχέσεις και εξισώσεις ένα σύστημα. Τα μοντέλα που αναπαριστούν με χρήση μαθηματικών προτύπων ονομάζονται μαθηματικά μοντέλα. Τα μοντέλα αυτά αδυνατούν να αναπαραστήσουν πολύπλοκα συστήματα τα οποία προϋποθέτουν την χρήση μη γραμμικών εξισώσεων που είναι δύσκολο να επιλυθούν αναλυτικά. Η ραγδαία ανάπτυξη και εξάπλωση των υπολογιστών συνέβαλλε σε μια νέα κατεύθυνση για την περιγραφή συστημάτων με την ανάπτυξη υπολογιστικών μοντέλων. Τα μοντέλα αυτά δεν κάνουν χρήση μαθηματικών σχέσεων αλλά κατάλληλων υπολογιστικών προγραμμάτων για την λειτουργία του εκάστοτε συστήματος.

Παράλληλα η πεποίθηση ότι η αναζήτηση λύσεων για πολύπλοκα υπολογιστικά προβλήματα, στον χώρο της βιολογίας, κατακτά όλο και περισσότερο το ενδιαφέρον των επιστημόνων. Προβλήματα που προϋποθέτουν την παράλληλη και ταυτόχρονη διερεύνηση πολλών υποψήφιων λύσεων ώστε να βρεθεί η πιο βέλτιστη από το σύνολο αυτό. Έτσι αναπτύχθηκαν οι εξελικτικοί αλγόριθμοι που παρουσίαζαν έντονη γοητεία, αφού η δομή και η λειτουργία τους εμπνεύσθηκαν από τις αρχές και τους μηχανισμούς της εξέλιξης των ειδών και του νόμου της επιβίωσης των καλύτερων ατόμων (survival of the fittest) που διέπουν τα φυσικά βιολογικά συστήματα, όπως καθορίσθηκαν από τον Δαρβίνο.

Κύριος και πιο αντιπροσωπευτικός κλάδος των εξελικτικών αλγορίθμων είναι οι γενετικοί αλγόριθμοι που είναι αλγόριθμοι γενικής βελτιστοποίησης (Global Optimization Algorithms) και αναζήτησης (Search Algorithms). Αυτοί δίνουν τη δυνατότητα της παράλληλης και εξαντλητικής διερεύνησης πολλών υποψήφιων λύσεων. Σε κάθε λύση αντιστοιχίζεται ένας βαθμός που δείχνει την καταλληλότητα της λύσης και αξιολογούνται ανάλογα με το είδος του προβλήματος. Οι καλύτερες από αυτές έχουν μεγαλύτερη πιθανότητα να συμμετέχουν σε μια διαδικασία

παρόμοια με αυτή της φυσικής επιλογής, έτσι ώστε να μπορέσουν να ανασχηματίσουν σε καινούργιες λύσεις πιθανότατα καλύτερες, τα "γενετικά" χαρακτηριστικά τους. Υπάρχει βέβαια και η πιθανότητα να μην είναι τόσο ευεργετικές οι καινούργιες λύσεις, αλλά αφομοιώνοντας την αρχή της επιβίωσης των καλύτερων κατευθυνόμαστε κυρίως στην βαθμιαία βελτίωση των λύσεων. Σε αυτό συμβάλλει και ο μικρός αλλά όχι μηδαμινός παράγοντας της μετάλλαξης των χαρακτηριστικών των λύσεων, που εμπεριέχει τυχαιότητα, ευελιξία και οφείλεται στις αλλαγές του περιβάλλοντος από εξωτερικούς παράγοντες. Αν επαναλάβουμε την διαδικασία για αρκετές γενιές κρατώντας σαν τρέχοντα πληθυσμό των πληθυσμό των απογόνων στην αρχή κάθε γενιάς τότε ο αλγόριθμος θα συγκλίνει στις βέλτιστες λύσεις. Αυτή η δυναμική της βελτίωσης των λύσεων και της προσαρμογής τους στις εκάστοτε αλλαγές του περιβάλλοντος είναι που καθιστούν τους πληθυσμούς βιώσιμους και βελτιστοποιήσιμους.

Πρωτοπόρος των γενετικών αλγορίθμων θεωρείται ο John Holland, που με την εργασία του "Adaptation in Natural and Artificial Systems" θεμελίωσε τις αρχές των ΓΑ και τα πρώτα θεωρήματα της θεωρίας σχημάτων (Schema Theorem). Μαζί με τους μαθητές του, προσπάθησαν:

a) Να προσδιορίσουν τις διαδικασίες που διέπουν τα φυσικά συστήματα

b) Να αναπτύξουν υπολογιστικά προγράμματα για την πραγματοποίηση τεχνητών συστημάτων που θα αναπαριστούσαν πραγματικά συστήματα, και θα τα αξιοποιούσαν σε πρακτικές εφαρμογές.

Λειτουργία Γενετικών Αλγορίθμων

Ένας Γενετικός Αλγόριθμος αποτελείται από τα ακόλουθα στάδια:

1. Αρχική παραγωγή πληθυσμού. Ο πληθυσμός αυτός διατηρεί κωδικοποιημένες λύσεις του προβλήματος.
2. Αξιολόγηση των ατόμων-λύσεων. Μέσα από την συνάρτηση ποιότητας (fitness quality function) αποτιμάται κάθε άτομο. Βάση αυτής της πληροφορίας εξελίσσεται ο ΓΑ.

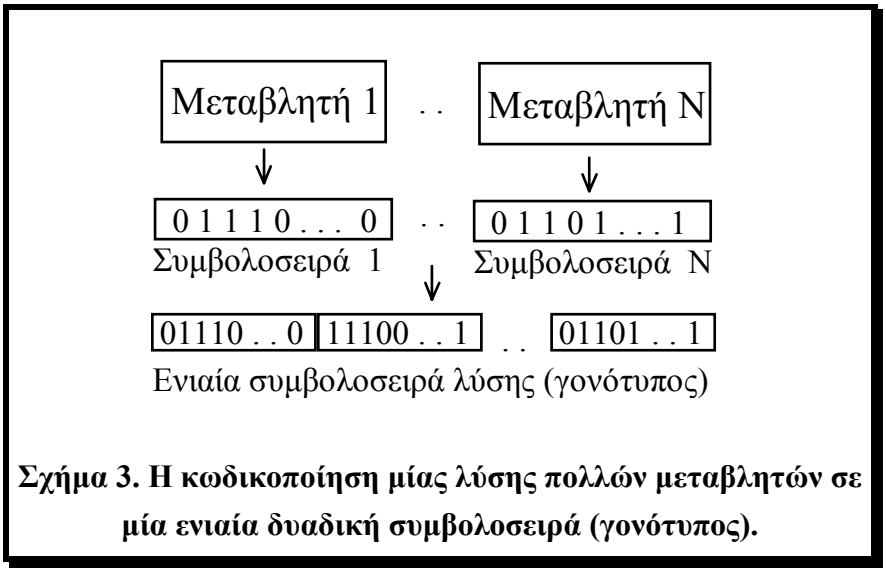
3. Πιθανοκρατική επιλογή των ατόμων σύμφωνα με την αρχή της επιβίωσης του καλύτερου. Λύσεις με μεγάλη τιμή αξιολόγησης είναι πιθανότερο να επιλεγούν για την διαδικασία της αναπαραγωγής.
4. Εφαρμογή Γενετικών Τελεστών όπως η διασταύρωση και η μετάλλαξη. Τα άτομα που επιλέχθηκαν είτε θα ανασυνδυαστούν είτε θα περάσουν αυτούσια στην επόμενη γενιά σαν ένα είδος κλωνοποίησης.
5. Αντικατάσταση των ατόμων της τρέχουσας γενιάς από τους απογόνους. Γενεαλογική αντικατάσταση των ατόμων που πραγματοποιείται ολικά (ολόκληρος ο πληθυσμός) ή μερικά (ένα τμήμα του βάση κάποιου ποσοστού).
6. Τερματισμός του Γενετικού Αλγόριθμου. Αν ικανοποιηθούν κάποια κριτήρια τερματισμού (αριθμός γενεών ή σύγκλιση της ποιότητας των ατόμων σε μια επιθυμητή τιμή) η εξέλιξη τερματίζει.

ΤΕΧΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Η μορφή με την όποια διατηρούνται τα άτομα-λύσεις του πληθυσμού δεν είναι η κανονική τους αλλά μια κωδικοποιημένη μορφή. Για να περιγράψουμε αυτήν την μορφή κωδικοποίησης θα πρέπει να χρησιμοποιήσουμε κάποιους όρους από τον χώρο των βιολογικών επιστημών, αφού οι ΓΑ στην ουσία είναι εξομοίωση φυσικών βιολογικών συστημάτων. Έτσι η κωδικοποιημένη μορφή των ατόμων-λύσεων, η οποία εξαρτάται από το είδος του προβλήματος κάθε φορά, ονομάζεται *γονότυπος* (genotype), ενώ η αποκωδικοποιημένη πραγματική παραμετρική της μορφή ονομάζεται *φαινότυπος* (phenotype). Ο γονότυπος αυτός αποτελείται από *χρωμοσώματα* (chromosomes). Στην περίπτωση που ο σχεδιασμός απαιτεί ένα μόνο χρωμόσωμα τότε το χρωμόσωμα ταυτίζεται με τον γονότυπο. Έτσι κάθε λύση-γονότυπος/χρωμόσωμα είναι κωδικοποιημένη σε μια συμβολοσειρά (string), τα σύμβολα της οποίας ανήκουν σε ένα όσο το δυνατό μικρό σύνολο συμβόλων. Το πιο συνηθισμένο σύνολο συμβόλων είναι το δυαδικό σύνολο {0,1} όπου κάθε λύση είναι μία αλληλουχία από 0 και 1. Αν και αρχικά η δυαδική μορφή των λύσεων παρουσίαζε μεγάλο ενδιαφέρον, η συνεχώς αυξανομένη πολυπλοκότητα των προβλημάτων είχε ως αποτέλεσμα η δυαδική κωδικοποίηση να αδυνατεί να δώσει λύσεις. Αυτό οδήγησε στο να εμφανιστούν γενετικοί αλγόριθμοι με κωδικοποίηση ατόμων πιο ρεαλιστική και την προσαρμογή των διάφορων τελεστών των ΓΑ σε αυτήν. Στην κατεύθυνση

αυτή φυσικά βοήθησε και η εμφάνιση διάφορων εργαλείων αντικειμενοστραφούς προγραμματισμού. Για ένα παράδειγμα μιας τριαδικής κωδικοποίησης με σύνολο συμβόλων {0,1,#} , θα πραγματοποιούνταν αναπαράσταση των ατόμων με 0,1 και #, και ο γενετικός τελεστής μετάλλαξης σ' αυτήν την περίπτωση θα μετέτρεπε το 0 σε 1και # με ίση πιθανότητα και αντίστοιχα τα υπόλοιπα σύμβολα.

Στο σχήμα 3 φαίνεται ένας γονότυπος αποτελούμενος από πολλές μεταβλητές ή χρωμοσώματα που είναι κωδικοποιημένος σε δυαδική μορφή. Αν κάθε μεταβλητή έπαιρνε τιμή από το διάστημα {-χ … χ} θα μπορούσε να κωδικοποιηθεί με έναν ν-bit αριθμό σε 2^{ν} διαφορετικές τιμές. Το βήμα διακριτοποίησης που αντιστοιχεί το κάθε bit θα είναι $2\chi / 2^{\nu}$. Το βήμα μπορούμε να το προσαρμόσουμε κατάλληλα ώστε να αυξήσουμε την απόδοση της κωδικοποίησης.



Σχήμα 3. Η κωδικοποίηση μίας λύσης πολλών μεταβλητών σε μία ενιαία δυαδική συμβολοσειρά (γονότυπος).

Μια περαιτέρω ανάλυση της κωδικοποίησης είναι τα *γονίδια* (genes), από τα οποία αποτελούνται τα χρωμοσώματα. Το κάθε σύμβολο δηλαδή που είναι διατεταγμένο στη γραμμική ακολουθία. Ακόμη τα γονίδια που επηρεάζουν την συμπεριφορά του γονότυπου βρίσκονται σε συγκεκριμένες θέσεις που ονομάζονται locus (position). Τέλος κάθε χρωμόσωμα μπορεί να πάρει μόνο επιτρεπτές τιμές και οι οποίες καλούνται *alleles* (καταστάσεις ενός γνωρίσματος).

Αξιολόγηση του Πληθυσμού

Μετά την παράγωγη του πληθυσμού όπου το γονίδιο κάθε χρωμοσώματος (ή γονότυπου στην περίπτωση που έχουμε ένα χρωμόσωμα) παίρνει τυχαία τιμή από το σύνολο των συμβόλων, έρχεται η διαδικασία της αξιολόγησης των λύσεων. Η διαδικασία αυτή πραγματοποιείται με την βοήθεια μιας συνάρτησης καταλληλότητας που δείχνει την σπουδαιότητα κάθε λύσης-ατόμου μέσα στον πληθυσμό. Ονομάζεται *συνάρτηση ποιότητας* ή *συνάρτηση αξιολόγησης* (quality ή fitness function). Μέσω αυτής σε κάθε άτομο δίδεται μια τιμή η οποία αντιπροσωπεύει πόσο καλά προσαρμοσμένη είναι η κάθε λύση στις εξωτερικές συνθήκες του περιβάλλοντος. Συνεπώς όσο πιο προσαρμοσμένο είναι ένα άτομο τόσο μεγαλύτερη θα είναι και η τιμή που του αντιστοιχίζεται βάση της οποίας θα επιλεγεί για αναπαραγωγή. Έτσι οι υψηλόβαθμες λύσεις έχουν μεγαλύτερη πιθανότητα να κληροδοτήσουν τα "γενετικά" τους χαρακτηριστικά. Η συνάρτηση αυτή είναι δηλαδή ο λόγος που οδεύουμε στην βελτιστοποίηση του ΓΑ.

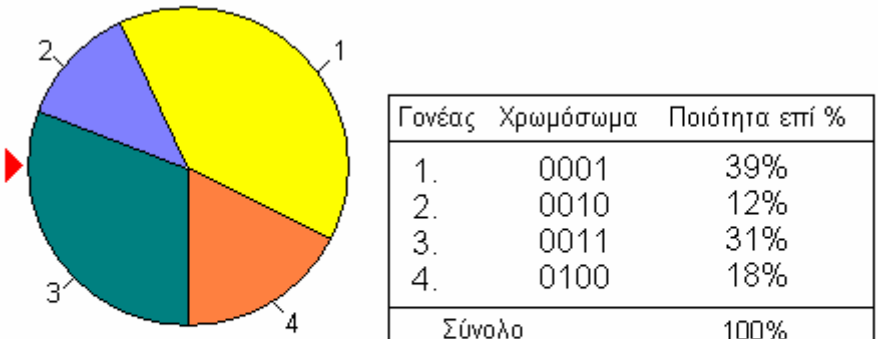
Επιλογή Γονέων

Αφού αποτιμηθούν όλα τα άτομα-λύσεις του πληθυσμού, ξεκινάει η διαδικασία *επιλογής των γονέων* (selection procedure) ώστε να ανασυνδυαστούν τα άτομα και να παραχθεί η νέα γενιά. Ο καινούργιος πληθυσμός δημιουργείται με την επιλογή των όσο καλυτέρα προσαρμοσμένων ατόμων, δηλαδή των ατόμων με τις μεγαλύτερες τιμές καταλληλότητας που προκύπτουν από την συνάρτηση ποιότητας. Βέβαια δεν αποκλείονται από την επιλογή άτομα με μικρή τιμή καταλληλότητας, εφόσον ενδέχεται ο συνδυασμός μιας καλής και μιας κακής λύσης ή ακόμη και ο συνδυασμός δύο κακών λύσεων να δώσει λύσεις με καλύτερη ποιότητα. Για τον σκοπό αυτό έχουν αναπτυχθεί διάφορες μέθοδοι επιλογής, όπως:

- **Αναλογική επιλογή** (proportionate selection), όπου τα άτομα επιλέγονται αναλογικά, σύμφωνα με την ποιότητας τους
- **Επιλογή με Διαβάθμιση** (ranking selection), όπου τα άτομα του πληθυσμού ταξινομούνται είτε με αύξουσα είτε με φθίνουσα σειρά και ο αριθμός των φορών που θα επιλεγούν είναι συνάρτηση του αριθμού ταξινόμησης τους.

- **Boltzmann επιλογή** (Boltzmann selection), όπου σε κάθε άτομο η τιμή αξιολόγησης f τροποποιείται σε f' που παράγεται με βάση την συνάρτηση f'= $(1+\exp(f/T)^{-1})$, και την μεταβλητή T να χρησιμοποιείται με παρόμοιο τρόπο με αυτόν της θερμοκρασίας στην κατανομή Boltzmann
- **Επιλογή με Τουρνουά** (tournament selection), όπου για την επιλογή του κάθε γονέα επιλέγεται τυχαία ένας υποπληθυσμός μέσα στον οποίο ο καλύτερος κερδίζει και συμμετέχει στην αναπαραγωγή.

Στα προγράμματα που ακολουθούν (SCS-CCS) χρησιμοποιήθηκε επιλογή γονέων με τον *Τροχό της Ρουλέτας* (**Roulette Wheel Parent Selection**), που είναι ένα αντιπροσωπευτικό είδος αναλογικής επιλογής. Στην μέθοδο αυτή ο γονέας επιλέγεται με πιθανότητα ανάλογη της ποιότητας του. Πήρε την ονομασία της επειδή θυμίζει το γύρισμα του τροχού της ρουλέτας στην οποία το εμβαδό κάθε κομματιού της, αντιστοιχίζεται στην ποιότητα των γονέων (βλ. σχήμα 4).



Γονέας	Χρωμόσωμα	Ποιότητα επί %
1.	0001	39%
2.	0010	12%
3.	0011	31%
4.	0100	18%
Σύνολο		100%

Σχήμα 4. Επιλογή με τον Τροχό της Ρουλέτας

Ανασυνδυασμός των Λύσεων-Διασταύρωση

Είναι καιρός να δούμε τους γενετικούς τελεστές πιο αναλυτικά. Ένας γενετικός τελεστής είναι μια διαδικασία που προσομοιάζει την αντίστοιχη διαδικασία του ζευγαρώματος των βιολογικών ειδών , στην ουσία της ανάμιξης των χρωμοσωμάτων κατά την φάση της αναπαραγωγής. Κύριος και πιο σημαντικός γενετικός τελεστής

είναι ο *τελεστής διασταύρωσης* (crossover). Ο τελεστής αυτός συνδυάζει τα χαρακτηριστικά γνωρίσματα των ατόμων που έχουν επιλεγεί σύμφωνα με τις μεθόδους που έχουν περιγραφεί στην προηγούμενη παράγραφο, ώστε να δημιουργηθούν καινούρια άτομα που θα πάρουν την θέση των γονέων τους στον πληθυσμό. Έτσι τα γενετικά χαρακτηριστικά μεταφέρονται από γενιά σε γενιά και λόγω της αρχής της επιβίωσης των καλύτερων ο αλγόριθμος συγκλίνει προς την βέλτιστη ή βέλτιστες λύσεις.

Οι πιο συνηθισμένοι τρόποι διασταύρωσης είναι οι εξής:

- **Διασταύρωση ενός σημείου (One Point Crossover)**. Έστω ότι έχουν επιλεγεί δύο άτομα για αναπαραγωγή τα οποία είναι κωδικοποιημένα όχι σε δυαδική μορφή αλλά με τα εξής 5 γονίδια:

  $$A= \{1, 2, 3, 4, 5\} \text{ και } B= \{\alpha, \beta, \gamma, \delta, \varepsilon\}$$

  Σύμφωνα με την διασταύρωση ενός σημείου τα παραγόμενα άτομα θα είναι τα έξης:

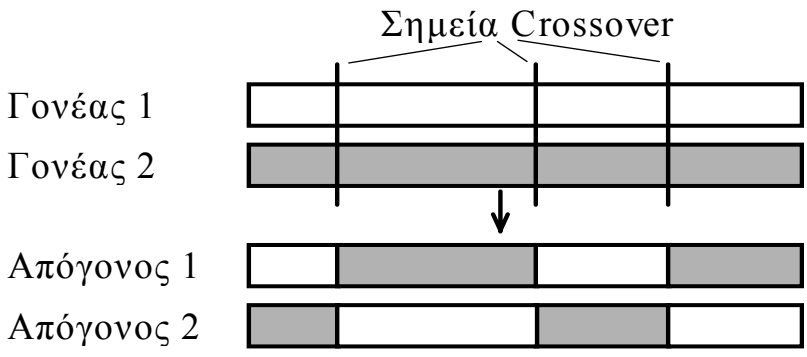
Γονείς		Απόγονοι
1, 2, 3, | 4, 5		1, 2, 3, | δ, ε
	→	
α, β, γ, | δ, ε		α, β, γ, | 4, 5

  Η κάθετη γραμμή είναι το σημείο διασταύρωσης. Στην συγκεκριμένη περίπτωση το σημείο 3.

