Ατομική Διπλωματική Εργασία

ΟΛΟΚΛΗΡΩΜΕΝΗ ΔΙΑΔΙΚΤΥΑΚΗ ΕΦΑΡΜΟΓΗ ΑΝΑΛΥΣΗΣ ΚΑΙ ΑΥΤΟΜΑΤΟΥ ΕΛΕΓΧΟΥ ΚΩΔΙΚΑ ΛΟΓΙΣΜΙΚΟΥ ΣΕ JAVA.

Παναγιώτης Πέτσας

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΟΛΟΚΛΗΡΩΜΕΝΗ ΔΙΑΔΙΚΤΥΑΚΗ ΕΦΑΡΜΟΓΗ ΑΝΑΛΥΣΗΣ ΚΑΙ ΑΥΤΟΜΑΤΟΥ ΕΛΕΓΧΟΥ ΚΩΔΙΚΑ ΛΟΓΙΣΜΙΚΟΥ ΣΕ JAVA.

Παναγιώτης Πέτσας

Επιβλέπων Καθηγητής
Ανδρέας Ανδρέου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2009

Ευχαριστίες

Ένα μεγάλο ευχαριστώ χρωστάω στους γονείς μου για τη συμπαράσταση, στήριξη, κατανόηση και υπομονή που μου δείξανε καθ'όλη τη διάρκεια της φοίτησής μου.
Θα ήθελα επίσης να ευχαριστήσω τον καθηγητή κ. Ανδρέα Ανδρέου που επέβλεπε αυτή τη διπλωματική εργασία όπως επίσης και τον καθηγητή κ. Αναστάσιο Σοφοκλέους, που με την καθοδήγησή τους, την εμπειρία, τις εισηγήσεις και το υλικό που παρείχαν, διευκόλυναν σε πολύ μεγάλο βαθμό την εκπόνηση της εργασίας. Χωρίς την επίβλεψή και βοήθειά τους η αποπεράτωση μιας τέτοιας εργασίας θα ήταν αδύνατη.
Ακόμη θέλω να εκφράσω τις ευχαριστίες μου στη φίλη μου Angela Molina, στον φίλο μου Κωνσταντίνο Κουλίνα και τους συμφοιτητές μου Γιώργο Νικολάου, Παναγιώτη Μάρκου, και Άντρια Κρόκου για την ψυχολογική τους συμπαράσταση και βοήθεια, τον έλεγχο εκδόσεων του εργαλείου και την κατάθεση της γνώμης του όσον αφορά προβλήματα που παρουσιάζονταν κατά καιρούς με τη διεπιφάνεια του εργαλείου. Η πολύτιμη γνώμη τους βοήθησε στην απάλειψη λαθών και τη βελτίωση της εργασίας.

Περίληψη

Τίτλος Ατομικής Διπλωματικής Εργασίας

Ολοκληρωμένη διαδικτυακή εφαρμογή ανάλυσης και αυτόματου ελέγχου κώδικα λογισμικού σε Java.

Εργασία κατευθυνόμενη από

Δρα Ανδρέα Ανδρέου, Καθηγητή, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

Η Ατομική αυτή Πτυχιακή Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, στα πλαίσια του προπτυχιακού μαθήματος Ατομική Διπλωματική Εργασία.

Σκοπός της εργασίας μου είναι η δημιουργία ενός ενοποιημένου εργαλείου ανάλυσης κώδικα, κατασκευής και γραφικής απεικόνισης γράφων ροής δεδομένων, ροής ελέγχου και γράφου εξαρτήσεων καθώς επίσης και σύστημα αυτοματοποιημένης παραγωγής περιπτώσεων ελέγχου με τη βοήθεια γενετικών αλγορίθμων. Ανάμεσα στους στόχους ήταν επίσης η αντικατάσταση παλαιών βιβλιοθηκών με νέες, βελτιωμένες και αναβάθμιση υπαρχουσών βιβλιοθηκών με τις τελευταίες εκδόσεις τους και όλες τις αλλαγές που αυτό συνεπαγόταν. Τέλος, το εργαλείο αυτό προορίζεται για διαδικτυακή χρήση και για το σκοπό αυτό, μετά από έρευνα, χρησιμοποιήθηκε το Java Web Start. Το παρόν εργαλείο χρησιμοποιεί λειτουργίες προηγούμενων εργασιών όπως αυτή του κ. Αναστάσιου Σοφοκλέους, και άλλων προπτυχιακών φοιτητών. Η υλοποίηση ξεκίνησε σε μια καινούρια βάση και η σχεδίαση έγινε με νέα φιλοσοφία και προσεκτικό σχεδιασμό που να διευκολύνει όχι μόνο τη συνένωση των υπαρχόντων συστημάτων αλλά να κάνει και το έργο της περαιτέρω ανάπτυξης του συστήματος πιο εύκολο. Όταν ο πυρήνας της εφαρμογής σχηματίστηκε, και η συνένωση έλαβε χώρα με επιτυχία, ακολούθησε αναδιάρθρωση της διεπιφάνειας της εφαρμογής και η προσθήκη νέων λειτουργιών (καλύτερη διαχείριση πολυνηματικότητας, Οδηγός Ανοίγματος-Φόρτωσης αρχείου, αποθήκευση γράφων σε αρχείο εικόνας, Εγχειρίδιο Χρήσης, εισαγωγή αλγορίθμων διάταξης γράφων, δυναμική αλληλεπίδραση με γράφους, διαδραστική παρουσίαση αποτελεσμάτων παραγωγή περιπτώσεων ελέγχου, συμβατότητα ΛΣ, κ.λπ.) με στόχο να βοηθήσουν την πλοήγηση του χρήστη στο εργαλείο και την παροχή περισσότερων πληροφοριών όταν αυτές χρειάζονται, στα πλαίσια μιας φιλικής προς τον χρήστη διεπαφή.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Υποκίνηση Έρευνας - Κίνητρο

Όλοι όσοι εμπλέκονται στον χώρο της τεχνολογίας λογισμικού γνωρίζουν ότι ο έλεγχος λογισμικού είναι μια διαδικασία επαναλαμβανόμενη και άκρως απαραίτητη καθ' όλες τις φάσεις ανάπτυξης του λογισμικού. Είναι ένα αναγκαίο κακό που ταλαιπωρεί διευθυντές, σχεδιαστές, αναλυτές και προγραμματιστές με ερωτήσεις του τύπου :

Τί είδους έλεγχος θα πρέπει να γίνει ;

Από ποιόν θα πρέπει να γίνει ;

Πως θα πρέπει να πραγματοποιηθεί:

Πότε θα μπορεί να θεωρηθεί ότι ο έλεγχος που έγινε είναι επαρκής;

Δύσκολες ερωτήσεις σαν αυτές βρίσκουν απαντήσεις σε τεχνικές ελέγχου λογισμικού οι οποίες μπορούν να αναγνωρίσουν αδυναμίες σε κάποια σημεία της εφαρμογής, να καθορίσουν το βαθμό της ποιότητας του κώδικα και να επιβεβαιώσουν την πεποίθηση ότι η

εφαρμογή όχι μόνο είναι χρησιμοποιήσιμη αλλά αποδοτική , αποτελεσματική και ότι ικανοποιεί τις απαιτήσεις του πελάτη-χρήστη χωρίς να παρουσιάσει απρόβλεπτες συμπεριφορές. Οι απρόβλεπτες συμπεριφορές αυτές μπορεί να σημαίνουν λάθη, ασυνέπειες στον κώδικα, σφάλματα, κακή διαχείριση πόρων, κλπ. Τέτοιου είδους συμπεριφορές μπορεί να προκαλέσουν από απώλεια εμπιστοσύνης προς την εφαρμογή, εκνευρισμό στο χρήστη, απώλεια σημαντικών δεδομένων, να κοστίσουν χρόνο και χρήμα στους χρήστες ή ακόμη και να θέσουν σε κίνδυνο ανθρώπινες ζωές, όταν αναξιόπιστες εφαρμογές με ελλιπή έλεγχο ενσωματώνονται σε συστήματα πλοήγησης αυτοκινήτων, αεροπλάνων, συστήματα ασφαλείας, μηχανήματα ιατρικής περίθαλψης, κ.ο.κ. Αντιλαμβανόμαστε λοιπόν πως ο έλεγχος λογισμικού δεν είναι απλά μια διαδικασία δαπανηρή σε χρόνο και χρήμα αλλά ακρογωνιαίος λίθος στην παραγωγή λογισμικού και μια βασική προϋπόθεση για κάθε εφαρμογή που πρόκειται να κυκλοφορήσει στην αγορά. Για το λόγο αυτό δημιουργείται η ανάγκη ύπαρξης μιας ολοκληρωμένης εφαρμογής που να παρέχει αυτοματοποιημένο έλεγχο, παράγοντας περιπτώσεις ελέγχου και έχοντας γνώση της λειτουργικότητας του προγράμματος (white-box) να εγγυάται συγκεκριμένα, ευρέως χρησιμοποιούμενα κριτήρια κάλυψης με έναν ευφυή και αποτελεσματικό τρόπο.

1.2 Έλεγχος λογισμικού σήμερα

Υπάρχουν πολλές προσεγγίσεις όσον αφορά τα είδη ελέγχου λογισμικού που μπορούν να γίνουν. Αυτό εξαρτάται από το ποια οπτική γωνία της εφαρμογής μας ενδιαφέρει να ελέγξουμε. Ανάμεσα στα είδη αυτά, είναι ο έλεγχος μονάδων (unit), απόδοσης, λειτουργικότητας, διεπιφάνειας, ασφάλειας, κ.ο.κ. Ο έλεγχος μπορεί να γίνει στατικά, από ομάδες ελεγκτών (προγραμματιστές, αναλυτές) οι οποίοι πραγματοποιούν reviews, walkthroughs ή inspections στον κώδικα της εφαρμογής υπό-έλεγχο ή δυναμικά με ανάλυση του κώδικα και εκτέλεσή του με δεδομένο σύνολο από περιπτώσεις ελέγχου (test-cases). Αυτό μπορεί να γίνει και με τη χρήση αυτοματοποιημένων εργαλείων. Σήμερα υπάρχουν πολλά εργαλεία που παρέχουν διάφορα είδη ελέγχου λογισμικού, όπως έλεγχο κάλυψης κώδικα, έλεγχο για διαρροή μνήμης, ανάλυση του κώδικα, εύρεση εξαρτήσεων μέσα στον κώδικα, έλεγχος οπισθοδρόμησης (regression testing) κλπ.

Υπάρχουν δυο κύριες μέθοδοι, όπως θα δούμε αναλυτικότερα σε επόμενο κεφάλαιο για τον έλεγχο λογισμικού. Η πρώτη μέθοδος, μαύρο κουτί (black box) αντιμετωπίζει τον κώδικα της εφαρμογής σαν ένα μαύρο κουτί, του οποίου η εσωτερική υλοποίηση και τρόπος λειτουργίας είναι άγνωστα. Συνοπτικά, αυτό που θέλει να επιτύχει ο έλεγχος black box είναι δεδομένων κάποιων εισόδων (inputs) να διαπιστώσει εάν τα αποτελέσματα είναι αναμενόμενα και αν τηρούν τις προδιαγραφές και απαιτήσεις της εφαρμογής. Είδη ελέγχου black box [9] είναι τα εξής :
equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.
Τα πλεονεκτήματα αυτού του είδους ελέγχου είναι ότι ο ελεγκτής που κάνει έλεγχο black box έχει ένα απλό σκεπτικό : "Πρέπει να βρω σφάλματα". Δεν υπάρχει κανένας δεσμός ανάμεσα στον ελεγκτή και τον κώδικα και αυτό πολλές φορές σημαίνει ότι μπορεί πιο εύκολα να βρει περισσότερα λάθη από ότι ο συγγραφέας του κώδικα. Από την άλλη πλευρά, ο έλεγχος black box είναι σαν να περπατάει ο ελεγκτής σε έναν σκοτεινό λαβύρινθο χωρίς φως : μπορεί να χρειαστεί να παράγει πολλές περιπτώσεις ελέγχου για να βρει ένα σφάλμα, κάτι που αν είχε γνώση του κώδικα θα μπορούσε ίσως να γίνει με μια μόνο περίπτωση ελέγχου.
Η άλλη μέθοδος είναι η μέθοδος άσπρου κουτιού (white-box) ελέγχου. Σύμφωνα με αυτή τη μέθοδο ο ελεγκτής του κώδικα έχει γνώση της υλοποίησης του κώδικα, των εσωτερικών δομών δεδομένων και αλγορίθμων. Είδη τέτοιου ελέγχου είναι :

- API Testing – έλεγχος διεπιφάνειας εφαρμογών
- Code Coverage – Δημιουργία ελέγχων για την ικανοποίηση κριτηρίων κάλυψης κώδικα.
- Fault Injection Methods – Μέθοδοι προσθήκης σφαλμάτων
- Mutation Testing Methods – Μέθοδοι ελέγχου μετάλλαξης
- Static Testing – Στατικός έλεγχος. Η τεχνική άσπρου κουτιού περιλαμβάνει όλο τον στατικό έλεγχο.

Το είδος που μας ενδιαφέρει περισσότερο και θα αναλυθεί σε βάθος στο επόμενο κεφάλαιο είναι η κάλυψη του κώδικα. Οι πιο τυπικές τεχνικές σχεδίασης κάλυψης κώδικα βασίζονται στους γράφους ροής ελέγχου (control flow graph) και ροής δεδομένων (data

flow graph).

Θα πρέπει να σημειωθεί σε αυτό το σημείο ότι στις μέρες μας υπάρχει και μια τρίτη μέθοδος ελέγχου που συνδυάζει τις δυο προηγούμενες [10,11]. Είναι η μέθοδος ελέγχου γκρι κουτού (grey-box testing). Η φιλοσοφία αυτού του ελέγχου είναι ότι υπάρχει γνώση των εσωτερικών δομών και αλγορίθμων του κώδικα για σκοπούς σχεδίασης και μόνο των περιπτώσεων ελέγχου, ενώ ο έλεγχος γίνεται με την αφαιρετικότητα του black-box ελέγχου. Στην μέθοδο αυτή μπορεί να συμπεριληφθεί και η τεχνική του reverse engineering για καθορισμό μηνυμάτων λαθών και ακραίων τιμών (boundary values). Περισσότερο βάθος στον έλεγχο λογισμικού, τις τεχνικές, εργαλεία και αποτελέσματα σε διάφορα λειτουργικά συστήματα δίνεται στη βιβλιογραφία [8, 9, 10, 11, 13].

1.3 Ανάλυση πηγαίου κώδικα

Η ανάλυση του πηγαίου κώδικα είναι μια διαδικασία στατική (static analysis), δηλαδή διεκπεραιώνεται χωρίς την εκτέλεση του λογισμικού. Στις περισσότερες περιπτώσεις γίνεται με βάση τον πηγαίο κώδικα και όχι το εκτελέσιμο αρχείο. Η πολύ σημαντική αυτή διαδικασία πραγματοποιείται αυτόματα από εργαλεία-εφαρμογές και παρέχει πολλές σημαντικές πληροφορίες για τον κώδικα. Το παρόν εργαλείο πραγματοποιεί τη διαδικασία με αναλυτές (parsers), όπως θα δούμε σε μεγαλύτερο βάθος στη συνέχεια και δημιουργεί γράφο ροής δεδομένων (data flow graph – DFG) , γράφο ροής ελέγχου (control flow graph – CFG) και γράφο απαιτήσεων (dependence graph – DPG).

1.4 Δημιουργία γράφων – γραφική απεικόνιση

Η δημιουργία των γράφων, που έπεται της ανάλυσης κώδικα είναι μια από τις δυνατότητες

του εργαλείου. Η γραφική αναπαράσταση γράφων εξαρτήσεων, ροής δεδομένων και ροής ελέγχου είναι διαθέσιμη για τον χρήστη που πραγματοποιεί έλεγχο. Βλέποντας γραφικά έναν γράφο μπορούν να εξαχθούν χρήσιμα συμπεράσματα και να εντοπισθούν λάθη ή ατέλειες ή περιπτώσεις στις οποίες η υλοποίηση δεν ανταποκρίνεται στην σχεδίαση του συστήματος. Εκτός από αυτό, η δημιουργία των γράφων και η γραφική τους αναπαράσταση στο πάνελ είναι ένα ενδιάμεσο βήμα για την εξαγωγή μονοπατιών και εξέταση του είδους κάλυψης που έχει επιλεχθεί.

1.5 Αυτοματοποιημένη Παραγωγή Περιπτώσεων Ελέγχου

Η παραγωγή περιπτώσεων ελέγχου, είναι μια επαναληπτική επίπονη και απαιτητική διαδικασία αλλά άκρως απαραίτητη ούτως ώστε να μπορεί να εγγυηθεί ότι το προϊόν λογισμικού έχει ελεγχθεί διεξοδικά, τηρεί τις προδιαγραφές του και δεν θα προσφέρει δυσάρεστες εκπλήξεις στους χρήστες. Είναι προφανές λοιπόν ότι αν αυτή η διαδικασία μπορεί να αυτοματοποιηθεί, ο έλεγχος θα γίνεται ταχύτερα, αποδοτικότερα και με μεγαλύτερη αξιοπιστία από την αντίστοιχη χειρωνακτική παραγωγή περιπτώσεων ελέγχου. Το παρόν εργαλείο, έχει σχεδιαστεί για να διευκολύνει την παραγωγή περιπτώσεων ελέγχου με την αυτοματοποίησή της. Το μόνο που απαιτείται από τον χρήστη είναι να επιλέξει το αρχείο κώδικα που επιθυμεί να ελέγξει, και το είδος κάλυψης στο οποίο στοχεύει (ανάλογα με το είδος γράφου που δημιουργείται). Αυτό βέβαια δε σημαίνει ότι δεν μπορεί να δει τις προεπιλεγμένες παραμέτρους και να τις τροποποιήσει. Η παραγωγή περιπτώσεων ελέγχου γίνεται με τη βοήθεια μιας ευφυούς ευρετικής τεχνικής, τους γενετικούς αλγορίθμους, όπως θα δούμε σε βάθος στο τέταρτο κεφάλαιο. Η διαδικασία αυτή εμπεριέχει δυναμική εκτέλεση του προγράμματος με τις περιπτώσεις ελέγχου που παράγονται, αξιολόγησή τους και καταμέτρηση των ακμών/μονοπατιών που καλύπτονται κατά την εκτέλεση.

1.6 Περιγραφή εργαλείου

Το παρόν εργαλείο, BPAS & ATCGS , σχεδιάστηκε για να καλύψει τις ανάγκες αυτές που προκύπτουν από την πολυδάπανη και χρονοβόρα διαδικασία του ελέγχου άσπρου κουτιού. Είναι ένα ολοκληρωμένο εργαλείο ανάλυσης κώδικα, παραγωγής γράφων και αυτοματοποιημένης παραγωγής περιπτώσεων ελέγχου με ευφυείς τεχνικές που αποδεδειγμένα προσεγγίζουν την βέλτιστη λύση ταχύτατα, σε σχέση με παραδοσιακές τεχνικές ελέγχου. Μπορεί να εγγυηθεί στον χρήστη υψηλά ποσοστά κάλυψης του πηγαίου κώδικα που επιλέγει με γνωστά κριτήρια κάλυψης όπως edge/condition, Ntafos coverage [4,5] , κ.ο.κ. Οι παραπάνω επιγραμματικά αναφερόμενες δυνατότητες του εργαλείου πλαισιώνονται σε ένα φιλικό προς τον χρήστη γραφικό περιβάλλον, με εύκολο στην πλοήγηση μενού επιλογών, online εγχειρίδιο χρήσης και πολλές ενημερωτικές οθόνες για την κατάσταση της επιλεγμένης λειτουργίας. Όλα αυτά και νέες πρόσθετες λειτουργίες είναι διαθέσιμα σε κάθε χρήστη αφού το μόνο που χρειάζεται για χρήση του εργαλείου είναι ένας φυλλομετρητής και εγκατεστημένη την JAVA (έκδοση 5 ή νεότερη).

1.7 Δομή Διπλωματικής Εργασίας

Το πρώτο κεφάλαιο στοχεύει να εισαγάγει τον αναγνώστη στο στόχους της διπλωματικής αυτής εργασίας, στην ανάγκη που ώθησε στη σχεδίαση και υλοποίηση του εργαλείου αυτού και να παρουσιάσει επιγραμματικά μερικές από τις βασικές λειτουργίες του. Παράλληλα, γίνεται οριοθέτηση του ρόλου του παρόντος εργαλείου στον χώρο ελέγχου λογισμικού με βάση τις μεθόδους και τεχνικές ανάλυσης και ελέγχου που χρησιμοποιούνται. Στο δεύτερο κεφάλαιο, γίνεται αναφορά στον έλεγχο λογισμικού, ως αναπόσπαστο κομμάτι του κύκλου ζωής του λογισμικού. Αναφέρονται περιγραμματικά οι κύριες μέθοδοι ελέγχου λογισμικού που χρησιμοποιούνται σήμερα, και επιμέρους διαχωρισμοί αναφορικά με τον τρόπο που διεξάγεται ο έλεγχος. Γίνεται αναφορά στα είδη γράφων που δημιουργούνται από το εργαλείο και εισάγει τον αναγνώστη στα είδη κάλυψης που επιτυγχάνει το εργαλείο. Στο τρίτο κεφάλαιο επεξηγείται η παραγωγή περιπτώσεων

ελέγχου, γνωστά είδη της, η αυτοματοποίηση της διαδικασίας που προσφέρει το εργαλείο και ο τρόπος με τον αυτή πραγματοποιείται. Στο επόμενο, τέταρτο, κεφάλαιο γίνεται μια διεξοδική ανάλυση του εργαλείου, των λειτουργικών και μη-λειτουργικών απαιτήσεών του, της λειτουργικότητάς και της εξέλιξής του. Αναλύονται οι νέες λειτουργίες που παρέχονται, η μεθοδολογία εργασίας που ακολουθήθηκε, αναφορές και συγκρίσεις με άλλα παρόμοια εργαλεία, και τέλος γίνεται μια ξενάγηση στο εργαλείο με σενάρια χρήσης του. Το πέμπτο κεφάλαιο συνοψίζει τα συμπεράσματα που προκύπτουν από την εκπόνηση της διπλωματικής εργασίας και γίνονται αναφορές στην μελλοντική δουλειά που μπορεί να πραγματοποιηθεί. Τέλος, το Παράρτημα Α περιέχει το Εγχειρίδιο χρήσης το οποίο είναι γραμμένο στην αγγλική γλώσσα και είναι ενσωματωμένο ακριβώς όπως αυτό είναι διαθέσιμο online μέσω του εργαλείου. Όπως αντιλαμβανόμαστε, λόγω του ότι το Εγχειρίδιο Χρήσης (Manual) αποτελεί τμήμα της εφαρμογής, η γλώσσα συγγραφής του κρίθηκε σκόπιμο να είναι η αγγλική για λόγους διεθνοποίησης του εργαλείου και διευκόλυνσης μη-ελληνόφωνων χρηστών. Όσον αφορά το παρόν κείμενο, πλην του Παραρτήματος Α, η γλώσσα συγγραφής είναι η ελληνική. Ξένη ορολογία σχετική με την Επιστήμη της Πληροφορικής μεταφράζεται στα ελληνικά, ωστόσο σε ορισμένες περιπτώσεις διατηρείται μόνο η αγγλική ορολογία αφού η μετάφραση στα ελληνικά κρίνεται άσκοπη και θα μπορούσε να αλλοιώσει την ερμηνεία.

Κεφάλαιο 2

Έλεγχος Λογισμικού

2.1 Σημασία – Οφέλη Ελέγχου Λογισμικού

Έλεγχος λογισμικού είναι η διαδικασία που στοχεύει στην εύρεση σφαλμάτων με την εκτέλεση ενός προγράμματος [11] ή, εναλλακτικά, κάποια σειρά ενεργειών που στοχεύει στην αξιολόγηση ενός χαρακτηριστικού η μιας δυνατότητας ενός προγράμματος ή συστήματος και προσπαθώντας να καθορίσουμε κατά πόσο ικανοποιεί τις απαιτήσεις του και αποφέρει τα αναμενόμενα αποτελέσματα. Αυτό που επιδιώκουμε με τον έλεγχο λογισμικού είναι να εντοπίσουμε και να μπορέσουμε κατ' επέκταση να προσδιορίσουμε και να διορθώσουμε τους τρόπους που μπορεί να αποτύχει το λογισμικό, να εμφανίσει δηλαδή ανεπιθύμητη συμπεριφορά λόγω σφαλμάτων. Σε αντίθεση με άλλα συστήματα, όπως για παράδειγμα ηλεκτρολογικά, δεν μπορούμε να προσδιορίσουμε με ακρίβεια τα αίτια της αποτυχίας ενώ σε μεγάλου μεγέθους κώδικα, όπως σε μια μεγάλης κλίμακας εφαρμογή. Το να εντοπίσουμε όλα τα πιθανά είδη αποτυχίας λογισμικού φαίνεται να είναι ανέφικτο αφού δεν είναι προβλήματα κατασκευαστικά που μπορεί να εμφανιστούν λόγω φυσικών φαινόμενων, όπως ένα ηλεκτρολογικό κύκλωμα ή ένα κτίριο, αλλά είναι λογικά λάθη σχεδιασμού τα οποία αν γίνουν και δεν εντοπισθούν και διορθωθούν θα παραμείνουν για πάντα ενσωματωμένα στην εφαρμογή μέχρι να γίνει συντήρηση της εφαρμογής ή να αποσυρθεί. Μέχρι τότε, η εφαρμογή θα περιέχει λάθη τα οποία θα αγνοούν οι σχεδιαστές και προγραμματιστές της όπως επίσης και οι χρήστες της, τα οποία μπορούν κάτω από κάποια σενάρια χρήσης να εμφανιστούν και να έχουνε ανεπιθύμητα ή και καταστροφικά αποτελέσματα.

Από τις περασμένες δυο δεκαετίες, υπήρχε ένας άτυπος κανόνας που θέλει τους μηχανικούς λογισμικού σε κάθε μεγάλης κλίμακας σύστημα που κατασκευάζουν, να ξοδεύουν περίπου το 50% του χρόνου τους στον έλεγχο του λογισμικού και περισσότερο από το 50% του κόστους του λογισμικού να δαπανάται για την διαδικασία του ελέγχου επίσης. Σήμερα, παρά τη βοήθεια που παρέχεται για τυπικό έλεγχο από σύγχρονα εργαλεία, και παρά τις έρευνες και τις τεχνικές ελέγχου που έχουν αναπτυχθεί, ο έλεγχος λογισμικού παραμένει μια από τις λιγότερο αναπτυγμένες διαδικασίες όσον αφορά τις φάσεις ανάπτυξης λογισμικού. Επιπροσθέτως, στην εποχή μας, με την ραγδαία εμφάνιση και εξέλιξη πολλών γλωσσών προγραμματισμού, λειτουργικών συστημάτων και την ανάπτυξη πλατφορμών υλικού ο έλεγχος λογισμικού γίνεται ακόμη πιο δύσκολος αλλά και

περισσότερο αναγκαίος. Αναγκαίος γιατί πέραν του γεγονότος ότι ο έλεγχος είναι μια χρονοβόρα διαδικασία, και πέραν των οικονομικών παραγόντων (αύξηση κόστους στα projects, κ.λπ.) τείνει να λάβει και μια άλλη, πιο ανθρωποκεντρική διάσταση: Είναι αναμφίβολο ότι στις μέρες μας οι πλειονότητα των ανθρώπων χρησιμοποιεί τους ηλεκτρονικούς υπολογιστές για επαγγελματικούς λόγους, σε διάφορα λειτουργικά συστήματα, κάνοντας χρήση εκατοντάδων επαγγελματικών εφαρμογών από τις οποίες αναμένουν αποδοτικότητα και αποτελεσματικότητα και όχι απρόβλεπτες συμπεριφορές από πλευράς εφαρμογής που μπορεί να οδηγήσουν σε σφάλματα, απώλεια δεδομένων και εκνευρισμό. Έχοντας αυτά υπόψη μας, αντιλαμβανόμαστε ότι το λογισμικό που γράφεται θα επηρεάσει δυνητικά εκατομμύρια ανθρώπους και δεν υπάρχει χώρος για λογισμικό που είναι ασταθές, μη-αποδοτικό και απρόβλεπτο [11, 16].