- **Διασταύρωση πολλών σημείων (Multi-Point Crossover)**. Η περίπτωση αυτή είναι μια πιο γενική, όπου έχουμε περισσότερα σημεία κοπής/διασταύρωσης. Εδώ οι γονείς χωρίζονται σε περισσότερα από 2 τμήματα και οι απόγονοι παράγονται με εναλλαγή των τμημάτων των γονέων.

- **Ομοιόμορφη Διασταύρωση (Uniform Crossover)**. Μια πιο ειδική περίπτωση είναι αυτή της ομοιόμορφης διασταύρωσης όπου κάθε γονίδιο έχει την ίδια πιθανότητα να ληφθεί είτε από τον ένα είτε από τον άλλο γονέα.

Είναι φανερό ότι ο πιο αποδοτικός τρόπος διασταύρωσης είναι αυτός της ομοιόμορφης. Παρακάτω φαίνεται σχηματικά πως λειτουργεί η διασταύρωση:



Σχήμα 5. Ο τελεστής Διασταύρωσης (Crossover) ανασυνδυάζει τις κωδικοποιημένες λύσεις.

Είναι σημαντικό να αναφερθεί ότι ο τελεστής αυτός δεν εφαρμόζεται πάντα. Υπάρχει μια πιθανότητα για να εφαρμοστεί που ονομάζεται *πιθανότητα διασταύρωσης* (crossover probability). Η πιθανότητα αυτή λαμβάνει τιμές στο διάστημα [0,1], με συνήθεις τιμές από 0.7 έως 0.9 . Έτσι κατά την διάρκεια του ΓΑ παράγεται ένας τυχαίος αριθμός όπου αν είναι μικρότερος ή ίσος της πιθανότητας διασταύρωσης τότε τα επιλεγμένα άτομα-γονείς ανασυνδυάζουν τα γενετικά χαρακτηριστικά τους, αλλιώς αν είναι μεγαλύτερος στην επόμενη γενιά περνάνε οι ίδιοι οι γονείς και τα γενετικά τους χαρακτηριστικά παραμένουν αναλλοίωτα.

Παρόλο που ο τελεστής διασταύρωσης δημιουργεί καινούργιες λύσεις δεν παράγει καινούργια πληροφορία. Αν για παράδειγμα ανασυνδυαστούν δύο όμοιοι γονείς το αποτέλεσμα θα είναι ίδιο με τις αρχικές λύσεις-άτομα.

Μετάλλαξη

Όπως είπαμε ο μηχανισμός της διασταύρωσης δεν παράγει καινούργια πληροφορία για αυτό χρησιμοποιείται ένας άλλος τελεστής που ονομάζεται τελεστής *μετάλλαξης* (mutation). Ο τελεστής αυτός παράγει τυχαιότητα και ευελιξία στον ΓΑ και εφαρμόζεται επίσης με κάποια πιθανότητα την *πιθανότητα μετάλλαξης* (mutation probability). Η παράμετρος αυτή παίρνει τιμή κοντά στο 0.01 ανά bit (ή γονίδιο). Βέβαια η εφαρμογή της δεν μας εγγυάται ότι η τροποποίηση των ατόμων με τον τελεστή αυτό θα μας δώσει καλύτερες λύσεις αλλά σίγουρα θα μας δώσει την ευκαιρία να εξερευνήσουμε όλο τον χώρο λύσεων χωρίς να αποκλείσουμε κάποιο τμήμα του.

Παρακάτω θα αναφέρουμε δύο τρόπους εφαρμογής αυτού του τελεστή.

1. Αν για παράδειγμα σε έναν γενετικό αλγόριθμο που χρησιμοποιεί δυαδική κωδικοποίηση έχουμε το προς μετάλλαξη άτομο των 10 bits,

0011000101

Διατρέχουμε το κάθε bit ξεχωριστά και παράγουμε έναν τυχαίο αριθμό. Αν είναι μικρότερος από την πιθανότητα μετάλλαξης τότε το αντίστοιχο bit θα μεταλλαχθεί. Στην περίπτωση της δυαδικής μορφής που έχουμε και στο παράδειγμα μας θα ισχύουν οι εξής μετατροπές:

$$0 \rightarrow 1 \quad \text{και} \quad 1 \rightarrow 0$$

Αν λοιπόν υποθέσουμε ότι για το bit στη θέση τρία έχουμε ίση ή μικρότερη πιθανότητα από αυτή της μετάλλαξης τότε το άτομο θα μεταβληθεί σε:

00 0 1000101

2. Ένας άλλος τρόπος είναι να βρούμε το γινόμενο μεταξύ του πλήθους των bits και της πιθανότητας μετάλλαξης. Στη συνέχεια κρατάμε το ακέραιο μέρος από τον παραγόμενο αριθμό και παράγουμε έναν τυχαίο αριθμό που αν είναι μικρότερος ή ίσος του δεκαδικού μέρους τότε το ακέραιο μέρος το αυξάνουμε κατά ένα. Ο ακέραιος δηλώνει πόσα bit θα μεταλλαχθούν. Για να βρούμε ποιες είναι οι θέσεις των bit που θα μεταλλαχθούνε παράγουμε τόσους τυχαίους αριθμούς όση είναι και η τιμή του ακεραίου.

Ελιτισμός

Κάτι που δεν αναφέρθηκε μέχρι τώρα αλλά χρήζει μεγάλης σημασίας για έναν ΓΑ είναι ο μηχανισμός της *Επιλεκτικότητας ή Ελιτισμού* (Elitism). Συνίσταται στο να διατηρεί την καλύτερη λύση κάθε γενιάς και να την περνάει στην επόμενη, ανέπαφη. Ο Ελιτισμός είναι σημαντικός σε έναν γενετικό αλγόριθμο λόγω του ότι οι δύο προηγούμενοι τελεστές μπορεί να μεταβάλλουν την απόδοση ενός ατόμου (που μπορεί να τυγχάνει να είναι η καλύτερη λύση της τρέχουσας γενιάς) αρνητικά είτε με τον ανασυνδυασμό είτε με την τυχαία αλλαγή κάποιου bit. Μ' αυτόν τον τρόπο καθορίζεται σίγουρα ότι μέσα στον πληθυσμό θα υπάρχει μία λύση με απόδοση ίση ή μεγαλύτερη από την καλύτερη λύση της προηγούμενης γενιάς. Έτσι ο ΓΑ παράγει σε κάθε γενιά ένα άτομο λιγότερο αφού το τελευταίο άτομο είναι αυτό που προκύπτει από τον ελιτισμό. Τέλος, να σημειωθεί ότι ο Ελιτισμός έχει εφαρμοστεί στα προγράμματα.

Άλλοι Γενετικοί Τελεστές

Υπάρχουν βέβαια και παραλλαγές των τελεστών που είδαμε μέχρι τώρα. Για τον τελεστή διασταύρωσης:

Η αντιστροφή (Inversion), όπου επιλέγεται κάθε άτομο ξεχωριστά και πάνω σ' αυτό παίρνουμε δύο σημεία με ομοιόμορφη τυχαία κατανομή. Ύστερα αναδιατάσσουμε αντίστροφα τα bit που βρίσκονται ανάμεσα στα δύο σημεία κοπής του γονότυπου. Ο τελεστής αυτός έχει κυρίως εφαρμογή σε προβλήματα διάταξης. Για παράδειγμα:

Επιλεγμένος Γονότυπος: 1 2 3 | 4 5 6 | 7 8 9

Σημεία κοπής

Παραγόμενος Γονότυπος: 1 2 3 | 6 5 4 | 7 8 9

Ανασυνδυασμός Μερικού Ταιριάσματος (Partially Matched Crossover PMX). Ο τελεστής αυτός εφαρμόζεται σε δύο επιλεγμένους γονείς. Σε κάθε γονότυπο λαμβάνονται δύο σημεία κοπής. Μέσα στην περιοχή αυτή το κάθε bit από τον έναν γονότυπο αντιστοιχίζεται βάση της θέσης του με το αντίστοιχο bit στον άλλο

γονότυπο. Τέλος αλλάζουμε τα bit αναμεταξύ τους όχι μόνο ανάμεσα στα σημεία κοπής άλλα σε όλο το μήκος του γονότυπου. Για παράδειγμα:

Γονότυπος 1: 5 6 4 | 9 1 2 | 7 3 8
Γονότυπος 2: 2 1 3 | 4 7 6 | 5 8 9

Σημεία κοπής

Απόγονος 1: 5 2 9 | 4 7 6 | 1 3 8
Απόγονος 1: 6 7 3 | 9 1 2 | 5 8 4

Άλλοι σημαντικοί ανασυνδυαστικοί τελεστές είναι:

- Ανασυνδυασμός Διάταξης (Order Crossover OX)
- Κυκλικός Ανασυνδυασμός (Cycle Crossover CX)
- Ανασυνδυασμός Ακμών (Edge Recombination ER)

Όσον αφορά τον τελεστή μετάλλαξης, μερικές παραλλαγές είναι:

- **Μετάλλαξη τυχαίου χρωμοσώματος (Random Chromosome)**. Εδώ επιλέγεται ένα χρωμόσωμα τυχαία και επανακαθορίζεται μέσα από το πεδίο ορισμού του.

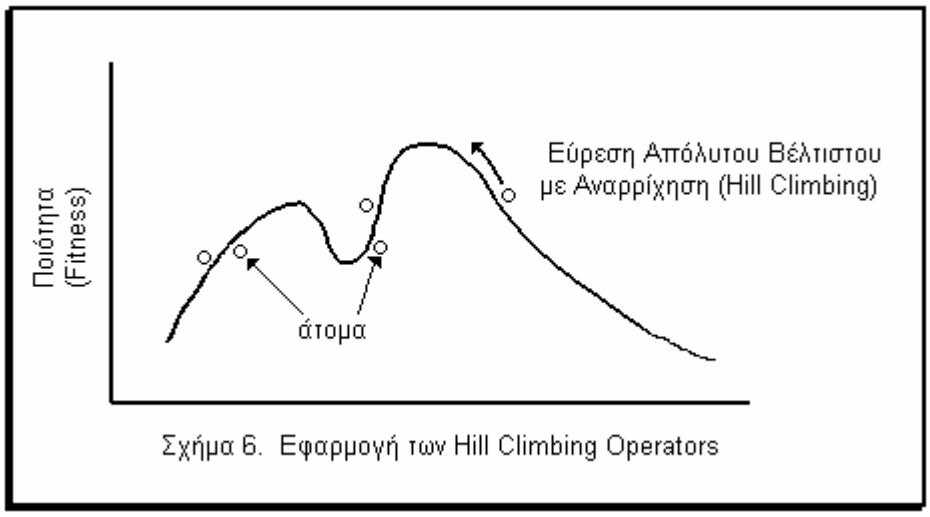
- **Μετάλλαξη Τυχαίου Γονότυπου (Random Genotype)**. Ολόκληρος ο γονότυπος επανακαθορίζεται όπως ένας γονότυπος καθορίζεται κατά την φάση αρχικοποίησης του πληθυσμού.

- **Μετάλλαξη Εναλλαγής χρωμοσωμάτων (Swap Chromosome)**. Χρησιμοποιείται σε προβλήματα με πολλές μεταβλητές. Εδώ η διαδικασία που λαμβάνει χώρα είναι η ανταλλαγή δυο χρωμοσωμάτων μεταξύ τους.

- **Μετάλλαξη Εναλλαγής Bits/Τμημάτων Χρωμοσωμάτων**. Λειτουργεί παρόμοια με τον παραπάνω τελεστή μόνο που αντί να εναλλάσσει ολόκληρο το χρωμόσωμα με ένα άλλο, εναλλάσσει ένα ή περισσότερα bit του συγκεκριμένου χρωμοσώματος.

Όλες οι παραπάνω παραλλαγές του ανασυνδυαστικού τελεστή και του τελεστή μετάλλαξης αποτελούν ειδικές μορφές και χρησιμοποιούνται σε συγκεκριμένα προβλήματα (Problem Specific), ώστε να επιτύχουν βελτιστοποίηση του ΓΑ για γρηγορότερη σύγκλιση προς τις καλύτερες λύσεις.

Τελεστές Αναρρίχησης (Hill Climbing Operators). Οι γενετικοί αλγόριθμοι, λόγω της δομής τους πολλές φορές είναι αναγκασμένοι να επιτρέψουν να περάσει ένα μεγάλο πλήθος γενεών ώσπου να συγκλίνουν προς το βέλτιστο. Για τον λόγο αυτό συνεργάζονται πολλές φορές με διάφορες μεθόδους. Μια τέτοια συνεργασία είναι και αυτή των γενετικών αλγορίθμων με τους τελεστές αναρρίχησης οι οποίοι κατευθύνονται από την τοπική κλίση της συνάρτησης ποιότητας (σχήμα 6). Αυτό που συμβαίνει στον παραπάνω συνδυασμό, είναι πως η καλύτερη λύση κάθε γενιάς στέλνεται στην μέθοδο αυτή. Η εφαρμογή της μεθόδου συνίσταται ώστε να εξερευνηθεί καλύτερα η "γειτονιά" του χώρου λύσεων του καλύτερου ατόμου. Μετά την εκτέλεση της μεθόδου, και την μικρή μεταβολή που επέρχεται, η λύση επαναξιολογείτε. Αν η παραγόμενη είναι καλύτερη σε ποιότητα από την αρχική λύση τότε εισάγεται στον πληθυσμό στην θέση της αρχικής, αλλιώς κρατάμε την αρχική λύση και είτε δοκιμάζεται μια νέα μεταβολή, είτε η διαδικασία αυτή σταματά.



Σχήμα 6. Εφαρμογή των Hill Climbing Operators

Μερικοί σημαντικοί τελεστές αναρρίχησης είναι:

- **Τελεστής Αναρρίχησης Μετάλλαξης Φαινομένου (Mutation Phenotype HCO)**

- **Συνδυαστικός Τελεστής Αναρρίχησης (Swap Bit HCO)**
- **Τελεστής Συμπλήρωσης Κενών (Fill Gap HCO)**

Αντικατάσταση Πληθυσμού

Εφόσον έχουν παραχθεί και αξιολογηθεί οι απόγονοι υπάρχει άλλη μια διεργασία που λαμβάνει μέρος στους γενετικούς αλγόριθμους. Θα πρέπει λοιπόν αυτές οι καινούργιες λύσεις να εισέλθουν στον πληθυσμό. Για τον λόγο αυτό έχουν αναπτυχθεί διάφορες μέθοδοι αντικατάστασης των γονέων από τους απογόνους (Γενεαλογική Αντικατάσταση) όπως:

- **Ολική Αντικατάσταση (Total Replacement)**. Ο πληθυσμός των απογόνων διαδέχεται πλήρως τον πληθυσμό των γονέων.
- **Μερική Αντικατάσταση (Partial Replacement)**. Ένα τμήμα των γονέων (συνήθως οι χειρότεροι) αντικαθιστάτε από ένα ισοδύναμο τμήμα των απογόνων (συνήθως τους καλύτερους). Το τμήμα αυτό καθορίζεται με κάποιο ποσοστό.
- **Αντικατάσταση Σταθερής Κατάστασης (Steady State Replacement).** Στην περίπτωση αυτή όταν παραχθεί μια λύση αντικαθιστά μία υπάρχουσα (συνήθως την χειρότερη του τρέχοντα πληθυσμού).

Τερματισμός του Γενετικού Αλγόριθμου

Ως πότε θα "τρέχει" ένας ΓΑ ή αλλιώς ως πότε θα διαδέχεται η μία γενιά την άλλη;

Ο τερματισμός των ΓΑ γίνεται με βάση την ποιότητα των λύσεων ή βάση του αριθμού των γενεών. Στην πρώτη περίπτωση όπου σαν κριτήριο έχουμε την ποιότητα, ο αλγόριθμος σταματά όταν οι βέλτιστες λύσεις έχουν συγκλίνει σε μια επιθυμητή τιμή ποιότητας. Δεν είναι απαραίτητο μια μόνο λύση να φτάσει αυτήν την ποιότητα αλλά ακόμη και ένα μεγάλο ποσοστό του πληθυσμού να συγκεντρωθεί σε ένα τοπικό βέλτιστο. Στην περίπτωση που ο αριθμός των γενεών είναι το κριτήριο τερματισμού η εξέλιξη σταματά όταν φτάσουμε ένα απόλυτο όριο γενεών (π.χ. 1000 γενιές) ή επίσης μπορεί να τροποποιηθεί ώστε όταν περάσει ένας αριθμός γενεών και δεν παραχθεί καλύτερη λύση, η εξέλιξη του ΓΑ να σταματά πάλι. Βέβαια θα μπορούσαν

να αναπτυχθούν κι άλλα κριτήρια τερματισμού ανάλογα με την φύση του προβλήματος.

Με τα όσα αναλύσαμε μέχρι τώρα θα μπορούσαμε να αναπαραστήσουμε σε ένα διάγραμμα ροής την εξέλιξη ενός γενετικού αλγόριθμου όπως φαίνεται παρακάτω (σχήμα 7).

Σχήμα 7. Λογικό Διάγραμμα ενός Γενετικού Αλγόριθμου

Ιδιότητες Γενετικών Αλγορίθμων

Οι γενετικοί αλγόριθμοι είναι όπως είπαμε αλγόριθμοι αναζήτησης και βελτιστοποίησης. Τα χαρακτηριστικά που τους κάνουν τόσο χρήσιμους και τους παρέχουν την ιδιότητα της ευρωστίας είναι τα εξής:

1. **Επεξεργάζονται κωδικοποιημένες λύσεις (γονότυπους) και όχι την ίδια την λύση**

2. **Έχουν την δυνατότητα της παράλληλης και ταυτόχρονης διερεύνησης των υποψήφιων λύσεων του προβλήματος**. Σε αντίθεση με άλλες μεθόδους που επεξεργάζονται μια προς μία της λύσης με κίνδυνο η αναζήτηση του βέλτιστου να εγκλωβιστεί σε ένα τοπικό ακρότατο, οι γενετικοί με την δυνατότητα της ταυτόχρονης επεξεργασίας πολλών λύσεων έχουν μεγαλύτερη πιθανότητα να φτάσουν σε ένα απόλυτο τοπικό μέγιστο. Στην πορεία αυτή βοηθάνε ακόμη περισσότερο και οι τελεστές αναρρίχησης που αναφέρθηκαν πιο στην προηγούμενη ενότητα.
3. **Όλες οι απαραίτητες πληροφορίες που χρειάζονται πηγάζουν από την συνάρτηση ποιότητας**. Εδώ όμως δημιουργείται η απορία μήπως αποκλείουμε έτσι χρήσιμη πληροφορία από άλλα στοιχεία. Για τον λόγο αυτό υπάρχουν και γενετικοί αλγόριθμοι που λαμβάνουν και αξιοποιούν και από άλλα μέσα πληροφορία.
4. **Λειτουργούν με τρόπο πιθανοκρατικό, στοχαστικό, και ευριστικό**. Χρησιμοποιούν δηλαδή την πιθανότητα για να έχουν μεγαλύτερη τυχαιότητα και ευελιξία και να διερευνούν έτσι ανεξερεύνητες περιοχές του χώρου λύσεων.

Πλεονεκτήματα-Μειονεκτήματα

Πλεονεκτήματα:

- Μπορούν να ανιχνεύουν ανεξερεύνητες περιοχές (exploration), και να εκμετάλλευονται τις καλύτερες λύσεις (exploitation).
- Έχουν την δυνατότητα να ενσωματώνουν κι άλλες μεθόδους στη δομή τους.
- Βρίσκουν εφαρμογή σε πολλούς τομείς.
- Παρέχουν πολλές λύσεις.

Μειονεκτήματα

- Πολλές φορές χρειάζονται αρκετό χρόνο μέχρι να συγκλίνουν στο βέλτιστο.
- Για μεγαλύτερη αποδοτικότητα θα πρέπει κάθε φορά να καθορίζονται οι διάφορες παράμετροι.
- Χρειάζονται αρκετή υπολογιστική ισχύ.

Γιατί λειτουργούν οι Γενετικοί Αλγόριθμοι;

Με τα όσα αναπτύξαμε κατά την ανάλυση αυτού του κεφαλαίου γεννιέται η απορία γιατί οι γενετικοί αλγόριθμοι συγκλίνουν προς το βέλτιστο; τι είναι αυτό που τους δίνει την ιδιότητα της ευρωστίας; Για να μπορέσουμε να δώσουμε μια απάντηση, θα πρέπει στο σημείο αυτό να εισάγουμε την έννοια των *σχημάτων* και γενικά να αναλύσουμε την Θεωρία των Σχημάτων (**Schema Theorem**).

Υποθέτοντας λοιπόν πως έχουμε ένα πληθυσμό από άτομα-λύσεις και παρατηρώντας τα, θα βλέπαμε πως υπάρχουν κάποιες ομοιότητες ανάμεσα σε κάποια άτομα. Υπάρχουν δηλαδή θέσεις στον γονότυπο κάποιων ατόμων που έχουν την ίδια τιμή. Αυτά τα σύνολα των ατόμων που έχουν κοινά στοιχεία παρουσιάζουν συστηματικά καλές ή κακές ποιότητες. Συμπεραίνουμε έτσι πως υπάρχει ένας συσχετισμός ανάμεσα στα άτομα καλών λύσεων αλλά και μεταξύ ποιότητας και λύσεων. Πριν ακόμη δώσουμε τον ορισμό του σχήματος να εισάγουμε στο αλφάβητο μας και το σύμβολο μπαλαντέρ * (don't care symbol) και να πούμε πως σαν σχήμα (H) μέσα σ' έναν γενετικό αλγόριθμο που κωδικοποιείται δυαδικά, ορίζεται ένα πρότυπο string που αναπαριστά όλες τις συμβολοσειρές οι οποίες έχουν όλα τα bit ίδια εκτός τις θέσεις που υπάρχει ο χαρακτήρας *.

Για παράδειγμα, στα παρακάτω άτομα μήκους 10 bit:

1. 0101110010
2. 1101110010
3. 1111110010
4. 0111110010

το σχήμα που τα αναπαριστά είναι το *1*1110010. Επίσης το σχήμα 0101110010 αναπαριστά μία μόνο λύση, την 0101110010 , και το σχήμα ********** αναπαριστά όλες τις λύσεις που έχουν μήκος 10 bit.

Ο αριθμός των σχημάτων που υπάρχουν σ' έναν πληθυσμό με γονότυπους μήκους ℓ είναι 3^{ℓ}, αρκετά μεγαλύτερος από τον αριθμό των λύσεων που είναι 2^{ℓ}. Κάθε γονότυπος όμως περιέχει 2^{ℓ} σχήματα αφού κάθε θέση παίρνει είτε '*' είτε κανονική τιμή. Είναι προφανές ότι κάθε σχήμα που περιέχει έναν αριθμό ν συμβόλων μπαλαντέρ '*' , μπορεί να αναπαραστήσει 2^{v} συμβολοσειρές. Έτσι αν ο πληθυσμός έχει M άτομα, ο δυνατός αριθμός των σχημάτων που μπορούν να υπάρξουν σ' αυτόν είναι:

$$2^{\ell} \leq \text{σχήματα} \leq M * 2^{\ell}$$

Στο σημείο αυτό θα πρέπει να αναφέρουμε δύο μεγέθη που χαρακτηρίζουν τα σχήματα, είναι: η *τάξη* (order) και το *μήκος ορισμού* (defining length).

Τάξη ενός σχήματος o(H), ορίζεται ο αριθμός των σταθερών συμβολών ή αλλιώς των θέσεων που δεν περιέχουν το σύμβολο μπαλαντέρ '*' (wildcard symbol). Δηλαδή είναι η διαφορά του μήκους του σχήματος μείον το πλήθος των αδιάφορων συμβόλων.

Έστω τα ακόλουθα σχήματα μήκους 10 bits:

$$H_1 = \{*01*00***1\},$$
$$H_2 = \{******01**\},$$
$$H_3 = \{100101**10\},$$

οι τάξεις των συμβόλων θα είναι: $o(H_1) = 5$, $o(H_2)=2$, $o(H_3)= 8$. Από τα παραπάνω σχήματα καταλαβαίνουμε πως αυτό που έχει τα περισσότερα σταθερά σύμβολα {0,1} είναι πιο ειδικό ή με άλλα λόγια, το λιγότερο γενικό αφού αναπαριστά μικρότερο πλήθος συμβολοσειρών. Το μέγεθος αυτό παίζει σημαντικό ρόλο στην επιβίωση του σχήματος κατά την διάρκεια της μετάλλαξης αφού η πιθανότητα να τροποποιηθεί ένα bit είναι p_m και άρα η πιθανότητα να επιβιώσει 1- p_m. Οπότε η πιθανότητα να επιβιώσει ένα σχήμα δεδομένου ότι η τάξη o(H) εκφράζει τον αριθμό των σταθερών συμβόλων ισούται με:

$$p_s= (1- p_m)^{o(H)}$$

Επειδή όμως η πιθανότητα μετάλλαξης είναι σχετικά πολλή μικρή $p_m<<1$ (συνήθως ισούται με 0.01) , η πιθανότητα επιβίωσης προσεγγίζεται από την σχέση:

$$p_s= 1- o(H)\cdot p_m$$

Σαν *μήκος ορισμού* ενός σχήματος δ(H), ορίζουμε την απόσταση μεταξύ του πρώτου και του τελευταίου σταθερού bit. Για τα παραπάνω σχήματα το οριζόμενο μήκος θα είναι:

$$\delta(H_1)= 10-2 = 8$$
$$\delta(H_2)= 8-7 = 1$$
$$\delta(H_3)= 10-1 = 9$$

Στην περίπτωση που υπάρχει ένα μόνο σταθερό bit μέσα στον γονότυπο το οριζόμενο μήκος είναι μηδέν.

Το μέγεθος αυτό παίζει σημαντικό ρόλο στην επιβίωση του σχήματος κατά την διάρκεια της διασταύρωσης. Αυτό φαίνεται αν αναλογιστούμε το παρακάτω παράδειγμα.

Έστω το άτομο A = 0111000 με μήκος 7 και τα παρακάτω σχήματα:

$$H_1 = * \ 1 \ * \ * \ * \ 0$$

$$H_2 = * \ * \ * \ 1 \ 0 \ * \ *$$

Η τάξη και το οριζόμενο μήκος για τα δύο σχήματα είναι αντίστοιχα:

$$o(H_1) = 2 \text{ και } \delta(H_1) = 5$$

$$o(H_2) = 2 \text{ και } \delta(H_2) = 1$$

Ενώ και τα δύο σχήματα έχουν την ίδια τάξη, είναι φανερό πως το πρώτο σχήμα μπορεί να διαμελιστεί πιο εύκολα.

Σε μια εφαρμογή του τελεστή Crossover με σημείο κοπής τη θέση μεταξύ του $3^{ου}$ και $4^{ου}$ ψηφίου θα βλέπαμε πως μόνο το δεύτερο σχήμα θα διατηρούνταν.

$$A = 0 \ 1 \ 1 \mid 1 \ 0 \ 0 \ 0$$

$$H_1 = * \ 1 \ * \mid * \ * \ * \ 0$$

$$H_2 = * \ * \ * \mid 1 \ 0 \ * \ *$$

Με βάση αυτά υπολογίζεται η πιθανότητα επιβίωσης του σχήματος αλλά και της "καταστροφής" αυτού, συναρτήσει του μήκους ορισμού και του ολικού μήκους του. Συγκεκριμένα:

$p_{d1}(H_1) = \delta(H_1) / (\ell - 1) = 5/6$, Πιθανότητα καταστροφής H_1

$p_{s1}(H_1) = 1 - p_{d1}(H_1) = 1/6$, Πιθανότητα επιβίωσης H_1

$p_{d2}(H_2) = \delta(H_2) / (\ell - 1) = 1/6$, Πιθανότητα καταστροφής H_2

$p_{s2}(H_2) = 1 - p_{d2}(H_2) = 5/6$, Πιθανότητα επιβίωσης H_2

όπου "ℓ" στις παραπάνω σχέσεις είναι το ολικό μήκος του σχήματος. Άρα προκύπτει πως με πιθανότητα Crossover 'p_c' , η επιβίωση ενός σχήματος H θα είναι:

$$p_s(H) \geq 1 - p_c \times \delta(H_1) / (\ell - 1)$$

Στη συνέχεια θα προσπαθήσουμε να εξάγουμε μια σχέση που να δείχνει την εκτίμηση του αριθμού των ατόμων m(H,t+1) που αναπαριστώνται από ένα σχήμα H στη γενιά t+1 και εξαρτάται από την αντίστοιχη τιμή της προηγούμενης γενιάς (t).

Αν για κάποιο σχήμα Η υπάρχει ένας υποπληθυσμός συμβολοσειρών p που ταιριάζει μ' αυτό την χρονική στιγμή t, τότε η μέση απόδοση αυτού του υποπληθυσμού θα δίδεται από την σχέση:

$$f(H) = \sum_{i=1}^{p} fi \; / \; p$$

Όμοια για ολόκληρο τον πληθυσμό A(t) συνολικού πλήθους M συμβολοσειρών, η μέση τιμή καταλληλότητας του για την γενιά t θα δίνεται από τον αντίστοιχο τύπο:

$$f(A) = \sum_{i=1}^{M} fi \; / \; M$$

Αφού επιλεγούν τα M άτομα του πληθυσμού ο εκτιμώμενος αριθμός των ατόμων του σχήματος Η θα είναι:

$$m(H, t+1) = M \times m(H, t) \times \frac{f(H)}{\sum_{i=1}^{M} fi}$$

Από τις δύο τελευταίες σχέσεις συνεπάγεται ο ακόλουθος τύπος που αναφέρεται στον ανασυνδυασμό:

$$m(H, t+1) = m(H, t) \times \frac{f(H)}{f(A)}$$

Συνδυάζοντας στον παραπάνω τύπο τους τύπους που αφορούσαν την διασταύρωση και την μετάλλαξη καταλήγουμε στην ζητούμενη σχέση:

$$m(H, t+1) \geq m(H, t) \times \frac{f(H)}{f(A)} \times \left(1 - pc \times \frac{\delta(\mathrm{H})}{\ell - 1} - o(H) \times pm\right)$$

Τα συμπεράσματα που προκύπτουν από αυτήν την σχέση είναι ότι κάθε σχήμα εξαρτάται από τον λόγο της μέσης απόδοσης των συμβολοσειρών που ταιριάζουν με το εκάστοτε σχήμα προς την μέση τιμή της καταλληλότητας των συμβολοσειρών όλου του πληθυσμού. Επίσης δηλώνει πως σχήματα με απόδοση μεγαλύτερη της μέσης τιμής του συνολικού πληθυσμού και με μικρό οριζόμενο μήκος και μικρή τάξη

θα διαιωνίζονται με ρυθμό εκθετικά αυξανόμενο σε διαδοχικές γενιές ενός γενετικού αλγορίθμου, ενώ αντίθετα σχήματα ποιότητας μικρότερης του μέσου όρου θα έχουν φθίνοντα ρυθμό αντιπροσώπων στις επόμενες γενιές. Αυτό αποτελεί και το Θεώρημα Σχημάτων (Schema Theorem). Τα σχήματα λοιπόν και οι σχέσεις τους με την ποιότητα είναι αυτά που εξελίσσουν έναν γενετικό αλγόριθμο προς το βέλτιστο.

Υπόθεση Δομικών Στοιχείων (Building Block Hypothesis)

Αυτό που συμβαίνει στην ουσία είναι πως ο γενετικός χρησιμοποιεί σχήματα μικρής τάξης τα οποία μπορούν και περνούν την πληροφορία μέσα από την διασταύρωση. Σε αυτά τα μικρής τάξης σχήματα που καλούνται *Δομικά Στοιχεία* (Building Blocks) οφείλεται η σύγκλιση του αλγορίθμου προς τις καλύτερες λύσεις.

Τέλος αυτά τα μικρά σχήματα επεξεργάζονται παράλληλα, από τον γενετικό αλγόριθμο (implicit parallelism), χωρίς να χρειάζεται άλλο είδος μνήμης πέραν του ίδιου του πληθυσμού.

ΥΛΟΠΟΙΗΣΗ ΑΠΛΟΥ CLASSIFIER SYSTEM

ΣΚΟΠΟΣ

Στόχος του κεφαλαίου αυτού είναι να υλοποιηθεί ένα παράδειγμα ενός απλού Classifier συστήματος χωρίς όμως τον αλγόριθμο κατανομής βαθμών (Bucket Brigade Algorithm). Το σύστημα θα εξελίσσει γενετικά κανόνες (if then rules) για την εύρεση ενός βέλτιστου σετ κανόνων που θα περιγράφει την συνάρτηση εισόδου-εξόδου ενός δυαδικού πολυπλέκτη 4 σε 1 αλλά και 8 σε 1.

Δυαδικοί Πολυπλέκτες

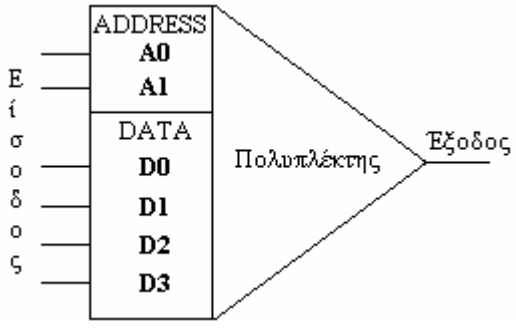
Ένας πολυπλέκτης (multiplexer ή mux) είναι μία συσκευή ή ένα κύκλωμα που επιλέγει ποια κατάσταση (data) θα σταλθεί στην έξοδο του, βάση της διεύθυνσης (address) που βρίσκεται η κάθε κατάσταση. Οι πολυπλέκτες υλοποιούνται με τρανζίστορς. Συνήθως την χρήση του πολυπλέκτη ακολουθεί ο αποπολυπλέκτης (demultiplexer) που εκτελεί την αντίθετη διαδικασία.

Ένας πολυπλέκτης βρίσκει εφαρμογή κυρίως:

- Σε **αναλογικά** και **ψηφιακά** κυκλώματα.
- Στα **δίκτυα επικοινωνίας** (δρομολογητές, μεταγωγείς, γέφυρες…).
- Σε **διατάξεις αποθήκευσης-μνήμες.**

Παρακάτω φαίνονται σχηματικά οι δύο τύποι πολυπλέκτη που χρησιμοποιήθηκαν καθώς και οι πίνακες με τα βέλτιστα σετ κανόνων τους:

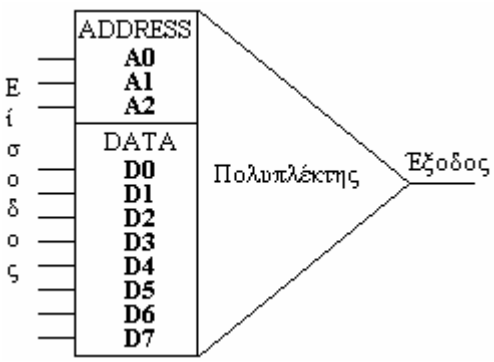
Πολυπλέκτης 4 σε 1:



Κανόνες	Περίγραμμα
00 0### : 0	0 Address/0 Signal
00 1### : 1	0 Address/1 Signal
01 #0## : 0	1 Address/0 Signal
01 #1## : 1	1 Address/1 Signal
10 ##0# : 0	2 Address/0 Signal
10 ##1# : 1	2 Address/1 Signal
11 ###0 : 0	3 Address/0 Signal
11 ###1 : 1	3 Address/1 Signal

Ο πολυπλέκτης 4 σε 1 έχει Address = 2 bit και Data = $2^{Address}$ = 4. Επίσης, η Λίστα Μηνυμάτων είναι χωρητικότητας 1 bit.

Πολυπλέκτης 8 σε 1:



Κανόνες	Περίγραμμα
000 0###### : 0	0 Address/0 Signal
000 1###### : 1	0 Address/1 Signal
001 #0###### : 0	1 Address/0 Signal
001 #1###### : 1	1 Address/1 Signal
010 ##0##### : 0	2 Address/0 Signal
010 ##1##### : 1	2 Address/1 Signal
011 ###0#### : 0	3 Address/0 Signal
011 ###1#### : 1	3 Address/1 Signal
100 ####0### : 0	4 Address/0 Signal
100 ####1### : 1	4 Address/1 Signal
101 #####0## : 0	5 Address/0 Signal
101 #####1## : 1	5 Address/1 Signal
110 ######0# : 0	6 Address/0 Signal
110 ######1# : 1	6 Address/1 Signal
111 #######0 : 0	7 Address/0 Signal
111 #######1 : 1	7 Address/1 Signal

Αντίστοιχα στον πολυπλέκτη 8 σε 1 έχουμε Address = 3 bit και Data = $2^{Address}$ = 8 bit. Κι εδώ, η Λίστα Μηνυμάτων είναι χωρητικότητας 1 bit.

Η ανάλυση που ακολουθεί είναι ίδια και για τις δύο περιπτώσεις αφού πριν την εκτέλεση του προγράμματος το μόνο που χρειάζεται για να καθορίσουμε ποιόν πολυπλέκτη θέλουμε να χρησιμοποιήσουμε, είναι να ορίσουμε την τιμή του Address σε «2» για την περίπτωση του πολυπλέκτη 4 σε 1 και σε «3» για την περίπτωση του 8 σε 1.

Υλοποίηση Προγράμματος

Ήρθε ο καιρός λοιπόν να εφαρμόσουμε όλα όσα αναπτύξαμε στα προηγούμενα κεφάλαια για την υλοποίηση του στόχου μας.

Σαν πρώτο βήμα λοιπόν όπως αναφέρθηκε και στο κεφάλαιο των γενετικών αλγορίθμων ήταν η παραγωγή του αρχικού πληθυσμού. Κάθε άτομο αποτελείται από ένα σετ κανόνων-μηνυμάτων. Το σετ για τον πολυπλέκτη 4 σε 1 είναι ένα σύνολο από 8 Classifiers, ενώ για τον 8 σε 1 είναι 16. Οι τιμές αυτές, τις οποίες θεωρούμε εξαρχής γνωστές, αποτελούν και το βέλτιστο σύνολο των σετ για τις δύο περιπτώσεις μας. Οι κανόνες αυτοί κωδικοποιήθηκαν με βάση το τριαδικό αλφάβητο {0,1,#} για την συνθήκη και το δυαδικό αλφάβητο {0,1} για το μήνυμα. Ως γνωστόν το σύμβολο '#' παίζει διπλό ρόλο είτε δηλαδή του '0' είτε του '1'.

'Όπως αναφέρθηκε και στο κεφάλαιο των *Classifier Systems*, κάθε κανόνας χαρακτηρίζεται από μια «δύναμη». Στο πρόγραμμα αυτό του ψηφιακού πολυπλέκτη, που αποτελεί μια απλή περίπτωση εκμάθησης ενός Classifier συστήματος, δεν χρησιμοποιήθηκε το «Σύστημα Καταμερισμού Βαθμών» (Credit Apportionment System) και κατ' επέκτασιν ο «αλγόριθμος Bucket Brigade». Επομένως η έννοια της δύναμης των κανόνων αποκτάει διαφορετικό νόημα.

Η δύναμη λοιπόν στην περίπτωση αυτή χαρακτηρίζει ολόκληρο το άτομο (σετ κανόνων με βέλτιστο πλήθος κανόνων) και δηλώνει το βαθμό προσαρμοστικότητας του ατόμου στο περιβάλλον. Πιο συγκεκριμένα, ο ΓΑ διατηρεί έναν πληθυσμό από λύσεις. Κάθε λύση αποτελείται από ένα σετ κανόνων. Για να αξιολογήσουμε μία λύση (σετ κανόνων) παρουσιάζουμε στην είσοδο του Classifier System όλους τους δυνατούς συνδυασμούς εισόδων(2^(address+data)). Για κάθε είσοδο το σετ των κανόνων θα αποκριθεί στην έξοδο με ένα 0 ή ένα 1 που μπορεί να είναι σωστό ή λάθος. Ο αριθμός των φορών που το CS θα δώσει σωστή έξοδο λαμβάνεται ως η ποιότητα της συγκεκριμένης λύσης. Για τον πολυπλέκτη 4σε1 ο αριθμός των συνδυασμών των εισόδων είναι 64 συνδυασμοί, οπότε ο μέγιστος αριθμός σωστών απαντήσεων (και άρα μέγιστη τιμή ποιότητας) θα είναι το 64, ενώ η ελάχιστη τιμή είναι προφανώς το 0, ενώ για τον 8σε1 είναι αντίστοιχα 2048 για την μέγιστη τιμή ποιότητας και 0 για την ελάχιστη τιμή. Ο αριθμός αυτός αποτελεί την ποιότητα του ατόμου και σύμφωνα με αυτόν επιλέγονται τα άτομα ώστε να τους εφαρμοστούν οι

γενετικοί τελεστές της διασταύρωσης και της μετάλλαξης ή και διάφοροι άλλοι τελεστές.