Λάθη στον κώδικα, bugs, θα υπάρχουνε πάντα σε λογισμικό, και αυτό δε συμβαίνει λόγω αμέλειας, ανευθυνότητας ή ανικανότητας των προγραμματιστών αλλά γιατί η πολυπλοκότητα του λογισμικού μπορεί να είναι απεριόριστη και οι ανθρώπινες δυνατότητες, όντας περιορισμένες, δεν μπορούν να την αντιμετωπίσουν. Η ανίχνευση λαθών στον κώδικα είναι μια αναλόγως επίπονη διαδικασία λόγω αυτής της πολυπλοκότητας. Λόγω της έλλειψης αλληλουχίας μεταξύ του υλικού και του λογισμικού, ο έλεγχος των ακραίων τιμών (boundary values) δεν επαρκεί για να εγγυηθεί ορθότητα κώδικα. Όλες οι πιθανές τιμές θα πρέπει να ελεγχθούν αλλά κάτι τέτοιο φαίνεται ανέφικτο. Για παράδειγμα, για ένα απλό πρόγραμμα που το μόνο που κάνει είναι να προσθέτει δυο ακέραιους αριθμούς 32 bits, θα χρειαζόμασταν 2^64 διαφορετικές περιπτώσεις ελέγχου (test cases). Ωστόσο για να εξετάσουμε 18446744073709551616 περιπτώσεις θα χρειαζόμασταν εκατοντάδες χρόνια, ακόμη και αν είχαμε τη δυνατότητα να πραγματοποιούμε χιλιάδες ελέγχους το δευτερόλεπτο, και αυτό για να μπορούμε να πούμε με βεβαιότητα ότι δεν υπάρχει κάποιο σφάλμα και ότι τίποτα δεν θα πάει στραβά κατά την πρόσθεση δυο οποιονδήποτε ακεραίων αριθμών, ανεξάρτητα με το υλικό και τις τιμές που θα πάρουν οι αριθμοί αυτοί. Σε ρεαλιστικά σενάρια γίνεται σαφές ότι η πολυπλοκότητα είναι κατά πολύ πολλαπλάσια από αυτήν. Παράλληλα, η πολυπλοκότητα αυξάνεται κατακόρυφα αν ληφθούν υπόψη παράμετροι που μπορούν να δοθούν από δεδομένα

πραγματικού κόσμου και από ποικιλότροπες και συχνά απρόβλεπτες αλληλεπιδράσεις του χρήστη με την εφαρμογή και το περιβάλλον της και ο χρόνος που αυτά θα συμβούν.

Πιο σημαντικά, μια μεγάλη ιδιαιτερότητα του λογισμικού είναι η δυναμική του φύση του και η ικανότητα να αλλάζει και να εξελίσσεται κατά τη διάρκεια χρήσης του. Ο τρόπος με τον οποίο εξελίσσεται και εκτελείται είναι σε μεγάλο βαθμό απρόβλεπτος αφού βασίζεται συνήθως σε εισόδους που λαμβάνει από τον χρήστη σε συνδυασμό με τρέχουσες συνθήκες του μηχανήματος στο οποίο τρέχει και άλλες συνθήκες που αφορούν το περιβάλλουν της εφαρμογής. Ακόμη, σε περίπτωση που βρεθεί κάποιο σφάλμα σε προκαταρκτικό στάδιο ελέγχου με χρήση μιας σειράς περιπτώσεων ελέγχου, και στη συνέχεια διορθωθεί, τότε δεν μπορούμε πλέον, μετά την αλλαγή που επιφέρουμε να εγγυηθούμε ότι όλες οι περιπτώσεις ελέγχου (test cases) που είχαν δοκιμασθεί μέχρι να βρεθεί το σφάλμα θα ισχύουν.

2.2 Είδη ελέγχου

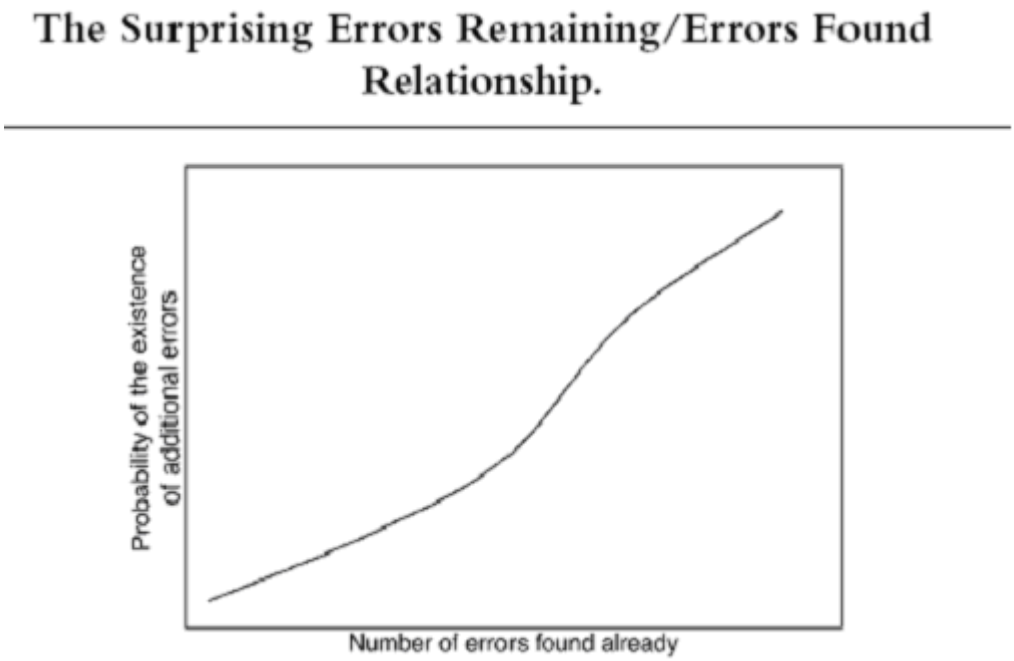
Ο έλεγχος λογισμικού μπορεί να πραγματοποιηθεί είτε 'χειρωνακτικά' (manually) από προγραμματιστές που θα διαβάσουν των κώδικα, θα ελέγξουν αποτελέσματα, θα προσπαθήσουν να εντοπίσουν απρόβλεπτες συμπεριφορές ή λάθος αποτελέσματα ακολουθώντας μια σειρά διαδικασιών και αρχών είτε θα γίνει αυτοματοποιημένα, με τη χρήση κάποιου εργαλείου το οποίο θα λάβει ως είσοδο τον υπο-έλεγχο κώδικα και με βάση κάποια κριτήρια θα ελέγξει αν αυτός πληροί τις προδιαγραφές του και ενδεχομένως με χρήση περιπτώσεων ελέγχου (test-cases) να καλύψει κάποια κριτήρια που εγγυώνται ικανοποιητικό έλεγχο (white-box ή black-box).

2.2.1 Χειρωνακτικός έλεγχος

Όσον αφορά το χειρωνακτικό έλεγχο [11] (που δεν θα εξετάσουμε σε λεπτομέρεια), μπορούμε να πούμε επιλεκτικά μερικές αρχές που μπορούν να ακολουθηθούν κατά τον έλεγχο (guidelines) :

Αρχές

- *Ένα αναγκαίο τμήμα μιας περίπτωσης ελέγχου είναι ο ορισμός του αναμενόμενου αποτελέσματος της εξόδου.*
- *Ο προγραμματιστής θα πρέπει να αποφεύγει να επιχειρεί τον έλεγχο των δικών του προγραμμάτων.*
- *Ένας οργανισμός/επιχείρηση θα ήταν καλό να μην επιχειρεί να ελέγχει τα δικά του/της προγράμματα.*
- *Επιβάλλεται διεξοδικός έλεγχος κάθε τεστ.*
- *Περιπτώσεις ελέγχου θα πρέπει να γράφονται τόσο για έγκυρες τιμές εισόδου όσο και για άκυρες και απρόβλεπτες τιμές.*
- *Η εξέταση του προγράμματος για να διαπιστωθεί αν δεν κάνει αυτό που έχει υλοποιηθεί να κάνει είναι η μισή μάχη. Η άλλη μισή είναι να εξεταστεί αν κάνει αυτά που δεν θα έπρεπε να κάνει.*
- *Αποφυγή περιπτώσεων ελέγχου μιας χρήσης, εκτός και αν η εφαρμογή είναι μιας χρήσης.*
- *Να μη γίνεται σχεδιασμός μιας προσπάθειας ελέγχου με το σκεπτικό ότι δεν θα βρεθούν λάθη.*
- *Η πιθανότητα ύπαρξης περισσότερων λαθών σε μια περιοχή του κώδικα είναι ανάλογη με τα λάθη τα οποία έχουν ήδη βρεθεί στην περιοχή αυτή.* [Πίνακας 2.1]
- *Ο έλεγχος είναι ένα εξαιρετικά δημιουργικό έργο και μια διανοητικά απαιτητική λειτουργία.*



Πίνακας 2.1 [11]

Ο χειρωνακτικός έλεγχος, ή αλλιώς 'ανθρώπινος έλεγχος', είναι η διαδικασία ελέγχου που δεν βασίζεται σε έλεγχο με την χρήση υπολογιστή (non computer-based testing) και επιτυγχάνεται με τρεις κυρίως τρόπους :

- Program inspections
- Walkthroughs
- Reviews

Τα program inspections (ή code inspection) και walkthroughs είναι αρκετά παρόμοιες διαδικασίες.

Κατά τη διάρκεια του program inspection μια ομάδα ανθρώπων διαβάζει, ή ελέγχει οπτικά τον κώδικα. Στην ομάδα αυτή ηγείται ένας συντονιστής ο οποίος δεν είναι ο συγγραφέας του κώδικα υπό έλεγχο και είναι υπεύθυνος για τον συντονισμό, οργάνωση και καταγραφή των λαθών κατά τη διάρκεια ενός inspection session. Τόσο τα program inspections όσο και

τα walkthroughs απαιτούν μια προκαταρκτική δουλειά. Στόχος είναι η ανίχνευση των πιθανών λαθών (bugs) αλλά όχι η διόρθωσή τους.

Walkthrough είναι η διαδικασία κατά την οποία μια ομάδα, συνήθως 3-4 προγραμματιστών από τους οποίους μόνο ένας από αυτούς είναι ο συγγραφέας του λογισμικού, εκπληρώνοντας έτσι μια από τις αρχές (guidelines) που προαναφέρθηκαν, δηλαδή ότι αυτός που έχει γράψει τον κώδικα τείνει να είναι λιγότερο αποτελεσματικός στην εύρεση λαθών στον κώδικά του.
Τα walkthroughs είναι αποτελεσματικά γιατί συμμετέχουν και άλλοι προγραμματιστές εκτός του συγγραφέα του λογισμικού κι έτσι ανακαλύπτονται περισσότερα λάθη. Ένα ακόμη πλεονέκτημα είναι ότι η μέθοδος αυτή αποκαλύπτει αρκετά λάθη σε συγκεκριμένα σημεία του κώδικα και επιτρέπει σε μετέπειτα στάδιο την μαζική τους διόρθωση, μειώνοντας σημαντικά το κόστος της αποσφαλμάτωσης.

Ωστόσο οι παραπάνω μέθοδοι συνήθως συνδράμουν στην εύρεση του 30%-70% των λαθών στον σχεδιασμό και τη λογική του κώδικα αλλά δεν είναι τόσο αποτελεσματικές όσο οι μέθοδοι βασισμένες στην χρήση εργαλείων ελέγχου λογισμικού με υπολογιστή (computer-based) οι οποίες μπορούν να ανακαλύψουν, όπως θα δούμε αναλυτικότερα στη συνέχεια, περισσότερα λάθη υψηλότερου επιπέδου σχεδιασμού.

2.2.2. Αυτοματοποιημένος έλεγχος

Με τον όρο αυτό, εννοούμε τον έλεγχο ο οποίος λαμβάνει χώρα υποβοηθούμενος από κάποια εφαρμογή η οποία τρέχει σε ένα υπολογιστικό σύστημα (computer-based testing/ computer-aided testing). Υπάρχουν πολλές ανεξάρτητες εφαρμογές ή εργαλεία ενσωματωμένα με προγράμματα σύνταξης και μεταγλώττισης κώδικα τα οποία πραγματοποιούν έλεγχο με βάση τις τεχνικές back-box ή white-box. Παράλληλα, υπάρχουν εργαλεία που προσφέρουν άλλων ειδών ελέγχου, όπως έλεγχο οπισθοδρόμησης (regression testing), έλεγχο για διαρροή μνήμης (memory leak), αναλυτές απόδοσης (performance

analysis), συμβολική αποσφαλμάτωση (symbolic debugging), παραγωγή benchmarks, κ.ο.κ. Ο χειρωνακτικός έλεγχος είναι μια χρονοβόρα και επαναληπτική διαδικασία, καθόλου ελκυστική στους ελεγκτές λογισμικού. Ο αυτοματοποιημένος έλεγχος ωστόσο μπορεί μεταξύ άλλων να εντοπίσει μαζικά λάθη ή αδυναμίες του κώδικα, να καθορίσει αν το λογισμικό υστερεί να καλύψει κάποιες από τις προδιαγραφές του, να εγγυηθεί την αποδοτικότητα και αποτελεσματικότητα, να παράξει μαζικά περιπτώσεις ελέγχου και να παρέχει διάφορα είδη κάλυψης του πηγαίου κώδικα.

2.3 Έλεγχος μαύρου κουτιού (Black-box)

Αυτή η τεχνική ελέγχου λογισμικού, βλέπει την εφαρμογή που εξετάζουμε από μια εξωτερική οπτική γωνία. Οι ελεγκτές που χρησιμοποιούν αυτήν την μέθοδο απλά γράφουν περιπτώσεις ελέγχου και τις παρέχουν ως είσοδο στο πρόγραμμα ή τμήμα εφαρμογής που υφίσταται τον έλεγχο [9, 10, 11]. Στη συνέχεια, βλέπουν αν τα αποτελέσματα-έξοδοι (outputs) της εφαρμογής είναι αυτά που αναμένονταν, σύμφωνα με τις προδιαγραφές του λογισμικού. Συνεπώς, χωρίς γνώση της εσωτερικής δομής του κώδικα, δίνονται έγκυρες και άκυρες είσοδοι και ελέγχονται οι έξοδοι. Αυτή η μέθοδος ελέγχου μπορεί να πραγματοποιηθεί σε όλα τα επίπεδα : μονάδων (unit), συνένωσης (integration), συστήματος (system)και αποδοχής (acceptance).

2.4 Έλεγχος άσπρου κουτιού (White-box)

Ο έλεγχος άσπρου κουτιού (white-box testing) ή αλλιώς δομικός έλεγχος (structural testing) είναι η μέθοδος ελέγχου του κώδικα, με χρήση της γνώσης των εσωτερικών δομών δεδομένων και αλγορίθμων για τη δημιουργία περιπτώσεων ελέγχου που βασίζονται σε αυτήν την εσωτερική δομή. Όταν ο έλεγχος αυτός γίνεται χειρωνακτικά, από ομάδες προγραμματιστών, είναι σαφές ότι απαιτεί εξαιρετικές προγραμματιστικές ικανότητες για την εύρεση όλων των πιθανών μονοπατιών μέσα στον κώδικα. Ο προγραμματιστής ή ομάδα προγραμματιστών που επιτελούν τον έλεγχο αυτό θα πρέπει να παράξουν

περιπτώσεις ελέγχου που θα δοθούν ως είσοδοι σε σημεία του προγράμματος και αφού διαπεράσουν κάποια μονοπάτια, θα καθορίσουν τις εξόδους του προγράμματος.
Πιο γνωστές τεχνικές ελέγχου white-box είναι ο έλεγχος ροής ελέγχου (control flow testing) και ο έλεγχος ροής δεδομένων (data flow testing) [10, 11].

2.4.1 Σχεδιασμός περιπτώσεων-ελέγχου (test-cases)

Ένα από τα σημαντικότερα θέματα για τον έλεγχο ενός προγράμματος είναι ο καλός σχεδιασμός test-cases. Ο ολοκληρωμένος έλεγχος (διεξοδικός έλεγχος για κάθε περίπτωση εκτέλεσης ενός προγράμματος) είναι εκ των πραγμάτων αδύνατος. Αυτό όμως που μπορεί να επιτευχθεί και αποτελεί μια γενική στρατηγική είναι η καλύτερη δυνατή επιλογή test-cases. Η στρατηγική αυτή μπορεί να περιοριστεί στα πλαίσια μιας απλής ερώτησης : "Ποιό υποσύνολο έχει περισσότερες πιθανότητες να ανιχνεύσει περισσότερα λάθη ; "
Μια απλή μέθοδος είναι η τυχαία επιλογή test-cases, δίνοντας δηλαδή τυχαίες τιμές από ένα πεδίο πιθανών τιμών. Αυτή η μέθοδος συνήθως αποκλίνει κατά πολύ από την εύρεση του βέλτιστου υποσυνόλου test-cases.

Μια άλλη μέθοδος είναι η επιλογή test-cases με μεθόδους περισσότερο black-box-oriented και στη συνέχεια η εξέταση της λογικής του προγράμματος με white-box-oriented μεθόδους.
Μπορούμε να δούμε στη συνέχεια μερικές από αυτές τις μεθόδους

Black Box	***White Box***
Equivalence partitioning	Statement coverage
Boundary-value analysis	Decision coverage
Cause-effect graphing	Condition coverage
Error guessing	Decision-condition coverage
	Multiple-condition coverage

2.4.2 Κάλυψη με βάση CFG

Θα πραγματοποιηθεί εδώ μια εισαγωγική επεξήγηση ορισμών για είδη κάλυψης και σύντομη σύγκρισή τους. Η συνοπτική αυτή επεξήγηση γίνεται με έμφαση σε ορισμούς ειδών κάλυψης που γίνονται με βάση τον CFG (Control Flow Graph) και εμφανίζονται στο παρόν κείμενο καθώς και την υλοποίησή τους στο εργαλείο.

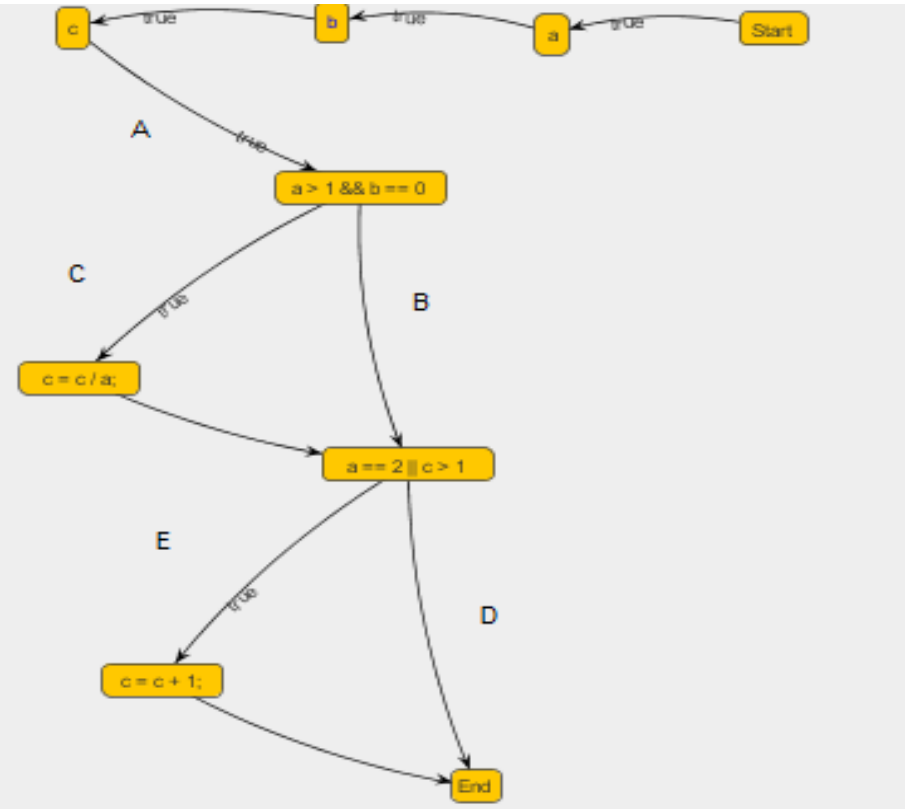
2.4.2.1 Κάλυψη τύπου statement (statement coverage)

Έχει διαπιστωθεί ότι η υπέρτατη τεχνική ελέγχου white-box είναι η εκτέλεση κάθε μονοπατιού του κώδικα αλλά κάτι τέτοιο (ολοκληρωμένη κάλυψη όλων των μονοπατιών) δεν είναι εφικτή σε προγράμματα με δομές επανάληψης (loops). Θα μπορούσε ωστόσο κάποιος να ισχυριστεί ότι απλά η κάλυψη κάθε statements του κώδικα θα ήταν ένα ικανοποιητικό κριτήριο. Κάτι τέτοιο όμως δεν ισχύει.

Ας εξετάσουμε για παράδειγμα το παρακάτω κομμάτι κώδικα (παρμένο από τον editor του εργαλείου), γραμμένο σε java όπως φαίνεται στον πίνακα παρακάτω (Πίνακας 2.2)

```
public void methodA(int a, int b, int c) {
  if (a > 1 && b == 0) {
    c = c / a;
  }
  if (a == 2 || c > 1) {
    c = c + 1;
  }
}
```

Πίνακας 2.2 : Παράδειγμα μιας μεθόδου κλάσης σε java



Σχήμα 2.1 : Παραγόμενος γράφος ροής ελέγχου του Πίνακα 2.1

Θεωρούμε τα παραπάνω υπο-μονοπάτια (Σχήμα 2.1) A,B,C,D,E, όπως έχουνε ονομαστεί στις αντίστοιχες ακμές, για διευκόλυνση τις επεξήγησης.

Αν θεωρήσουμε το παρόν κριτήριο (κάλυψη τουλάχιστον μια φορά κάθε statement) ικανοποιητικό, τότε θα δούμε ότι μπορούμε πολύ εύκολα να κατασκευάσουμε ένα test-case, συγκεκριμένα α=2, b=0 και c=3, το οποίο πράγματι καλύπτει κάθε statement του κώδικα, δηλαδή κάθε κόμβο του παραπάνω control-flow γράφου. Θα έχουμε κάλυψη του υπο-μονοπατιού A,C,E.

Αν όμως είχαμε στον πρώτο έλεγχο a>1 || b==0 αντί για a>1 && b==0 τότε το λάθος θα μας ξέφευγε, αφού δεν θα εκτελούνταν όπως είναι λογικό το statement b==0 καθώς a>1 == TRUE και άρα σε έναν έλεγχο TRUE || VALUE, δεν θα υπολογιστεί/εκτελεστεί το statement VALUE, που στην περίπτωσή μας είναι το b==0.

Παρόμοια, αν στον δεύτερο έλεγχο a==2 || c>1 είχαμε αντ'αυτού a==2 || c>0, θα αποτυγχάναμε να ανιχνεύσουμε ακόμη ένα λάθος.

Τέλος, αν αυτό που θέλαμε να πετύχουμε με αυτό το κομμάτι κώδικα ήταν να είναι η αλλαγή της τιμής της μεταβλητής c, τότε θα μας ξέφευγε ακόμη ένα λάθος αφού υπάρχει ένα μονοπάτι, το A,B,D το οποίο αφήνει το c αμετάβλητο.
Μπορούμε λοιπόν να συμπεράνουμε ότι κριτήριο αυτό, κάλυψη κάθε statement τουλάχιστον μια φορά, είναι γενικά άχρηστο, αφού μπορεί να αφήσει πολλά λάθη να περάσουν απαρατήρητα.

2.4.2.2 Κάλυψη τύπου decision (decision/branch coverage)

Ένα πιο δυνατό κριτήριο λογικής κάλυψης (logic coverage), είναι το decision-coverage (κάλυψη αποφάσεων), ή αλλιώς branch-coverage (κάλυψη κλάδων), γνωστό και ως all-edges coverage [22]. Σύμφωνα με αυτό το κριτήριο, κάθε κλαδί ενός ελέγχου, θα πρέπει ελεγχθεί τουλάχιστον μια φορά. Με άλλα λόγια, για κάθε δομή ελέγχου που η ροή ελέγχου θα ακολουθήσει δυο πιθανά μονοπάτια (true, αν ισχύει ο έλεγχος ή false αν δεν ισχύει), θα πρέπει να δημιουργηθούν αρκετά test-cases τα οποία θα μας εγγυώνται ότι κάθε κλαδί του ελέγχου έχει εκτελεστεί τουλάχιστον μια φορά.
Συνήθως, το branch-coverage μπορεί να μας εγγυηθεί και statement-coverage (αφού κάθε statement ανήκει σε κάποιο υπο-μονοπάτι που πηγάζει είτε από κάποιο κλάδο ελέγχου είτε από σημείο εισόδου του προγράμματος (entry point),και άρα κάθε statement θα πρέπει να εκτελεστεί αν έχουμε κάθε κλαδί εκτελεσμένο). Αυτό δεν ισχύει για τις τρεις παρακάτω εξαιρέσεις :

- προγράμματα που δεν περιέχουν αποφάσεις (χωρίς δομές ελέγχου)
- προγράμματα ή υπο-ρουτίνες (μέθοδοι) με πολλαπλά σημεία εισόδου (multiple entry points).
- statements μέσα σε ON-units. Διασχίζοντας κάθε κατεύθυνση των κλάδων ελέγχου δεν σημαίνει αναγκαστικά ότι θα εκτελεστούν όλα τα ON-units.

Το decision-coverage απαιτεί κάθε απόφαση (πχ if-statement και while-statement) να έχει ένα true ή false αποτέλεσμα και κάθε κλάδος αυτής της απόφασης να εκτελεστεί

τουλάχιστον μια φορά. Για γλώσσες που επιτρέπουν δομές ελέγχου με πολλαπλούς κλάδους ως αποτέλεσμα του ελέγχου, όπως για παράδειγμα select (case) statements , θα πρέπει να τροποποιηθεί κατάλληλα ώστε να εγγυάται πάλι ότι θα εκτελεστούν όλοι οι κλάδοι που απορρέουν από τον έλεγχο.
Το πρόβλημα με αυτό το κριτήριο είναι ότι αγνοεί κλάδους μέσα σε boolean εκφράσεις που προκύπτουν από βραχύκυκλους τελεστές (short-circuit operators). Σε γλώσσες όπως η java, C/C++,κλπ που το επιτρέπουν. Όπως στο παρακάτω στον Πίνακα 2.3.

```
if (a_condition && (another_condition || function() ) )
  statement;
else
  statement_else;
```

Πίνακας 2.3 : Παράδειγμα βραχυκυκλωμένων τελεστών (short-circuit operators)

2.4.2.3 Κάλυψη τύπου condition (condition coverage)

Ένα άλλο κριτήριο που δεν πρέπει να παραληφθεί είναι το *condition coverage*
Το *condition coverage* είναι γενικά καλύτερο κριτήριο από το decision-coverage αφού σύμφωνα με το κριτήριο αυτό πρέπει να παράξουμε αρκετά test-cases τα οποία θα μας εγγυώνται ότι κάθε υπόθεση (condition) σε μια απόφαση (decision) καλύπτει όλα τα πιθανά αποτελέσματα που απορρέουν από αυτή τουλάχιστον μια φορά. Επιπλέον, κάθε σημείο εισόδου στο πρόγραμμα (entry point) η σε υπο-ρουτίνα, θα πρέπει να κληθεί τουλάχιστον μια φορά.

2.4.2.4 Επισκόπηση

Εν κατακλείδι, μπορούμε να ισχυριστούμε ότι η κάλυψη τύπου statement, φαινομενικά είναι ικανοποιητική. Κάτι τέτοιο όμως δεν ισχύει αφού ισχυρότερα κριτήρια όπως η

κάλυψη τύπου decision είναι καλύτερη, αφού μπορεί ,εκτός περιπτώσεων να συμπεριλάβει την κάλυψη τύπου statement. Τέλος, η κάλυψη τύπου condition είναι καλύτερο κριτήριο από τα προηγούμενα δυο κριτήρια. Αργότερα, άλλες τεχνικές και κριτήρια κάλυψης που υπερτερούν αυτών επινοήθηκαν τα οποία , όπως θα δούμε βασίζονται στον γράφο που αποκαλούμε γράφο ροής δεδομένων.