Για το πρόγραμμα αυτό υλοποιήθηκε ο γενετικός τελεστής διασταύρωσης πολλών σημείων (Multi-Point Crossover). Επίσης η μετάλλαξη εφαρμόστηκε σύμφωνα με το δεύτερο παράδειγμα που ορίστηκε στο κεφάλαιο των Γενετικών αλγόριθμών στην αντίστοιχη παράγραφο. Και οι δύο αυτοί τελεστές λειτουργούν πιθανοκρατικά, δηλαδή βάση κάποιας πιθανότητας που ορίζεται κατά την εκτέλεση του προγράμματος. Επιπλέον εφαρμόστηκε και ένας τελεστής αναρρίχησης ο οποίος αποτελεί ένα είδος παραλλαγής του Ανασυνδυαστικού Τελεστή Αναρρίχησης (Swap Bit HCO).

Ο πληθυσμός των απογόνων που παράγεται βάση των παραπάνω διαδικασιών είναι ισοπλήθης με τον πληθυσμό των γονέων και η γενεαλογική αντικατάσταση που πραγματοποιείται είναι ολική (Total Replacement).

Όλη αυτή η διαδικασία επανεκτελείται μέχρι την σύγκλιση του αλγορίθμου προς το βέλτιστο.

Σαν κριτήρια λοιπόν για τον τερματισμό χρησιμοποιήθηκαν τρεις περιπτώσεις:

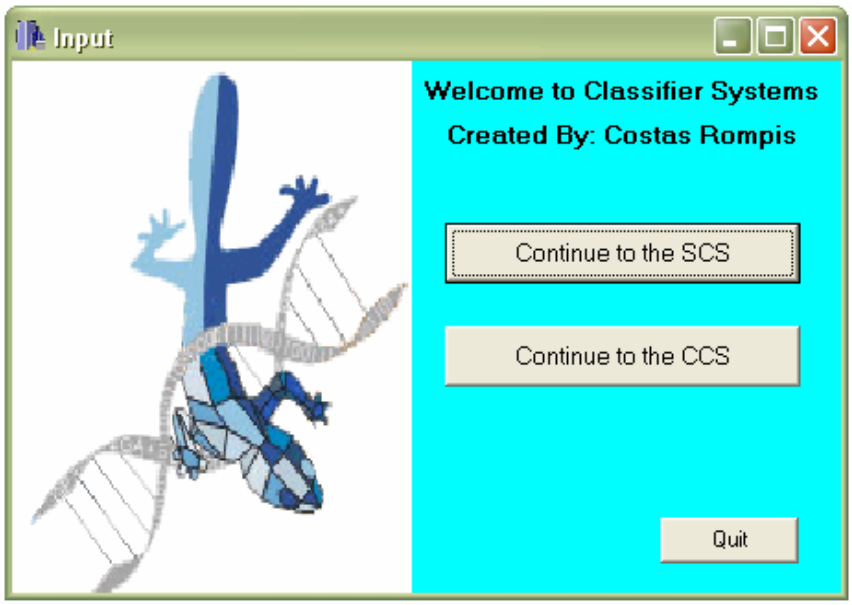
1. Τερματισμός με βάση το πλήθος των γενεών
2. Τερματισμός με βάση την ποιότητα
3. Τερματισμός με βάση και τα δύο παραπάνω κριτήρια.

Το πρόγραμμα αυτό λόγω της έλλειψης του «αλγόριθμου Bucket Brigade» αποτελεί στην ουσία έναν σύνθετο γενετικό αλγόριθμο κι όχι ένα απόλυτο Classifier System. Στο επόμενο κεφάλαιο παρουσιάζεται ένα πιο πολύπλοκο Classifier σύστημα με χρήση και του Bucket Brigade όπου κάθε άτομο-λύση είναι μια ξεχωριστή οντότητα-κανόνας και όχι ένα σετ κανόνων.

ΟΘΟΝΕΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

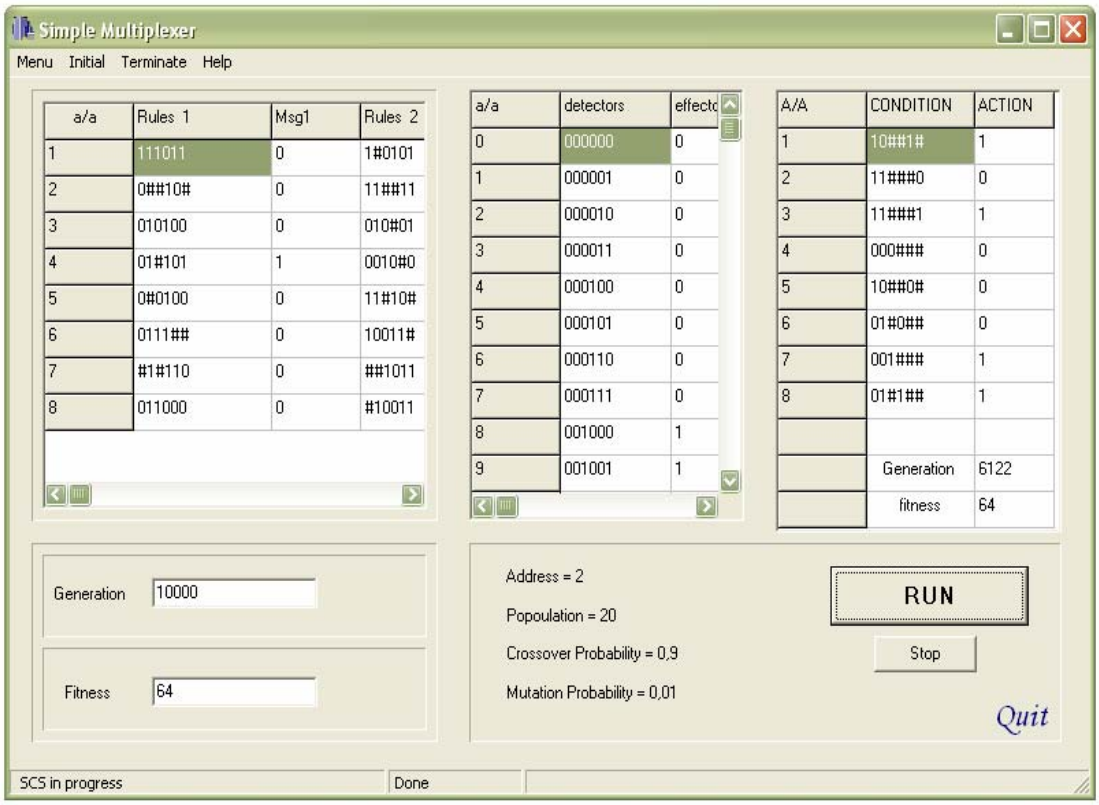
Εισαγωγική Οθόνη Επιλογής

Όπως φαίνεται αποτελείται από τρία Buttons.



Η Κύρια οθόνη του προγράμματος

Το παρακάτω σχήμα αποτελεί την φόρμα του Simple Classifier System:



Παρατηρούνται τα εξής Components:

StringGrid1 (Αριστερά): Σε αυτό εμφανίζεται ο αρχικός παραγόμενος πληθυσμός

StringGrid2 (Μέση): Εδώ εμφανίζονται οι δυνατοί συνδυασμοί εισόδων με τις αντίστοιχες εξόδους

StringGrid3 (Δεξιά): Εδώ εμφανίζεται η καλύτερη λύση κάθε γενιάς.

RUN Button: Ξεκινά την γενετική εξέλιξη του Classifier System.

STOP Button: Σταματά ανά πάσα στιγμή την διαδικασία.

Quit Label: Τερματισμός της εφαρμογής.

Labels (Address, Population, Crossover Probability, Mutation Probability): Δείχνουν τις τιμές που έχουν οριστεί από τον χρήστη.

Main Menu: Εμφανίζει τις επιλογές του προγράμματος που είναι δυνατές για τον χρήστη.

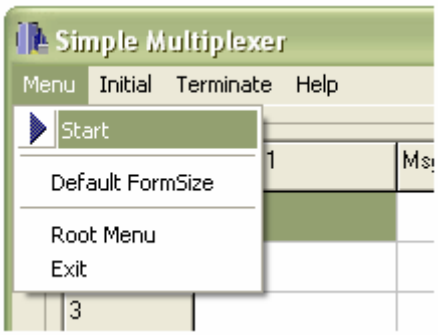
Generation Edit Box: Εδώ εισάγεται το πλήθος των επιθυμητών γενεών.

Fitness Edit Box: Εδώ εισάγεται η τιμή της επιθυμητής ποιότητας σύγκλισης.

Status Bar: Δείχνει την κατάσταση στην οποία βρίσκεται το πρόγραμμα.

MAIN MENU

1. FILE



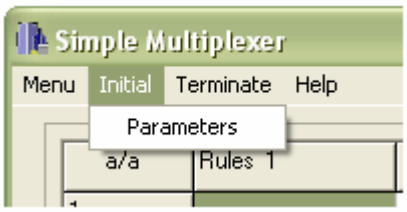
File->Start: Ξεκινά την γενετική εξέλιξη του Classifier System.

File->Default FormSize: Επανακαθορισμός του αρχικού μεγέθους της φόρμας.

File->Root Menu: Επιστροφή στην αρχική φόρμα.

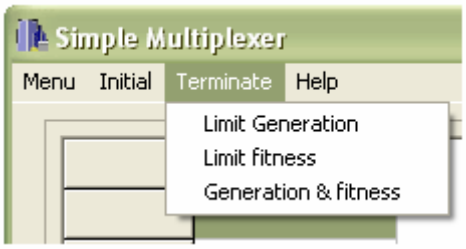
File->Exit: Τερματισμός εφαρμογής.

2. INITIAL



Initial->Parameters: Εμφανίζει την φόρμα Initial.

3. TERMINATE

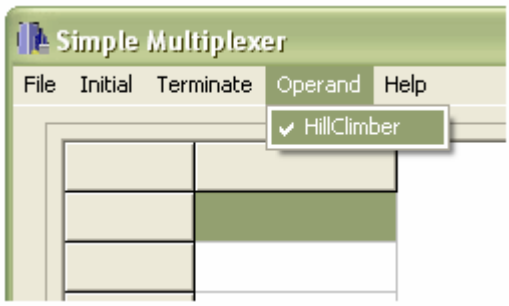


Terminate-> Limit Generation: Τερματισμός με απόλυτο όριο γενεών

Terminate->Limit Fitness: Τερματισμός με απόλυτο όριο ποιότητας.

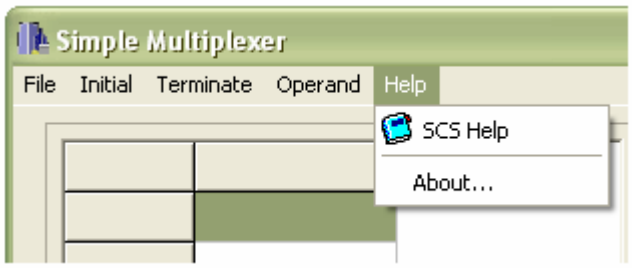
Terminate->Generation & Fitness: Τερματισμός με συνδυασμό και των δύο.

4. OPERAND



Operand-> HillClimber: Ο χρήστης επιλέγει αν στην γενετική εξέλιξη θα συμμετάσχει και ο τελεστής Αναρρίχησης (αποτελεί μια παραλλαγή του Swap Bit HCO).

5. HELP



Help-> SCS Help: Εμφανίζει το Help της εφαρμογής.

Help-> About: Εμφανίζει την φόρμα About.

Η Οθόνη Αρχικοποίησης Τιμών INITIAL

Αποτελείται από τα εξής Components:

Address Edit: Εδώ εισάγετε η τιμή του address των κανόνων.

Population Edit: Εδώ εισάγετε το πλήθος των κανόνων (Population of Classifier Store).

Crossover Points Edit: Εδώ εισάγετε ο αριθμός των σημείων κοπής από τον χρήστη.

Crossover Probability Edit: Εδώ εισάγετε η πιθανότητα διασταύρωσης.

Mutation Probability Edit: Εδώ εισάγετε η πιθανότητα μετάλλαξης.

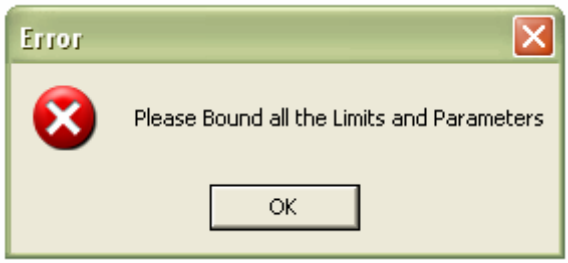
Αν δεν οριστούν τα παραπάνω μεγέθη καθώς και τα όρια τερματισμού τότε δεν μπορούμε να τρέξουμε τον αλγόριθμό και η εφαρμογή μας ενημερώνει με αντίστοιχο μήνυμα για την εισαγωγή των δεδομένων.

Η Οθόνη About

Εδώ εμφανίζεται η Modal φόρμα About και οι πληροφορίες που περιέχονται σ' αυτήν.

ΜΗΝΥΜΑΤΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

ΕΛΛΕΙΠΗ ΑΡΧΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ



Εμφανίζετε όταν εκτελείται το Button RUN ή το File-> Start, χωρίς να έχουν οριστεί όλα τα δεδομένα (Address, Population, Crossover Probability, Mutation Probability, Fitness or Generation).

ΕΠΑΝΑΚΑΘΟΡΙΣΜΟΣ ΤΩΝ ΠΑΡΑΜΕΤΡΩΝ

Παρόμοιο μήνυμα παράγεται για όλα τα δεδομένα στην περίπτωση που αντί για αριθμός πατηθεί κατά λάθος κάποιος χαρακτήρας στο αντίστοιχο Edit Box.

Απόδοση του προγράμματος

Ακολουθεί ένα πείραμα που δείχνει την αποτελεσματικότητα του προγράμματος. Για τον σκοπό αυτό έγινε εκτέλεση του προγράμματος με τα παρακάτω δεδομένα:

Address = 2
Population = 100
Crossover Points= 4 (Διαλέγεται ένα νούμερο από το 1 έως το 55)
Crossover Probability = 1,0
Mutation Probability = 0,01
Generation = 3000

Ο παρακάτω πίνακας δείχνει τα αποτελέσματα της εκτέλεσης για πλήθος 10 φορών:

a/a	Generation	Fitness
1	3000	58
2	808	64
3	3000	61
4	2247	64
5	3000	55
6	2038	64
7	3000	61
8	3000	59
9	3000	53
10	3000	61

Μερικά συμπεράσματα που διεξάγονται από αυτόν τον πίνακα είναι πως ο αλγόριθμος συγκλίνει με μέσο όρο γενεών: 2609 γενεές και με μέσο όρο ποιότητας: 57 ταιριάσματα ποιότητας. Επίσης παρατηρείτε, πως ο αλγόριθμος τερματίζει βάση και του κριτηρίου ποιότητας, και δεν σταματά μόνο με το όριο του πλήθους των γενεών, πράγμα που δείχνει πως λειτουργεί ικανοποιητικά.

Με πειραματισμό του παραπάνω σετ μεγεθών (address, population,…) μπορούμε να πετύχουμε καλύτερη απόδοση του συστήματος. Για παράδειγμα η αύξηση του πληθυσμού δίνει την δυνατότητα μεγαλύτερης διερεύνησης του χώρου λύσεων και συνεπώς γρηγορότερη σύγκλιση.

ΥΛΟΠΟΙΗΣΗ ΣΥΝΘΕΤΟΥ CLASSIFIER SYSTEM

ΣΚΟΠΟΣ

Αντικείμενο αυτού του κεφαλαίου είναι η θεωρητική περιγραφή του δευτέρου μέρους αυτής της εργασίας.

Στο προηγούμενο κεφάλαιο αναπτύχθηκε ένα απλοϊκό μοντέλο ενός Classifier συστήματος για την εκμάθηση της συνάρτησης εισόδου εξόδου ενός ψηφιακού πολυπλέκτη. Η έλλειψη όμως του «Συστήματος Καταμερισμού Βαθμών» μας οδήγησε στο να προσομοιώσουμε ένα σύστημα όχι και τόσο ρεαλιστικό. Για αυτόν το λόγο θα αυξήσουμε την πολυπλοκότητα και θα εντάξουμε τον «Bucket Brigade» στο σύστημα μας για ώστε να δούμε μια πιο ολοκληρωμένη μορφή της εκμάθησης ενός Classifier System.

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Εντάσσοντας λοιπόν τον «Bucket Brigade» αλλάζει όλη η δομή του προγράμματος. Ενώ στην προηγούμενη εφαρμογή κάθε άτομο-λύση του συστήματος αποτελούνταν από ένα σετ κανόνων (με γνωστό τον αριθμό των βέλτιστων κανόνων εξαρχής), εδώ κάθε άτομο-λύση είναι μια ξεχωριστή οντότητα-κανόνας (-condition-:-action-). Ο πολυπλέκτης που χρησιμοποιήθηκε ήταν ο 4 σε 1 με 64 δυνατούς συνδυασμούς εισόδου.

Αρχικά στο πρόγραμμα παράγεται μια τυχαία είσοδος (περιβάλλον) από το σύνολο των 64 διαφορετικών εισόδων. Έπειτα αυτή η είσοδος ελέγχεται πόσα και ποια άτομα του πληθυσμού (Classifier Store) διέγειρε. Η διαδικασία μέχρι εδώ αποτελεί την "ραχοκοκαλιά" του συστήματος μας και ονομάζεται «Performance System». Έπειτα ακολουθεί άλλη μια θεμελιακή διαδικασία, ο αλγόριθμος «Bucket Brigade». Τα διεγερμένα άτομα συναγωνίζονται μεταξύ τους με την προσφορά ενός ποσού από την δύναμη τους για το ποιο θα είναι αυτό που θα εκπέμψει το μήνυμα του. Το προσφερόμενο ποσό υπολογίζεται από μια γραμμική σχέση της δύναμης και του συντελεστή προσφοράς. Εδώ φαίνεται ότι κάθε άτομο έχει την δική του δύναμη που αποτελεί την ποιότητα ενός ατόμου βάση της οποίας επιλέγεται για αναπαραγωγή. Το επικρατέστερο άτομο (αυτό που πρόσφερε το μεγαλύτερο ποσό) ενεργοποιείται και αποστέλλει στην έξοδο το μήνυμα του. Το επόμενο βήμα είναι η επιβράβευση του

ατόμου που ενεργοποιήθηκε μόνο όμως στην περίπτωση που το μήνυμα που παράχθηκε ταιριάζει με την σωστή έξοδο της αντίστοιχης εισόδου η οποία ενεργοποίησε το άτομο. Για να μην αυξηθεί όμως υπέρμετρα η δύναμη των κανόνων εφαρμόζονται και φόροι που μειώνουν την δύναμη και βοηθούν στο να διακριθούν οι πιο σωστοί κανόνες. Η διαδικασία που περιγράφηκε μέχρι στιγμής καλείται "iteration". Αν τώρα εκτελέσουμε ένα μεγάλο πλήθος *iterations* θα βαθμολογήσουμε σωστά όλους τους κανόνες για όλες τις εισόδους. Αυτός είναι και ο αλγόριθμος του Bucket Brigade.