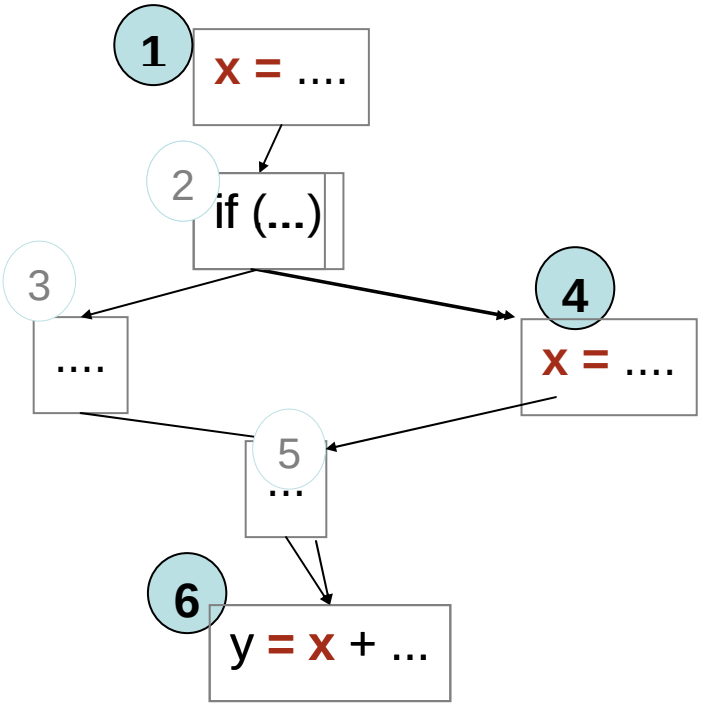
2.4.3 Κάλυψη με βάση DFG

Θα ακολουθήσει στο σημείο αυτό μια συνοπτική επεξήγηση ορισμών που χρησιμοποιούνται από το εργαλείο και για τη διευκόλυνση του αναγνώστη στην κατανόηση των τύπων κάλυψης που προκύπτουν με βάση τον DFG. Περισσότερη έμφαση δίνεται στα [6, 7, 8].
Μερικοί χρήσιμοι ορισμοί βασικών εννοιών ακολουθούν παρακάτω :

- dn(x) : Στη μεταβλητή x ανατίθεται τιμή στον κόμβο/δήλωση n.

- um(y) : Η μεταβλητή y χρησιμοποιείται στον κόμβο/δήλωση m.

- Μονοπάτι ‘definition clear’ p ως προς τη μεταβλητή x είναι ένα υπο-μονοπάτι στο οποίο η μεταβλητή x δεν ορίζεται (defined) σε κανέναν από τους κόμβους/δηλώσεις στο p.



Σχήμα 2.2 : Παράδειγμα Def-clear μονοπατιού σε CFG

Όπως μπορούμε να τα δούμε στο παραπάνω σχήμα, το μονοπάτι 1,2,3,5,6, είναι ένα definition clear (def-clear) μονοπάτι για τη μεταβλητή x αφού δεν γίνεται ξανά καμιά ανάθεση τιμής στο x, σε αντίθεση με το μονοπάτι 1,2,4,5,6 το οποίο δεν είναι def-clear αφού στον κόμβο 4 η μεταβλητή x παύει να είναι ζωντανή διότι στον κόμβο/δήλωση αυτό ανατίθεται μια νέα τιμή.

 Ένας ορισμός (definition) dm(x) φτάνει σε μια χρήση (use) un(x) αν και μόνο αν υπάρχει ένα υπο-μονοπάτι

(m) • p • (n)

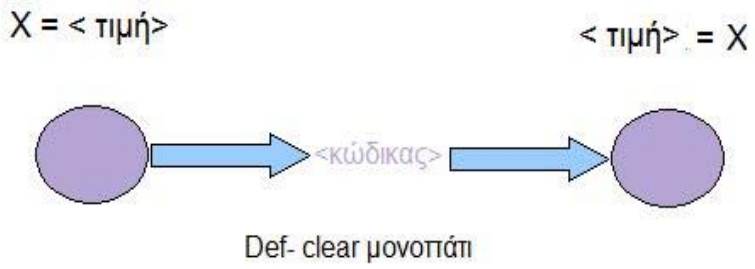
Τέτοιο ώστε το μονοπάτι p να είναι ένα definition clear μονοπάτι ως προς το x.

Στη συνέχεια θα δούμε τα κριτήρια κάλυψης Rapps and Weyuker's για γράφο ροής δεδομένων :

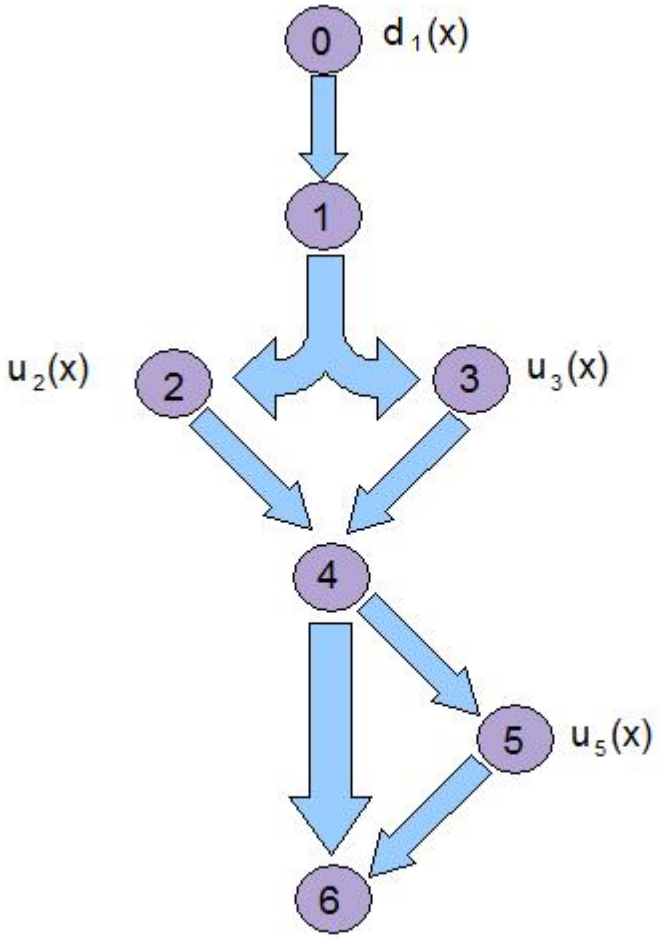
- All-Defs
- All-Uses
- All-C-Uses, Some-P-Uses
- All-P-Uses, Some-C-Uses
- All-P-Uses
- All-Du-Paths

2.4.3.1 All-defs κριτήριο

All-defs μονοπάτι είναι ένα definition-clear υπο-μονοπάτι (subpath) από τη δήλωση (declaration) μιας μεταβλητής μέχρι τη χρησιμοποίησή της (Σχήμα 2.3).



Σχήμα 2.3 : Παράδειγμα Def-clear μονοπατιού



Σχήμα 2.4 : Παράδειγμα All-defs κριτηρίου

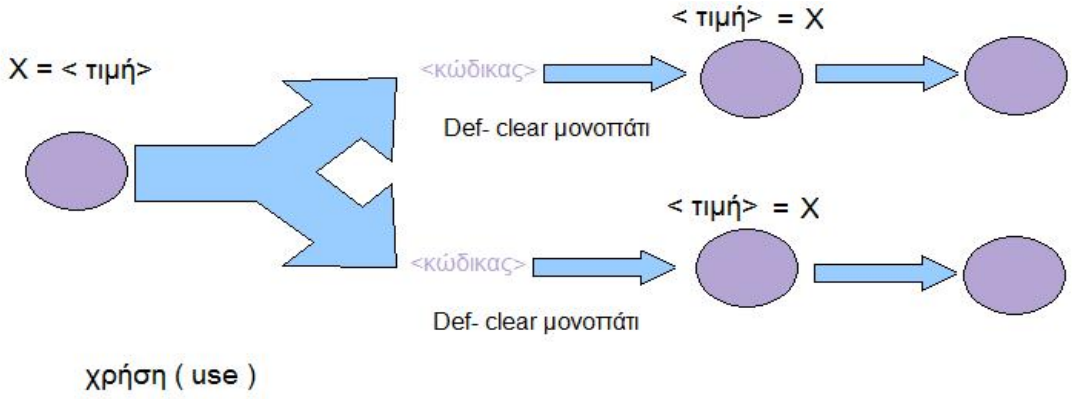
Βλέπουμε στο παραπάνω σχήμα (Σχήμα 2.4), σε μια αφαιρετική απεικόνιση γράφου ενός προγράμματος, όπου με d(y) συμβολίζουμε τον κόμβο στον οποίο γίνεται δήλωση της μεταβλητής y (declaration), και με u(y) υποδηλώνεται η χρήση της μεταβλητής y ενώ, όπως στο παράδειγμα, μπορούμε να έχουμε πολλές χρήσεις ή δηλώσεις μεταβλητών, για το διαχωρισμό τους χρησιμοποιείται ως υποδείκτης ο αριθμός χρήσης της δήλωσης/χρήσης.

Για το παράδειγμα που θέλουμε All-Defs απαιτείται μονοπάτι από το d1(x) μέχρις ότου μια

χρήση. Στη συγκεκριμένη περίπτωση το μονοπάτι που αποτελείται από την ακολουθία κόμβων 0,1,2,4,6 επαρκεί.

2.4.3.2 All-uses κριτήριο

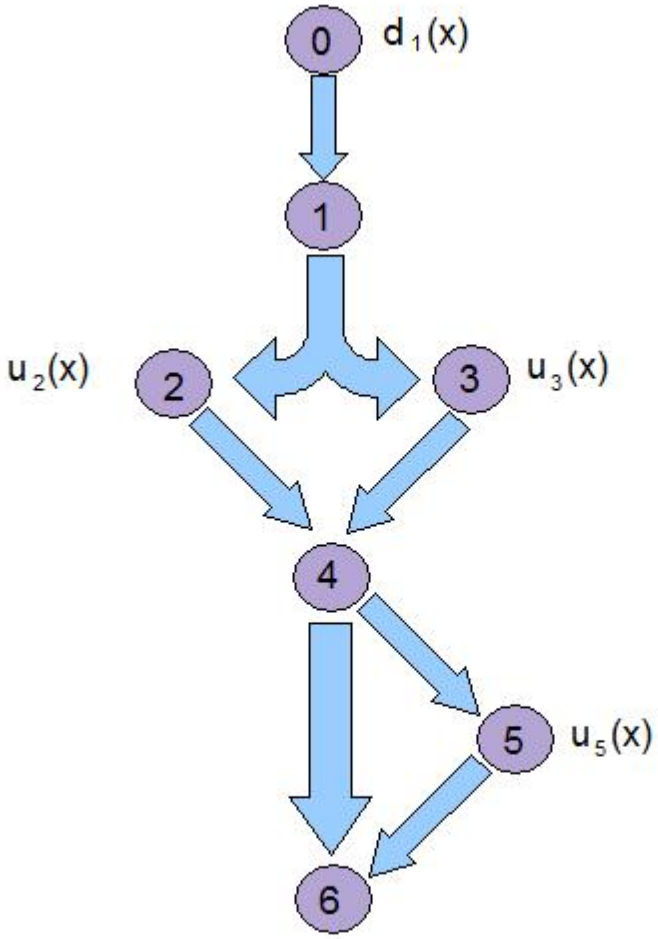
All-uses είναι τα definition-clear υπο-μονοπάτια από κάθε ορισμό μιας μεταβλητής (δήλωση-αρχικοποίηση) και κάθε χρήση της δήλωσης αυτής και κάθε απόγονο κόμβο αυτής της χρήσης (Σχήμα 2.5).



Σχήμα 2.5 : Παράδειγμα All-uses κριτηρίου

άλλα κριτήρια κάλυψης αφορούν την χρήση της μεταβλητής, αν αυτή η μεταβλητή δηλαδή χρησιμοποιείται για υπολογισμούς ή αν είναι προαπαιτούμενη (πχ για ελέγχους)
C-use (computation use) έχουμε για παράδειγμα για τη μεταβλητή x όταν κάνουμε την πράξη y = x * 4

ενώ P-use (predicate use) είναι όταν η μεταβλητή x χρειάζεται ως προαπαιτούμενη για την ολοκλήρωση μιας πράξης ελέγχου όπως if (x > 8)



Σχήμα 2.6 : Παράδειγμα δεύτερο All-uses κριτηρίου

Σύμφωνα με το All-Uses, θα πρέπει να βρούμε μονοπάτια (Σχήμα 2.6) τα οποία να οποία να ξεκινούν από τους κόμβους δήλωσης και να καταλήγουν σε όλες τις χρήσης των αντίστοιχων μεταβλητών. Έτσι, στο συγκεκριμένο παράδειγμα απαιτούνται μονοπάτια από :

- το $d_1(x)$ στο $u_2(x)$
- το $d_1(x)$ στο $u_3(x)$
- το $d_1(x)$ στο $u_5(x)$

Μπορούμε να δούμε λοιπόν από τον γράφο ότι μονοπάτια :

- 0,1,2,4,5,6 και
- 0,1,3,4,6

μπορούν να ικανοποιήσουν το κριτήριο αυτό.

2.4.3.3 All C-uses, Some P-uses κριτήριο

Το κριτήριο All-C-uses, Some-P-uses αφορά τον έλεγχο για :

- μερικά definition-clear υπο-μονοπάτια από κάθε ορισμό μέχρι κάθε C-use του μονοπατιού που καταλήγει από τη δήλωση της μεταβλητής
- και αν δεν καταλήγει σε κάποιο C-use ο ορισμός της μεταβλητής, τότε βρίσκουμε κάποιο definition-clear υπο-μονοπάτι από αυτόν τον ορισμό μέχρι τουλάχιστον μια P-use στην οποία να καταλήγει ο ορισμός της μεταβλητής.

2.4.3.4 All P-uses, Some C-uses κριτήριο

Το κριτήριο All-P-Uses, Some-C-Uses αφορά τον έλεγχο για :

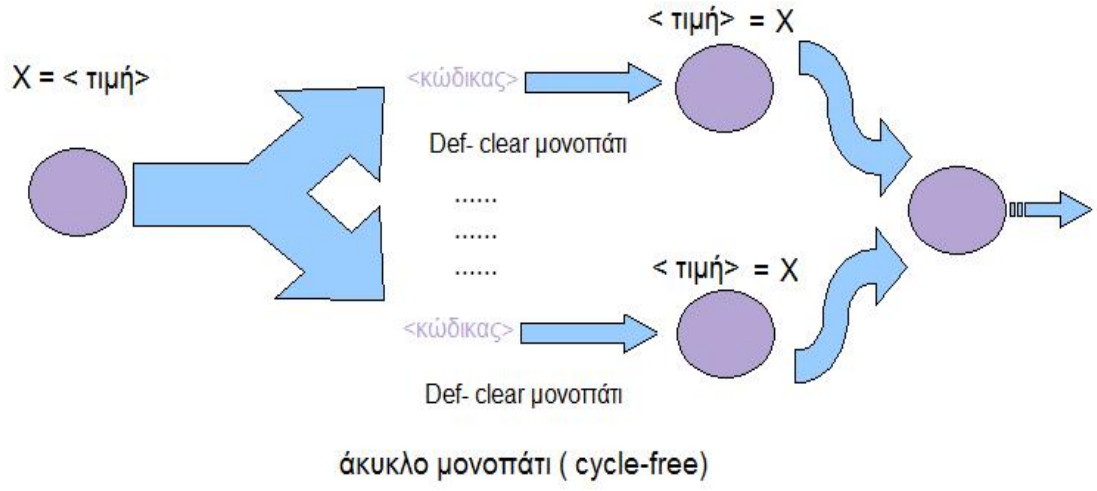
- μερικά definition-clear υπο-μονοπάτια από κάθε ορισμό μέχρι κάθε P-use του μονοπατιού που καταλήγει από τη δήλωση της μεταβλητής και κάθε απόγονο του κόμβου της χρήσης της μεταβλητής.
- αν δεν καταλήξουμε σε κάποιο P-use σε κανένα μονοπάτι που ξεκινά από δήλωση της μεταβλητής, τότε επιλέγουμε τουλάχιστον ένα definition-clear υπο-μονοπάτι στο οποίο καταλήγει η δήλωση της μεταβλητής.

2.4.3.5 All P-uses κριτήριο

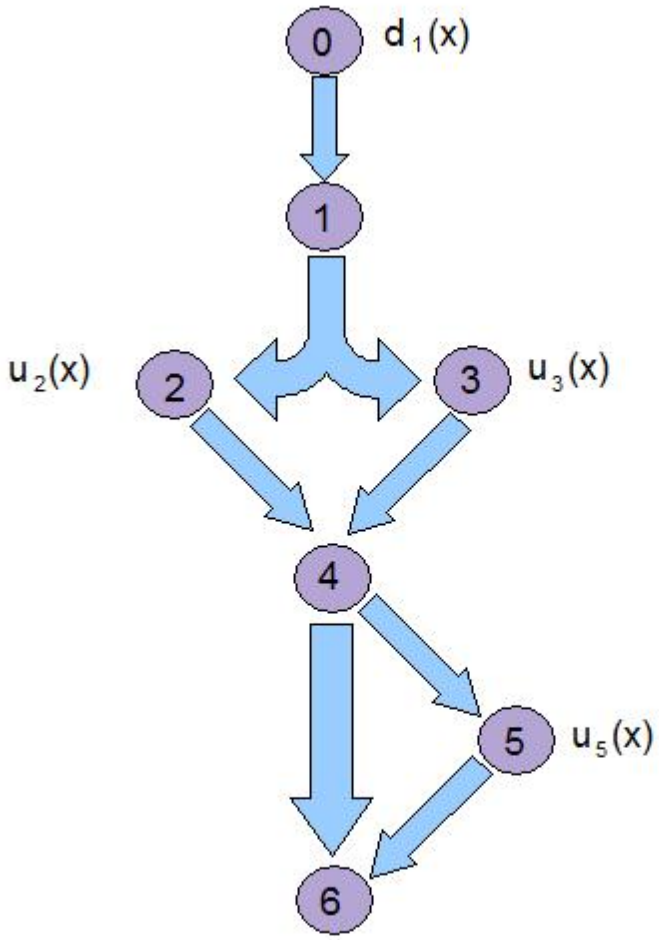
Definition-clear υπο-μονοπάτια που ξεκινάνε από κάθε ορισμό μεταβλητής μέχρι κάθε P-use της μεταβλητής αυτής και κάθε απόγονο κόμβο του μονοπατιού που ακολουθεί τη χρήση.

2.4.3.6 All DU-Paths κριτήριο

Με βάση αυτό το κριτήριο ελέγχου, βρίσκουμε όλα τα definition-clear υπο-μονοπάτια τα οποία δεν έχουν κύκλους (άκυκλοι γράφοι) ή έχουνε απλούς κύκλους από κάθε ορισμό - δήλωση μιας μεταβλητής μέχρι κάθε χρήση της μεταβλητής αυτής και κάθε απόγονο κόμβο που ακολουθείται από την χρήση της μεταβλητής (Σχήμα 2.7).



Σχήμα 2.7: Παράδειγμα All-DU κριτηρίου



Σχήμα 2.8: Παράδειγμα δεύτερο με All-DU-μονοπάτια

Στο παράδειγμα αυτό θέλουμε να βρούμε τα μονοπάτια που να ικανοποιούν το κριτήριο All-DU-μονοπάτια. Σύμφωνα με αυτό, θέλουμε να βρούμε μονοπάτια από :

- το $d_1(x)$ στο $u_2(x)$
- το $d_1(x)$ στο $u_3(x)$
- και τα δυο μονοπάτια που ξεκινούν από το $d_1(x)$ και καταλήγουν στον κόμβο με το

$u_5(x)$

Συνεπώς, τα μονοπάτια που θα ικανοποιούσαν αυτές τις συνθήκες του κριτηρίου είναι προφανώς τα :

- 0,1,2,4,5,6
- 0,1,3,4,5,6

2.4.3.7 Ntafos Coverage κριτήριο

Σύμφωνα με τον Ntafos, προτείνεται μια οικογένεια από κριτήρια ελέγχου δεδομένων, βασισμένα σε αλυσίδες από άμεσες ροές δεδομένων αποκαλούμενο "k-dr αλληλεπιδράσεις" (k-dr interactions). Πιο επίσημα, μια k-dr αλληλεπίδραση μέσα στον γράφο ροής δεδομένων ενός προγράμματος είναι μια ακολουθία από k κόμβους, όπου $k>=2$, $<v_1, v_2, v_3, \dots , v_k>$ και μια ακολουθία από k-1 μεταβλητές $<x_1, x_2, x_3, \dots x_{k-1}>$ τέτοιες ώστε για $i = 2, \dots, k$ ο κόμβος vi να είναι άμεσα εξαρτώμενος από δεδομένα (directly data dependent) με τον κόμβο vi σε συμφωνία με το xi – 1. Οι απαιτούμενες k-πλειάδες που προτείνονται από το κριτήριο ελέγχου αυτό ικανοποιούνται από ένα σύνολο T από δεδομένα ελέγχου εάν, μεταξύ άλλων , το σύνολο T πραγματοποιεί κάθε δυνατή 1-dr αλληλεπίδραση στον γράφο ροής του προγράμματος τουλάχιστον μια φορά για $2 \leq l \leq k$. Σε μεγαλύτερη ανάλυση μπορούμε να δούμε την κάλυψη με το κριτήριο αυτό στην βιβλιογραφία [4, 5, 6].

2.4.3.8 Επισκόπηση

Συμπερασματικά, για όσα έχουν ειπωθεί για τα προαναφερθέντα κριτήρια, πρέπει να σημειωθεί ότι τα σύνολα από μονοπάτια τα οποία ικανοποιούν ένα κριτήριο δεν είναι απαραίτητα μοναδικά. Θα μπορούσαμε να αξιολογήσουμε τα κριτήρια αυτά αν ορίζαμε μια συνάρτηση – συσχέτιση υποσυνόλων για την οποία να ισχύει ότι το κριτήριο Β εντάσσεται

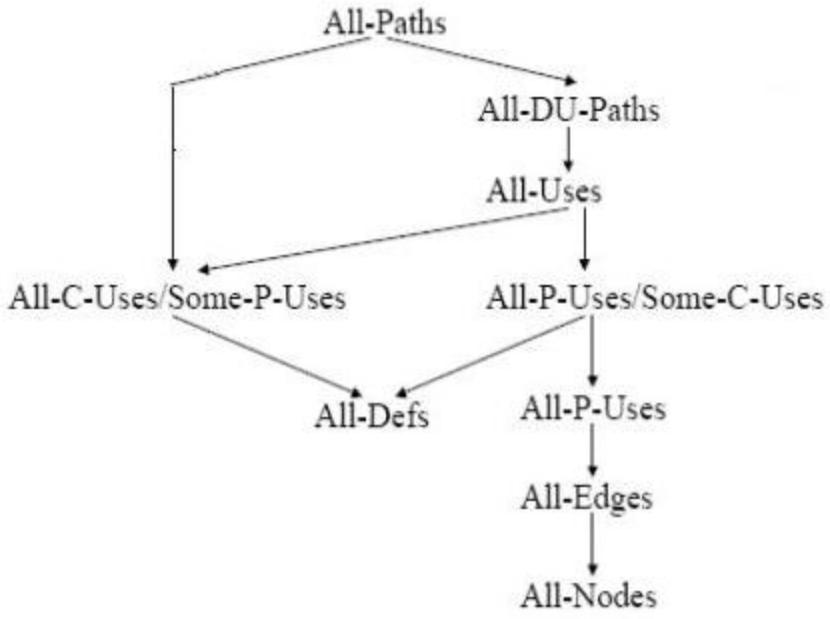
στο κριτήριο A αν και μόνο αν ισχύει για κάθε γράφο ροής :

S ικανοποιεί A = => S ικανοποιεί B

όπου S είναι ένα σύνολο μονοπατιών.

Τα κριτήρια A και B είναι ισοδύναμα (equivalent) αν και μόνο αν το A εντάσσεται στο B και το B εντάσσεται στο A.

Εν κατακλείδι, μπορούμε να συνοψίσουμε διαγραμματικά τις συσχετίσεις αυτές με το παρακάτω σχήμα (Σχήμα 2.9).



Σχήμα 2.9: Παράδειγμα συσχετίσεων κριτηρίων DFG

Μπορούμε να πούμε ότι τα κριτήρια αυτά μας παρέχουν ένα πλαισιωμένο τρόπο σκέψης με κανόνες και κριτήρια σύγκρισης και μας επιτρέπουν να διαμορφώσουμε τη λογική με την οποία θα επιλέξουμε τους συνδυασμούς των μονοπατιών που θα λάβουμε υπόψη μας κατά τον έλεγχο κάλυψης.

Το πιο ευρέως χρησιμοποιούμενο κριτήριο κάλυψης είναι το all-uses. Γενικότερα θεωρούνται ως βελτίωση σε σχέση με τις τεχνικές ελέγχου της ροής ελέγχου (control flow). Ένα από το προβλήματα που συναντάμε ωστόσο είναι ότι με την data flow coverage έχουμε ανέφικτα μονοπάτια και συνήθως δεν μπορούμε να έχουμε 100% κάλυψη.

Κεφάλαιο 3

Αυτόματη παραγωγή Περιπτώσεων Ελέγχου

3.1 Περιπτώσεις ελέγχου (test-cases)

Στην τεχνολογία λογισμικού, με τον όρο περίπτωση-ελέγχου (test-case) εννοείται ένα σύνολο από συνθήκες ή μεταβλητές, με βάση τις οποίες η ομάδα που αναλαμβάνει τον έλεγχο της εφαρμογής θα καθορίσει αν αυτή ανταποκρίνεται στις προδιαγραφές της. Συχνά απαιτείται η παραγωγή πολλών περιπτώσεων ελέγχου για να μπορεί να διαπιστωθεί ότι πράγματι το σύστημα ή το λογισμικό λειτουργεί σωστά. Οι περιπτώσεις ελέγχου γράφονται με βάση τις απαιτήσεις και προδιαγραφές του κώδικα που ελέγχεται. Ανάλογα με το είδος του ελέγχου που πραγματοποιείται, και με βάση τις συνθήκες κάλυψης που οι ελεγκτές θεωρούν ικανοποιητικές, παράγονται τουλάχιστον τόσες περιπτώσεις ελέγχου όσες και οι

προδιαγραφές που υπάρχουν (βάση Formal, requirement-based περιπτώσεις ελέγχου).

3.2 Παραγωγή περιπτώσεων ελέγχου για έλεγχο Αντικειμενοστραφών προγραμμάτων.

Όσον αφορά την παραγωγή περιπτώσεων ελέγχου για αντικειμενοστραφή προγράμματα (Object-Oriented programs) υπάρχουν διαφορετικές παράμετροι που λαμβάνονται υπόψη από αυτές στον διαδικαστικό (procedural) κώδικα. Παράμετροι όπως :

- Το ότι η μικρότερη δυνατή μονάδα που μπορεί να υποστεί έλεγχο είναι η ενσωματωμένη κλάση (encapsulated class).
- Ο έλεγχος γίνεται σε κάθε λειτουργία ως μέρος της ιεραρχίας μιας κλάσης αφού κάθε ιεραρχία της κλάσης καθορίζει το περιεχόμενο της χρήσης.
- Έλεγχος κάθε μεθόδου (method) και κατασκευαστή (constructor) της κλάσης.
- Έλεγχος της συμπεριφοράς της κατάστασης, γνωρισμάτων της κλάσης (attributes) μεταξύ μεθόδων.

Βλέπουμε λοιπόν από τις παραμέτρους που αναφέρθηκαν επιγραμματικά παραπάνω, ότι υπάρχουν διαφορές ανάμεσα στον έλεγχο κλασικών διαδικαστικών προγραμμάτων και προγραμμάτων αντικειμενοστραφούς κώδικα. Στην πρώτη περίπτωση δίνεται έμφαση στο τρίπτυχο είσοδος-επεξεργασία-έξοδος ενώ στην δεύτερη περίπτωση, με τα αντικειμενοστραφή προγράμματα, ο έλεγχος των κλάσεων επικεντρώνεται στον έλεγχο κάθε μεθόδου ξεχωριστά αλλά και το σχεδιασμό ακολουθιών μεθόδων που επηρεάζουν την κατάσταση των κλάσεων. Ωστόσο, οι τεχνικές που προτείνει ο έλεγχος άσπρου κουτιού ισχύουν και στον αντικειμενοστραφή προγραμματισμό.

Η σχεδίαση περιπτώσεων ελέγχου για αντικειμενοστραφή προγράμματα γίνεται για κάθε μέθοδο ξεχωριστά, με βάση την τρέχουσα μέθοδο που ελέγχεται. Περιπτώσεις ελέγχου μπορεί να περιέχουν πληροφορίες με τις λειτουργίες και τις τιμές που περιλαμβάνουν, την ακολουθία δηλώσεων που εκτελούνται, την ακολουθία εξαιρέσεων (exceptions) που

πιθανόν να συναντηθούν κατά την εκτέλεση, εξωτερικών παραμέτρων και συνθηκών, κ.ο.κ. Μεταξύ άλλων, ο έλεγχος αντικειμενοστραφούς λογισμικού έχει να αντιμετωπίσει και άλλες προκλήσεις, όπως :

Η ενθυλάκωση (encapsulation), που επιφέρει την δυσκολία αποτύπωσης στιγμιότυπου της κλάσης χωρίς να κατασκευαστούν επιπρόσθετες μέθοδοι που θα περιγράφουν επακριβώς την κατάσταση της κλάσης.

Ο πολυμορφισμός (polymorphism), που σημαίνει ότι κάθε χρήση διαφορετικού περιεχομένου κλάσης απαιτεί ξεχωριστό έλεγχο, αφού μια μέθοδος μπορεί να έχει υλοποιηθεί πολλές φορές με διαφορετικό τρόπο.

Η κληρονομικότητα (inheritance), η οποία μεταξύ άλλων επιβάλλει τον έλεγχο άλλων, μη τροποποιημένων μεθόδων μέσα σε μια υποκλάση (subclass) που μπορεί να χρησιμοποιούν δηλωμένες διαφορετικά μεθόδους. Με άλλα λόγια, μπορεί να γίνεται χρήση δηλωμένων (διαφορετικών) μεθόδων στην θυγατρική κλάση ή μεθόδων που ανήκουν σε πατρικές υπερ-κλάσεις (superclass) .

Αυτοματοποιημένα εργαλεία, όπως το παρόν σκοπεύουν στη διευκόλυνση των ομάδων ελέγχου λογισμικού, παράγοντας αυτόματα περιπτώσεις ελέγχου, ελέγχοντας και αξιολογώντας αυτές, και επιτυγχάνοντας είδη κάλυψης με βάση τους δημιουργημένους γράφους. Η κάλυψη αυτή, στα πλαίσια ελέγχου άσπρου κουτιού γίνεται σε επίπεδο κάλυψης μονοπατιών δημιουργημένων από το κριτήριο k-πλειάδων (Ntafos Coverage) [4, 5, 7] , κάλυψη ακμών, με βάση το γράφο ροής ελέγχου, και κάλυψη συνθήκης (edge/condition coverage).

3.3 Είδη Παραγωγής Περιπτώσεων ελέγχου

Μπορεί να ειπωθεί ότι υπάρχουν αρκετά είδη παραγωγής περιπτώσεων ελέγχου, με βάση

την οπτική γωνία από την οποία πραγματοποιείται ο έλεγχος και η παραγωγή test-cases. Στη συνέχεια θα αναφερθούν περιληπτικά οι κυριότερες κατηγορίες παραγωγής ελέγχου στοχεύοντας στο να διασαφηνιστεί ο τρόπος παραγωγής περιπτώσεων ελέγχου αυτού του εργαλείου. Ανάμεσα στις κατηγορίες αυτές είναι η παραγωγή περιπτώσεων ελέγχου με βάση τις προδιαγραφές (Specification-Based), με βάση το μοντέλο (Model-Based) και με βάση τον πηγαίο κώδικα και τα μονοπάτια.

3.3.1 Παραγωγή περιπτώσεων ελέγχου με βάση προδιαγραφές

Όπως προτάθηκε από τους [19, 20, 21, 23] , μια τεχνική η οποία προσαρμόζει προκαθορισμένες προδιαγραφές βασισμένες στην κατάσταση για να παράξει περιπτώσεις ελέγχου από UML διαγράμματα καταστάσεων (state-charts). Παράδειγμα αυτού αυτοματοποιημένου εργαλείου που λειτουργεί με τον παραπάνω τρόπο είναι το UMLTEST, ένα εργαλείο παραγωγή περιπτώσεων ελέγχου ενσωματωμένο στο Rational Rose. Ο έλεγχος λογισμικού, όπως αυτός περιγράφεται από Marlon E Viera et al [8], χρησιμοποιεί επίσης διαγράμματα καταστάσεων UML ως βάση για την παραγωγή οδηγών ελέγχου (test drivers) και scripts για τον έλεγχο των τμημάτων που υφίστανται έλεγχο. Οι συγγραφείς της προσέγγισης αυτής υλοποίησαν και ένα πρωτότυπο αποκαλούμενο Design And Specification-Based Object-Oriented Testing (DAS-BOOT) . Στο εργαλείο αυτό, ο χρήστης επιλέγει την κλάση java που επιθυμεί να ελέγξει και το αντίστοιχο διάγραμμα-καταστάσεων προδιαγραφών της κλάσης και καθορίζει την αναπαράσταση-αντιστοίχηση με το να συσχετίσει τον κώδικα στις προδιαγραφές. Έπειτα μπορούν να επιλεχθούν τα κριτήρια κάλυψης, και ξεκινάει η αυτοματοποιημένη παραγωγή περιπτώσεων ελέγχου.

3.3.2 Παραγωγή περιπτώσεων ελέγχου με βάση μοντέλου

Η συνεχής έρευνα σε αυτήν την προσέγγιση, οδήγησε , όπως μπορούμε να δούμε στο Jeff Offutt et al [19, 21] στην επέκταση της εργασίας τους την ανάλυση με χρήση

διαγραμμάτων κατάστασης σε επίπεδο συστήματος στην ανάλυση με χρήση διαγραμμάτων συνεργασίας (collaboration-diagrams) στη φάση της συνένωσης. Οι συγγραφείς περιγράφουν κριτήρια τόσο για στατικό όσο και για δυναμικό έλεγχο σε επίπεδο προδιαγραφών και σε επίπεδο στιγμιότυπων διαγράμματα συνεργασίας. Έχει σχεδιαστεί αλγόριθμος ο οποίος επιβεβαιώνει ότι οι έλεγχοι ικανοποιούν τα επίσημα κριτήρια ελέγχου και βοηθά τον χρήστη να παρακολουθεί τα μονοπάτια. Σε επίπεδο unit, τα διαγράμματα κατάστασης είχαν καλύτερα αποτελέσματα σε σχέση με τα ακολουθίας (sequence) ωστόσο σε επίπεδο συνένωσης συνέβη το αντίθετο. Για την παραγωγή ακόμη ισχυρότερων περιπτώσεων ελέγχου σχεδιάζεται η συνένωση προδιαγραφών UML με με την OCL (Object Constraint Language). Στα πλαίσια αυτής της φιλοσοφίας, για την αυτοματοποιημένη παραγωγή περιπτώσεων ελέγχου με βάση μοντέλου, έχουν κατασκευαστεί πολλά εργαλεία, σχεδιασμένα για να δουλεύουν με άλλα γνωστά εργαλεία, όπως το TGV, και το UMLAUT.

3.3.3 Παραγωγή περιπτώσεων ελέγχου με βάση μονοπατιών

Ένα εργαλείο που χρησιμοποιεί τον γνωστό αλγόριθμο δυαδικής αναζήτησης στα πλαίσια της παραγωγής περιπτώσεων ελέγχου με βάση μονοπάτια παρουσιάζεται από τους Sami Baydeda and Volker Gruhn [26], και κατατάσσεται στις προσεγγίσεις δυναμικών μονοπατιών. Το μονοπάτι που επιχειρείται να καλυφθεί λαμβάνεται υπόψη βήμα-βήμα και μετά ακολουθεί η αναζήτηση των περιπτώσεων ελέγχου που θα το καλύψουν. Στην έρευνα αυτή, ο αλγόριθμος δυαδικής αναζήτησης απαιτεί ορισμένες υποθέσεις ενώ στα θετικά της συγκαταλέγεται το γεγονός ότι δεν απαιτείται τροποποίηση παραμέτρων ούτε τεχνικές βελτιστοποίησης για την παραγωγή περιπτώσεων ελέγχου.

3.3.4 Ευφυείς μετα-ευρετικές τεχνικές παραγωγής περιπτώσεων ελέγχων

Μπορούμε να θεωρήσουμε μια ξεχωριστή κατηγορία, ένα σύνολο ευφυών – εξελικτικών τεχνικών για αυτοματοποιημένη παραγωγή περιπτώσεων ελέγχου οι οποίες ανήκουν στα προβλήματα αναζήτησης [1,3], εύρεσης δηλαδή της λύσης ενός προβλήματος, εάν αυτή υπάρχει. Η φιλοσοφία της προσέγγισης αυτής, η οποία θεωρείται από τις πιο αποτελεσματικές και αποδοτικές τεχνικές [2] και χρησιμοποιείται από το BPAS & ATCGS είναι η εξέλιξη των παραγόμενων περιπτώσεων ελέγχου και επιλογή των καλύτερων (υγιέστερων) από αυτές, κατ' αναλογία με την εξέλιξη των χρωμοσωμάτων και την φυσική επιλογή που γνωρίζουμε από τη βιολογία. Πιο αναλυτικά, σχηματίζεται ένας αρχικός πληθυσμός από χρωμοσώματα, με βάση την μέθοδο της κλάσης που εξετάζουμε, αν για παράδειγμα έχουμε τις παραμέτρους A, B, C ως είσοδο σε μια μέθοδο, τότε ένα πιθανό χρωμόσωμα θα μπορούσε να είναι το [A= 320, B= -143, C= 0]. Οι γενετικοί τελεστές (αναφερόμενοι εκτενώς σε προηγούμενο κεφάλαιο) της μετάλλαξης (mutation), της διασταύρωσης (crossover), καθώς και η διαδικασία της φυσικής επιλογής, σε συνδυασμό με διαφορετικές (ανάλογα με το είδος της κάλυψης μονοπατιών που θέλουμε να πετύχουμε) συναρτήσεις αξιολόγησης της φυσικής κατάστασης των χρωμοσωμάτων συντελούν στον σχηματισμό μέσω συνεχούς εξέλιξης (evolution) των ελίτ χρωμοσωμάτων. Με αυτό εννοείται ότι σε κάθε εξέλιξη (στην οποία τα υποψήφια χρωμοσώματα-λύσεις υφίστανται γενετικές λειτουργίες από γενετικούς τελεστές) πλησιάζουν τα επίλεκτα χρωμοσώματα – περιπτώσεις ελέγχου όλο και πιο κοντά στη βέλτιστη λύση. Η συνάρτηση αξιολόγησης φυσικής κατάστασης (fitness evaluation function) είναι ένα από τα πιο καθοριστικά και σημαντικά τμήματα, που υλοποιείται ξεχωριστά και αφορά αποκλειστικά το πρόβλημα εύρεσης που έχουμε να αντιμετωπίσουμε. Έτσι, διαφορετικές συναρτήσεις χρησιμοποιούνται για την κάλυψη Ntafos από τον επονομαζόμενο γράφο ροής δεδομένων [5, 6, 7] και άλλη για την κάλυψη ακμών/συνθηκών βασισμένα στο γράφο ροής ελέγχου του εργαλείου. Τροποποιώντας παραμέτρους όπως το μέγεθος του πληθυσμού, τον αριθμό των εξελίξεων, το ποσοστό φυσικής επιλογής, κ.ο.κ επηρεάζεται άμεσα η λύση η οποία θα βρεθεί (για παράδειγμα αν μειώσουμε τον αριθμό των εξελίξεων τότε είναι πολύ πιθανό η βέλτιστη λύση που θα βρεθεί από το εργαλείο να απέχει πολύ από την πραγματική βέλτιστη λύση ή σύνολο βέλτιστων λύσεων) αλλά και ο χρόνος που θα χρειαστεί να

δαπανηθεί για την εύρεση της λύσης. Οι γενετικοί αλγόριθμοι για την παραγωγή περιπτώσεων ελέγχου όπως και οι συναρτήσεις αξιολόγησης φυσικής κατάστασης ενσωματώθηκαν στο εργαλείο με μερικές τροποποιήσεις (λόγω αντικατάστασης της παλιάς βιβλιοθήκης γενετικών αλγορίθμων με τη νεότερη έκδοση) από προηγούμενες εργασίες φοιτητών και μελών ακαδημαϊκού προσωπικού του τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου όπως στα [6, 7, 14].

3.3.5 Γενετικοί αλγόριθμοι και η χρήση τους στο εργαλείο

Οι γενετικοί αλγόριθμοι είναι μια τεχνική αναζήτησης που χρησιμοποιείται στον υπολογισμό με στόχο την ακριβή εύρεση, όπου είναι εφικτό, ή την προσέγγιση λύσεων σε προβλήματα αναζήτησης και βελτιστοποίησης. Περισσότερα για τη χρήση των γενετικών αλγορίθμων σε προβλήματα αναζήτησης, βελτιστοποίησης, αυτόματης μάθησης και εύρεσης σε μεγάλο χώρο καταστάσεων μπορούμε να δούμε στα [1,2,3]. Είναι με άλλα λόγια ένας ευρετικός προσαρμοστικός αλγόριθμος αναζήτησης (adaptive heuristic search algorithm) που είναι μια υπο-κατηγορία των εξελικτικών αλγορίθμων, οι οποίοι χρησιμοποιούν τεχνικές εξελικτικής βιολογίας, όπως την κληρονομικότητα (inheritance), τη μετάλλαξη (mutation), την επιλογή (selection) και τη διασταύρωση (crossover).

Ενσωματώνοντας την έξυπνη αυτή τεχνική εύρεσης στην εφαρμογή επωφελούμαστε με πολλούς τρόπους, όπως :

- ο χώρος αναζήτησης, οι περιπτώσεις ελέγχου που θέλουμε δηλαδή να παράξουμε για να καλύψουν τις ακμές ή σύνθετα μονοπάτια μέσα σε ένα γράφο, είναι πολύ μεγάλος και πολύπλοκος. Ας έχουμε υπόψη μας ότι στις περισσότερες περιπτώσεις χρειαζόμαστε όχι μόνο μια τιμή για κάλυψη ενός μονοπατιού αλλά ζεύγη τιμών, ή τριάδες, ή σύνολα τιμών προσεκτικά επιλεγμένα για να τηρούν τις προϋποθέσεις κάλυψης, που μπορεί να είναι ένας σύνθετος έλεγχος.
- η δυσκολία της μηχανής που παρά την υπολογιστική της ανωτερότητα, να

κατανοήσει και να λύσει προβλήματα απλά για τον ανθρώπινο εγκέφαλο όπως μια πολύπλοκη εξίσωση, έναν πολυσύνθετο έλεγχο περιορισμών, κλπ. Με τη χρήση των γενετικών αλγορίθμων επιτυγχάνουμε να περιορίσουμε τον χώρο αναζήτησης αφού κάθε σύνολο τιμών που παράγονται από τις μεταλλάξεις του πληθυσμού, τις διασταυρώσεις και την επιλογή αξιολογούνται από συναρτήσεις αξιολόγησης της αποτελεσματικότητας τους και με τον έξυπνο αυτό τρόπο ευνοούνται οι καλύτερες λύσεις έναντι λιγότερο αποδοτικών βελτιστοποιώντας στο σύνολο αποτελεσμάτων και αυξάνοντας το ποσοστό κάλυψης με βάση τα κριτήρια που τίθενται σε λιγότερο χρόνο.

- Είναι επίσης φανερό το ότι η χρήση γενετικών αλγορίθμων υπερτερεί πολύ από πολλές 'παραδοσιακές' μεθόδους επίλυσης προβλημάτων, ειδικά στον τομέα του ελέγχου λογισμικού και της τεχνητής νοημοσύνης.

Συνοπτικά , οι γενετικοί αλγόριθμοι ξεκινούν με έναν πληθυσμό, ο οποίος ξεκινά με έναν τυχαία παραγόμενο πληθυσμό από κωδικοποιημένα υποψήφια χρωμοσώματα –λύσεις. Αρχικά γίνεται η ρύθμιση (configuration) στην οποία δίνονται ως παράμετροι :

- το σύνολο χρωμοσωμάτων,
- το ελάχιστο ποσοστό μεγέθους του πληθυσμού,
- τον φυσικό επιλογέα (natural selector),
- τον τελεστή διασταύρωσης (crossover operator),
- τον τελεστή μετάλλαξης (mutation operator),
- τον ελιτισμό (elitism), την διαφύλαξη δηλαδή, των καλύτερων χρωμοσωμάτων,
- την συνάρτηση αξιολόγησης (fitness evaluator),
- δείγματα χρωμοσωμάτων τα οποία περιέχει γονίδια (genes) τα οποία παίρνουμε από την στατική ανάλυση του προγράμματος, τα αποτελέσματα δηλαδή που προκύπτουν από μη χρόνο-εκτέλεσης (program non-runtime results).

Στη συνέχεια, όταν η παραγωγή περιπτώσεων ελέγχου ξεκινήσει, και αφού τα μονοπάτια που θέλουμε να καλύψουμε είναι διαθέσιμα (έχουν αποθηκευτεί ή εξαχθεί από τον γράφο), τότε για κάθε μονοπάτι γίνεται εφαρμογή των γενετικών τελεστών πάνω στον πληθυσμό.

Αυτό γίνεται για όλες τις διαθέσιμες εξελίξεις που έχει επιλέξει ο χρήστης να γίνουν μέχρι να καλυφθεί το μονοπάτι που στοχεύει κάθε φορά ή μέχρι να εξαντληθούν όλες οι διαθέσιμες μεταλλάξεις.

Σε κάθε μετάλλαξη και διασταύρωση, ο πληθυσμός εξελίσσεται πλησιάζοντας σε μια βέλτιστη λύση. Η βέλτιστη λύση όμως δεν είναι κάτι που μπορεί να εγγυηθεί , όμως με τον έξυπνο σχεδιασμό ενός τρόπου αξιολόγησης των υποψήφιων λύσεων είναι ένα πολύ σημαντικό βήμα προς αυτήν την κατεύθυνση, την αύξηση της πιθανότητας εύρεσης της βέλτιστης λύσης. Αυτό που γίνεται στην πράξη είναι η σωστή αξιολόγηση της καταλληλότητας (fitness) κάθε υποψήφιας λύσης στον τρέχοντα πληθυσμό και η επιλογή της καταλληλότερης υποψήφιας λύσης του τρέχοντος πληθυσμού για να λειτουργήσουν ως γονείς στην επόμενη γενιά υποψήφιων λύσεων (εκμεταλλευόμενοι το γεγονός της καλής κατάστασης των γονέων, αναμένουμε ότι κάποιοι από τους απογόνους θα είναι ακόμη καταλληλότεροι, κ.ο.κ.).

Μετά την επιλογής τους για αναπαραγωγή, οι γονείς ξανά-συνδυάζονται, με χρήση τελεστή διασταύρωσης (crossover operator) και μεταλλάσσονται με χρήση τελεστή μετάλλαξης (mutation operator). Οι καταλληλότεροι γονείς και οι νέοι απόγονοι σχηματίζουν το νέο πληθυσμό και η ίδια διαδικασία συνεχίζεται.

Οι λειτουργίες της αξιολόγησης , της επιλογής, του επανασυνδυασμού και της μετάλλαξης είναι λειτουργίες οι οποίες εκτελούνται πολλές φορές σε έναν γενετικό αλγόριθμο. Η επιλογή, η μετάλλαξη και η διασταύρωση είναι γενικές λειτουργίες των γενετικών αλγορίθμων επί του πληθυσμού. Ωστόσο η αξιολόγηση είναι ανάλογη με το είδος του προβλήματος που επιθυμούμε να επιλύσουμε και εξαρτάται από τη δομή των λύσεων. Το παρόν εργαλείο χρησιμοποιεί δυο διαφορετικές συναρτήσεις αξιολόγησης, οι οποίες ενσωματώθηκαν από προηγούμενες εργασίες σε διπλωματικές εργασίες φοιτητών, όπως προαναφέρθηκε. Η πρώτη αφορά την αξιολόγηση των υποψήφιων λύσεων που παράγονται με στόχο την εύρεση βέλτιστη λύση που θα καλύπτει τα μονοπάτια που έχουν εξαχθεί από τον γράφο ροής δεδομένων. Η δεύτερη είναι η αντίστοιχη, η οποία στοχεύει στην αξιολόγηση λύσεων που θα επιτύχουν κάλυψη ακμής/συνθήκες (edge/condition coverage) στον γράφο ροής ελέγχου του πηγαίου κώδικα που βρίσκεται υπό έλεγχο.

Κρίνεται άσκοπο να δοθεί περισσότερο βάθος σε αυτό το κομμάτι του κώδικα αφού ως επί

το πλείστον βασίζεται σε παλαιότερες εργασίες φοιτητών και μπορούν να βρεθούν σε προηγούμενες εργασίες και έρευνες [6, 7, 14].

Ανάμεσα στα μεγαλύτερα πλεονεκτήματα αυτής της προσέγγισης, της παραγωγής δηλαδή περιπτώσεων ελέγχου με γενετικούς αλγορίθμους, είναι ότι οι ΓΑ παρέχουν την ευελιξία και αποτελεσματικότητα, κάνοντας την μια πολύ εύρωστη μέθοδο αναζήτησης [15, 16, 17]. Ανάλογα και με το πρόβλημα που επιδιώκεται να επιλυθεί, μπορεί να ειπωθεί ότι οι ΓΑ κάνουν σχετικά λίγες υποθέσεις όσον αφορά το πρόβλημα, όμως για τον έλεγχο λογισμικού, με κατάλληλη υλοποίηση συνάρτησης αξιολόγησης φυσικής κατάστασης μπορεί να αποδειχθούν πάρα πολύ αποτελεσματικοί, με μεγάλα ποσοστά κάλυψης για τα κριτήρια που τίθενται [2, 17]. Παράλληλα, τέτοιου είδους προσέγγιση είναι ανοικτή για εκμετάλλευση από παράλληλη επεξεργασία, γεγονός που σε σύγχρονα μηχανήματα σημαίνει ότι επιτρέπει την παράλληλη υλοποίηση και περεταίρω αύξηση της αποδοτικότητάς τους. Αυτό ισχύει διότι παράγουν πολλαπλούς απογόνους, οι οποίοι μπορούν να καλύψουν τεράστιους χώρους αναζήτησης ικανοποιητικών περιπτώσεων ελέγχου και άρα να έχουν υψηλά ποσοστά κάλυψης και σε κώδικα με μεγάλη πολυπλοκότητα, αφού οι απόγονοι μπορούν να εξεταστούν και να αξιολογηθούν από τη συνάρτηση αξιολόγησης παράλληλα και όχι σειριακά, κάτι που θα ήταν ιδιαίτερα χρονοβόρο.

Στα μειονεκτήματα της παραγωγή περιπτώσεων ελέγχου με ΓΑ μπορεί να συμπεριληφθεί το γεγονός ότι είναι πιθανό να συμβεί πρόωρη σύγκλιση (premature convergence), δηλαδή αν αναπτυχθούν μερικά γονίδια τα οποία είναι σε άριστη φυσική
κατάσταση σε σχέση με τα υπόλοιπα, σε πρώτα στάδια των εξελίξεων, τότε μπορεί πολύ γρήγορα τα γονίδια αυτά να κυριαρχήσουν στον πληθυσμό, ωθώντας τον στο να τείνει να συγκλίνει προς ένα τοπικό μέγιστο (local maximum) [1, 15, 16, 17]. Αν αυτό συμβεί, τότε η ικανότητα του ΓΑ να συνεχίσει την αναζήτηση καλύτερων λύσεων μειώνεται σημαντικά αφού η διασταύρωση μεταξύ πολύ παρόμοιων χρωμοσωμάτων δεν έχει να προσφέρει πολλά στην αναζήτηση του χώρου τον πιθανών λύσεων και μένουν στάσιμοι προσεγγίζοντας απλά τοπικά μέγιστα (local maxima convergence). Ο μόνος τελεστής που απομένει για να διευρύνει τον χώρο αναζήτησης είναι η μετάλλαξη, και αυτή συνήθως

πραγματοποιεί μια σχετικά αργή, τυχαία αναζήτηση.

Ένα άλλο πιθανό εμπόδιο πηγάζει από την ευελιξία των ΓΑ, και αυτό κάνει επιτακτική την ανάγκη κωδικοποίησης με σωστό τρόπο του προβλήματος και υλοποιώντας μια πολύ αποτελεσματική συνάρτηση αξιολόγησης που να αναθέτει τιμές λογικές σημασιολογικά οι οποίες να ωθούν την εξέλιξη σε ολοένα καλύτερες λύσεις. Τέλος, η παραγωγή περιπτώσεων ελέγχου με ΓΑ μπορεί να γίνει υπολογιστικά απαιτητική. Αναλυτικότερα μπορούμε να δούμε πλεονεκτήματα και μειονεκτήματα των ΓΑ σε εφαρμογές στη βιβλιογραφία [1, 2 , 3, 16].

Κεφάλαιο 4

Περιγραφή του Εργαλείου BPAS & ATCGS

4.1 Στόχος εφαρμογής

Σκοπός της εργασίας ήταν η δημιουργία μιας ολοκληρωμένης εφαρμογής η οποία να δεδομένου ενός προγράμματος γραμμένου σε java να πραγματοποιεί μια βασική ανάλυση του κώδικα (Basic Program Analyzer System), να δημιουργεί τρία είδη γράφων κατ'επιλογήν του χρήστη και να παράγει test cases αυτόματα (Automatic Test Cases Generation System).

Αποτελείται δηλαδή από δυο κύρια τμήματα, τον αναλυτή προγράμματος (BPAS) και το σύστημα που παράγει περιπτώσεις ελέγχου και πραγματοποιεί white-box έλεγχο στον επιλεγμένο πηγαίο κώδικα (ATCGS).

Παράλληλα, μια από τις προδιαγραφές του εργαλείου αυτού είναι ότι είναι διαθέσιμο για χρήση από το διαδίκτυο με τη χρήση του Java Web Start (JAWS). Με άλλα λόγια, οποιοσδήποτε διαθέτει σύνδεση στο διαδίκτυο και JAVA, με JDK 6 ή νεότερη έκδοση, μπορεί να έχει πρόσβαση και μπορεί να κάνει χρήση του εργαλείου αυτού.

4.2 Μη - Λειτουργικές Απαιτήσεις

Μπορεί να ειπωθεί, κατόπιν εμπειρικών μελετών ότι ως ελάχιστες απαιτήσεις για τη λειτουργία του εργαλείου σε ένα ικανοποιητικό βαθμό χωρίς δηλαδή καθυστερήσεις στις εργασίες ανάλυσης του κώδικα, παρουσίασης και αλληλεπίδρασης με τον γράφο και η αυτόματη παραγωγή περιπτώσεων ελέγχου, μπορούμε να θέσουμε τις εξής απαιτήσεις :

Ελάχιστες απαιτήσεις συστήματος :

- ✓ Επεξεργαστής 1.60 GHz
- ✓ Μνήμη τυχαίας προσπέλασης (RAM) 1GB
- ✓ Σκληρός δίσκος 20 GB
- ✓ Σύνδεση στο διαδίκτυο DSL με ταχύτητα μεταφόρτωσης (download speed) 1.5Mbps
- ✓ Εγκατεστημένη Java έκδοση 5 ή μεταγενέστερη

- ✓ Συσχέτιση αρχείων τύπου JNLP /MIME (JNLP file/ MIME Association) και άνοιγμά τους με Java Web Start.
- ✓ Λειτουργικό σύστημα : Microsoft Windows XP (service pack 2)
- ✓ Ή Ubuntu (Feisty Fawn) 7.04 ή μεταγενέστερη έκδοση

Προτεινόμενες απαιτήσεις συστήματος :

- ✓ Επεξεργαστής Intel® Core ™ 2 Duo T5800 @ 2.00GHz
- ✓ Μνήμη τυχαίας προσπέλασης (RAM) 3GB
- ✓ Σκληρός δίσκος 94,1 GB
- ✓ Σύνδεση στο διαδίκτυο DSL με ταχύτητα μεταφόρτωσης (download speed) 2Mbps
- ✓ Εγκατεστημένη Java έκδοση 6 ή μεταγενέστερη
- ✓ Συσχέτιση αρχείων τύπου JNLP /MIME (JNLP file/ MIME Association) και άνοιγμά τους με Java Web Start.
- ✓ Λειτουργικό σύστημα : Microsoft Windows Vista (service pack 1)
- ✓ Ή Ubuntu (Intrepid Ibex) 8.10 ή μεταγενέστερη έκδοση

Να σημειωθεί εδώ ότι οι απαιτήσεις αυτές λήφθηκαν από μηχανές στις οποίες έγινε η ανάπτυξη και ο έλεγχος του εργαλείου. Επίσης η ταχύτητα σύνδεσης στο διαδίκτυο είναι σημαντική κυρίως για τη μεταφόρτωση των απαιτούμενων αρχείων για το άνοιγμα της εφαρμογής μέσω Java Web Start η οποία με την ταχύτητα των ελάχιστων απαιτήσεων αναμένεται να διαρκέσει 3-4 λεπτά, και όχι τόσο για τη μεταφόρτωση αρχείων πηγαίου κώδικα στην εφαρμογή η οποία διαρκεί κλάσματα του δευτερολέπτου. Ακόμη, όσον αφορά τη μεταφόρτωση της εφαρμογής με το JAWS, όταν η διαθέσιμη ταχύτητα μεταφόρτωσης είναι μικρή, ενδέχεται να σταματήσει προσωρινά (stall) και να συνεχίσει στη συνέχεια.
Επίσης η εγκατεστημένη έκδοση Java, ως παλαιότερη συμβατή έκδοση θεωρείται η έκδοση 5 λόγω χρήσης generics στον κώδικα τα οποία δεν υποστηρίζονται σε προγενέστερες εκδόσεις.