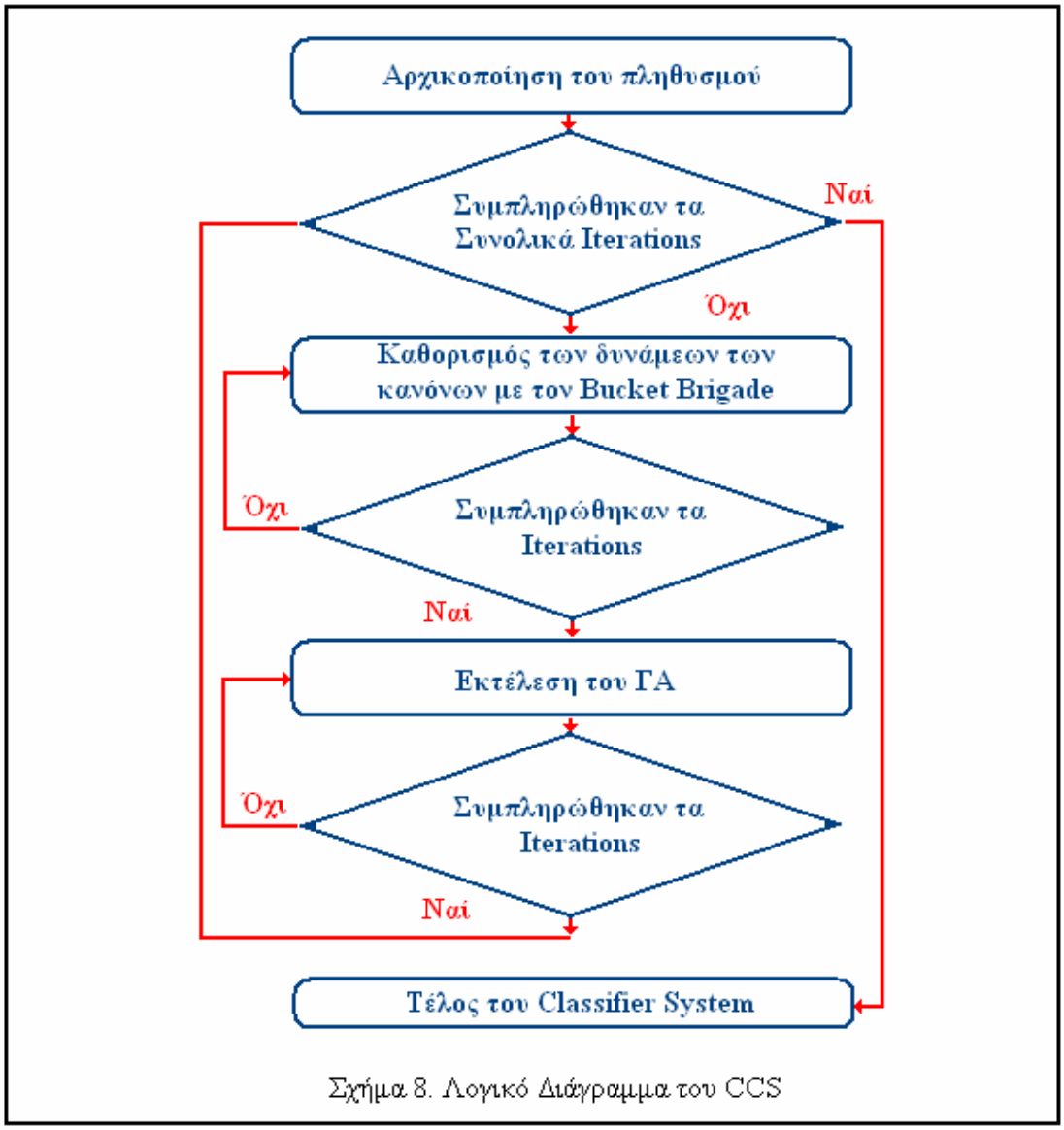
Συνέχεια έχει ο Γενετικός Αλγόριθμος που εισάγει νέα πληροφορία στο σύστημα μας. Κάθε φορά παράγονται δύο απόγονοι μέσω της διασταύρωσης (Crossover) και της μετάλλαξης (Mutation), οι γονείς των οποίων επιλέγονται βάση της μεθόδου Roulette Wheel και σύμφωνα με την ποιότητα τους (δύναμη κανόνα). Έπειτα τα άτομα αυτά παίρνουν την θέση των πιο «αδύναμων» κανόνων. Εφόσον αντικαθιστώνται λιγότερα άτομα από το συνολικό πλήθος των ατόμων του πληθυσμού αναφερόμαστε σε Αντικατάσταση Σταθερής Κατάστασης. Τα χειρότερα ποιοτικά άτομα αντικαθιστώνται άμεσα από τους απογόνους χωρίς να χρειάζεται να χρησιμοποιηθεί και δεύτερος πληθυσμός που να κρατά τους απογόνους.

Κι εδώ το αλφάβητο για τη συνθήκη και το μήνυμα είναι {0,1,#} και {0,1} αντίστοιχα. Έτσι ελέγχεται στην μετάλλαξη που ανήκει η θέση του προς μετάλλαξη στοιχείου ή στοιχείων (bits) , και αν ανήκει στη συνθήκη ή το μήνυμα μεταλλάσσεται βάση του αντίστοιχου αλφάβητου.

Η διαδικασία που ορίζεται από την επιλογή των γονέων μέχρι την αντικατάσταση του χειρότερου αυτών, ονομάζεται iteration του ΓΑ.

Ο ΓΑ εκτελείται μια φορά μετά από τα iteration του ΒΒ. Αν επαναληφθούν οι δύο αυτές διαδικασίες μέσα σε μια επαναληπτική ρουτίνα, το πρόγραμμα θα οδηγηθεί στο να παραχθούν κανόνες με μεγάλη δύναμη που θα ταιριάζουν με τις εισόδους του περιβάλλοντος και θα παράγουν τα σωστά μηνύματα.

Η εφαρμογή τερματίζει αφού συμπληρωθεί το συνολικό πλήθος των iterations που έθεσε ο χρήστης σαν κριτήριο.



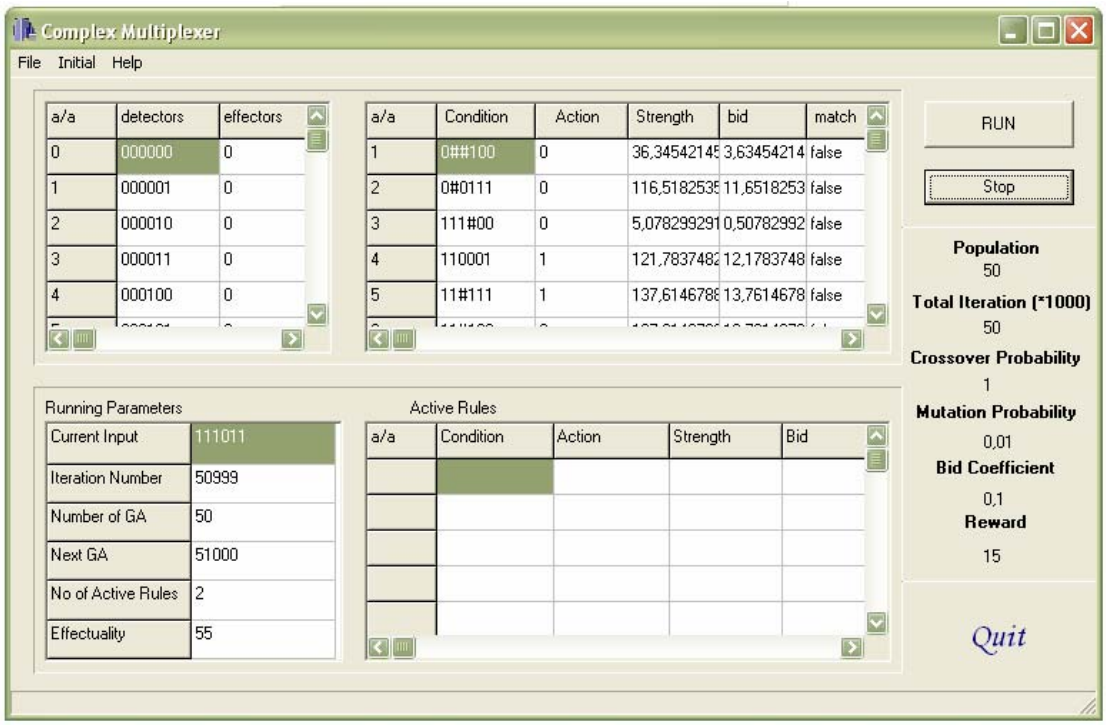
Σχήμα 8. Λογικό Διάγραμμα του CCS

Για να φανεί πόσο αποδοτικό είναι το Classifier System που παράχθηκε εφαρμόζονται και 64 διαφορετικοί συνδυασμοί εισόδων και υπολογίζεται πόσοι από αυτούς ταίριαξαν σωστά με τους εξελιγμένους κανόνες.

ΟΘΟΝΕΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Η Κύρια οθόνη του προγράμματος

Το παρακάτω σχήμα αποτελεί την φόρμα του Complex Classifier System:



Παρατηρούνται τα εξής Components:

StringGrid4 (Πάνω Δεξιά): Σε αυτό εμφανίζεται ο αρχικός παραγόμενος πληθυσμός.

StringGrid5 (Πάνω Αριστερά): Εδώ εμφανίζονται οι δυνατοί συνδυασμοί εισόδων με τις αντίστοιχες εξόδους.

StringGrid6 (Κάτω Αριστερά): Εδώ εμφανίζονται διάφορες σημαντικοί παράμετροι κατά την διάρκεια της εκτέλεσης.

StringGrid7 (Κάτω Δεξιά): Εδώ εμφανίζονται οι ενεργοί κανόνες σε κάθε iteration.

RUN Button: Ξεκινά την εξέλιξη του Classifier System.

STOP Button: Σταματά ανά πάσα στιγμή την διαδικασία.

Quit Label: Τερματισμός της εφαρμογής.

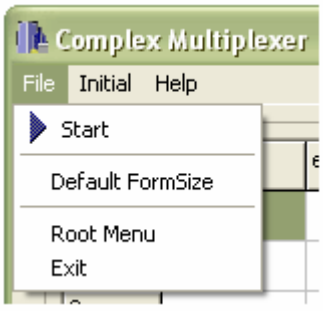
Labels (Population, Total Iteration, Crossover Probability, Mutation Probability, Bid Coefficient, Reward): Δείχνουν τις τιμές που έχουν οριστεί από τον χρήστη.

Main Menu: Εμφανίζει τις επιλογές του προγράμματος που είναι δυνατές για τον χρήστη.

Status Bar: Δείχνει την κατάσταση στην οποία βρίσκεται το πρόγραμμα.

MAIN MENU

1. FILE



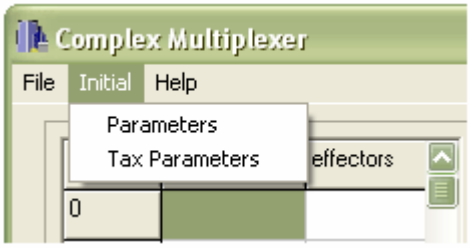
File->Start: Ξεκινά την εξέλιξη του Classifier System.

File->Default FormSize: Επανακαθορισμός του αρχικού μεγέθους της φόρμας.

File->Root Menu: Επιστροφή στην αρχική φόρμα.

File->Exit: Τερματισμός εφαρμογής.

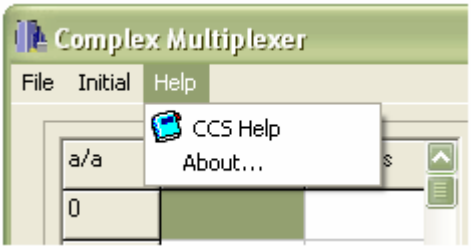
2. INITIAL



Initial->Parameters: Εμφανίζει την φόρμα Initial.

Initial->Tax Parameters: Εμφανίζει την φόρμα Taxes.

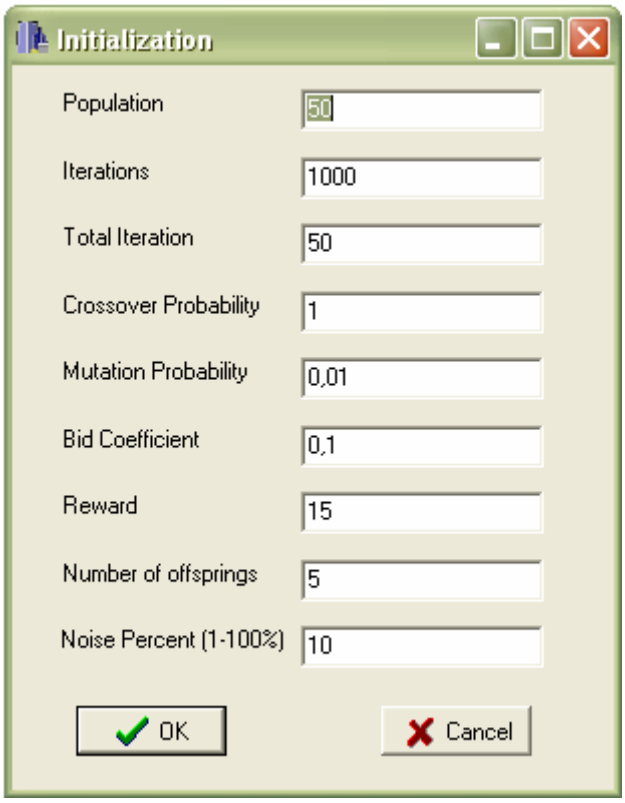
3. HELP



Help-> CCS Help: Εμφανίζει το Help της εφαρμογής.

Help-> About: Εμφανίζει την φόρμα About.

Η Οθόνη Αρχικοποίησης Τιμών INITIAL



Αποτελείται από τα εξής Components:

Population Edit: Εδώ εισάγετε το πλήθος των κανόνων (Population of Classifier Store).

Iteration Edit: Εδώ εισάγετε το πλήθος των επαναλήψεων σε κάθε εκτέλεση του BB.

Total Iteration Edit: Εδώ εισάγετε το συνολικό πλήθος των εκτελέσεων του BB μαζί με τον ΓΑ. Δηλαδή το συνολικό πλήθος των iterations.

Crossover Probability Edit: Εδώ εισάγετε η πιθανότητα διασταύρωσης.

Mutation Probability Edit: Εδώ εισάγετε η πιθανότητα μετάλλαξης.

Coefficient Bid Edit: Εδώ εισάγετε ο συντελεστής Προσφοράς.

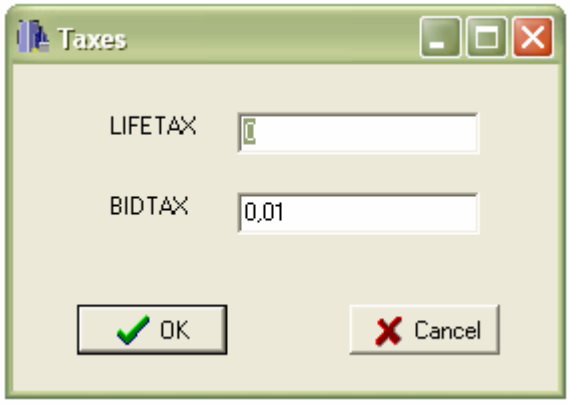
Reward Bid Edit: Εδώ εισάγετε η τιμή της επιβράβευσης για τους ενεργούς κανόνες που παρήγαγαν σωστά μηνύματα.

No of Offspring Edit: Εδώ εισάγετε η τιμή του αριθμού των απογόνων που δημιουργούνται σε κάθε εκτέλεση του ΓΑ.

Noise Percent Edit: Εδώ εισάγετε η τιμή του ποσοστού θορύβου που χρησιμοποιείται για τον προσδιορισμό του επικρατέστερου κανόνα.

Τα παραπάνω μεγέθη έχουν προκαθορισμένες τιμές. Βέβαια ο χρήστης μπορεί ανά πάσα στιγμή να τις αλλάξει. Στην περίπτωση που έστω μία τιμή δεν έχει καθοριστεί ή είναι μηδέν η εφαρμογή δεν θα μπορέσει να εκτελεστεί πατώντας RUN ή File->Start και θα μας ειδοποιήσει με αντίστοιχο μήνυμα.

Η Οθόνη Αρχικοποίησης Φόρων



Lifetax Edit: Εδώ εισάγετε η τιμή της παραμέτρου για την φορολογία όλων των κανόνων.

Bidtax Edit : Εδώ εισάγετε η τιμή της παραμέτρου για την φορολογία μόνο των ενεργοποιημένων κανόνων.

Τα παραπάνω μεγέθη έχουν προκαθορισμένες τιμές , αρκεί να επιλεγούν μια φορά στην αρχή. Βέβαια ο χρήστης μπορεί ανά πάσα στιγμή να τις αλλάξει. Στην περίπτωση που μόνο η μεταβλητή BIDTAX είναι μηδέν η εφαρμογή δεν θα μπορέσει να εκτελεστεί πατώντας RUN ή File->Start και θα μας ειδοποιήσει με αντίστοιχο μήνυμα.

Οι οθόνες που αφορούν την φόρμα About αλλά και την εισαγωγική φόρμα έχουν δηλωθεί στο αντίστοιχο κεφάλαιο του SCS και εφόσον είναι κοινές δεν θα ξαναδηλωθούν στο σημείο αυτό. Επίσης ισχύουν τα ίδια μηνύματα λάθους που εμφανίζονται κατά την εκτέλεση του προγράμματος.

Απόδοση του προγράμματος

Ακολουθεί ένα πείραμα που δείχνει την αποτελεσματικότητα του προγράμματος. Για τον σκοπό αυτό έγινε εκτέλεση του προγράμματος με τα παρακάτω δεδομένα:

Population = 100

Iterations = 1000

Total Iterations 100

Crossover Probability = 1,0

Mutation Probability = 0,03

Bid Coefficient = 0,1

Reward = 30

New Offsprings = 5

Noise = 10%

Ο παρακάτω πίνακας δείχνει τα αποτελέσματα της εκτέλεσης για πλήθος 10 φορών:

a/a	Αρχική Ποιότητα	Καλύτερη Ποιότητα	Τελική Ποιότητα
1	45	61	61
2	39	60	60
3	43	63	63
4	40	60	60
5	36	62	62
6	36	63	63
7	33	63	63
8	39	63	59
9	40	64	62
10	33	59	56

Η αποδοτικότητα του συστήματος είναι εμφανής, εφόσον όπως παρατηρείτε σε κάθε εκτέλεση ο αλγόριθμος του BB με τον καθορισμό των δυνάμεων των κανόνων μαζί με την γενετική εξέλιξη αυτών από τον ΓΑ, καταφέρνουν να επιτύχουν αποδόσεις που ξεκινάνε από μια μέτρια τιμή και φτάνουν σε αρκετά καλές προσεγγίσεις του βέλτιστου ή ακόμη και στο βέλτιστο. Ακόμη λόγω του ότι δεν υπάρχει ελιτισμός στο σύστημα, η μέγιστη τιμή της αποδοτικότητας μπορεί να εμφανιστεί και στις ενδιάμεσες γενιές κι όχι απαραίτητα στην τελική. Αυτό φαίνεται

στις στήλες της καλύτερης ποιότητας και της τελικής ποιότητας, όπου οι τιμές της πρώτης είναι πάντα μεγαλύτερες ή ίσες της δεύτερης.

Με πειραματισμό του παραπάνω σετ μεγεθών (population, iteration, reward, …) μπορούμε να πετύχουμε καλύτερες αποδόσεις του συστήματος. Για παράδειγμα η αύξηση του πληθυσμού δίνει την δυνατότητα μεγαλύτερης διερεύνησης του χώρου λύσεων και συνεπώς γρηγορότερη σύγκλιση.

Ο ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΤΟΥ SCS

ΤΕΧΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Σαν πρώτο βήμα λοιπόν όπως αναφέρθηκε και στο κεφάλαιο των γενετικών αλγορίθμων ήταν η παραγωγή του αρχικού πληθυσμού. Κάθε άτομο αποτελείται από ένα σετ κανόνων-μηνυμάτων. Το σετ για τον πολυπλέκτη 4 σε 1 είναι ένα σύνολο από 8 Classifiers, ενώ για τον 8 σε 1 είναι 16. Ο τύπος μεταβλητής που χρησιμοποιείται για κάθε Classifier είναι τύπου δομής η οποία ονομάζεται «classif» και έχει την εξής μορφή:

```
typedef struct          //classif struct
{
    rules rule[N_MAX];
    AnsiString genotype;
    double fit;
}classif;
```

Κάθε άτομο λοιπόν αποτελείται από έναν πίνακα N_MAX στοιχείων, με κάθε στοιχείο να είναι μια δομή τύπου *rules*. Για τον πολυπλέκτη 4 σε 1 χρησιμοποιούνται μόνο 8 στοιχεία σε κάθε δομή (N=8), ενώ για τον πολυπλέκτη 8 σε 1 χρησιμοποιούνται 16 όσοι δηλαδή είναι και οι βέλτιστοι κανόνες σε κάθε περίπτωση. Η τελευταία δομή έχει δηλωθεί πιο πριν για να μπορεί να χρησιμοποιηθεί στο σημείο αυτό. Ο ορισμός της είναι ο ακόλουθος:

```
typedef struct          //rule struct uses in classif struct
{
    AnsiString condition;
    AnsiString action;
}rules;
```

Εδώ φαίνεται και η μορφή ενός κανόνα όπως περιγράφηκε στο πρώτο κεφάλαιο αφού η δομή αυτή περιλαμβάνει αλφαριθμητικές μεταβλητές, μία για την συνθήκη (condition) και μία για το μήνυμα (action). Έτσι αν είχαμε ένα άτομο με όνομα *classname*, θα αναφερόμασταν στο condition ως εξής:

classname.rule.condition

Συνεχίζοντας λοιπόν παρατηρούμε ακόμη μια αλφαριθμητική μεταβλητή, την *genotype*, η τιμή της οποίας είναι το αλφαριθμητικό άθροισμα όλων των κανόνων του ατόμου. Δηλαδή είναι μια ακολουθία από L bits

$$L= N\ (\text{κανόνες}) \times (\text{condition.Length}\ () + \text{action.Length}\ ())$$

Τέλος βλέπουμε και την μεταβλητή *fit* που είναι τύπου double και δηλώνει την ποιότητα κάθε ατόμου σε σχέση με το περιβάλλον. Η τιμή αυτή πηγάζει από την συνάρτηση ποιότητας που στο πρόγραμμα μας είναι η συνάρτηση match(). Βλέπουμε ότι κάθε κανόνας δεν έχει μια τιμή που να δηλώνει την δύναμη του έτσι η fit που αντιστοιχεί στο σύνολο των κανόνων δηλαδή στο κάθε άτομο, είναι η μόνη πηγή πληροφορίας μέσω της οποίας θα εξελιχθεί το σύστημα μας.