Οι απαιτήσεις σκληρού δίσκου είναι πολύ μικρές αφού τα απαιτούμενα αρχεία και βιβλιοθήκες για να τρέξει η εφαρμογή απαιτούν μέγεθος λιγότερο από 20MB και τα αρχεία πηγαίου κώδικα που μεταφορτώνονται έχουν στην πλειοψηφία τους μέγεθος μικρότερο από 10KB, με τα μικρότερα να είναι 1KB και το μεγαλύτερο αρχείο 1000 γραμμών 103KB.

Τέλος, τα λειτουργικά συστήματα έχουν ελεγχθεί για συμβατότητα καθ'όλη την υλοποίηση του συστήματος και αναμένεται να τρέχουν χωρίς διαφορές, εκτός από το Look&Feel της εφαρμογής σε κάθε προαναφερθέν λειτουργικό σύστημα.

4.3 Λειτουργικές Απαιτήσεις

4.3.1. Εισαγωγή

Ανάμεσα στις λειτουργικές απαιτήσεις τις οποίες τέθηκαν στα αρχικά στάδια εκπόνησης της διπλωματικής εργασίας ήταν η ανάπτυξη ενός νέου εργαλείου έχοντας παράλληλα συλλέξει και συνενώσει τμήματα προηγούμενων εργασιών αναφορικά με ανάλυση κώδικα και παραγωγή περιπτώσεων ελέγχου. Επιπρόσθετα, μέρος των απαιτήσεων ήταν να αναβαθμιστούν ορισμένες από τις προϋπάρχουσες λειτουργίες των προγραμμάτων που συνενώθηκαν. Οι λειτουργίες αυτές αφορούν την αντικατάσταση της παλαιότερης βιβλιοθήκης δημιουργίας γράφων με καινούρια, η βελτίωση της γραφικής αλληλεπίδρασης της εφαρμογής, η αναβάθμιση της παλιάς βιβλιοθήκης των γενετικών αλγορίθμων με την τελευταία της έκδοση και επέκταση των δυνατοτήτων του εργαλείου αναφορικά με την βελτίωση της απόδοσης της εφαρμογής με αύξηση χρήσης πολυνηματικότητας. Τέλος, μεταξύ άλλων προσθηκών, η καλύτερη παρουσίαση αποτελεσμάτων στο γράφο και η πλαισίωση του εργαλείου με το Java Web Start και ενεργοποίησής του για χρήση από το διαδίκτυο.

4.3.2. Συνένωση εφαρμογών

Αναλυτικότερα, όταν ολοκληρώθηκε η συνένωση των προηγούμενων εργασιών σε ένα καινούριο εργαλείο που ξεκίνησε σε μια νέα βάση, τα κομμάτια των διπλωματικών εργασιών που χρησιμοποιήθηκαν ενσωματώθηκαν όπως είχαν. Αυτό σημαίνει ότι δεν έγινε καμιά αλλαγή στον τρόπο ανάλυσης του κώδικα, τον τρόπο δημιουργίας και παρουσίασης του κάθε γράφου και περιλήφθηκαν μόνο τα τμήματα που αφορούσαν την ανάλυση του κώδικα και παρουσίαση του γράφου και όχι την παραγωγή περιπτώσεων ελέγχου με τους γενετικούς αλγορίθμους.

4.3.3. Αναβάθμιση Λειτουργιών – Βιβλιοθηκών

Η παλιά βιβλιοθήκη δημιουργίας γράφων, η ibmgraph.jar , μια πεπαλαιωμένη βιβλιοθήκη παραγωγής απλών, λιτών γραφικά γράφων και με περιορισμένες δυνατότητες αλληλεπίδρασης χρήστη-γράφου, αντικαταστάθηκε με μια σύγχρονη, καλύτερα οργανωμένη, με περισσότερες δυνατότητες και βελτιωμένη οπτικά βιβλιοθήκη, την jung.jar. Η βιβλιοθήκη αυτή επιλέχθηκε μεταξύ άλλων, όπως η Jgraph, JgraphT και yWorks. Οι λόγοι επιλογής της συγκεκριμένης είναι η εγγύτητα αυτών που προσφέρει σε αυτά που ζητούσαν οι απαιτήσεις του εργαλείου, δηλαδή δημιουργία ενός βελτιωμένου οπτικά γράφου, με ευχάριστη αλληλεπίδραση, που να μην επιβαρύνει το σύστημα και να έχει καλή οργάνωση που να επιτρέπει τη συντήρηση και μελλοντική επέκταση του εργαλείου (maintanability, reusability), που να είναι ταυτόχρονα προϊόν ανοιχτού κώδικα. Ανάμεσα στις απαιτήσεις αυτές να σημειωθεί ότι ο γράφος είναι κατευθυνόμενος, με πιθανό μεγάλο βαθμό εισόδων σε αρκετούς κόμβους, και με την ανάγκη εμφάνισης ετικετών τόσο στις ακμές του γράφου (πχ συνθήκες ελέγχου ή true/false) όσο και μέσα στους κόμβους του. Για του λόγους αυτούς επιλέχθηκε η βιβλιοθήκη jung.jar έκδοση 1.7.6, ένα open-source project ευρέως χρησιμοποιούμενο το οποίο εξελίσσεται και βελτιώνεται συνεχώς.

Όταν έγινε η επιλογή της νέας βιβλιοθήκης, για να μπορεί να λειτουργήσει η εφαρμογή με

αυτήν έπρεπε να δημιουργηθούν νέες κλάσεις στο πακέτο του γράφου, δηλαδή νέες κλάσεις για τον γράφο, τον κόμβο και την ακμή. Οι κλάσεις αυτές, αντίστοιχα pGraph, pVertex, pEdge, αντικατέστησαν τις παλαιότερες tGraph, tVertex, tEdge. Αναγκαίες κρίθηκαν ακόμη οι προσθήκες στις κλάσεις των αναλυτών (parsers) και τροποποιήσεις στη διεπιφάνεια των μεθόδων για να μπορούν οι νέες κλάσεις να χρησιμοποιηθούν με παρόμοιο τρόπο όπως γινόταν με τις προηγούμενες εργασίες και να διατηρηθεί η ομοιογένεια μεταξύ των διαφορετικών αναλυτών που ενσωματώθηκαν. Σε πολλές περιπτώσεις τροποποιήθηκαν οι αναλυτές για να μπορούν να διεκπεραιώνουν τη λειτουργία τους χωρίς σφάλματα και εξίσου, αν όχι περισσότερο αποδοτικά. Αλλαγές αυτές περιλαμβάνουν χρήση έτοιμων μεθόδων που παρέχει η νέα βιβλιοθήκη, όπως η διάταξη του γράφου στον χώρο, τα σχήματα και χρώματα των κόμβων, οι πληροφορίες που περιέχουν οι κόμβοι και οι ακμές, οι δυνατότητες αλληλεπίδρασης του γράφου με το ποντίκι, η διαχείριση των συμβάντων (events), και οι διακοσμητές του γράφου (decorators). Σε άλλες περιπτώσεις, που η βιβλιοθήκη δεν παρείχε κάποιες συναρτήσεις, αυτές υλοποιήθηκαν ώστε να ανταποκρίνονται στις ανάγκες των αναλυτών, όπως έγινε για παράδειγμα με την κατά πλάτος αναζήτηση (Breadth First Search). Τροποποιήσεις έγιναν και σε βασικές λειτουργίες του γράφου όπως την εισαγωγή ακμών που ανήκουν σε διαφορετικό γράφο, τη διαγραφή κόμβων και ακμών που έχουν συνάφεια με άλλους κόμβους χωρίς να κατακερματίζεται ο γράφος και ταυτόχρονα χωρίς να καταπατούνται οι περιορισμοί (constraints) της βιβλιοθήκης, όπως επεξηγείται και σε επόμενο κεφάλαιο.

Όσων αφορά την αναβάθμιση της βιβλιοθήκης γενετικών αλγορίθμων, απαιτήθηκαν οι ακόλουθες εργασίες, η ενσωμάτωση της νέας έκδοσης της βιβλιοθήκης jgap.jar, και οι τροποποίηση των κλάσεων που την χρησιμοποιούν. Οι αλλαγές αυτές αφορούν κυρίως τον τρόπο που γίνεται η ρύθμιση (Configuration) για να καταστεί δυνατή η εκτέλεση των γενετικών αλγορίθμων. Στη νέα βιβλιοθήκη σύμφωνα και με την τεκμηρίωση πολλοί αλγόριθμοι τρέχουν αποδοτικότερα και οι διεπιφάνειες πολλών κλάσεων έχουν τροποποιηθεί καθιστώντας τον προϋπάρχοντα κώδικα μη συμβατό με τη νέα βιβλιοθήκη. Το γεγονός αυτό έκανε επιτακτική την ανάγκη των αλλαγών που αναφέρθηκαν, σε μεθόδους, κατασκευαστές κλάσεων και στις διεπιφάνειες και σε αρκετές περιπτώσεις στη σειρά με την οποία γίνονται οι ρυθμίσεις των τελεστών των γενετικών αλγορίθμων,

ρυθμίσεις του πληθυσμού, των δειγμάτων-χρωμοσωμάτων, κ.ο.κ. Όταν οι αλλαγές ολοκληρώθηκαν ακολούθησε ο έλεγχος για τη συνέπεια των αποτελεσμάτων της παραγωγής περιπτώσεων ελέγχου με τα αποτελέσματα όπως αυτά παρουσιάζονται στις προηγούμενες εργασίες με την παλαιότερη βιβλιοθήκη των γενετικών αλγορίθμων ώστε να είμαστε σε θέσεις να διαβεβαιώσουμε ότι το εργαλείο όχι μόνο δεν έχει χάσει τμήμα της λειτουργικότητάς του, αλλά την αύξησε σε συνδυασμό με την μεγαλύτερη αποδοτικότητα που προσφέρει η νέα βιβλιοθήκη. Έτσι, η παλιά βιβλιοθήκη διαγράφηκε από την εφαρμογή και η αναβάθμισή της ήταν επιτυχής.

Στα πλαίσια της συντηρισιμότητας (maintenability) και της δυνατότητας επαναχρησιμοποίησης (reusability) της εφαρμογής, και για να μπορέσει να διευκολύνει το έργο της πιθανής επέκτασής της, έχει δημιουργηθεί από αυτοματοποιημένο εργαλείο μοντελοποίησης του Netbeans ένα σύνολο από εύκολα πλοηγίσιμες σελίδες με συνδέσεις υπερκειμένου (hypertext links) που έχουνε σκοπό, να παρέχουν πληροφορίες για τον τρόπο με τον οποίο οι κλάσεις συνδέονται μεταξύ τους, τις μεθόδους που έχουν, τους κατασκευαστές των κλάσεων και τα πεδία τους. Ακόμη δείχνει τις εξαρτήσεις της κάθε κλάσης και κάθε πακέτου, τις μεταξύ των κλάσεων , μεταξύ πακέτων αλλά και εξαρτήσεις μεταξύ κλάσεων και πακέτων. Μπορεί να πει κανείς ότι είναι κάτι σαν το API (Application Interface) της εφαρμογής χωρίς την τεκμηρίωση για τη διεργασία που επιτελεί η κάθε μέθοδος. Η μοντελοποίηση αυτή είναι διαθέσιμη στον σύνδεσμο :

http://www.cs.ucy.ac.cy/~cs04pp2/report/index.html

4.3.4. Διαδικτυακή λειτουργία με JAWS

Σε επόμενο στάδιο, εφόσον είχε επιτευχθεί ενοποίηση του τμημάτων της εργασίας στο νέο σκελετό που κατασκευάστηκε και η συνένωση των πακέτων των γράφων σε ένα ενιαίο πακέτο, δόθηκε έμφαση στην ενεργοποίηση του Java Web Start (JAWS). Στο σημείο αυτό πρέπει να σημειωθεί ότι από τις πρώτες προεργασίες που έγιναν πριν την υλοποίηση του εργαλείου ήταν ο σχεδιασμός του, και ανάμεσα στις προδιαγραφές, η επιλογή και κατόπιν χρήση ενός τρόπου που θα επέτρεπε στο εργαλείο να είναι διαθέσιμο στους χρήστες μέσω

του διαδικτύου. Η επιλογή ήταν ανάμεσα σε Java Applet και του νεότερου Java Web Start. Έπειτα από έρευνα και ζυγοστάθμισης των πλεονεκτημάτων και μειονεκτημάτων και των δυο, επιλέχθηκε το Java Web Start.

Οι λόγοι για την επιλογή αυτή είναι οι εξής :

- Το JAWS είναι μια τεχνολογία λογισμικού που ενσωματώνει εύκολη μεταφερσιμότητα (portability) εφαρμογών με χρήση φυλλομετρητή (browser). Σε αντίθεση με τα applets, δεν χρειάζεται τον φυλλομετρητή να είναι ανοιχτός καθώς τρέχει, αφού η εφαρμογή που τρέχει με JAWS ανοίγει σε νέο παράθυρο (φόρμα εφαρμογής), ακριβώς όπως θα έτρεχε και τοπικά (locally) σε μια μηχανή χωρίς χρήση του Java Web Start.
- Παρέχει μεγάλες ευκολίες στο χρήστη αφού το μόνο που έχει να κάνει για να κατεβάσει και να ανοίξει την εφαρμογή είναι ένα κλικ (one click install).
- Το JAWS αναλαμβάνει να κατεβάζει τα αρχεία από τον εξυπηρετητή (server) που απαιτούνται για να τρέχει η εφαρμογή (executable jars, κλπ.)
- Επιτρέπει τροποποίηση του κώδικα για την διαδικασία εγκατάστασης (επιλογή αρχείων, δικαιωμάτων (permissions), αλλαγή προτεραιοτήτων για μεταφόρτωση των αρχείων (download eager / lazy).
- Τα applets πρέπει να μοιράζονται τη μνήμη τυχαίας προσπέλασης (RAM) με έναν απαιτητικό και συχνά «λαίμαργο» φυλλομετρητή, ενώ το JAWS τρέχει ανεξάρτητα, με το java.exe Hotspot.
- Αυτόματη ενημέρωση αλλαγών της εφαρμογής (automatic update). Αυτό σημαίνει ότι αν η εφαρμογή έχει αλλάξει, τότε θα κατεβάσει την νέα έκδοση της αυτόματα και θα την φορτώσει. Με άλλα λόγια, όταν ο χρήστης επιλέξει να τρέξει την εφαρμογή για πρώτη φορά, θα φορτωθεί η εφαρμογή και μετά θα τρέξει. Αν στη συνέχεια κλείσει την εφαρμογή (ακόμη και τον φυλλομετρητή) και την ξανα-ανοίξει, τότε δε χρειάζεται να ξανακατεβάσει την εφαρμογή από τον εξυπηρετητή αφού είναι ήδη αποθηκευμένη, αλλά ανοίγει άμεσα. Σε περίπτωση όμως που έχει αλλαχθεί η εφαρμογή ή έχουν γίνει ανανεώσεις (updates) τότε θα φορτώσει εκ νέου την νέα εφαρμογή πριν ανοίξει.
- Μηχανισμούς αυτόματης επιλογής και χρήσης του βέλτιστου JRE (Java Runtime

Environment), το εκτελέσιμο javaw.exe και ένα μηχανισμό για δέσμευση μνήμης στο δίσκο που δεν αναγκάζει το χρήστη να βρει μοναδικό όνομα για αυτό.

- Σε αντίθεση με τα applets, θα αποθηκευτούν στην κρυφή μνήμη (cached) του πελάτη (client) όλα τα .jars της εφαρμογής και δεν διαγράφονται ποτέ αυτόματα. Έτσι, έχοντας ήδη κατεβάσει την εφαρμογή, δεν χρειάζεται σύνδεση για να τρέξει.
- Για χρήστης με σύνδεση μέσω modem – γραμμής τηλεφώνου (dial-up connection), το JAWS λειτουργεί έξυπνα. Αν υπάρχει σύνδεση, τότε ελέγχει για ανανεώσεις (updates) και τις κατεβάζει αν χρειάζονται. Αν δεν υπάρχει σύνδεση την τρέχουσα στιγμή, τότε δεν επιχειρεί να δημιουργήσει νέα σύνδεση – κλήση αλλά χρησιμοποιεί την αποθηκευμένη, παλιά σύνδεση της εφαρμογής και όταν υπάρχει σύνδεση στο διαδίκτυο, τότε θα ελέγξει για ενημερώσεις.
- Τέλος, μπορεί να ειπωθεί με ασφάλεια ότι διευκολύνει τον προγραμματιστή στο γράψιμο κώδικα, κάνοντας το εύκολα συντηρήσιμο. Αυτό ισχύει γιατί ο προγραμματιστής δεν χρειάζεται να γράψει δυο εκδόσεις του ίδιου προγράμματος σε διαφορετικές βάσεις (code bases) ,μια για την αυτή καθαυτή εφαρμογή που τρέχει ως Java Application και είναι ανεξάρτητη (standalone application) και μια για το Java Web Start. Έτσι, εάν σε περίπτωση που το JAWS αποσυρθεί από την χρήση ή αντικατασταθεί από κάτι εντελώς διαφορετικό, ο προγραμματιστής μπορεί απλά να συνεχίσει την ανάπτυξη ή συντήρηση του λογισμικού χωρίς ξοδέψει χρόνο για να τροποποιήσει τον κώδικα.
- Το JAWS προσφέρει ασφάλεια, ελέγχοντας για τις ψηφιακές υπογραφές (digital signatures) της εφαρμογής και των βιβλιοθηκών που χρειάζεται (όλα τα jars πρέπει να έχουνε την ίδια υπογραφή, από τον ίδιο κατασκευαστή, εκτός αν ανήκουν σε άλλους οργανισμούς και έχουνε δικές του υπογραφές που μπορούν να εμπιστευθούν). Ακόμη και αν υπάρχουν επεκτάσεις (extensions) σε βιβλιοθήκες, αυτές πρέπει να φέρουν αποδεκτές και ασφαλείς υπογραφές. Μπορεί ωστόσο μια εφαρμογή JAWS να μην φέρει υπογραφή. Αυτό όμως σημαίνει πως οι εφαρμογές αυτές έχουν αρκετούς περιορισμούς. Όπως ότι δεν μπορούν να έχουν πρόσβαση στον τοπικό σκληρό δίσκο, δεν επιτρέπονται ιθαγενείς βιβλιοθήκες (native libraries), επιτρέπονται μόνο συνδέσεις από τον host που κατεβαίνουν όλα τα jars,

και τέλος , δεν εγκαθίσταται ο ελεγκτής ασφάλειας (security manager).

Αξίζει ωστόσο να αναφερθούν και μερικά μειονεκτήματα της χρήσης του JAWS. Αυτά βέβαια διαφέρουν από έκδοση ε έκδοση και ενδεχομένως ορισμένα από αυτά να εξαλειφθούν στο εγγύς μέλλον.

- Το JAWS καθυστερεί εξίσου πολύ κατά τη διάρκεια μεταφόρτωσης (download) των ενημερώσεων όσο και κατά τη διάρκεια μεταφόρτωσης ολόκληρης της εφαρμογής στο ξεκίνημα.
- Απαιτεί ο πελάτης (client) να έχει JRE εγκατεστημένο στο μηχάνημά του (όπως και στα applets) και επιπρόσθετα απαιτεί application/x-java-jnlp-file MIME τύπου συσχέτιση (association) με το jaws.exe στις ρυθμίσεις του φυλλομετρητή. Ακόμη, συσχέτιση των αρχείων .jnlp τύπου με το javaws.exe
- Όσον αφορά τον εξυπηρετητή (server), απαιτείται τα .jnlp αρχεία με application/x-java-jnlp-file MIME τύπο.
- Ιδανικά , χωρίς όμως να είναι απαραίτητο, μπορεί να εγκατασταθεί στον εξυπηρετητή JNLP πρωτόκολλο για να εξυπηρετούνται οι αλλαγές στα .jar αρχεία πιο αποτελεσματικά.

Συγκρίνοντας τις δυο αυτές τεχνολογίες και λαμβάνοντας υπόψη όλα τα χαρακτηριστικά τους, ζυγίζοντας πλεονεκτήματα και μειονεκτήματα προτιμήθηκε η τεχνολογία του Java Web Start, μια νέα , πρακτική, πολύ αποδοτική και φιλική προς το χρήστη η οποία φαίνεται να είναι και πιο εξελίξιμη και αναμενόμενη να βελτιωθεί περεταίρω στο μέλλον. Με την επιλογή αυτή ακόμη, δεν τίθεται σε ρίσκο η περαιτέρω ανάπτυξη ή συντήρηση της εφαρμογής.

Έτσι, στο σημείο αυτό, έχοντας επιλέξει το εργαλείο λογισμικού JAWS για τη χρήση της εφαρμογής από το διαδίκτυο, έπρεπε να τροποποιηθεί και ο σχεδιασμός για την ανάπτυξη ορισμένων τμημάτων του. Επομένως ορισμένες προσθήκες έγιναν αναγκαίες για να μπορεί να δουλέψει από το διαδίκτυο. Όπως προαναφέρθηκε, για να λειτουργήσει η εφαρμογή, χρειάζεται να πάρει ως είσοδο ένα (ή περισσότερα) αρχεία πηγαίου κώδικα γραμμένα σε

java τα οποία και στη συνέχεια θα αναλύσει και ελέγξει.
Τα αρχεία αυτά, που σε πρότερο στάδιο, όταν η εφαρμογή έτρεχε τοπικά, βρίσκονταν σε ένα ξεχωριστό πακέτο, το testfiles packet, στη συνέχεια κρίθηκε σκόπιμο να τοποθετηθούν στον κατάλογο public_html, στο προσωπικό μου domain του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, όπως άλλωστε και η διανομή (distribution) της εφαρμογής για το Java Web Start, η απλή ιστοσελίδα (launch.html) μέσω της οποίας ανοίγει η εφαρμογή με το JAWS, και το .jnlp αρχείο. Για να λειτουργήσει η εφαρμογή με το JAWS χωρίς προβλήματα, θα πρέπει να τηρεί τους περιορισμούς που θέτει το Java Web Start. Περιορισμοί όπως το ότι όλες οι βιβλιοθήκες που απαιτούνται για να τρέξει η εφαρμογή να περιλαμβάνονται στον ίδιο κατάλογο και να είναι σε μορφή .jar και όχι σαν αρχεία συμπίεσης όπως .zip ή .rar, κ.οκ. Ακόμη θα πρέπει να έχουνε μια έγκυρη ψηφιακή υπογραφή όλα τα πακέτα τις εφαρμογής επειδή απαιτείται πρόσβαση στη μηχανή που θα τρέξει η εφαρμογή για ανάγνωση απαιτούμενων αρχείων, για μεταφόρτωση και δημιουργία νέων αρχείων (όπως τα αρχεία πηγαίου κώδικα για έλεγχο). Κάτι τέτοιο δεν θα ήταν εφικτό να γίνει αν δεν υπήρχε κάποια ψηφιακή υπογραφή αφού όπως θα δούμε αργότερα στα παραδείγματα εκτέλεσης σεναρίων της εφαρμογής πριν ξεκινήσει να τρέχει ως εφαρμογή JAWS θα πρέπει ο χρήστης να αποδεχθεί ότι εμπιστεύεται την εφαρμογή και την υπογραφή του συγγραφέα της για να μπορεί να ξεκινήσει η εκτέλεση του εργαλείου. Επιπρόσθετα, θα πρέπει να προσεχθεί ότι το .jnlp αρχείο είναι ορθό και περιέχει τη σωστή διεύθυνση της εφαρμογής, αφού αυτή κατά προεπιλογή τίθεται να δείχνει στο εκτελέσιμο αρχείο της εφαρμογής που βρίσκεται τοπικά στη μηχανή που δημιουργείται η εφαρμογή. Έτσι θα πρέπει να τροποποιηθεί για να δείχνει στο σωστό αρχείο στο σωστό απομακρυσμένο κατάλογο του διαδικτυακού τόπου. Τέλος, θα πρέπει να ελεγχθεί ότι ο κατάλογος αυτός έχει τα απαιτούμενα δικαιώματα χρηστών και ομάδων χρηστών ούτως ώστε να μπορεί να εκτελεστεί (πχ rwx-rx-x). Να σημειωθεί εδώ ότι δημιουργήθηκε και είναι διαθέσιμο στη σελίδα αυτή ένα απλό API (Application Interface) με σκοπό να παρέχει περισσότερες πληροφορίες σε χρήστες και προγραμματιστές και να διευκολύνει την περαιτέρω ανάπτυξη του συστήματος αν και όταν αυτό καταστεί αναγκαίο. Το απλό αυτό API περιλαμβάνει όλες τις

- κλάσεις,
- διεπιφάνειες (interfaces),

- τύπους δομών,
- πακέτα και όλα τα στοιχεία που συνδυάζονται για να αποτελέσουν αυτό το ολοκληρωμένο εργαλείο, όπως όλες οι διασυνδέσεις μεταξύ κλάσεων και πακέτων.

Όσον αφορά τις περιεχόμενες κλάσεις, οι πληροφορίες που εμφανίζονται είναι οι (δημόσιες) μέθοδοι που είναι διαθέσιμες, οι ιδιότητες (attributes) της κλάσης, οι κατασκευαστές (constructors), τα χαρακτηριστικά της κλάσης (αν είναι τελική (final), αν είναι ορατή στις άλλες κλάσεις , αφαιρετική (abstract) , κλπ). Ακόμη, για κάθε κλάση συμπεριλαμβάνονται στην αρχή τις σελίδας και όλες οι γνωστές συσχετίσεις (all known associations) με άλλες κλάσεις από το ίδιο ή διαφορετικό πακέτο. Στον χώρο αυτό λοιπόν τοποθετήθηκαν τα δοκιμαστικά αρχεία για έλεγχο, κάτω από τον κατάλογο testfiles/, και συμπεριλαμβανομένου ενός επιπλέον αρχείου με πληροφορίες για τα αρχεία που υπάρχουν στον φάκελο αυτό (ονόματα αρχείων), το testfiles_index. Το αρχείο testfiles_index χρησιμοποιείται από την αρχή στην αρχή, όπως θα δούμε παρακάτω, όταν ο χρήσης επιλέξει να φορτώσει τη λίστα με τα αρχεία και να επιλέξει ένα από αυτά. Εκείνη τη στιγμή μεταφορτώνεται το αρχείο testfiles_index με χρήση http (πρωτόκολλο μεταφοράς υπερ-κειμένου) και διαβάζονται έτσι τα περιεχόμενα του φακέλου /testfiles, και τα ονόματα των διαθέσιμων αρχείων προβάλλονται στη λίστα, στο πρώτο βήμα (step)του Οδηγού Ανοίγματος-Φόρτωσης Αρχείου (Load Wizard). Έτσι ο χρήστης μπορεί να επιλέξει ένα από αυτά τα αρχεία και να μεταφορτώσει μόνο αυτό που χρειάζεται και όχι όλο τον κατάλογο με τα αρχεία ελέγχου, κάτι που θα προκαλούσε καθυστέρηση, άσκοπη χρήση του bandwidth και της μνήμης της μηχανής. Αν, φυσικά στη συνέχεια επιθυμεί να χρησιμοποιήσει κάποιο άλλο αρχείο, τότε με την ίδια διαδικασία μπορεί να το επιλέξει και να το μεταφορτώσει. Τα περιεχόμενα του αρχείου testfiles_index είναι το αποτέλεσμα εκτέλεσης της εντολής ls . | sort > testfiles_index
στη γραμμή εντολών (command-line) του unix, αν εκτελεστεί στο κατάλογο testfiles/ . Περιέχει δηλαδή όλα τα περιεχόμενα του καταλόγου testfiles/ σε αλφαβητική σειρά, αποτυπωμένα στο αρχείο testfiles_index, που είναι ένα αρχείο κειμένου.
Τα αρχεία αυτά είναι μερικά κλασικά προβλήματα σε κώδικα java, όπως ταξινόμηση, εύρεση, ταξινόμηση τριγώνων, χρήση constructors, χρήση κληρονομικότητας και μερικά άλλα απλά και πιο σύνθετα και εκτενή προγράμματα που γράφτηκαν για τον έλεγχο της εφαρμογής. Ενδεικτικά, μερικά από αυτά τα αρχεία πηγαίου κώδικα είναι τα παρακάτω :

```
BinarySearch.java
BouncySolidBall.java
BubbleSort.java
BucketSort.java
Car.java
Factorial.java
FExam06.java
Fibonacci.java
Fighter.java
FindMaximum.java
Foo1000Line.java
Foo1000LineSimple.java
Foo11.java
Foo2000Complex.java
Foo2.java
Foo3.java
Foo4.java
Foo5.java
Foo6.java
Foo7.java
Foo7Run.java
FooArray.java
FooComplex.java
FooFighter.java
FooFor.java
Foo.java
FooTest01.java
FooTest02.java
FooTest03.java
FooTest04.java
FooToTestWithDG.java
FooWhile.java
GeometricSeriesSumClosedFormExpr.java
GeometricSeriesSumHornerRule.java
GeometricSeriesSum.java
Goo1000M14.java
InsertionSort.java
Moo1.java
PowerExample.java
PrefixSums.java
Quadratic.java
SumExample.java
SumHornerRuleExample.java
Test0102.java
Test01.java
Test02.java
Test03.java
Test04.java
Test05.java
Test06.java
Test07.java
Test08.java
Test12.java
Test15.java
Test25.java
Test26.java
```

```
Test28.java
Test29.java
Test30.java
TestFile01.java
TestFile02.java
TestFile03.java
TestFile04.java
TestFile05.java
TestFile06.java
TriangClassification.java
TriangleClassificationold.java
TriangleClassificationRunold.java
```

Για τη μεταφόρτωση των αρχείων αυτών, όπως και του αρχείου testfiles_index, με τη χρήση του πρωτοκόλλου http , δημιούργησα ένα νέο πακέτο, το http, το οποίο περιέχει την κλάση FileDownloader.java , η οποία είναι μια απλή υλοποίηση μεταφοράς αρχείων με το πρωτόκολλο http και διαθέτει εύκολες στη χρήση μεθόδους, όπως για παράδειγμα η μέθοδος public static void download (String address) η οποία μεταφορτώνουν (download) το επιθυμητό αρχείο δίνοντας απλά τη διεύθυνση του αρχείου. Για παράδειγμα,
FileDownloader.download(http://www.cs.ucy.ac.cy/~cs04pp2/testfiles_index.txt);
Θα μεταφορτώσει το αρχείο που περιέχει τα ταξινομημένα περιεχόμενα του καταλόγου testfiles/ .

4.3.5. Βελτιωμένη αλληλεπίδραση με τον χρήστη

Στα πλαίσια των απαιτήσεων για βελτίωση της γραφικής αλληλεπίδρασης ακολουθήθηκε μια σειρά ενεργειών, όπως η οργάνωση των κουμπιών σε ομάδες λειτουργιών, χρήση παρόμοιων φορμών για συναφής λειτουργίες, διευκολύνοντας τον χρήστη στον κατατοπισμό του, βελτιωμένη παρουσίαση αποτελεσμάτων στα πρότυπα των αρχών της αλληλεπίδρασης ανθρώπου-υπολογιστή, όπως θα δούμε πιο κάτω. Τα μενού, οι φόρμες και γενικότερα το γραφικό πλαίσιο αντικαταστάθηκε με τα ελαφρύτερα, αποδοτικότερα και σύγχρονα εργαλεία που παρέχει η java, τα συστατικά του swing (swing components). Με την αποτελεσματικότερη χρήση πολλαπλών νημάτων μπορεί πλέον ο χρήστης να εκτελεί πολλές εργασίες παράλληλα και χωρίς να καθυστερεί αναμένοντας αποτελέσματα

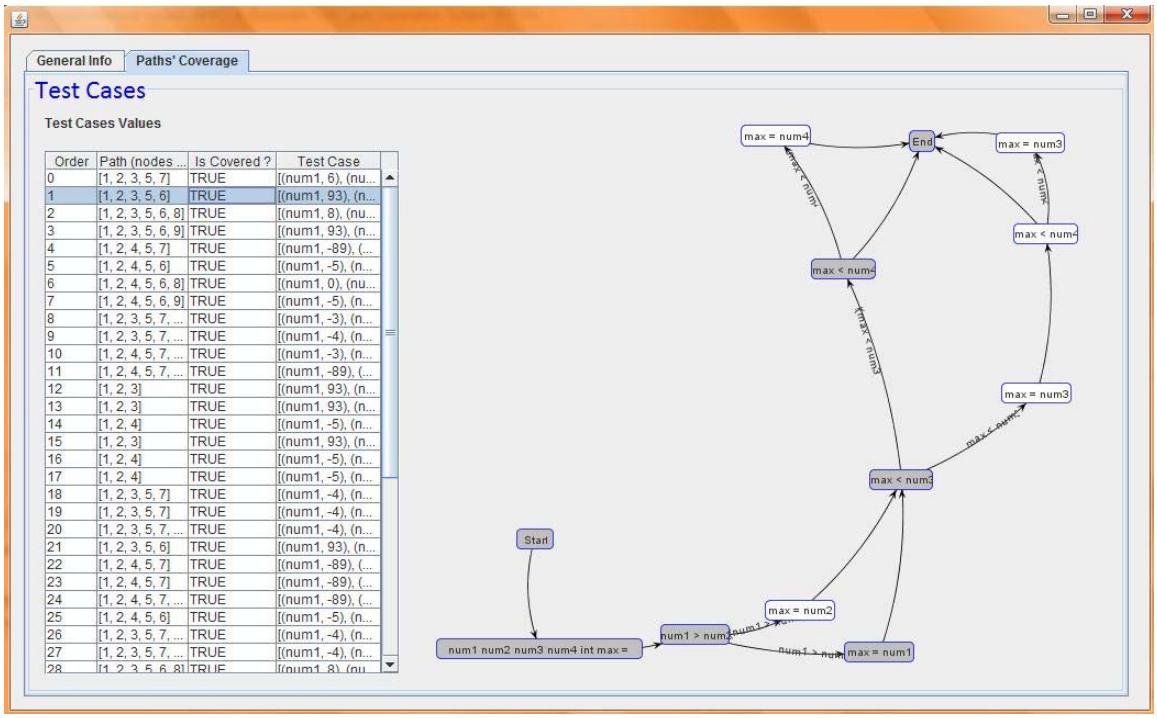
προηγούμενων εργασιών. Ακόμη, όταν μια διεργασία τείνει να γίνεται χρονοβόρα, η εφαρμογή δεν παγώνει, αποκλείοντας το χρήστη από κάθε είδος αλληλεπίδρασης με αυτήν αλλά του επιτρέπει να πλοηγείται στα μενού, να αλληλεπιδρά με τους γράφους και να επεξεργάζεται αποτελέσματα ολοκληρωμένων διεργασιών, ενώ μπορεί ταυτόχρονα να ενημερώνεται ανά πάσα στιγμή για την πρόοδο και το έργο που εκτελείται ανά πάσα στιγμή από την τρέχουσα χρονοβόρα διεργασία. Ο τρόπος εμφάνισης των αποτελεσμάτων των περιπτώσεων ελέγχου έχει αλλάξει επίσης και έχει γίνει περισσότερο διαδραστικός, αφού όπως επεξηγείται αναλυτικότερα στη συνέχεια, για κάθε περίπτωση ελέγχου που έχει παραχθεί, μπορεί ο χρήστης να δει γραφικά στον γράφο να εμφανίζεται το υπομονοπάτι που έχει καλυφθεί. Η πλοήγηση του χρήστη για την επιλογή του αρχείου που θέλει να ελέγξει, τη μεταφόρτωσή του από την απομακρυσμένη τοποθεσία, και την επιλογή του γράφου που θέλει να δημιουργήσει έχει γίνει πιο εύκολη με την βοήθεια του οδηγού Ανοίγματος-Μεταφόρτωσης αρχείου (Load Wizard) ενώ παράλληλα παρέχει όλες τις απαραίτητες πληροφορίες που χρειάζεται ο χρήστης σε μόλις τρία βήματα.

Έχοντας ενσωματώσει τις προαναφερθείσες αλλαγές, ακολούθησαν αλλαγές στην διεπιφάνεια της εφαρμογής οι οποίες, ακολουθώντας τις αρχές τις Αλληλεπίδρασης Ανθρώπου-Υπολογιστή (Human-Computer Interaction) στοχεύουν στην διευκόλυνση του χρήστη στην πραγματοποίηση των εργασιών του εν μέσω μιας φιλικής προς τον χρήστη εφαρμογής. Αρχές όπως ένα ευχάριστο οπτικά μενού, με κουμπιά και επιφάνειες προσεκτικά στοιχισμένες και τοποθετημένες στο γραφικό περιβάλλον της εφαρμογής και πληροφορίες αναφορικά με την λειτουργία του καθενός που είναι διαθέσιμες στον χρήστη μέσα από το ίδιο το εργαλείο με βοηθητικά μηνύματα κειμένου (tool-tip text) που εμφανίζονται με την κίνηση του ποντικιού πάνω από το κάθε κουμπί (mouse over) ή περιοχή που εμφανίζει ένα αντικείμενο ή μενού που επιτελούν μια λειτουργία. Τα μηνύματα αυτά είναι απλά, σύντομα και κατατοπιστικά και σε συνδυασμό με σχόλια σε μενού ρυθμίσεων και αυτοεπεξηγηματικές εικόνες αναφορικά με τις λειτουργίες των κουμπιών συντελούν στον προσανατολισμό του χρήστη και τη διακριτική καθοδήγησή του στην εργασία που θέλει να πραγματοποιήσει χωρίς να τον κουράζει με μεγάλο όγκο πληροφοριών και πολύπλοκα εκτενή αρχεία βοήθειας.

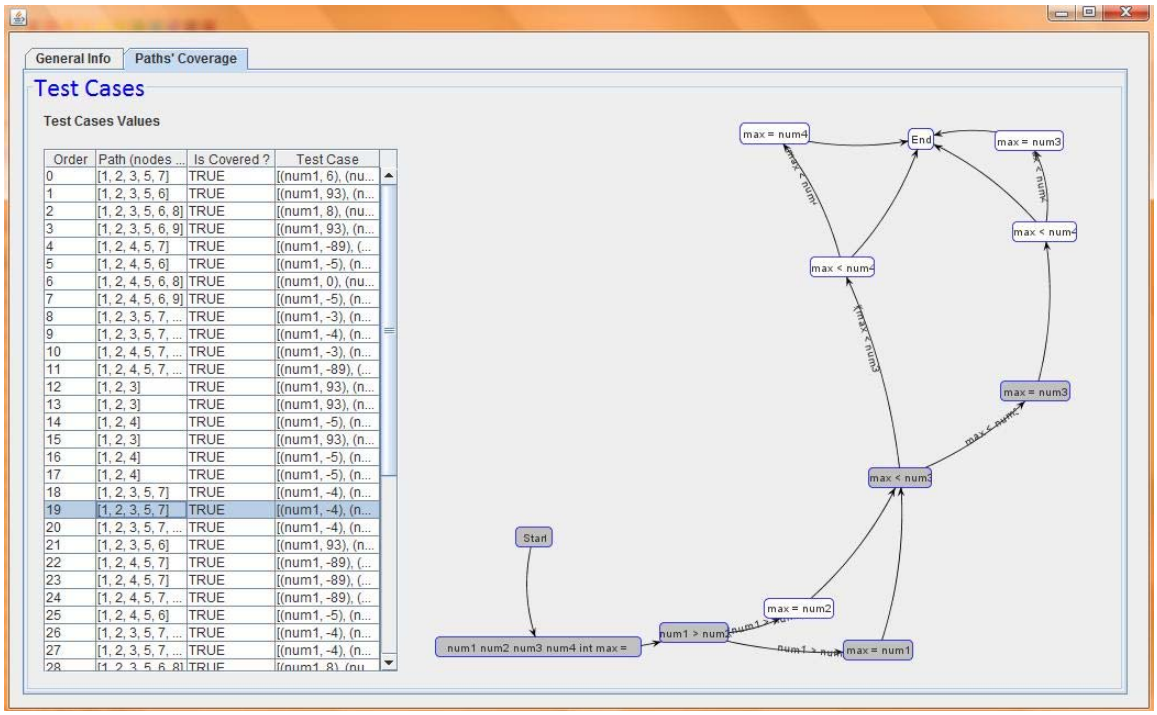
Τέλος, στα πλαίσια μιας αποδοτικής και εύχρηστης διεπιφάνειας εργασίας, αποκρύπτονται πληροφορίες στον χρήστη οι οποίες δεν χρειάζεται να γνωρίζει, χωρίς να επιβαρύνεται έτσι

με επιπλέον πληροφορίες όταν αυτός δεν τις χρειάζεται. Για παράδειγμα, στην εκκίνηση της εφαρμογής τα κουμπιά των λειτουργιών που θα είναι διαθέσιμα θα είναι ο Οδηγός Ανοίγματος-Μεταφόρτωσης Αρχείου, και οι ρυθμίσεις που αφορούν την εφαρμογή, όπως είναι και λογικό άλλωστε, αυτά είναι τα πρώτα αναμενόμενα βήματα χρήσης του εργαλείου από τον χρήστη. Όταν γίνουν τα βήματα αυτά, ειδικά όταν ολοκληρωθεί η μεταφόρτωση του αρχείου και η δημιουργία του γράφου, τότε το εργαλείο έχει στη διάθεση του μια επιπλέον πληροφορία : το είδος του γράφου που έχει επιλέξει να δημιουργήσει. Οπότε στο σημείο αυτό, ανάλογα με τον επιλεγμένο γράφο, ένα από τα κουμπιά που επιτελούν την παραγωγή περιπτώσεων ελέγχου με τη βοήθεια γενετικών αλγορίθμων θα ενεργοποιηθεί. Δηλαδή αν έχει επιλεγεί ο γράφος ροής ελέγχου, τότε θα ενεργοποιηθεί το αντίστοιχο κουμπί για τον έλεγχο του γράφο ροής δεδομένων, αν έχει επιλεγεί ο γράφος ροής ελέγχου τότε το κουμπί για τον έλεγχο που βασίζεται στο γράφο ροής δεδομένων και τα μονοπάτια που έχουν εξαχθεί με βάση το κριτήριο k-tuples. Αν επιλεγεί ο γράφος απαιτήσεων, τότε κανένα κουμπί δεν θα ενεργοποιηθεί. Με τον όρο ενεργοποίηση του κουμπιού εννοούμε ότι το κουμπί (Jbutton) μεταβαίνει από την αρχική κατάσταση a_button.isEnabled = false στην κατάσταση a_button.isEnabled = true. Με τον ίδιο τρόπο, όταν η παραγωγή και έλεγχος των περιπτώσεων ελέγχου του αντίστοιχου γράφου έχει ολοκληρωθεί, τότε ενεργοποιούνται τα αντίστοιχα κουμπιά με τα οποία μπορεί ο χρήστης να δει τα αντίστοιχα αποτελέσματα, δηλαδή τις περιπτώσεις ελέγχου που παράχθηκαν, τις ακμές ή τα μονοπάτια που στόχευαν να καλύψουν και αν τα κάλυψαν και να δει γραφικά το ισοδύναμο αποτέλεσμα πάνω στον γράφο με χρωματισμένες τις ακμές/μονοπάτια που καλύφθηκαν (γκρι) και με διαφορετικό χρώμα (άσπρο) αυτές που δεν καλύφθηκαν. Τα αποτελέσματα αυτά αποτυπώνονται στον γράφο και αλλάζουν κάθε φορά που ο χρήστης επιλέγει μια διαφορετική περίπτωση ελέγχου από αυτές που έχουν παραχθεί. Βλέπουμε λοιπόν ότι ακόμη μια τεχνική αλληλεπίδρασης ανθρώπου-υπολογιστή τηρήθηκε που θέλει τον χρήστη να μπορεί να βλέπει γραφικά κάτι (τα αποτελέσματα) και όχι μόνο μια ακολουθία από αριθμούς και τιμές. Όπως βλέπουμε παρακάτω στα παρατιθέμενα σχήματα, έχουμε τα αποτελέσματα παραγωγής και ελέγχου περιπτώσεων ελέγχου που βασίζονται στην εξαγωγή μονοπατιών από έναν γράφο ροής δεδομένων, και πως εμφανίζονται γραφικά στον γράφο, όπως περιγράφει προηγουμένως. Ο χρήστης μπορεί δηλαδή να αλληλεπιδράσει με τα αποτελέσματα και τον γράφο στον οποίο παρουσιάζονται. Έχουμε

λοιπόν στο (Σχήμα 4.1) επιλεγμένη μια περίπτωση ελέγχου και την αποτύπωση της κάλυψης που είχε σε ένα μονοπάτι στον γράφο, ενώ στο (Σχήμα 4.2) βλέπουμε τι συμβαίνει όταν ο χρήστης επιλέξει από τον πίνακα περιπτώσεων ελέγχου μια διαφορετική ομάδα τιμών που στοχεύει στην κάλυψη ενός μικρότερου μονοπατιού στον γράφο, το οποίο καλύπτεται.



Σχήμα 4.1 : Επιλογή περίπτωσης ελέγχου 1 και απεικόνιση αντίστοιχου μονοπατιού



Σχήμα 4.2 : Επιλογή περίπτωσης ελέγχου 19 και απεικόνιση αντίστοιχου μονοπατιού

4.3.6. Διαχείριση πολυνηματικότητας (Multithreading)

Ακολουθώντας μια ακόμη τεχνική βελτίωσης της αλληλεπίδρασης μεταξύ ανθρώπου-υπολογιστή, που θέλει τον χρήστη να είναι συνεχώς ενήμερος για την τρέχουσα κατάσταση της εφαρμογής, αλλά και τη γνώση ποιας διεργασίας εκτελείται από την εφαρμογή προστέθηκαν ενημερωτικές μπάρες προόδου των διεργασιών (progress bars) για τα πιο χρονοβόρα τμήματα του εργαλείου. Όσον αφορά τις εργασίες που είναι αρκετά χρονοβόρες είναι καλό να φαίνεται σε πιο στάδιο αυτές βρίσκονται, για αυτό το λόγο στην περίπτωση αυτή προστέθηκαν παράθυρα διαλόγου που δείχνουν ποιες εργασίες εκτελέστηκαν, ποιες εκκρεμούν και ποιες θα ακολουθήσουν στη συνέχεια.
Πιο συγκεκριμένα, οι λειτουργίες που αναφέρονται και χαρακτηρίζονται ως δυνητικά

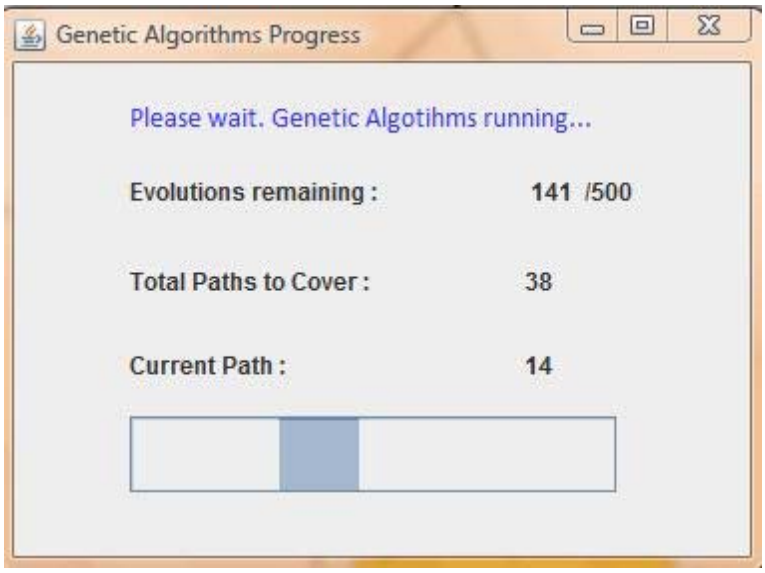
χρονοβόρες, είναι οι εξής :

Αυτές της εκτέλεσης των περιπτώσεων ελέγχου με τη βοήθεια γενετικών αλγορίθμων, οι εξελίξεις (evolutions) των πληθυσμών (population) και η εκτέλεση του προγράμματος και η αξιολόγηση των αποτελεσμάτων με τη συνάρτηση αξιολόγησης (fitness evaluation function) τόσο αυτά για τον γράφο ροής δεδομένων όσο και για το γράφο ροής ελέγχου. Σε αυτές τις περιπτώσεις, ανάλογα φυσικά με την πολυπλοκότητα και την έκταση του πηγαίου κώδικα που εξετάζεται, ενδέχεται ορισμένα από αυτά τα προγράμματα να καθυστερούν σημαντικά στις παραπάνω εργασίες.

Μια άλλη περίπτωση που μπορεί να αποδειχθεί χρονοβόρα με χρήση εκτενούς πηγαίου κώδικα είναι, σε αύξουσα σειρά απαίτησης χρόνου, η μεταφόρτωση του αρχείου από την απομακρυσμένη τοποθεσία (σε μικρότερο βαθμό), η ανάλυση του κώδικα (parsing) (σε αρκετά μεγαλύτερο βαθμό), το χτίσιμο του γράφου (σε πολύ μεγάλο βαθμό) και η οπτική απεικόνιση (visualization viewer) του γράφου και προσθήκη του στο πάνελ των γράφων (σε πολύ μεγάλο βαθμό).

Στην πρώτη περίπτωση αυτό αντιμετωπίστηκε με τη χρήση πολυνηματικότητας (multithreading) με μια νέα διευκόλυνση που προσφέρει η Java, το SwingWoker. Έτσι δημιουργήθηκαν δύο Εργασίες (Tasks), μια για την κάλυψη των αναγκών των γενετικών αλγορίθμων που τρέχουν με βάση τον γράφο ροής ελέγχου και μια για αυτή που γίνεται πάνω στο γράφο ροής δεδομένων. Οι εργασίες αυτές κληρονομούν από την κλάση SwingWorker η οποία παρέχει πολλές ευκολίες για χρήση και επεξεργασία νημάτων (threads) και για λειτουργίες που επιδιώκουμε να τρέχουν στο παρασκήνιο (run in background). Έτσι, όταν έχουν εκτελεστεί τα απαραίτητα προαπαιτούμενα, όπως η επιλογή του προγράμματος για έλεγχο, η φόρτωσή του, και δημιουργία του γράφου (είτε ροής δεδομένων είτε ροής ελέγχου) και ο χρήστης επιλέξει να τρέξει το εργαλείο παραγωγής περιπτώσεων ελέγχου, τότε, ανάλογα με τον γράφο που έχει δημιουργηθεί και τους γενετικούς αλγορίθμους που επιλεγούν να τρέξουν, δημιουργείται η αντίστοιχη Εργασία. Η Εργασία αυτή ξεκινάει να τρέχει σε ξεχωριστό νήμα χωρίς το υπόλοιπο εργαλείο να 'παγώνει' μέχρις ότου ολοκληρωθεί η παραγωγή και ο έλεγχος των περιπτώσεων ελέγχου με τους γενετικούς αλγορίθμους. Έτσι ο χρήστης μπορεί χωρίς κανένα πρόβλημα να συνεχίζει να χρησιμοποιεί το εργαλείο και να εκτελεί τις εργασίες που θέλει, ή να ελέγξει τον γράφο, να αλληλεπιδράσει με αυτόν, να φορτώσει κάποιο άλλο πρόγραμμα, κ.ο.κ.

χωρίς να αναμένει για τα αποτελέσματα τις εκτέλεσης των γενετικών αλγορίθμων από το πρόγραμμα που υπόκειται έλεγχο. Στην περίπτωση της επεξεργασίας και παραγωγής περιπτώσεων ελέγχου με βάση τον γράφο ροής ελέγχου (control flow graph) οι πληροφορίες που απεικονίζονται στο παράθυρο διαλόγου (Σχήμα 4.3) είναι η μπάρα προόδου (progress bar), ο αριθμός των εξελίξεων που απομένουν (evolutions remaining) από τον μέγιστο αριθμό εξελίξεων (κατά προεπιλογή είναι 500), ο αριθμός μονοπατιών που πρέπει να καλυφθούν, και το τρέχον μονοπάτι που ελέγχεται ανά πάσα στιγμή από κάποια περίπτωση ελέγχου. Στην άλλη περίπτωση, όπου η παραγωγή και εκτίμηση περιπτώσεων ελέγχου με γενετικούς αλγορίθμους βασίζεται στον γράφο ροής δεδομένων (data flow graph), οι πληροφορίες που εμφανίζονται στο παράθυρο διαλόγου είναι ανάλογες, αλλά αναφέρονται στις ακμές-μονοπάτια του γράφου που επιδιώκονται να καλυφθούν. Όταν το νήμα που παράγει και ελέγχει τις περιπτώσεις ελέγχου ολοκλήρωση την εκτέλεσή του, τότε ο χρήστης μπορεί να δει τα αποτελέσματα που παράχθηκαν από το αντίστοιχο κουμπί στη διεπιφάνεια.

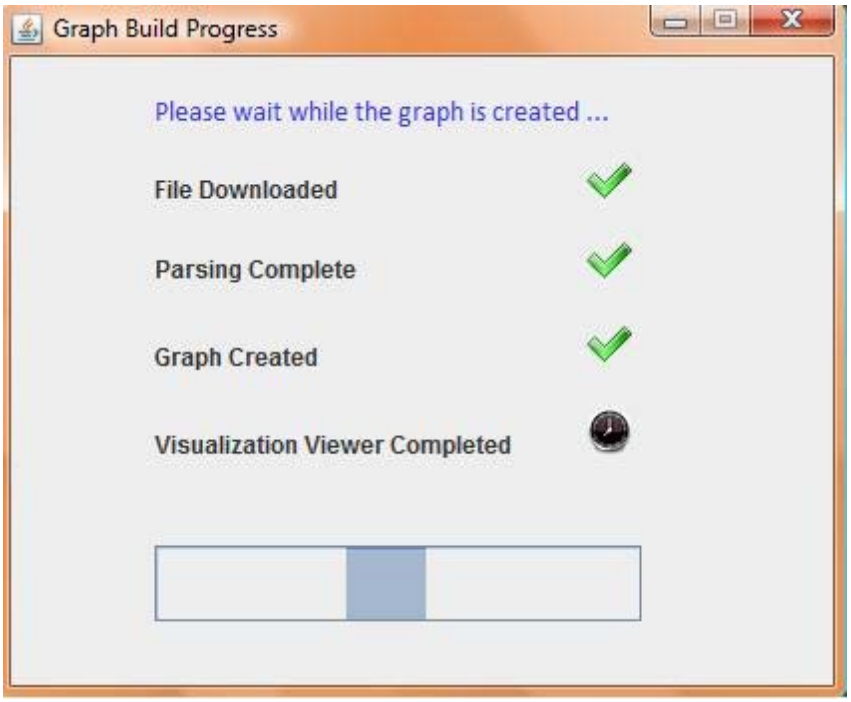


Σχήμα 4.3 : Πληροφορίες κατά τη διάρκεια εκτέλεσης GA

Όσον αφορά την δεύτερη περίπτωση που αναφέρθηκε, αντιμετωπίστηκε πάλι με χρήση νημάτων (threads) τα οποία τρέχουν ανεξάρτητα το ένα από το άλλο. Όταν ο χρήστης έχει

επιλέξει το αρχείο πηγαίου κώδικα που θέλει να ελέγξει και έχει επιλέξει το είδος/είδη του γράφου που θέλει να δημιουργήσει, στο τελευταίο βήμα του Οδηγού Ανοίγματος-Φόρτωσης Αρχείου (Load Wizard), δημιουργείται ένα Task το οποίο θα αναλάβει να μεταφορτώσει το αρχείο πηγαίου κώδικα από την απομακρυσμένη τοποθεσία του, να πραγματοποιήσει την ανάλυση του κώδικα, το χτίσιμο του γράφου και την δημιουργία της γραφικής απεικόνισής του στο πάνελ. Αυτές οι λειτουργίες πρέπει να εκτελεστούν με αυτή τη σειρά, και για το λόγο αυτό μπορούν να τρέξουν στο ίδιο νήμα. Αν ο χρήστης έχει επιλέξει περισσότερα από ένα είδη γράφων για να δημιουργηθούν, τότε ανάλογα με τον αριθμό των γράφων, θα δημιουργηθεί το αντίστοιχο πλήθος νημάτων που θα αναλάβουν να εκτελέσουν τις προαναφερθείσες λειτουργίες για τη δημιουργία του κάθε γράφου. Όπως αντιλαμβανόμαστε, κάθε ομάδα εργασιών που απαιτείται για την διεκπεραίωση της δημιουργίας και εμφάνισης ενός είδους γράφου μπορεί να εκτελεστεί παράλληλα με μια ομάδα εργασιών που στοχεύουν στην δημιουργία και εμφάνιση ενός άλλου είδους γράφου. Έτσι τα νήματα αυτά που δημιουργούν κάθε γράφο μπορούν να εκτελούνται παράλληλα. Για κάθε ένα από αυτά εμφανίζεται το ίδιο παράθυρο διαλόγου (Σχήμα 4.4) που δείχνει την τρέχουσα κατάσταση του αντίστοιχου νήματος και έτσι ο χρήστης μπορεί να δει για παράδειγμα ότι το αρχείο για τη δημιουργία του γράφου ροής δεδομένου έχει μεταφορτωθεί και γίνεται ανάλυση, ενώ το νήμα για τη δημιουργία του γράφου απαιτήσεων έχει ολοκληρώσει την ανάλυση του κώδικα και χτίζει τον γράφο, ενώ το νήμα που είναι υπεύθυνο για τη δημιουργία του γράφου ροής δεδομένων έχει δημιουργήσει το γράφο και εργάζεται στην γραφική απεικόνιση του γράφου. Όταν ένα νήμα ολοκληρώσει τη λειτουργία του και η τιμή του thread της μεθόδου isRunning είναι false, ο γράφος που έχει δημιουργήσει παρατίθεται στο πάνελ των γράφων στην κύρια οθόνη. Αν κάποιο άλλο νήμα τελειώσει μετά από αυτό, τότε ο γράφος που δημιουργεί το τελευταίο νήμα προστίθεται στο πάνελ των γράφων (και αντικαθιστά τον παλαιό γράφο που υπήρχε στο γράφο, αν υπάρχει) και προστίθεται ο γράφος στην μπάρα κύλισης με τους διαθέσιμους γράφους που έχουν κατασκευαστεί. Επομένως, το πάνελ απεικόνισης των γράφων θα απεικονίζει τον γράφο που έχει δημιουργήσει το τελευταίο νήμα που έχει ολοκληρώσει την εργασία του. Ο χρήστης επωφελείται από αυτό έχοντας την ευχέρεια να δημιουργεί όλα τα είδη γράφου που επιθυμεί χωρίς να περιμένει για τη δημιουργία κάθε γράφου σειριακά. Επίσης δε χρειάζεται να πραγματοποιήσει την ίδια λειτουργία τρεις φορές σε περίπτωση

που θέλει να δημιουργήσει τρία διαφορετικά είδη γράφων, και μπορεί να συνεχίζει να χειρίζεται την εφαρμογή καθώς τα νήματα τρέχουν και δημιουργούνται οι γράφοι. Όταν κάθε νήμα ολοκληρώσει την εκτέλεσή του το αντίστοιχο παράθυρο διαλόγου που παρουσιάζει την κατάσταση και την πρόοδο της εργασίας κλείνει αυτόματα.