Η αρχική παραγωγή του πληθυσμού γίνεται μέσα σε ένα βρόχο που εκτελείται τόσες φορές όσες και ο αριθμός των ατόμων. Το πλήθος των ατόμων καθορίζεται από το χρήστη κατά την εκτέλεση του προγράμματος, πριν ξεκινήσει η εκμάθηση των κανόνων. Για μεγάλες τιμές του πλήθους των ατόμων, ο αλγόριθμος συγκλίνει πιο γρήγορα όσων αφορά το πλήθος των γενεών εφόσον ερευνά μεγαλύτερο εύρος του χώρου λύσεων όμως απαιτεί μεγαλύτερη χρονική διάρκεια και άρα υπολογιστική ισχύς. Για μικρές τιμές του αριθμού των ατόμων η κάθε γενιά διαδέχεται γρήγορα την άλλη χρονικά αλλά αργεί να συγκλίνει φτάνοντας σε μεγάλο αριθμό γενεών. Έπειτα αρχικοποιούνται και άλλα στοιχεία του προγράμματος, όπως ο πίνακας αλφαριθμητικών *detc[]* που περιέχει τις εισόδους από το περιβάλλον (detectors) και που κάθε στοιχείο του λαμβάνει τιμή από την εκτέλεση της συνάρτησης IntToBinary(). Το πλήθος των στοιχείων του πίνακα για κάθε πολυπλέκτη ισούται με:

$$2^{\text{address + data}}$$

Επομένως για τον πολυπλέκτη 4 σε 1 θα έχουμε πλήθος στοιχείων 64 (address = 2 και data = 4), και για τον πολυπλέκτη 8 σε 1 το πλήθος αυτό θα είναι 2048 (address=3 και data = 8). Αυτό που κάνει η τελευταία συνάρτηση είναι να παίρνει σαν όρισμα έναν ακέραιο (τον δείκτη της θέσης του στοιχείου μέσα στον πίνακα) και να τον μετατρέπει στην δυαδική του μορφή. Το σώμα της συνάρτηση είναι το εξής:

```
AnsiString IntToBinary(int x)
{
    int i;
    int mask = pow(2,(BITS-1));
    AnsiString s="";
    for (i=0;i<BITS;i++)
    {
        if ( (x & mask) > 0) {s+="1";}
        else                {s+="0";}
        mask=mask >> 1;
    }
    return s;
}
```

Η συνάρτηση αυτή χρησιμοποιεί την δυαδική μορφή του ακεραίου *mask* και την δυαδική μορφή του ορίσματος. Ελέγχει κάθε θέση bit σύμφωνα με την λογική της πύλης AND και δημιουργεί κατάλληλα μια αλφαριθμητική τιμή η οποία επιστρέφεται σαν τιμή στην αντίστοιχη θέση του πίνακα *detc[]*. Οι δυνατές τιμές εισόδου είναι 000000…111111 για τον 4_1 και 00000000000…11111111111 για τον 8_1.

Όμοια για τον πίνακα ακεραίων *out[]* που περιλαμβάνει σαν τιμές τις σωστές εξόδους (effectors) για κάθε είσοδο. Κάθε στοιχείο του πίνακα παίρνει τιμή από την εκτέλεση της συνάρτησης OutputMsg(). Σαν ορίσματα εισάγονται οι τιμές των θέσεων του πίνακα detc[] και παράγονται οι αντίστοιχες έξοδοι. Άρα οι δύο τελευταίες συναρτήσεις εκτελούνται τον ίδιο αριθμό φορών. Το σώμα της OutputMsg() είναι:

```
int OutputMsg(AnsiString str)
{
      int m;
      int i,sum;
      sum=1;
      for(i=1;i<=address;i++)
      {
          if(str[i]=='1')
             sum+=1*pow(2,(address-i));
          else
             sum+=0;
      }
      if(str[address+sum]=='0')
          m=0;
      else
          m=1;
      return m;     }
```

Στη συνέχεια ακολουθεί η αποτίμηση των λύσεων. Υπεύθυνη για την διαδικασία αυτή είναι η συνάρτηση match() (συνάρτηση ποιότητας). Αυτό που κάνει η συνάρτηση αυτή είναι να ταιριάζει τις εισόδους του περιβάλλοντος με τους κανόνες κάθε ατόμου. Κάθε φορά που ταιριάζει ένας κανόνας με μία είσοδο, η δύναμη του ατόμου αυξάνεται κατά ένα, όταν βέβαια ταιριάζει και το μήνυμα του ενεργοποιημένου κανόνα με την σωστή έξοδο που αντιστοιχεί στην είσοδο που ενεργοποίησε τον κανόνα . Στην περίπτωση που ενεργοποιούνται πολλοί κανόνες ισχύει το επικρατέστερο μήνυμα από αυτούς. Για παράδειγμα αν ενεργοποιηθούν τρεις κανόνες εκ των οποίων οι δύο παράγουν σαν μήνυμα την μονάδα "1" και ο τρίτος το μηδέν "0", τότε επικρατέστερο μήνυμα είναι η μονάδα. Στην περίπτωση όμως που ενεργοποιηθούν κανόνες που παράγουν ίσο αριθμό μηνυμάτων με "0" και "1" , τότε επιλέγεται ένα από τα δύο μηνύματα με ποσοστό 50% και αν επαληθευθεί με την σωστή έξοδο τότε η δύναμη του ατόμου αυξάνεται κατά 0.5 και όχι κατά 1. Αυτό γίνεται γιατί μπορεί σε δύο διαδοχικές κλήσεις της match()το ίδιο άτομο να δώσει την μια φορά σαν έξοδο '0' και την άλλη '1' πράγμα που όπως είναι νοητό δεν συμφωνεί με το επιθυμητό, δηλαδή να μας δώσει και τις δύο φορές την ίδια έξοδο.

Παρακάτω φαίνεται το σώμα της *match()* :

```
double match(AnsiString detc1[], classif par1, int out1[])
{
    bool equal;
    bool eqFL=false;                    //it is true when one==true;
    int zero,one,i,j,c,eff,x;
    double fit;
    AnsiString str;

    fit=0.0;
    for(c=0;c<T;c++)
    {
        str=detc1[c];
        x=out1[c];
        zero=0;
        one=0;
        for(i=0;i<N;i++)
         {
            for(j=1;j<=BITS;j++)
            {
                if(str[j]==par1.rule[i].condition[j] || par1.rule[i].condition[j]=='#')
                  equal=true;
                else
                {
                  equal=false;
```

```
                    break;
                  }
              }
              if(equal)
              {
                if(par1.rule[i].action=='1')
                  one++;
                else
                  zero++;
              }
          }
          if(one!=0 || zero!=0)
          {
            if(one>zero)
              eff=1;
            else if(one<zero)
              eff=0;
            else if(one==zero)
            {
              if(rnd()<0.5)
                eff=0;
              else
                eff=1;
              eqFL=true;
            }
            if(x==eff && eqFL==false)
              fit++;
            else if(x==eff && eqFL==true)
            {
              fit+=0.5;
              eqFL=false;
            }
          }
      }
      return fit;
}
```

Στη συνέχεια του βασικού προγράμματος βρίσκεται η εξέλιξη του ΓΑ που πραγματοποιείται μέσα από έναν βρόχο **do{}while()**. Όπως πλέον γνωρίζουμε η επόμενη φάση της εξέλιξης είναι η επιλογή των ατόμων. Για την επιλογή των ατόμων χρησιμοποιήθηκε η μέθοδος επιλογής Roulette Wheel (αυτό είναι και το όνομα της αντίστοιχης συνάρτησης) σύμφωνα με την οποία τα άτομα επιλέγονται βάση της ποιότητας τους. Η συνάρτηση αυτή είναι η παρακάτω:

```
int roulettewheel(classif par1[])
{
      double sum, sumrnd ;
      int i;
```

```
    sum=0.0;
    for(i=0;i<P;i++)
    {
        sum+=par1[i].fit;
    }
    sumrnd=rnd()*sum;
    for(i=0,sum=0;i<P;i++)
    {
        sum+=par1[i].fit;
        if (sum>=sumrnd)
        break;
    }
    return i;
}
```

Ακολουθούν οι γενετικοί τελεστές της Διασταύρωσης (συνάρτηση Crossover()) και της μετάλλαξης (συνάρτηση Mutation())

Η συνάρτηση της διασταύρωσης παίρνει σαν ορίσματα τα δύο άτομα που επελέγησαν από την RouletteWheel() και τα ανασυνδυάζει. Ο ανασυνδυασμός γίνεται με τις μεταβλητές *genotype* των δύο ατόμων. Ο αριθμός των σημείων κοπής είναι και αυτός μία παράμετρος που καθορίζεται από τον χρήστη κατά την εκτέλεση του προγράμματος. Βέβαια η συνάρτηση είναι αυτή που καθορίζει ποιες θα είναι οι θέσεις των σημείων κοπής, οι οποίες καθορίζονται με τυχαίο τρόπο. Άλλο ένα πρόβλημα που μπορεί να προκύψει στο σημείο αυτό, είναι μήπως και η συνάρτηση παράγει δύο φορές το ίδιο σημείο κοπής. Αυτό αποφεύγεται χωρίζοντας τον γονότυπο σε τόσα τμήματα όσα είχε ορίσει ο χρήστης για τα σημεία κοπής και υπολογίζοντας ένα σημείο για κάθε τμήμα.

Για παράδειγμα αν είχαμε τους παρακάτω γονότυπους με μήκος L bits (όπως ορίστηκε στην σελίδα 32):

A.genotype= 00000000……..00000000
B.genotype= 11111111……..11111111

και ο χρήστης έδινε αριθμό σημείων κοπής 2 (έστω το πρώτο βρίσκεται μεταξύ του $5^{ου}$ και του $6^{ου}$ bit και το δεύτερο μεταξύ του L-4 και L-3) τότε τα παραγόμενα άτομα θα ήταν τα:

child1.genotype= 00000111……..11110000
child2.genotype= 11111000……..00001111

Συνάρτηση Crossover():

```
classif Crossover(classif s1, classif s2)
{
    int i,j,pos,tmima;
    classif child;
    bool chFL=true;

    int L=N*(BITS+ABITS);          //megethos genotype

    tmima=(int)(L/points);

    j=1;
    child.genotype="";
    for(i=0;i<points;i++)
    {
      pos=i*tmima + random(tmima) + 1;
      while(j<=pos)
      {
        if(chFL==true)
          child.genotype+=s1.genotype[j];
        else
          child.genotype+=s2.genotype[j];
        j++;
      }
      if(chFL==true) {chFL=false;}
      else          {chFL=true;}
    }
    while(j<=L)
    {
        if(chFL==true)
          child.genotype+=s1.genotype[j];
        else
          child.genotype+=s2.genotype[j];
        j++;
    }
    return child;
}
```

Όπως φαίνεται και από την συνάρτηση, επιστρέφεται μόνο ένα άτομο. Δεν έχει σημασία με ποια σειρά καθορίζεται το άτομο που θα επιστραφεί (αν δηλαδή παίρνει στοιχεία πρώτα από το s1 ή πρώτα από το s2) αφού και οι δύο περιπτώσεις παράγονται τυχαία. Η σειρά με την οποία ξεκινάει η αντιγραφή καθορίζεται από την μεταβλητή *chFL*, όπου αν αρχικοποιείται με την τιμή true θα πάρει στοιχεία πρώτα από το s1, αλλιώς αν είναι false θα πάρει πρώτα από το s2. Η αποτίμηση του είναι αυτή που θα δείξει το πόσο κατάλληλο είναι για την εξέλιξη.

Αξίζει να σημειωθεί ότι κατά την αρχική υλοποίηση του προγράμματος η μεταβλητή genotype δεν υπήρχε και η διασταύρωση μεταξύ δύο ατόμων γινόταν

μεταξύ των κανόνων. Δηλαδή ο απόγονος έπαιρνε τους κανόνες του ενός γονέα μέχρι το σημείο κοπής και τους υπόλοιπους από τον άλλον. Η συνάρτηση αυτή καλείται βάση κάποιας πιθανότητας *CrossProb*. Αν τύχει να μην εκτελεστεί η Crossover() τότε επιλέγεται να περάσει στον πληθυσμό των απογόνων *pop[]* ένας από τους δύο γονείς η επιλογή του οποίου γίνεται με την βοήθεια της συνάρτησης βιβλιοθήκης random(). Ο έλεγχος για την πιθανότητα να κληθεί η συνάρτηση γίνεται με την βοήθεια μιας τιμής που βρίσκεται στο διάστημα [0,1] και που επιστρέφεται από μια «βοηθητική» συνάρτηση την rnd() (βλ. Πηγαίο Κώδικα Προγράμματος – Unit2).

Η μετάλλαξη τώρα πραγματοποιείται στο παραγόμενο άτομο είτε αυτό προέκυψε από διασταύρωση είτε από "κλωνοποίηση". Στέλνεται σαν όρισμα το άτομο-απόγονος και εσωτερικά στο σώμα της συνάρτησης Mutation() μεταλλάσσεται ο γονότυπος του. Η παράμετρος PMUT (mutation probability) θα καθορίσει πόσα bit του γονότυπου θα μεταλλαχθούν. Οι δυνατές μεταλλάξεις για το condition είναι οι εξής:

$$0 \rightarrow 1 \text{ ή } \#$$
$$1 \rightarrow 0 \text{ ή } \#$$
$$\# \rightarrow 0 \text{ ή } 1$$

και για το action:

$$0 \rightarrow 1$$
$$1 \rightarrow 0$$

Προγραμματιστικά αυτό υλοποιήθηκε με το να ελέγχεται η θέση του κάθε bit και αν αυτή είναι ακέραιο πολλαπλάσιο του μήκους του κάθε κανόνα τότε να μεταλλάσσεται το action σύμφωνα με τον πιο πάνω τρόπο αλλιώς να μεταλλάσσεται το condition.

Συνάρτηση Mutation:

```
classif Mutation(classif par1)
{
    double point , rest;
    int j,bits,pos;

    point = PMUT * N * (BITS + ABITS);  // ta simeia pou tha ginei i metalaxi
    bits = (int)point;                  //to akeraio meros
```

```
    rest = point - bits;              //float meros
    if(rnd() < rest)                  //prob gia na metallaxthei allo ena bit
        bits++;
    for(j=0;j<bits;j++)
    {
        pos=random(N*(BITS + ABITS)) + 1;
        if(pos%(BITS + ABITS)==0)          //  gia to mnvuma(action) tha prepei na
                                           //doume an to ypoloipo
        {
           if(par1.genotype[pos]=='0')     // einai 0 kai tha to allaksoume se 0 h 1
               par1.genotype[pos]='1';
           else
               par1.genotype[pos]='0';
         }
         else
         {
             if(par1.genotype[pos]=='0')
                 if(rnd()<0.5)   par1.genotype[pos]='1';
                 else            par1.genotype[pos]='#';
             else
             if(par1.genotype[pos]=='1')
                 if(rnd()<0.5)   par1.genotype[pos]='0';
                 else            par1.genotype[pos]='#';
             else
                 if(par1.genotype[pos]=='#')
                     if(rnd()<0.5)   par1.genotype[pos]='1';
                     else            par1.genotype[pos]='0';
         }
    }
    return par1;
}
```

Στη συνέχεια αφού έχουν διαμορφωθεί οι γονότυποι των απογόνων στον πληθυσμό pop[], θα πρέπει να παραχθούν μέσα από την μεταβλητή *genotype* οι κανόνες στην κανονική τους μορφή:

condition : action

Υπάρχει για αυτό η συνάρτηση LineToArray() (Unit2) που παίρνει αυτήν την ακολουθία των bits και την μετατρέπει σε ένα σετ ή πίνακα από κανόνες. Έτσι τα καινούργια άτομα μπορούν να αξιολογηθούν και πάλι μέσα από την συνάρτηση match().

Υπάρχει και η αντίστροφη συνάρτηση ArrayToLine() (Unit2) που από το σετ κανόνων δημιουργεί τον ενιαίο γονότυπο αφού όπως είναι κατανοητό χρειάζεται στους γενετικούς τελεστές.

Για την γρηγορότερη σύγκλιση του αλγόριθμου εισάγουμε έναν τελεστή αναρρίχησης. Χρησιμοποιήθηκε μια παραλλαγή του Swap Bit HCO. Αλλάζει κάθε φορά μόνο ένα bit ενός τυχαίου κανόνα από κάποιο άτομο με τα υπόλοιπα σύμβολα του αλφάβητου (0, 1, # αν πρόκειται για την συνθήκη του κανόνα, και 0 , 1 για το μήνυμα) και ελέγχεται η ποιότητα του τροποποιημένου κανόνα. Αν αυτή είναι καλύτερη από την αρχική τότε θα υιοθετηθεί ο καινούριος κανόνας, αλλιώς αυτός θα ακυρωθεί και θα επικρατήσει ο αρχικός. Η αντίστοιχη συνάρτηση είναι η HillClimber().

Η διαδικασία που περιγράφηκε από το σημείο της επιλογής των γονέων-ατόμων μέχρι εδώ επαναλαμβάνεται ώσπου να γεμίσει ο πληθυσμός των απογόνων πλην της τελευταίας θέσης η οποία κρατείται για να καταχωρηθεί το άτομο από τον πληθυσμό των γονέων *par[]* με την καλύτερη ποιότητα.

Η γενιά ολοκληρώνεται με την εύρεση του πιο βέλτιστου κανόνα και την εισαγωγή του στην τελευταία θέση του πληθυσμού των απογόνων (Ελιτισμός):

```
//-----------elitism----------------
    max=par[0].fit;
    best=0;
    for(p=1;p<P;p++)
    {
        if(max<par[p].fit)
        {
            max=par[p].fit;
            best=p;
        }
    }
```

και την ολική αντικατάσταση του πληθυσμού των γονέων από τον πληθυσμό των απογόνων:

```
//Oi apofonoi_POP diadexontai toys goneis_PAR
    for(p=0;p<P;p++)
    {
        par[p] = pop[p];
    }
```

Μέσα στον βρόχο do..while{} υπάρχει ένας μετρητής, ο *count* που ξεκινάει από το μηδέν και αυξάνεται κατά ένα σε κάθε επανάληψη και δηλώνει το πλήθος των γενεών.

Τέλος οι συνθήκες τερματισμού του βρόχου αποτελούν τα κριτήρια τερματισμού και του ΓΑ.

Πιο ειδικά για τον τερματισμό, μέσα στην **while()** καλείται μια συνάρτηση η criterion() , που ελέγχει αν η τιμή της μεταβλητής *count* και η πιθανότητα του καλύτερου ατόμου κάθε γενιάς συμφωνούν με τα όρια που τέθηκαν από τον χρήστη και ανάλογα επιστρέφει μια **true** ή **false** τιμή.

ΤΑ UNIT ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

ΕΙΣΑΓΩΓΙΚΗ ΦΟΡΜΑ

Η εισαγωγική φόρμα της εργασίας αντιστοιχεί στο Unit1 και αποτελείται από τα εξής αρχεία:

- Unit1.cpp
- Unit1.h
- Unit1.dfm

Κλάση

TEntry: Η κλάση της εισαγωγικής φόρμας.