Σχήμα 4.4 : Παράθυρο με πληροφορίες για την πρόοδο κατασκευής γράφου

4.3.7. Νέες λειτουργίες

4.3.7.1. Λειτουργία εκτός σύνδεσης (Offline Mode)

Για τη διευκόλυνση του χρήση αλλά για την ευκολότερη ανάπτυξη και επέκταση της εφαρμογής δημιουργήθηκε μια νέα λειτουργία η οποία τελικά αποδείχθηκε πρακτική και εύχρηστη και κατόπιν με μερικών τροποποιήσεων προστέθηκε στο μενού ρυθμίσεων της εφαρμογής για να μπορεί και ο χρήστης/προγραμματιστής να επωφεληθεί από αυτήν την

δυνατότητα. Τη δυνατότητα να επιτρέπεται στο χρήστη να χρησιμοποιεί την εφαρμογή χωρίς σύνδεση. Αυτό είναι εφικτό εφόσον φυσικά έχει μεταφορτωθεί η εφαρμογή από τον φυλλομετρητή (με τη χρήση του συνδέσμου που ξεκινάει την μεταφόρτωση) και η εφαρμογή , κατά προεπιλογή από το JAWS παραμένει αποθηκευμένη τοπικά στη μηχανή. Έτσι , εφόσον υπάρχουν οι βιβλιοθήκες και η εφαρμογή τοπικά, μπορεί ο χρήστης να επιλέξει, μέσω του μενού ρυθμίσεων (settings) της εφαρμογής αν επιθυμεί να εργαστεί με την εφαρμογή με ή χωρίς σύνδεση. Η επιλογή εργασίας χωρίς σύνδεση στο διαδίκτυο (offline mode allowed) επιτρέπει στον χρήστη να εργαστεί και να χρησιμοποιήσει την εφαρμογή ανεξάρτητα από την ύπαρξη ή μη σύνδεσης στη μηχανή που χρησιμοποιεί. Με άλλα λόγια, δίνει τη δυνατότητα εργασίας χωρίς περιορισμούς και χωρίς τη σύνδεση στο διαδίκτυο. Το μόνο που χρειάζεται να γίνει από πλευράς χρήστη είναι να σιγουρευτεί ότι υπάρχει τοπικά, στον υπολογιστή του στον προσωπικό του κατάλογο, το αρχείο με το ευρετήριο των διαθέσιμων αρχείων για έλεγχο, δηλαδή το αρχείο testfiles_index.txt, πριν την έναρξη φόρτωση αρχείων για έλεγχο και αφού έχει ορισθεί η ρύθμιση που επιτρέπει στον χρήστη να εργάζεται χωρίς σύνδεση. Αυτό μπορεί να γίνει στο μενού με τις ρυθμίσεις (settings) και να επιλέξει την αντίστοιχη επιλογή και τέλος το κουμπί «εφαρμογή» (apply) που θα αποθηκεύση αυτήν τη ρύθμιση. Αν δεν γίνει αυτό πριν την έναρξη της φόρτωσης αρχείων , τότε από προεπιλογή η εφαρμογή θα δοκιμάσει να βρεί σύνδεση και θα επιχειρήσει να συνδεθεί με το domain που βρίσκονται τα απαιτούμενα αρχεία, και εφόσον κάτι τέτοιο δεν θα μπορεί να συμβεί, αν όντως δεν υπάρχει σύνδεση, τότε θα αντικατασταθεί το υπάρχον αρχείο testfiles_index με ένα άδειο αρχείο που φέρει το ίδιο όνομα. Στην πραγματικότητα επιχειρείται μεταφόρτωση του αρχείου αυτού και επειδή δεν μπορεί να ολοκληρωθεί, λόγω απουσίας σύνδεσης με τον παγκόσμιο ιστό, δημιουργείται ένα νέο αρχείο το οποίο είναι άδειο και εμφανίζεται ενημερωτικό μήνυμα (dialog) που υπενθυμίζει στον χρήστη ότι είτε δεν έχει σύνδεση στο διαδίκτυο, είτε το αρχείο αυτό δεν υπάρχει στον απομακρυσμένο κατάλογο που αναμένεται. Έτσι, αντιλαμβανόμαστε ότι το αρχείο αυτό πρέπει να υπάρχει ήδη στην μηχανή που εργάζεται ο χρήστης. Αυτό σημαίνει ότι το αρχείο έχει μεταφορτωθεί ήδη στη μηχανή κατά τη διάρκεια προηγούμενης χρήσης της εφαρμογής η οποία πραγματοποιήθηκε με την χρήση με το διαδίκτυο και μεταφέρθηκε επιτυχώς και εξ'ολοκλήρου, είτε να δημιουργηθεί χειρωνακτικά από τον χρήστη με το ίδιο όνομα και να τοποθετηθεί στον προσωπικό κατάλογο που μεταφορτώνεται και τρέχει η

εφαρμογή. Όταν γίνει αυτό τότε για να είναι ολοκληρωμένη η εφαρμογή, θα πρέπει να επισημανθεί ότι στον ίδιο κατάλογο αναμένονται να βρίσκονται και τα αρχεία-δείγματα να οποία θα υποστούν τον έλεγχο (και τα ονόματά τους περιέχονται στο αρχείο testfiles_index), ή τουλάχιστον μόνο τα αρχεία που επιθυμεί να αναλύσει και να ελέγξει ο χρήστης.

4.3.7.2. Οδηγός Ανοίγματος-Φόρτωσης αρχείου (Load Wizard)

Η πλοήγηση του χρήστη για την επιλογή του αρχείου που θέλει να ελέγξει, τη μεταφόρτωσή του από την απομακρυσμένη τοποθεσία, και την επιλογή του γράφου που θέλει να δημιουργήσει έχει γίνει πιο εύκολη με την βοήθεια του οδηγού Ανοίγματος-Μεταφόρτωσης αρχείου (Load Wizard) ενώ παράλληλα παρέχει όλες τις απαραίτητες πληροφορίες που χρειάζεται ο χρήστης σε μόλις τρία βήματα. Πληροφορίες που αφορούν:

- το είδος του γράφου που θα δημιουργηθεί,
- το επιλεγμένο αρχείο,
- το αν ο χρήστης χρησιμοποιεί την εφαρμογή με σύνδεση στο διαδίκτυο ή όχι,
- το χρόνο που αναμένεται να διαρκέσει η ανάλυση του αρχείου και η δημιουργία του γράφου,
- το χρόνο που αναμένεται να διαρκέσουν οι γενετικοί αλγόριθμοι,
- αν ο επιλεγμένος γράφος για το επιλεγμένο αρχείο πηγαίου κώδικα είναι διαθέσιμος,
- αν το αρχείο που έχει επιλεχθεί έχει μεταφορτωθεί με επιτυχία ή βρεθεί τοπικά (αν δε χρησιμοποιείται σύνδεση).

4.3.7.3. Πηγαίος κώδικας γραμμένος από τον χρήστη

Κατά προεπιλογή, συνίσταται η χρήση των δειγμάτων-αρχείων που διατίθενται στον

απομακρυσμένο κατάλογο και ανοίγουν αυτόματα από την εφαρμογή.
Εναλλακτικά, ο χρήστης μπορεί να δημιουργήσει και τα αρχεία αυτά, ή μόνο όσα θέλει να χρησιμοποιήσει και να προσθέσει τα ονόματά τους στο αρχείο ευρετηρίου. Με αυτόν τρόπο δίνεται παράλληλα η δυνατότητα στον χρήστη να δημιουργήσει ο ίδιος τα αρχεία πηγαίου κώδικα τα οποία θέλει να χρησιμοποιήσει, με μόνο περιορισμό το ότι θα πρέπει να προσθέσει το όνομά τους στο αρχείο ευρετηρίου και να τα τοποθετήσει στον προσωπικό του κατάλογο, μαζί με το ευρετήριο, εκεί που τρέχει το εργαλείο. Πρέπει να σημειωθεί ότι για να γίνει χρήση των αρχείων αυτών, δηλαδή ανάλυση, δημιουργία των γράφων τους και έλεγχός τους, θα πρέπει να είναι συντακτικά ορθά και να και να συμβαδίζουν με τους περιορισμούς που θέτει ο κάθε αναλυτής (parser). Για παράδειγμα, ο αναλυτής που δημιουργεί γράφο ροής δεδομένων απαιτεί ο πηγαίος κώδικας να μην περιέχει δομές αναδρομής (recursion) ή δομές πίνακα (Arrays) ή διανυσμάτων (Vectors) , ή συνόλων (sets) , κλπ.
Όσον αφορά το μέγεθος του πηγαίου κώδικα, δεν υπάρχει κάποιος ιδιαίτερος περιορισμός, ωστόσο σε αρχεία που έχουν ελεγχθεί μεγέθους μεγαλύτερου από 1000 γραμμές κώδικα (όπως το Foo1000Line.java) παρατηρείται καθυστέρηση κατά την ανάλυση του κώδικα και αργεί λίγο η ολοκλήρωση της οπτικοποίησης και απεικόνισης (visualization) του γράφου. Ακόμη ενδέχεται να υπερφορτωθεί το πάνελ (JPanel) που απεικονίζει τον γράφο και να γίνει ‘βαρύ’ και οι κόμβοι του γράφου να φαίνονται πολύ πυκνοί και κοντά μεταξύ τους.
Όπως είναι αναμενόμενο, στην περίπτωση που ο χρήστης επιθυμεί να χρησιμοποιήσει δικό του πηγαίο κώδικα για το εργαλείο, δεν θα παρέχονται οι επιπρόσθετες πληροφορίες στο τελευταίο βήμα του Οδηγού Ανοίγματος-Φόρτωσης Αρχείου (Load Wizard) αναφορικά με το κατά πόσο είναι εφικτή η δημιουργία κάποιου γράφου για το επιλεγμένο αρχείο και το πόσο χρόνο θα αναμένονται τα αποτελέσματα επεξεργασίας των γενετικών αλγορίθμων και η παραγωγή και παρουσίαση των test-cases. Αυτό συμβαίνει γιατί τα αποτελέσματα έχουν εξαχθεί και ενσωματωθεί μετά από ελέγχους για τα δείγματα-αρχεία (sample files) που παρέχονται κατά προεπιλογή από το εργαλείο στον χρήστη. Ακόμη και αν αυτές οι πληροφορίες είναι διαθέσιμες, επειδή ο χρήστης χρησιμοποίησε ένα από τα υπάρχοντα αρχεία που του παρέχονται και το τροποποίησε ή δημιούργησε ένα δικό του και το αποθήκευσε με το όνομα ενός από τα προκαθορισμένα αρχεία, τότε οι πληροφορίες αυτές

πιθανόν να μην είναι αληθείς ή ακριβείς.

4.3.7.4. Αποτύπωση γράφου σε αρχείο εικόνας

Μια από τις τελευταίες προσθήκες που προστέθηκαν στην εφαρμογή στην τελευταία έκδοση ήταν η δυνατότητα να μπορεί ο χρήστης να αποθηκεύει στιγμιότυπο του γράφου, όπως αυτό εμπεριέχεται μέσα στον πάνελ των γράφων (Graph Panel). Αυτή η λειτουργία είναι διαθέσιμη στον χρήστη από τη στιγμή που έχει δημιουργηθεί κάποιος γράφος, και έχει τοποθετηθεί η οπτική-γραφική του αναπαράσταση στο πάνελ των γράφων. Η αποθήκευση γίνεται με κλικ πάνω στο κουμπί με την εικόνα της δισκέτας, που βρίσκεται κάτω δεξιά από το Graph Panel. Πατώντας αυτό το κουμπί αναδύεται ένα παράθυρο αναζήτησης φακέλων (File Browser) για να βοηθήσει τον χρήστη να βρει την τοποθεσία που θέλει να αποθηκευτεί η εικόνα του γράφου. Να σημειωθεί στο σημείο αυτό ότι επιλέγεται μόνο η τοποθεσία, δηλαδή υπάρχοντες κατάλογοι (directories) και όχι αρχεία. Το όνομα της εικόνας παράγεται αυτόματα, περιέχοντας το όνομα του αρχείου που απεικονίζει ο γράφος, το είδος του γράφου και έναν αύξοντα αριθμό αποτύπωσης, σε περίπτωση που ο χρήστης δημιουργεί περισσότερες από μια εικόνες του ίδιου γράφου του ίδιου αρχείου. Η κωδικοποίηση της εικόνας γίνεται σε μορφή jpeg και png αλλά κατά προεπιλογή έχει επιλεχθεί η jpeg. Η λειτουργία αυτή είναι πολύ χρήσιμη καθώς έτσι μπορεί ο χρήστης να αποθηκεύει την γραφική αποτύπωση του γράφου και να την διαθέτει διαθέσιμη στον σκληρό δίσκο, ακόμη και όταν το πρόγραμμα τερματιστεί. Παράλληλα, μπορεί έτσι να δει διαφορετικούς γράφους παράλληλα, να πραγματοποιήσει συγκρίσεις, κ.ο.κ. Αρκετά από τα παρόμοια εργαλεία, όπως θα δούμε διαθέτουν αυτήν δυνατότητα, όπως θα δούμε αργότερα στις συγκρίσεις με τα εργαλεία αυτά, και για τους λόγους που προαναφέρθηκαν, κρίθηκε βοηθητικό να υλοποιηθεί και να ενσωματωθεί στην εφαρμογή.

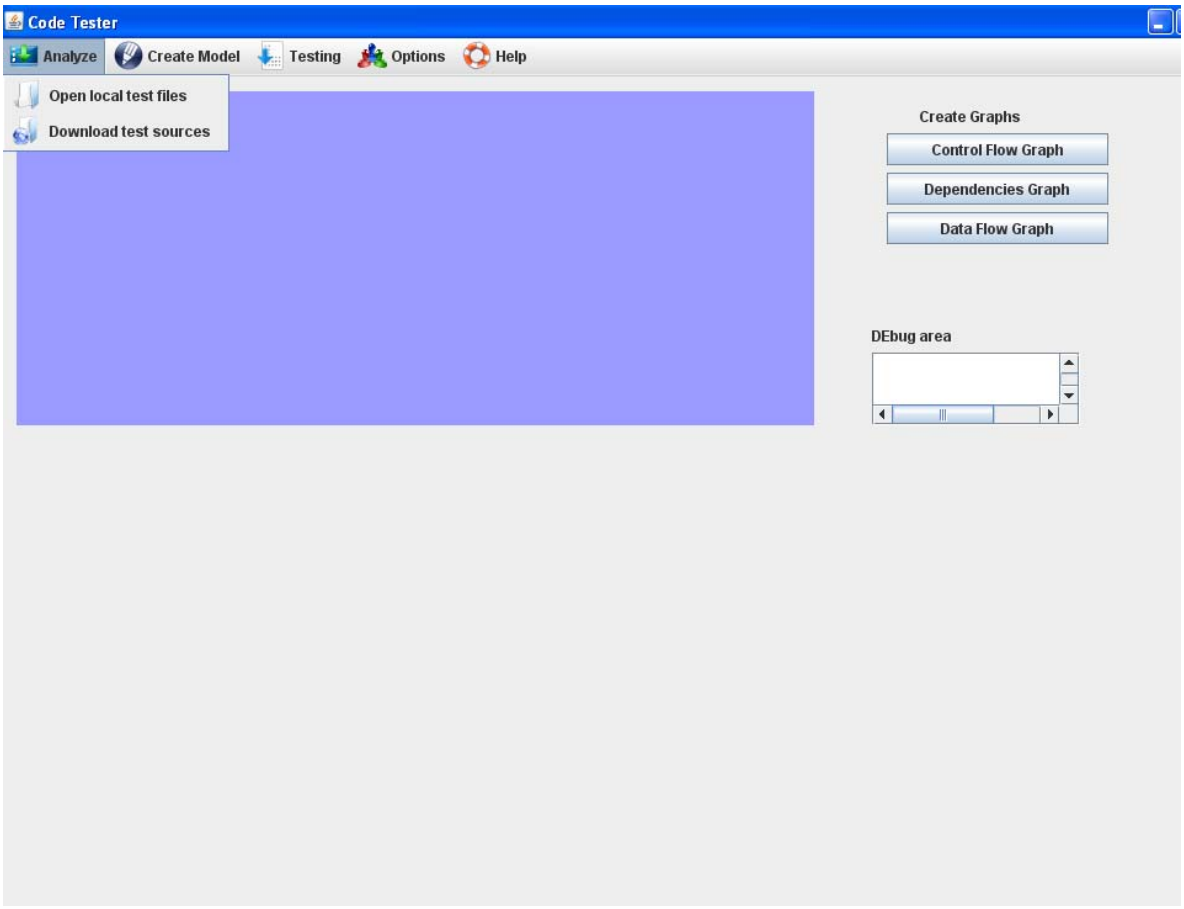
4.3.8. Επισκόπηση

Εν κατακλείδι, μπορούμε να παρατηρήσουμε ότι εκτός από την ενοποίηση αρκετών εφαρμογών – δυνατοτήτων σε ένα εργαλείο, το οποίο είναι μάλιστα διαθέσιμο με όλες τις λειτουργίες του και για χρήση από το διαδίκτυο, είναι σε θέση να προσφέρει μια ευχάριστη αλληλεπίδραση με τον χρήστη. Μπορεί δηλαδή να του επιτρέπει να χειρίζεται πληροφορίες που χρειάζεται, να αποκρύπτει αυτές που δεν είναι απαραίτητες ανά πάσα στιγμή και να προσφέρει ένα σταθερό γραφικό περιβάλλον εφαρμογής, και όχι κάτι επικεντρωμένο σε στατικές αναφορές με αποτελέσματα και εκτενή όγκο από αριθμούς ή βασισμένο σε γραμμή εντολών. Αντίθετα, δίνει τον χρήστη να αλληλεπιδράσει με τα αποτελέσματα και τους γράφους, να ενημερώνεται για την τρέχουσα κατάσταση που βρίσκεται το σύστημα και τη λειτουργία που επιτελείται, όταν αυτή τείνει να γίνεται χρονοβόρα.

4.4 Σχεδίαση Εφαρμογής

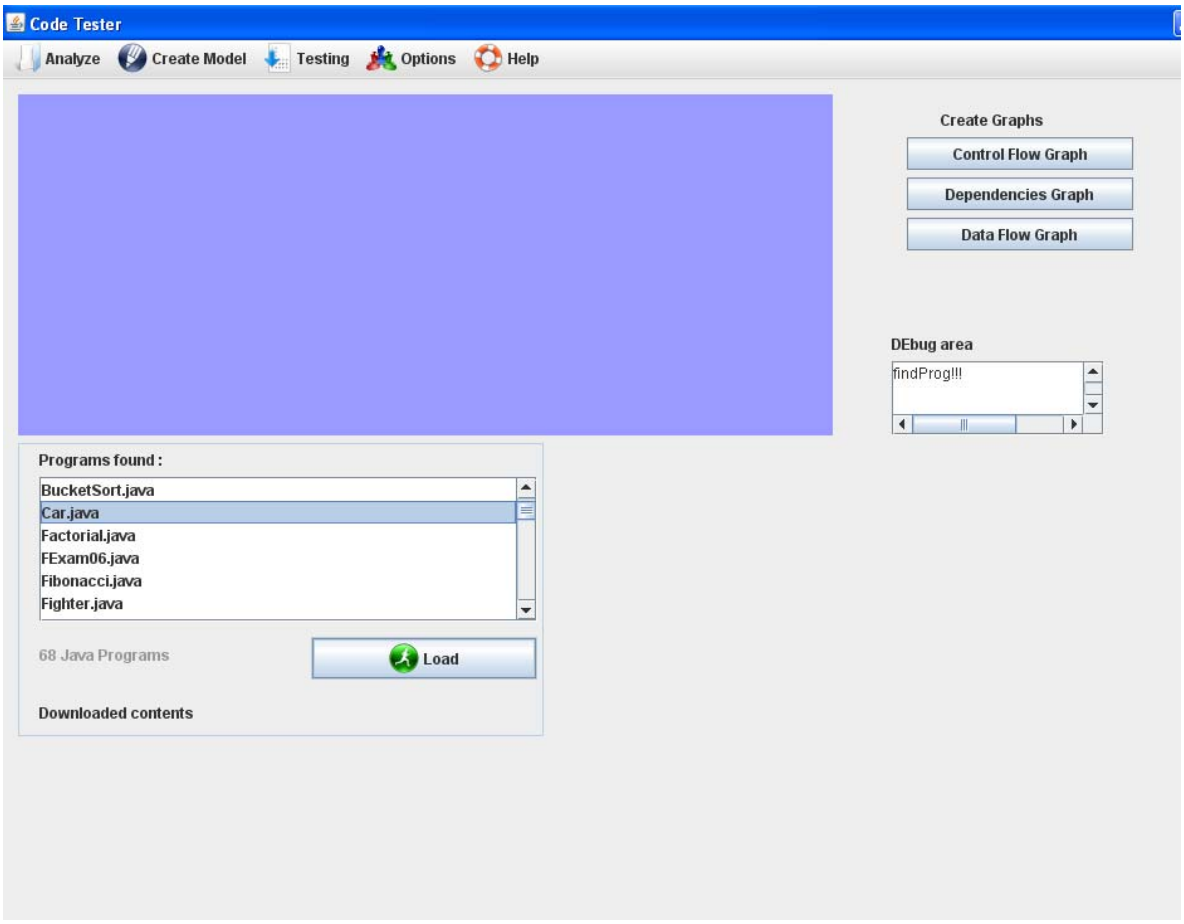
Η σχεδίαση της εφαρμογής ξεκίνησε σε μια εντελώς νέα βάση, ανεξάρτητη από τις προηγούμενες εργασίες, τμήματα αλγορίθμων των οποίων ενσωματώθηκαν στο εργαλείο. Μεταξύ άλλων, οι λόγοι για μια διαφορετική προσέγγιση στη σχεδίαση έγινε για να αποφευχθούν ατέλειες και σχεδιαστικά λάθη προηγούμενων εργασιών. Γενικότερη φιλοσοφία του σχεδιασμού του εργαλείου ήταν να διατηρήσει την αλγοριθμική ακεραιότητα και αποδοτικότητα των προηγούμενων εργασιών και αυτό όχι μόνο επιτεύχθηκε αλλά ταυτόχρονα έγινε εφικτό να αυξηθεί η λειτουργικότητα του εργαλείου. Χαρακτηριστικό παράδειγμα αυτού είναι οι νέες λειτουργίες που αναφέρονται στο κεφάλαιο 4.3.7 αλλά και λεπτομέρειες υλοποίησης όπως οι νέες βιβλιοθήκες που ενσωματώθηκαν και η εκμετάλλευση των δυνατοτήτων τους. Πιο συγκεκριμένα, με τη νέα βιβλιοθήκη γράφων και τις αλλαγές στη σχεδίαση των κλάσεων που αλληλεπιδρούν με αυτές επιτεύχθηκε η κατασκευή και δημιουργία γραφικής αναπαράστασης γράφων από πηγαίο κώδικα μεγάλης έκτασης (1000 γραμμών και μεγαλύτερο). Αυτό βέβαια δεν θα μπορούσε να επιτευχθεί χωρίς την τροποποίηση των αναλυτών (parsers), των κλάσεων των γράφων και τις αλλαγές στη σχεδίαση της γραφικής αναπαράστασης του γράφου. Ακόμη,

προσεκτική σχεδίαση έλαβε χώρα στα πλαίσια του τρόπου αλληλεπίδρασης των κλάσεων μεταξύ τους και το δυναμικό πάνελ γράφων που με τις μπάρες ολίσθησης (scrollbars), τις δυνατότητες του Swing, λειτουργίες μεγέθυνσης/σμίκρυνσης, μετακίνησης/περιστροφής και τους αλγορίθμους διάταξης γράφων κατέστη δυνατό να χωρέσουν να εμφανιστούν πολύπλοκοι πυκνοί γράφοι με έκταση πολύ μεγαλύτερη από το πάνελ, στο πάνελ των γράφων. Κάτι τέτοιο δεν ήταν δυνατόν σε προηγούμενες σχετικές εργασίες, και αν ακόμα οι γράφοι κατάφερναν να κατασκευαστούν, ήταν δυσνόητοι, δυσανάγνωστοι και καθιστούσαν πρακτικά αδύνατη την αλληλεπίδραση του χρήστη με τους γράφους. Παράλληλα, δόθηκε ιδιαίτερη έμφαση στην βελτίωση της διεπιφάνειας και της αλληλεπίδρασης με το σύστημα. Γίνεται προσπάθεια να δοθεί εργονομία και ευχρηστία στην εφαρμογή επιταχύνοντας την πλοήγηση στο εργαλείο με απόκρυψη αχρείαστων πληροφοριών, δίνοντας όμως ταυτόχρονα τη δυνατότητα, ανάλογα με την εμπειρία του χρήστη να δει όλες τις πληροφορίες που μπορεί να χρειαστεί και να τροποποιήσει παραμέτρους που επιθυμεί. Όπως για παράδειγμα με τη δημιουργία του Οδηγού Φόρτωσης-Ανοίγματος Αρχείου (Load Wizard) που αντικατέστησε την παλαιότερη οθόνη ανοίγματος αρχείων που μπορούμε να δούμε στο Σχήμα 4.13. Ταυτόχρονα με την καλύτερη διαχείριση της πολυνηματικότητας, αυξάνεται η ανεξαρτησία χρονοβόρων διεργασιών ενώ η αντικατάσταση βιβλιοθηκών με νεότερες, αποτελεσματικότερες, που παρέχουν μεγαλύτερη λειτουργικότητα. Η αντικατάσταση υπαρχουσών βιβλιοθηκών με νεότερες εκδόσεις ώθησε στην αναθεώρηση του σχεδιασμού και την αλλαγή αρκετών κλάσεων για την καλύτερη λειτουργία τους.