Όπως φαίνεται αποτελείται από τρία Buttons.

Συναρτήσεις-Μέλη

void __fastcall TEntry::QuitClick(TObject *Sender)

Quit: Τερματίζει την εφαρμογή.

void __fastcall TEntry::ButtonSCSClick(TObject *Sender)

Continue to the SCS: Μας εισάγει στο πρόγραμμα του απλού Classifier συστήματος.

void __fastcall TEntry::ButtonCCSClick(TObject *Sender)

Continue to the CCs: Μας εισάγει στο πρόγραμμα του σύνθετου Classifier συστήματος.

void __fastcall TEntry::FormCanResize(TObject *Sender, int &NewWidth, int &NewHeight, bool &Resize)

Κρατάει το **Resize=false** ώστε να μην αυξομειώνεται το μέγεθος της φόρμας.

ΦΟΡΜΑ ΤΟΥ SCS

Είναι η κύρια φόρμα του προγράμματος που προσομοιώνει την συνάρτηση εισόδου εξόδου του απλού ψηφιακού πολυπλέκτη χωρίς την χρήση του «Συστήματος Καταμερισμού Βαθμών».

Η φόρμα αυτή αντιστοιχεί στο Unit2 και αποτελείται από τα εξής αρχεία:

- Unit2.cpp
- Unit2.h
- Unit2.dfm

Κλάση

TSCSMainForm: Η κλάση της κύριας φόρμας.

Σταθερές

#define ABITS 1: Σταθερά για τον καθορισμό του μήκους του μηνύματος.

#define N_MAX 100: Σταθερά για τον καθορισμό του πλήθους των στοιχείων του πίνακα *rule[]* τα οποία είναι τύπου *rules*.

Μεταβλητές και Δομές

```
typedef struct          //rule struct uses in classif struct
{
  AnsiString condition;
  AnsiString action;
}rules;
```

Δομή τύπου *rules*.

```
typedef struct          //classif struct
{
   rules rule[N_MAX];   // N_MAX constant  = 100
  AnsiString genotype;
  double fit;
}classif;
```

Δομή τύπου *classif*.

TEdit *GenEdit :

Σε αυτό το Edit Box ο χρήστης εισάγει το όριο του κριτηρίου των γενεών.

TEdit *FitEdit :

Σε αυτό το Edit Box ο χρήστης εισάγει το όριο του κριτηρίου ποιότητας.

int address :

Μεταβλητή που ορίζεται από τον χρήστη και δηλώνει το μέγεθος του address των κανόνων.

int P :

Μεταβλητή που ορίζεται από τον χρήστη και δηλώνει το πλήθος του Classifier Store.

int points :

Μεταβλητή που ορίζεται από τον χρήστη και δηλώνει τον αριθμό των σημείων κοπής.

double CROSSPROB :

Ορίζεται από τον χρήστη και δηλώνει την πιθανότητα διασταύρωσης.

double PMUT :

Ορίζεται από τον χρήστη και δηλώνει την πιθανότητα μετάλλαξης.

int count :

Μετρητής που κρατάει το πλήθος των γενεών.

int data :

Μεταβλητή που δηλώνει το μέγεθος του *data* των κανόνων. Εξαρτάται από το *address*.

int BITS :

Μέγεθος του *Condition*. Εξαρτάται από τα *address* και *data.*

int T :

Πλήθος δυνατών συνδυασμών εισόδου-εξόδου. Εξαρτάται από το *BITS.*

int N :

Πλήθος βέλτιστων κανόνων σε κάθε άτομο. Εξαρτάται από το *data.*

bool genFL :

Λογική μεταβλητή για τον καθορισμό του κριτηρίου τερματισμού βάση του αριθμού των γενεών.

bool fitFL :

Λογική μεταβλητή για τον καθορισμό του κριτηρίου τερματισμού βάση της ποιότητας.

bool genfitFL :

Λογική μεταβλητή για τον καθορισμό του κριτηρίου τερματισμού βάση του συνδυασμού των δύο παραπάνω.

bool stopFL :

Λογική μεταβλητή για την διακοπή όλης της εφαρμογής.

Όλες οι παραπάνω μεταβλητές είναι GLOBAL Μεταβλητές.

Συναρτήσεις

double rnd(void)

Παράγει έναν τυχαίο αριθμό στο διάστημα [0,1].

AnsiString IntToBinary(int x)

Παίρνει σαν όρισμα έναν αριθμό μεταξύ του ‘0’ και του μέγιστου των δυνατών συνδυασμών εισόδων-εξόδων και επιστρέφει την δυαδική του μορφή.

bool criterion(classif parBest, int gen, double fitness)

Παίρνει σαν ορίσματα το καλύτερο άτομο, τον μέγιστο αριθμό γενεών και την ποιότητα που όρισε ο χρήστης και ελέγχει αν ικανοποιούνται τα κριτήρια τερματισμού.

Int OutputMsg(AnsiString str)

Παίρνει σαν όρισμα την δυαδική μορφή μιας εισόδου και επιστρέφει την σωστή έξοδο για αυτήν την είσοδο.

AnsiString ArrayToLine(classif par1)

Παίρνει σαν όρισμα ένα άτομο και από τον πίνακα των κανόνων παράγει και επιστρέφει τον ενιαίο γονότυπο.

classif LineToArray(classif par1)

Εκτελεί την αντίστροφη εργασία από την παραπάνω συνάρτηση. Από τον γονότυπο επανακαθορίζει το σετ κανόνων.

double match(AnsiString detc1[], classif par1, int out1[]) // ΓΑ

Είναι η συνάρτηση ποιότητας (fitness function). Παίρνει σαν ορίσματα ένα στοιχείο του πληθυσμού , τον πίνακα των εισόδων και τον πίνακα των εξόδων. Επιστρέφει την ποιότητα του κάθε ατόμου.

int roulettewheel(classif par1[]) // ΓΑ

Συνάρτηση για την επιλογή γονέων.

classif Crossover(classif s1, classif s2)

Συνάρτηση ανασυνδυασμού-διασταύρωσης γονέων.

classif Mutation(classif par1) // ΓΑ

Συνάρτηση μετάλλαξης παραγόμενου ατόμου είτε από διασταύρωση είτε από "κλωνοποίηση".

classif HillCLimber(classif par1,AnsiString detc[],int out[]) // ΓΑ

Συνάρτηση του τελεστή αναρρίχησης. Αποτελεί μια παραλλαγή του «SWAP BIT HCO».

Συναρτήσεις-Μέλη

void __fastcall TSCSMainForm::FormClose(TObject *Sender, TCloseAction &Action)

Είναι υπεύθυνη για τον τερματισμό όλης της εφαρμογής.

void __fastcall TSCSMainForm::Button1Click(TObject *Sender)

Είναι η βασική συνάρτηση μέσα στην οποία γίνονται όλες οι διεργασίες:

- Δημιουργία δυναμικού πληθυσμού γονέων και απογόνων

```
classif * par, * pop;
par=new classif[P];
pop=new classif[P];
```

- Δημιουργία δυναμικών πινάκων για τις εισόδους – εξόδους

```
int* out;
out=new int[T];
AnsiString* detc;
detc=new AnsiString[T];
```

- Κλήσεις όλων των παραπάνω συναρτήσεων που είναι υπεύθυνες για την γενετική εξέλιξη των ατόμων καθώς και κάποιων βοηθητικών συναρτήσεων (π.χ. rnd() , LineToArray(), …).

void __fastcall TSCSMainForm::FormCreate(TObject *Sender)

Καλείται με το που δημιουργείται η φόρμα SCSMainForm.

void __fastcall TSCSMainForm::Exit1Click(TObject *Sender)

Τερματίζει την εφαρμογή μέσα από το Main Menu.

void __fastcall TSCSMainForm::LimGenClick(TObject *Sender)

Προσδιορίζει σαν κριτήριο τερματισμού τον αριθμό των γενεών και εμφανίζει το αντίστοιχο Edit Box για την εισαγωγή του από τον χρήστη.

void __fastcall TSCSMainForm::LimFitClick(TObject *Sender)

Προσδιορίζει σαν κριτήριο τερματισμού την επιθυμητή ποιότητα των ατόμων και εμφανίζει το αντίστοιχο Edit Box για την εισαγωγή της από τον χρήστη.

void __fastcall TSCSMainForm::LimGenFitClick(TObject *Sender)

Συνδυάζει ταυτόχρονα τα δύο παραπάνω κριτήρια και εμφανίζει και τα δύο Edit Boxes.

void __fastcall TSCSMainForm::SCSHelp1Click(TObject *Sender)

Εμφανίζει το Help της εφαρμογής.

void __fastcall TSCSMainForm::About1Click(TObject *Sender)

Εκτελείται όταν πατηθεί το Help→ About από το Main Menu και εμφανίζει την φόρμα About.

void __fastcall TSCSMainForm::DefaultFormSize1Click(TObject *Sender)

Εκτελείται όταν πατηθεί το File→ Default FormSize από το Main Menu και επανακαθορίζει την φόρμα στην αρχική της μορφή.

void __fastcall TSCSMainForm::Variables1Click(TObject *Sender)

Εκτελείται όταν πατηθεί το Initial → Parameters και εμφανίζει την φόρμα Initial για την εισαγωγή δεδομένων από τον χρήστη. Επίσης αρχικοποιούνται και τα StringGrid του προγράμματος.

void __fastcall TSCSMainForm::StopClick(TObject *Sender)

Το πάτημα του πλήκτρου Stop που αντιστοιχεί σε αυτήν την συνάρτηση προκαλεί την διακοπή της εξέλιξης του προγράμματος.

void __fastcall TSCSMainForm::RootMenu1Click(TObject *Sender)

Μας επιστρέφει στην εισαγωγική φόρμα. Βρίσκεται μέσα στην Main Menu→ File→ Root Menu.

void __fastcall TSCSMainForm::QuitClick(TObject *Sender)

Προκαλεί την έξοδο από το πρόγραμμα.

void __fastcall TSCSMainForm::GenEditKeyPress(TObject *Sender, char &Key)

Καλείται όταν εισάγουμε τα δεδομένα στο GenEdit Box και ελέγχει αν έχουν δοθεί σωστά. Αν για παράδειγμα πατηθεί χαρακτήρας αντί αριθμού τότε εμφανίζεται αντίστοιχο μήνυμα που μας ειδοποιεί ότι πρέπει να επανακαθορίσουμε την τιμή του στοιχείου.

void __fastcall TSCSMainForm::FitEditKeyPress(TObject *Sender, char &Key)

Όμοια με το GenEdit και το FitEdit καλείται για να ελέγξει αν έχει δοθεί αριθμός και όχι χαρακτήρας.

Οι δύο παραπάνω συναρτήσεις-μέλη αναφέρονται στο κριτήριο τερματισμού και τα δύο Edit Boxes καθορίζουν τα όρια του αριθμού των γενεών και της ποιότητας. Ο έλεγχος διευκρινίζει και αν έχουν δοθεί δεκαδικοί αντί για ακέραιοι που είναι το απαιτούμενο.

ΦΟΡΜΑ ΑΡΧΙΚΟΠΟΙΗΣΗΣ ΤΙΜΩΝ INITIAL

Είναι μία Modal φόρμα που εμφανίζεται για να εισάγουμε τα αρχικά στοιχεία. Αποτελείται από 5 Edit Boxes , 5 Labels και 2 BitBtns.

Η φόρμα αυτή αντιστοιχεί στο Unit5 και αποτελείται από τα εξής αρχεία:

- Unit5.cpp
- Unit5.h
- Unit5.dfm

Κλάση

TInitial: Η κλάση της φόρμας.

Συναρτήσεις-Μέλη

void __fastcall TInitial::FormCanResize(TObject *Sender, int &NewWidth, int &NewHeight, bool &Resize)

Καθορίζει ότι το μέγεθος της φόρμας είναι αμετάβλητο (Resize = false).

void __fastcall TInitial::AddrEditKeyPress(TObject *Sender, char &Key)

Καλείται όταν εισάγουμε τα δεδομένα στο AddrEdit Box και ελέγχει αν έχουν δοθεί σωστά. Αν για παράδειγμα πατηθεί χαρακτήρας αντί αριθμού τότε εμφανίζεται αντίστοιχο μήνυμα που μας ειδοποιεί ότι πρέπει να επανακαθορίσουμε την τιμή του στοιχείου.

void __fastcall TInitial::PopEditKeyPress(TObject *Sender, char &Key)

Όμοια κι εδώ ελέγχεται τι πατήθηκε μέσα στο PopEdit Box και αντίστοιχα είτε εκχωρεί την τιμή στο στοιχείο είτε εμφανίζει μήνυμα για τον επανακαθορισμό του πληθυσμού.

void __fastcall TInitial::CPointsEditKeyPress(TObject *Sender, char &Key)

Όμοια κι εδώ ελέγχεται τι πατήθηκε μέσα στο CPointsEdit Box και αντίστοιχα είτε εκχωρεί την τιμή στο στοιχείο είτε εμφανίζει μήνυμα για τον επανακαθορισμό του στοιχείου.

Οι τρεις αυτές παράμετροι λαμβάνουν μόνο ακέραιους αριθμούς. Έτσι αν πατηθεί ακόμη και ο χαρακτήρας ‘,’ εμφανίζεται πάλι το μήνυμα λάθους.

void __fastcall TInitial::CrEditKeyPress(TObject *Sender, char &Key)

Όμοια ελέγχεται τι πληκτρολόγησε ο χρήστης του προγράμματος για την πιθανότητα Διασταύρωσης. Ο χρήστης μπορεί να πληκτρολογήσει και πραγματικό αριθμό. Επίσης ελέγχει αν αντί του χαρακτήρα ‘,’ που είναι για τους πραγματικούς πατηθεί ο χαρακτήρας ‘.’ και τον αλλάζει αυτόματα στον ‘,’ .

void __fastcall TInitial::MEditKeyPress(TObject *Sender, char &Key)

Όμοια ελέγχεται τι πληκτρολόγησε ο χρήστης του προγράμματος για την πιθανότητα μετάλλαξης. Ο χρήστης μπορεί να πληκτρολογήσει και πραγματικό αριθμό. Επίσης ελέγχει αν αντί του χαρακτήρα ‘,’ που είναι για τους πραγματικούς πατηθεί ο χαρακτήρας ‘.’ και τον αλλάζει αυτόματα στον ‘,’ .

ΦΟΡΜΑ ABOUT

Είναι μία Modal φόρμα που περιέχει πληροφορίες για τον συγγραφέα και το πρόγραμμα υλοποίησης της όλης εργασίας.

Η φόρμα αυτή αντιστοιχεί στο Unit3 και αποτελείται από τα εξής αρχεία:

- Unit3.cpp
- Unit3.h
- Unit3.dfm

Κλάση

TAbout: Η κλάση της modal φόρμας.

Συναρτήσεις-Μέλη

void __fastcall TAbout::FormCanResize(TObject *Sender, int &NewWidth, int &NewHeight, bool &Resize)

Καθορίζει ότι το μέγεθος της φόρμας είναι αμετάβλητο (Resize = false).

Σημειώνεται ότι τα Unit1 και Unit3 που αντιστοιχούν στις φόρμες Entry και Initial αντίστοιχα, είναι κοινά και για τις δύο εφαρμογές, του SCS και του CCS. Για αυτό αναφέρονται μόνο σε αυτό το κεφάλαιο.

Ο ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΤΟΥ CCS

ΤΕΧΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Το κεφάλαιο αυτό περιγράφει την δομή ενός πιο ρεαλιστικού μοντέλου των Classifier Systems για το πρόβλημα του πολυπλέκτη. Για λόγους που αφορούν τον χρόνο και την υπολογιστική ισχύ ασχολείται μόνο με τον πολυπλέκτη 4 σε 1 (64 δυνατοί συνδυασμοί εισόδων-εξόδων) εφόσον μπορεί εύκολα να επεκταθεί και σε πιο πολύπλοκους πολυπλέκτες (π.χ. 8 σε 1).

Το πρώτο βήμα είναι ο καθορισμός του πληθυσμού μαζί με τα χαρακτηριστικά κάθε ατόμου που τον αποτελούν. Για τον ορισμό ενός ατόμου χρησιμοποιείται η παρακάτω δομή:

```
typedef struct
{
  AnsiString condition;
  AnsiString action;
  double strength;
  double bid;
  bool matchflag;
}classtype;
```

Όπως παρατηρείτε κάθε άτομο αποτελείται από έναν ανεξάρτητο κανόνα (κι όχι σετ κανόνων). Οι πρώτες δύο αλφαριθμητικές μεταβλητές αναφέρονται στο *Condition* και το *Action* αντίστοιχα του κάθε Classifier. Επίσης υπάρχουν και κάποιες μεταβλητές που αναφέρονται στα χαρακτηριστικά του ατόμου. Η μεταβλητή *Strength* αναφέρεται στη δύναμη του και είναι στην ουσία η ποιότητα (fitness). Ακόμη η μεταβλητή *bid* αναφέρεται στην προσφορά που κάνουν τα Classifiers (άτομα) προκειμένου να ενεργοποιηθούν και καθορίζεται από την γραμμική σχέση της δύναμης και του συντελεστή προσφοράς (Coefficient Bid).

$$bid = strength \times Coefficient\_Bid$$

Τέλος η λογική μεταβλητή *matchflag* χρησιμεύει για να δηλώσει ποια άτομα ταίριαξαν με μία είσοδο (true) ή όχι (false).

Και για το πρόγραμμα αυτό χρειάστηκε να οριστούν οι δύο δυναμικοί πίνακες που περιλαμβάνουν τις δυνατές εισόδους *detc[]* και τις αντίστοιχες εξόδούς αυτών *out[]*.

Οι τιμές για κάθε θέση των δύο πινάκων καθορίζονται επίσης από τις συναρτήσεις IntToBinary() για τον *detc[]* και την OutputMsg() για τον *out[]*. Η ανάλυση των δύο αυτών συναρτήσεων καθώς και τα σώματα του ορισμού τους περιγράφονται στο προηγούμενο κεφάλαιο του SCS.

Πριν τελειώσουμε με το Performance System είναι αναγκαίο να περιγραφεί μια συνάρτηση με θεμελιακό ρόλο. Είναι η συνάρτηση match() που εμφανίζεται και στον SCS, ασφαλώς τροποποιημένη για τις ανάγκες του παρόντος προγράμματος του CCS. Κι ενώ πριν η συνάρτηση αυτή χρησίμευε στο να καθορίζει τον αριθμό των ταιριασμάτων μεταξύ όλων των δυνατών εισόδων και του κάθε ατόμου (σετ κανόνων) που παρήγαγε τη σωστή έξοδο για κάθε είσοδο, πλέον τροποποιείται στο να ελέγχει αν ταιριάζει μία μόνο είσοδος με το κάθε άτομο/κανόνα του πληθυσμού και να επιστρέφει *αληθή* ή *ψευδή* τιμή ανάλογα αν ταιριάζουν ή όχι και η οποία τιμή να λαμβάνεται από την μεταβλητή *matchflag* του κάθε ατόμου.

Συνέχεια έχουν τα iterations του BB. Σε κάθε iteration λαμβάνει χώρα η δημοπρασία, η πληρωμή επιταγών και η φορολογία των ατόμων.

Για την *δημοπρασία* (Auction), υπάρχει μια ομώνυμη υπεύθυνη συνάρτηση. Σ' αυτήν στέλνονται ορίσματα ο πληθυσμός, μια δυναμική λίστα η *clist[]* που κρατάει τα άτομα που ταίριαξαν με μια συγκεκριμένη είσοδο, και ακόμη το πλήθος των στοιχείων της *clist[]*. Εσωτερικά η συνάρτηση αυτή βρίσκει και επιστρέφει την θέση του επικρατέστερου ατόμου – άτομο με την μεγαλύτερη προσφορά bid-. Στην περίπτωση της ισοβαθμίας εισάγεται θόρυβος με την βοήθεια της συνάρτησης noise(), μέσω του οποίου διαφοροποιούνται τα ισοδύναμα άτομα και μπορούμε έτσι να επιλέξουμε το επικρατέστερο. Σώμα συνάρτησης της Auction():

```
double noise(double bidN)
{
   bidN+= bidN * (rnd()*(0.02*noiseP)-(0.01*noiseP));
   return bidN;
}

int Auction(classtype par[], int cl[], int N)
{
  int p,i,best;
  double max;
  double * temp;
  temp=new double[N];
```

```
  for(i=0;i<N;i++)
  {
    p=cl[i];
    temp[i]=noise(par[p].bid);
  }
  max=temp[0];
  best=0;
  for(i=1;i<N;i++)
  {
    if(max<temp[i])
    {
      max=temp[i];
      best=i;
    }
  }
  delete(temp);
  return cl[best];
}
```

Για την *πληρωμή Επιταγών* λαμβάνεται το άτομο νικητής της προηγούμενης διαδικασίας και του αφαιρείται το ποσό bid που πρόσφερε για την ενεργοποίηση του. Υπεύθυνη για αυτό είναι η Payment() που καλείται μέσα από την ευρύτερη συνάρτηση ClearingHouse(). Αν τύχει και το μήνυμα αυτού του ατόμου είναι σωστό τότε υπάρχει επιβράβευση γι' αυτό το άτομο που του δίνεται μέσα από την συνάρτηση Reward(). Παρακάτω φαίνονται τα σώματα αυτών των συναρτήσεων:

```
classtype ClearingHouse(classtype p1,int ROutput)
{
  p1=Payment(p1);
  p1.strength=Reward(p1,ROutput);
  return p1;
}

classtype Payment(classtype p2)
{
  p2.strength=p2.strength-p2.bid;     //new strength definition
  p2.bid=p2.strength*CBID;            //new bid definition
  return p2;
}

double Reward(classtype p2, int ROutput)
{
   if(p2.action==ROutput)         //if right output is equal to action give a reward
  {
    p2.strength=p2.strength+REWARDBID;
    p2.bid=p2.strength*CBID;
```

```
    }
    return p2.strength;
}
```

Η *είσπραξη φόρων* τώρα είναι μια διαδικασία που αποτρέπει την ανεξέλεγκτη αύξηση της δύναμης των κανόνων μέσα από δύο είδη φόρων που επιβάλλονται στα άτομα. Το πρώτο είδος εφαρμόζεται και λαμβάνεται από όλα τα άτομα και καθορίζεται από την πραγματική μεταβλητή LIFETAX. Το δεύτερο είδος εφαρμόζεται μόνο στα άτομα που έκαναν Bid και καθορίζεται από την πραγματική μεταβλητή BIDTAX. Αν και τα δύο αυτά μεγέθη είναι διάφορα του μηδενός, υπάρχει κατάλληλος μηχανισμός μέσα στο σώμα της συνάρτησης με το οποίο εισπράττετε το κατάλληλο ποσό φόρου από κάθε άτομο.

```
classtype TaxCollector(classtype p1)
{
  int bidtaxswitch;

  if(p1.matchflag) bidtaxswitch=1;
  else             bidtaxswitch=0;
  p1.strength= p1.strength – LIFETAX*p1.strength – BIDTAX*bidtaxswitch*
               p1.strength;
  p1.bid=p1.strength*CBID;
  return p1;
}
```

Εδώ τελειώνει ο αλγόριθμος του BB και στο προσκήνιο ανεβαίνει μια άλλη σημαντική διεργασία, ο Γενετικός Αλγόριθμος. Είναι ελαφρώς τροποποιημένος από το προηγούμενο μοντέλο του SCS, μιας και τα άτομα δεν αποτελούνται όπως έχει ήδη αναφερθεί από σετ κανόνων αλλά από ξεχωριστά ανεξάρτητα Classifiers. Και γι' αυτήν την εργασία χρησιμοποιήθηκαν οι ίδιες ή παραλλαγές των διαδικασιών επιλογής, αποτίμησης, ανασυνδυασμού και αντικατάστασης.

Για την επιλογή χρησιμοποιήθηκε και πάλι η μέθοδος Roulette Wheel τροποποιημένη για έναν κανόνα κι όχι για έναν πίνακα από αυτούς. Η συνάρτηση που της αντιστοιχεί είναι η RouletteWheel().

Ακολουθεί ο τελεστής διασταύρωσης που κι εδώ μαζί με τον τελεστή μετάλλαξης λειτουργούν πιθανοκρατικά. Χρησιμοποιήθηκε διασταύρωση ενός σημείου (βλ. ΓΑ-Τελεστής Διασταύρωσης). Εδώ όπως παρατηρεί κανείς κι από την συνάρτηση, τα δύο άτομα που επιλέχθηκαν ανασυνδυάζονται βάση ενός σημείου κοπής. Ακόμη για το παραγόμενο άτομο καθορίζονται εσωτερικά της συνάρτησης και τα βάρη της

δύναμης και της προσφοράς. Η δύναμη προκύπτει ως ο μέσος όρος των δυνάμεων των δύο γονέων και η προσφορά από την γνωστή γραμμική σχέση της με την δύναμη.

Ο τελεστής μετάλλαξης του CCS λειτουργεί παρόμοια με αυτόν του SCS αν στη θέση του ενιαίου γονότυπου που αποτελείται από N κανόνες βάλουμε μόνο έναν.

Η αντικατάσταση σταθερής κατάστασης υλοποιείται με έναν βρόχο **for()** που τρέχει τόσες φορές όσες και τα άτομα που θέλουμε να αντικαταστήσουμε.

Η εφαρμογή τερματίζει μόλις ικανοποιηθεί το κριτήριο του κύριο βρόχου που είναι μια επαναληπτική μέθοδος **do..while{}**. Ο βρόχος αυτός περιλαμβάνει τον ΒΒ και τον ΓΑ. Μόλις ολοκληρωθούν τα συνολικά iterations τερματίζει.

ΤΑ UNIT ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

ΦΟΡΜΑ ΤΟΥ SCS

Είναι η κύρια φόρμα του προγράμματος που προσομοιώνει την συνάρτηση εισόδου εξόδου του σύνθετου ψηφιακού πολυπλέκτη με την χρήση του «Συστήματος Καταμερισμού Βαθμών».

Η φόρμα αυτή αντιστοιχεί στο Unit4 και αποτελείται από τα εξής αρχεία:

- Unit4.cpp
- Unit4.h
- Unit4.dfm

Κλάση

TCCSMainForm: Η κλάση της κύριας φόρμας.

Σταθερές

#define ABITS 1: Σταθερά για τον καθορισμό του μήκους του μηνύματος.
#define LIFETAX 0.0 : Σταθερά για την φορολογία όλων των κανόνων.
#define BIDTAX 0.01 : Σταθερά για την φορολογία μόνο των κανόνων που

ταίριαξαν με την είσοδο.

#define address 2 : Σταθερά που δηλώνει το μέγεθος του Address των Classifiers.

Μεταβλητές και Δομές

```
typedef struct
{
  AnsiString condition;
  AnsiString action;
  double strength;
  double bid;
  bool matchflag;
}classtype;
```

Δομή τύπου *classtype*.

int P :

Μεταβλητή που ορίζεται από τον χρήστη και δηλώνει το πλήθος του Classifier Store

double CROSSPROB :

Ορίζεται από τον χρήστη και δηλώνει την πιθανότητα διασταύρωσης.

double PMUT :

Ορίζεται από τον χρήστη και δηλώνει την πιθανότητα μετάλλαξης.

double CBID :

Ορίζεται από τον χρήστη και δηλώνει τον Συντελεστή Προσφοράς.

int REWARDBID :

Ορίζεται από τον χρήστη και δηλώνει την επιβράβευση.

int iteration :

Δηλώνει τον αριθμό των iterations σε κάθε εκτέλεση του BB. Το total iteration ορίζει πόσα τέτοια iterations θα έχουμε.

int TotalIteration :

Ορίζεται από τον χρήστη και δηλώνει τα συνολικά iterations. Η τιμή που εισάγεται πολλαπλασιάζετε με την μεταβλητή iteration.

int offspringN :

Δηλώνει τον αριθμό των απογόνων που δημιουργούνται σε κάθε εκτέλεση του ΓΑ. Ορίζεται από τον χρήστη.

int noiseP :

Δηλώνει το ποσοστό του θορύβου που χρησιμοποιείται για τον ασαφή προσδιορισμό του επικρατέστερου κανόνα από ένα σύνολο ισοδύναμων κανόνων. Ορίζεται από τον χρήστη.

int data :

Μεταβλητή που δηλώνει το μέγεθος του *data* των κανόνων. Εξαρτάται από το *address*.

int BITS :

Μέγεθος του *Condition*. Εξαρτάται από τα *address* και *data.*

int T :

Πλήθος δυνατών συνδυασμών εισόδου-εξόδου. Εξαρτάται από το *BITS.*

Συναρτήσεις

double rnd(void)

Παράγει έναν τυχαίο αριθμό στο διάστημα [0,1].

AnsiString IntToBinary(int x)

Παίρνει σαν όρισμα έναν αριθμό μεταξύ του ‘0’ και του μέγιστου των δυνατών συνδυασμών εισόδων-εξόδων και επιστρέφει την δυαδική του μορφή.

Int OutputMsg(AnsiString str)

Παίρνει σαν όρισμα την δυαδική μορφή μιας εισόδου και επιστρέφει την σωστή έξοδο για αυτήν την είσοδο.

void match(classtype *par1,AnsiString str)

Ταιριάζει μία είσοδο με έναν κανόνα. Επιστρέφει true ή false ανάλογα με το αν ταιριάζουν ή όχι αντίστοιχα.

classtype TaxCollector(classtype p1)

Η συνάρτηση αυτή αποδίδει φόρους στα άτομα.

int Auction(classtype par[], int cl[], int N)

Η συνάρτηση δημοπρασίας. Επιστρέφει την θέση του ατόμου που έκανε την μεγαλύτερη προσφορά.

int noise(double bidN)

Η συνάρτηση αυτή προσθέτει θόρυβο στις τιμές *bid* των ατόμων χωρίς να αποθηκεύεται με σκοπό να αναδείξει με μια ελαφριά ασάφεια το ποιος από τους κανόνες κερδίζει κάθε φορά.

classtype ClearingHouse(classtype p1, int ROutput)

Συνάρτηση πληρωμής Επιταγών.

Classtype Payment(classtype p2)

Η συνάρτηση αυτή αφαιρεί το ποσό της προσφοράς από το ενεργοποιημένο άτομο.

double Reward(classtype p2,int ROutput)

Αν η έξοδος του ενεργοποιημένου ατόμου είναι σωστή, τότε η συνάρτηση αυτή επιβραβεύει το συγκεκριμένο άτομο.

int roulettewheel(classtype par1[])

Συνάρτηση επιλογής γονέων βάση της ποιότητας/δύναμης.

classtype Crossover1Point(classtype s1, classtype s2)

Συνάρτηση διασταύρωσης ενός σημείου.

classtype Mutation(classtype p)

Συνάρτηση μετάλλαξης ατόμων.

int Success(classtype * par, AnsiString * detc)

Συνάρτηση επιτυχίας. Εισάγοντας όλες τις δυνατές εισόδους στο σύστημα συγκρίνει και επιστρέφει πόσες από αυτές ταιριάζουν με τους κανόνες προκαλώντας σωστά μηνύματα.

Συναρτήσεις-Μέλη

void __fastcall TCCSMainForm::RunButtonClick(TObject *Sender)

Είναι η βασική συνάρτηση μέσα στην οποία γίνονται όλες οι διεργασίες:

- Δημιουργία δυναμικού πληθυσμού γονέων:

  classif * par;

par=new classif[P];

- Δημιουργία δυναμικών πινάκων για τις εισόδους – εξόδους

int* out;
out=new int[T];
AnsiString* detc;
detc=new AnsiString[T];

Κλήσεις όλων των παραπάνω συναρτήσεων που είναι υπεύθυνες για την γενετική εξέλιξη των ατόμων καθώς και κάποιων βοηθητικών συναρτήσεων (π.χ. rnd() , IntToBinary(), …).

void __fastcall TCCSMainForm::FormClose(TObject *Sender, TCloseAction &Action)

Τερματίζει την εφαρμογή.

void __fastcall TCCSMainForm::QuitCClick(TObject *Sender)

Όμοια με την προηγούμενη

void __fastcall TCCSMainForm::About1Click(TObject *Sender)

Εμφανίζει την φόρμα About.

void __fastcall TCCSMainForm::StopCClick(TObject *Sender)

Σταματάει την διαδικασία σε όποιο σημείο της εκτέλεσης θέλουμε.

void __fastcall TCCSMainForm::DefaultSizeForm1Click(TObject *Sender)

Επανακαθορίζει την φόρμα στο αρχικό της μέγεθος.

void __fastcall TCCSMainForm::RootMenu1Click(TObject *Sender)

Επιστρέφει στην εισαγωγική φόρμα.

void __fastcall TCCSMainForm::FormCanResize(TObject *Sender, int &NewWidth, int &NewHeight, bool &Resize)

Ελέγχει τα όρια της φόρμας να μην ξεπεράσουν τα επιθυμητά.

void __fastcall TCCSMainForm::Parameters1Click(TObject *Sender)

Εμφανίζει την φόρμα αρχικοποίησης των μεγεθών.

ΦΟΡΜΑ ΑΡΧΙΚΟΠΟΙΗΣΗΣ ΤΙΜΩΝ INITIAL

Είναι μία Modal φόρμα που εμφανίζεται για να εισάγουμε τα αρχικά στοιχεία. Αποτελείται από 9 Edit Boxes , 9 Labels και 2 BitBtns.

Η φόρμα αυτή αντιστοιχεί στο Unit6 και αποτελείται από τα εξής αρχεία:

- Unit6.cpp
- Unit6.h
- Unit6.dfm

Κλάση

TInitial_CCS: Η κλάση της φόρμας.

Συναρτήσεις-Μέλη

void __fastcall TInitial_CCS::FormCreate(TObject *Sender)

Καλείται με το που δημιουργείται η φόρμα. Χρησιμεύει στο να καθορίζει τις αρχικές τιμές των μεγεθών. Ο χρήστης βέβαια μπορεί να τις αλλάξει ανά πάσα στιγμή.

void __fastcall TInitial_CCS::FormCanResize(TObject *Sender, int &NewWidth, int &NewHeight, bool &Resize)

Καθορίζει ότι το μέγεθος της φόρμας είναι αμετάβλητο (Resize = false).

void __fastcall TInitial_CCS::popEditKeyPress(TObject *Sender, char &Key)

Καλείται όταν εισάγουμε τα δεδομένα στο popEdit Box και ελέγχει αν έχουν δοθεί σωστά. Αν για παράδειγμα πατηθεί χαρακτήρας αντί αριθμού τότε εμφανίζεται αντίστοιχο μήνυμα που μας ειδοποιεί ότι πρέπει να επανακαθορίσουμε την τιμή του στοιχείου.

void __fastcall TInitial_CCS::total_iterationEditKeyPress(TObject *Sender,

char &Key)

Όμοια κι εδώ ελέγχεται τι πατήθηκε μέσα στο total_iterationEdit Box και αντίστοιχα είτε εκχωρεί την τιμή στο στοιχείο είτε εμφανίζει μήνυμα για τον επανακαθορισμό του μεγέθους.

void __fastcall TInitial_CCS:: iterationEditKeyPress(TObject *Sender,
char &Key)

Όμοια κι εδώ ελέγχεται τι πατήθηκε μέσα στο iterationEdit Box και αντίστοιχα είτε εκχωρεί την τιμή στο στοιχείο είτε εμφανίζει μήνυμα για τον επανακαθορισμό του μεγέθους.

void __fastcall TInitial_CCS::rewardEditKeyPress(TObject *Sender,
char &Key)

Όμοια κι εδώ ελέγχεται τι πατήθηκε μέσα στο rewardEdit Box και αντίστοιχα είτε εκχωρεί την τιμή στο στοιχείο είτε εμφανίζει μήνυμα για τον επανακαθορισμό του μεγέθους.

void __fastcall TInitial_CCS::offspringNumberEditKeyPress(TObject *Sender,
char &Key)

Όμοια κι εδώ ελέγχεται τι πατήθηκε μέσα στο offspringNumberEdit Box και αντίστοιχα είτε εκχωρεί την τιμή στο στοιχείο είτε εμφανίζει μήνυμα για τον επανακαθορισμό του μεγέθους.

void __fastcall TInitial_CCS::noiseEditKeyPress(TObject *Sender, char &Key)

Όμοια κι εδώ ελέγχεται τι πατήθηκε μέσα στο noiseEdit Box και αντίστοιχα είτε εκχωρεί την τιμή στο στοιχείο είτε εμφανίζει μήνυμα για τον επανακαθορισμό του μεγέθους.

Οι έξι αυτές παράμετροι λαμβάνουν μόνο ακέραιους αριθμούς. Έτσι αν πατηθεί ακόμη και ο χαρακτήρας ‘,’ εμφανίζεται πάλι το μήνυμα λάθους.

void __fastcall TInitial_CCS::crosspEditKeyPress(TObject *Sender, char &Key)

Όμοια ελέγχεται τι πληκτρολόγησε ο χρήστης του προγράμματος για την πιθανότητα Διασταύρωσης. Ο χρήστης μπορεί να πληκτρολογήσει και πραγματικό αριθμό. Επίσης ελέγχει αν αντί του χαρακτήρα ‘,’ που είναι για τους δεκαδικούς πατηθεί ο χαρακτήρας ‘.’ και αλλάζει αυτόματα στον ‘,’ .

void __fastcall TInitial_CCS::mutpEditKeyPress(TObject *Sender, char &Key)

Όμοια ελέγχεται τι πληκτρολόγησε ο χρήστης του προγράμματος για την πιθανότητα μετάλλαξης. Ο χρήστης μπορεί να πληκτρολογήσει και πραγματικό αριθμό. Επίσης ελέγχει αν αντί του χαρακτήρα ‘,’ που είναι για τους δεκαδικούς πατηθεί ο χαρακτήρας ‘.’ και αλλάζει αυτόματα στον ‘,’ .

void __fastcall TInitial_CCS::cbidEditKeyPress(TObject *Sender, char &Key)

Όμοια ελέγχεται τι πληκτρολόγησε ο χρήστης του προγράμματος για τον συντελεστή προσφοράς. Ο χρήστης μπορεί να πληκτρολογήσει και πραγματικό αριθμό. Επίσης ελέγχει αν αντί του χαρακτήρα ‘,’ που είναι για τους δεκαδικούς πατηθεί ο χαρακτήρας ‘.’ και αλλάζει αυτόματα στον ‘,’ .

ΦΟΡΜΑ ΑΡΧΙΚΟΠΟΙΗΣΗΣ ΦΟΡΩΝ

Είναι μία Modal φόρμα που εμφανίζεται για να εισάγουμε τα αρχικά στοιχεία. Αποτελείται από 2 Edit Boxes , 2 Labels και 2 BitBtns.

Η φόρμα αυτή αντιστοιχεί στο Unit7 και αποτελείται από τα εξής αρχεία:

- Unit7.cpp
- Unit7.h
- Unit7.dfm

Κλάση

TTaxes: Η κλάση της φόρμας.

Συναρτήσεις-Μέλη

void __fastcall TTaxes::FormCanResize(TObject *Sender, int &NewWidth, int &NewHeight, bool &Resize)

Καθορίζει ότι το μέγεθος της φόρμας είναι αμετάβλητο (Resize = false).

void __fastcall TTaxes::FormCreate(TObject *Sender)

Καλείται με το που δημιουργείται η φόρμα. Χρησιμεύει στο να καθορίζει τις αρχικές τιμές των μεγεθών. Ο χρήστης βέβαια μπορεί να τις αλλάξει ανά πάσα στιγμή.

void __fastcall TTaxes::lifetaxEditKeyPress(TObject *Sender, char &Key)

Ελέγχεται τι πληκτρολόγησε ο χρήστης του προγράμματος για την παράμετρο αυτή. Ο χρήστης μπορεί να πληκτρολογήσει και πραγματικό αριθμό. Επίσης ελέγχει αν αντί του χαρακτήρα ‘,’ που είναι για τους δεκαδικούς πατηθεί ο χαρακτήρας ‘.’ και αλλάζει αυτόματα στον ‘,’ .

void __fastcall TTaxes::bidtaxEditKeyPress(TObject *Sender, char &Key)

Όμοια με την παραπάνω για το BIDTAX.

Το πρόγραμμα επίσης “βλέπει” και την εισαγωγική φόρμα καθώς και την φόρμα About, οι οποίες ήδη αναφέρθηκαν στο προηγούμενο πρόγραμμα και για αυτό δεν υπάρχει λόγος να αναλυθούν στο σημείο αυτό.

ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

Βιβλιογραφία

1. Σημειώσεις μαθήματος «Εξελικτικής Υπολογιστικής» του κ. Σπυρίδων Καζαρλή.
2. «GENETIC ALGORITHMS in Search, Optimization & Machine Learning», David E. Goldberg.
3. «Μαθηματικά Πρότυπα Φυσιολογίας-Επεξεργασία Σήματος και Εικόνας» , Αδαμόπουλος Αδάμ.