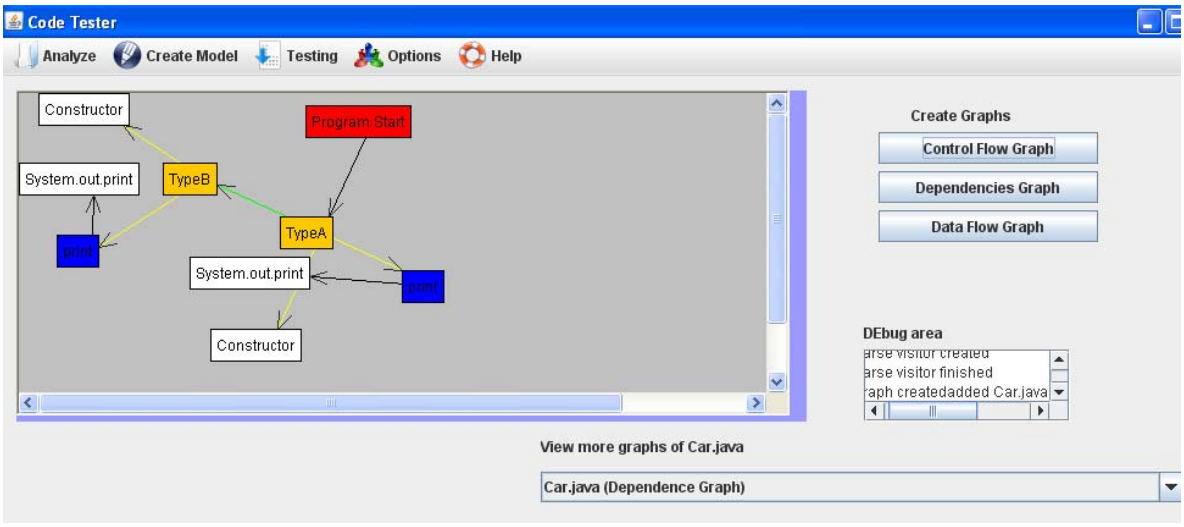


Σχήμα 4.5 : Πρωτότυπο σε αρχικά στάδια σχεδίασης

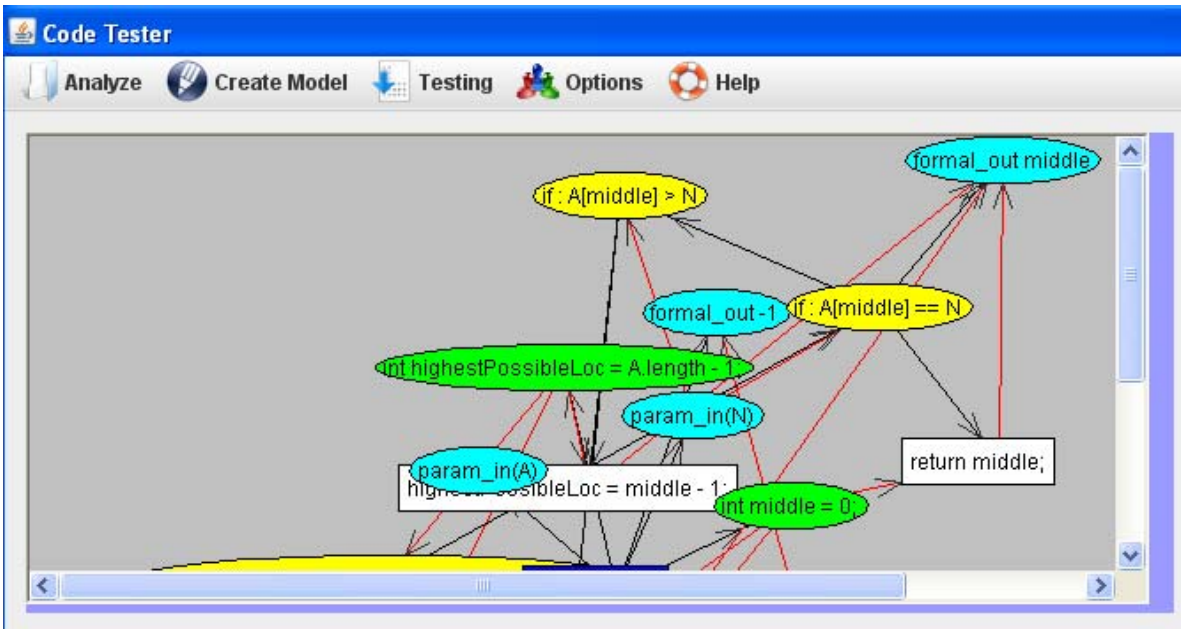
Μπορούμε να δούμε πιο παραστατικά στο σχήμα (Σχήμα 4.5) ένα από τα πρωτότυπα της εφαρμογής, με περιορισμένη λειτουργικότητα, όπως σχεδιάστηκε στα πρώτα στάδια του εργαλείου. Στο Σχήμα 4.6 βλέπουμε ένα στιγμιότυπο από την εκτέλεση της εφαρμογής σε μια από τις πρώτες εκδόσεις, ενώ στο Σχήμα 4.7 μπορούμε να δούμε τη γραφική αναπαράσταση του γράφου απαιτήσεων του προγράμματος Car.java με χρήση της παλιάς βιβλιοθήκης για κατασκευή γράφων ibmgraph.jar , σε προηγούμενη έκδοση, στο σημείο που έγινε συνένωση των προηγούμενων εργασιών στο νέο πρωτότυπο.



Σχήμα 4.6 : Στιγμιότυπο από εκτέλεση της εφαρμογής βασισμένο στο πρωτότυπο

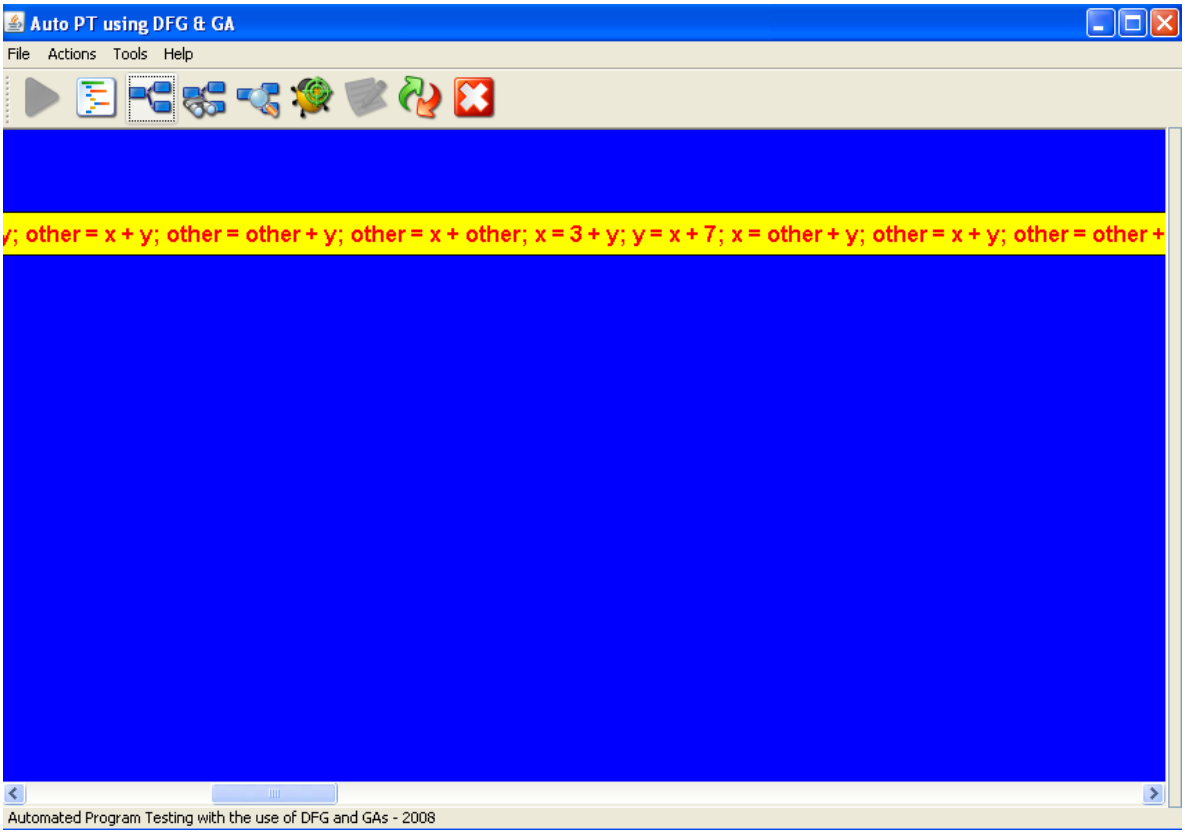


Σχήμα 4.7 : Γράφος απαιτήσεων με χρήση της παλιάς βιβλιοθήκης γράφων



Σχήμα 4.8 : Γράφος ροής δεδομένων με χρήση της παλιάς βιβλιοθήκης γράφων

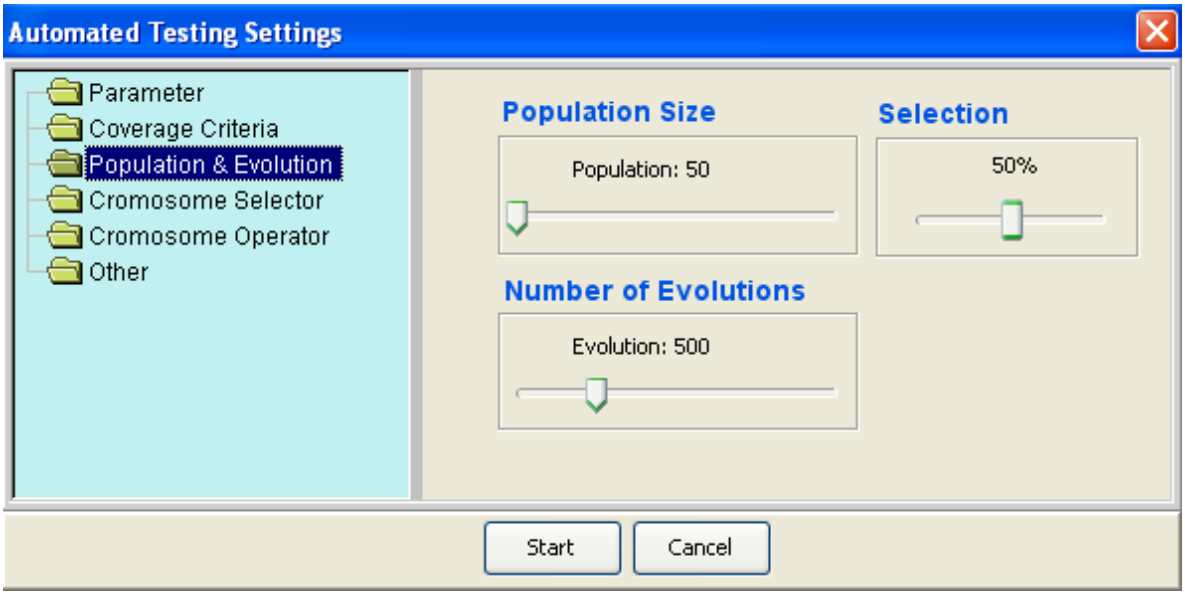
Μπορούμε να διαπιστώσουμε από το Σχήμα 4.8 ότι ένας όχι ιδιαίτερα πυκνός γράφος ροής δεδομένων μπορεί να γίνει αρκετά δυσανάγνωστος αφού δύσκολα διαχωρίζονται οι ακμές του γράφου, ενώ η μετακίνηση κόμβων γίνεται με δυσχέρεια. Παράλληλα, για να διαπεράσουμε όλο το γράφο χρειάζεται να μετακινήσουμε αρκετή απόσταση τις μπάρες κύλισης, κάτι που κάνει δύσκολη τη μελέτη του γράφου. Στο σχεδιασμό της τελικής έκδοσης όλες οι παράμετροι αυτοί ελήφθησαν υπόψη και για τη διευκόλυνση του χρήστη έγινε χρήση κατάλληλων αλγορίθμων διάταξης γράφων, δυναμικό πάνελ με δυναμικές μπάρες κύλισης που αλλάζουν με βάση τη μεγέθυνση / σμίκρυνση του γράφου και την μετατόπιση.



Σχήμα 4.9 : Γράφος ροής δεδομένων σε εφαρμογή παλαιότερης εργασίας

Στο Σχήμα 4.9 μπορούμε να δούμε ένα στιγμιότυπο από την εκτέλεση της εφαρμογής που είχε υλοποιηθεί σε προηγούμενη διπλωματική εργασία, οι αλγόριθμοι τις οποίας

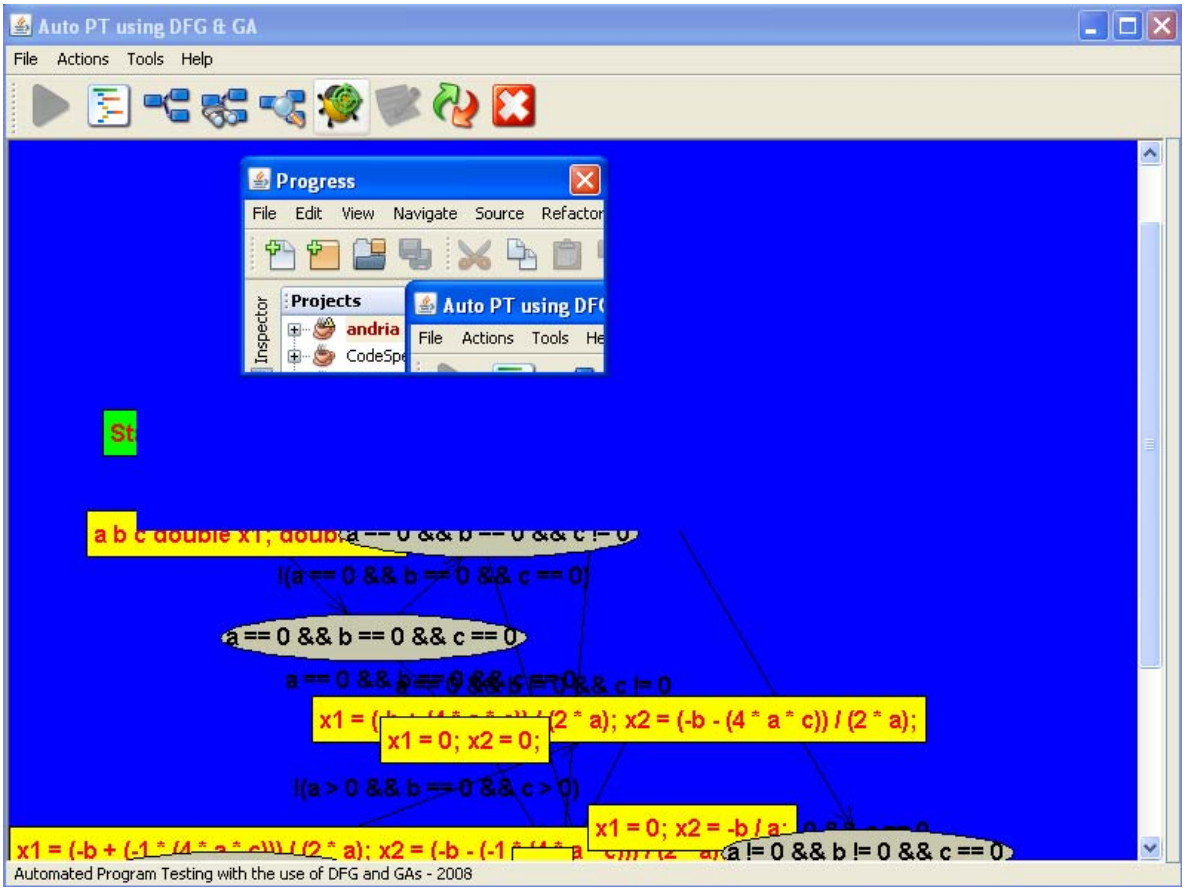
ενσωματώθηκαν στο εργαλείο για την κατασκευή του γράφου ροής δεδομένων και για την παραγωγή περιπτώσεων ελέγχου για την κάλυψη Ntafos [6, 7]. Είναι φανερή η έλλειψη αλγορίθμων διάταξης γράφων στην εφαρμογή αυτή, κάτι που δυσχεραίνει την οπτική αναπαράσταση του γράφου, αφού φαίνεται να αναπαρίσταται ως ένας τεράστιος κόμβος , ενώ ο γράφος εκτείνεται σε μεγάλο εύρος οριζόντια.



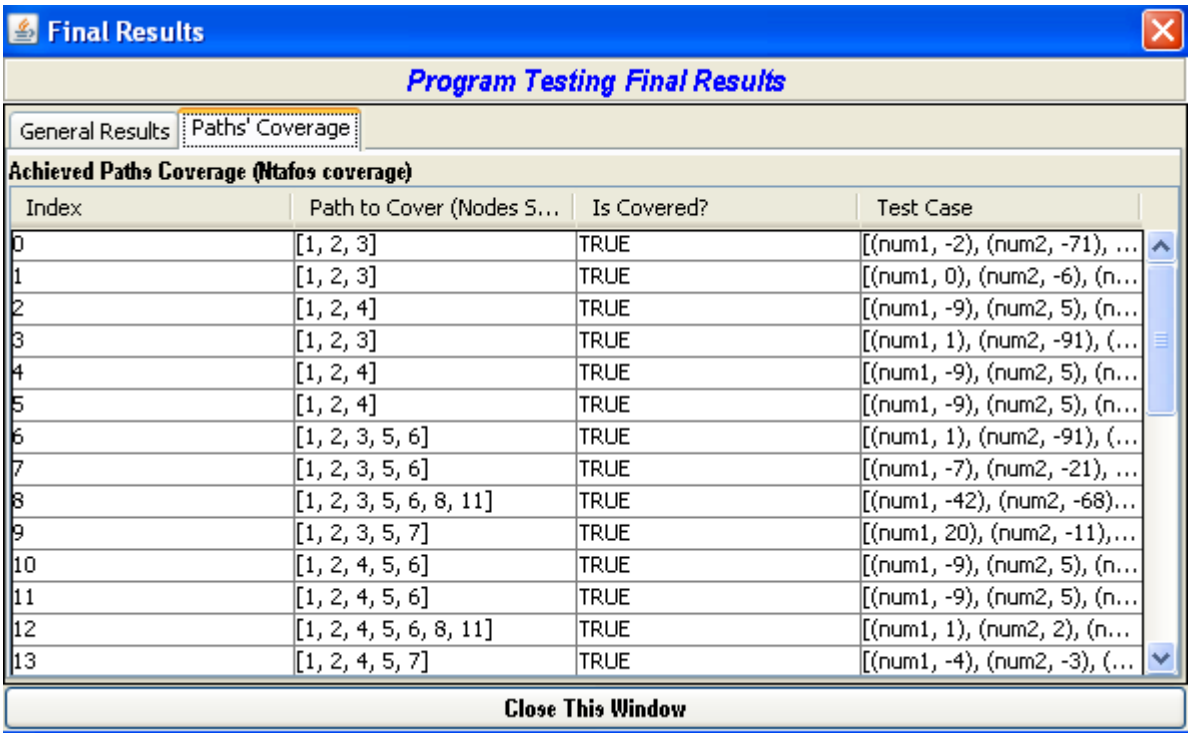
Σχήμα 4.10 : Ρυθμίσεις GA σε εφαρμογή παλαιότερης εργασίας

Στο Σχήμα 4.10 μπορούμε να δούμε το παράθυρο της ίδιας εφαρμογής για τη ρύθμιση των παραμέτρων των γενετικών αλγορίθμων ενώ στο επόμενο σχήμα (Σχήμα 4.11) παρατηρούμε ότι κατά την εκτέλεση παραγωγής περιπτώσεων ελέγχων με τους GA η εφαρμογή φαίνεται να 'παγώνει' για μεγάλο χρονικό διάστημα χωρίς να δείχνει καμιά πληροφορία σχετικά με την πρόοδο της διαδικασίας που βρίσκεται σε εξέλιξη. Όχι μόνο ο χρήστης δεν είναι σε θέση να γνωρίζει την πρόοδο των GA αλλά δε μπορεί να ξέρει αν πράγματι η παραγωγή περιπτώσεων ελέγχου και η αξιολόγησή τους βρίσκεται σε εξέλιξη η αν το σύστημα έχει κολλήσει, ενώ δεν μπορεί παράλληλα να πραγματοποιήσει κάποια άλλη εργασία μέχρις ότου η παραγωγή περιπτώσεων ελέγχου ολοκληρωθεί. Στο Σχήμα 4.12 μπορούμε να δούμε την οθόνη που παρουσιάζει τα αποτελέσματα που προκύπτουν μετά την εκτέλεση των γενετικών αλγορίθμων. Βλέπουμε ότι στην παλαιότερη εφαρμογή

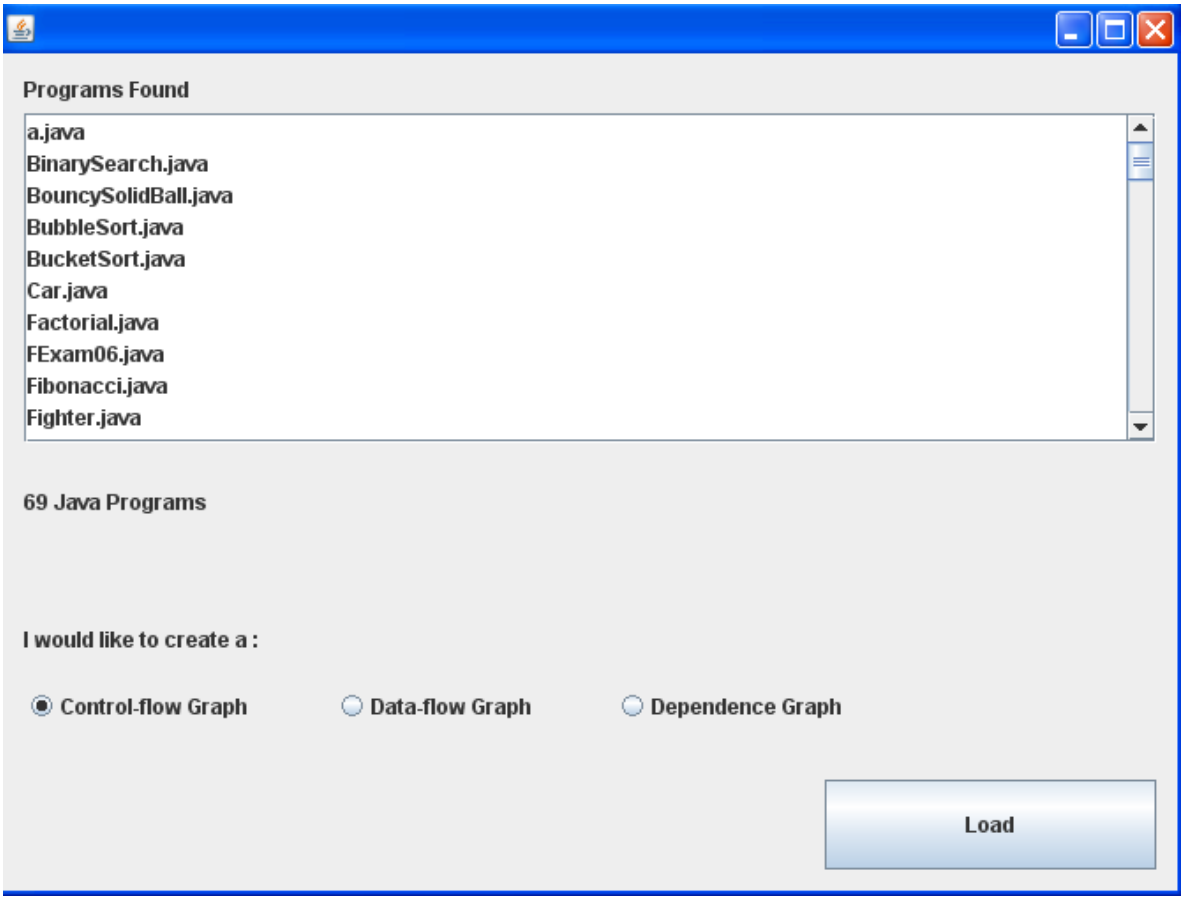
δεν υπάρχει γραφική αναπαράσταση του γράφου με τα αντίστοιχα μονοπάτια που καλύπτονται ούτε διαδραστική αλληλεπίδραση με τις περιπτώσεις ελέγχου και τον γράφο.



Σχήμα 4.11 : Εκτέλεση GA σε εφαρμογή παλαιότερης εργασίας

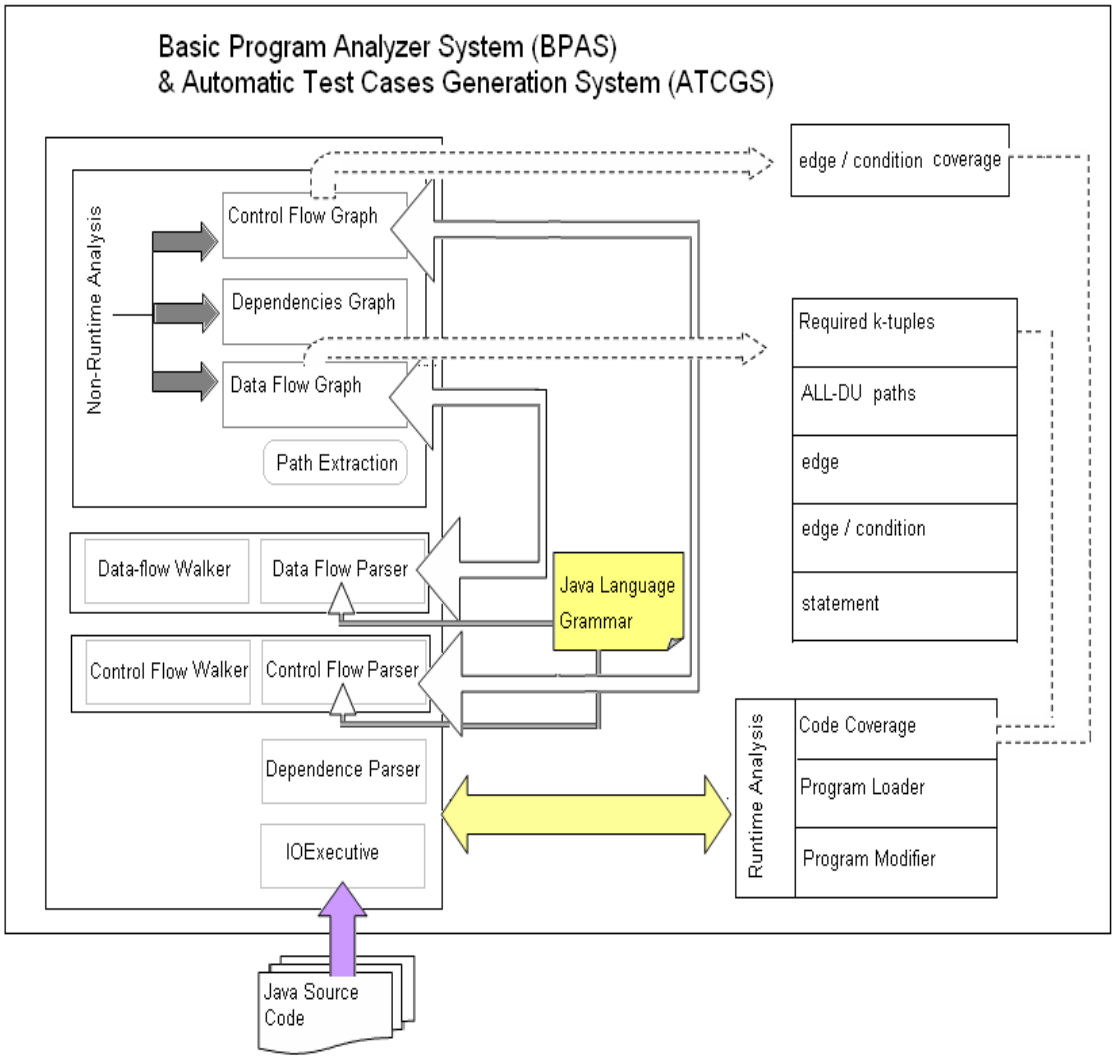


Σχήμα 4.12 : Αποτελέσματα παραγωγής περιπτώσεων ελέγχου σε εφαρμογή παλαιότερης εργασίας



Σχήμα 4.13: Άνοιγμα αρχείου πηγαίου κώδικα σε παλαιότερη έκδοση (χωρίς Load Wizard)

Συνοψίζοντας, μπορούμε να δούμε αφαιρετικά τη σχεδίαση του συστήματος στο Σχήμα 4.14 , αναφορικά με τις πιο κύριες λειτουργίες του εργαλείου.



Σχήμα 4.14: Σχεδίαση BPAS & ATCGS

4.5 Μεθοδολογία εργασίας

Ως επί το πλείστον, η διπλωματική αυτή εργασία κινείται σε προγραμματιστικά πλαίσια, δηλαδή η υλοποίηση ενός ενιαίου συστήματος το οποίο θα παρέχει στον χρήστη κάποια δείγματα κλασσικών και μη προγραμμάτων, γραμμένων σε java, και ο χρήστης θα μπορεί

να επιλέγει μεταξύ αυτών και στη συνέχεια να με τη βοήθεια του αναλυτή του εργαλείου (parser) να αναλύει το πρόγραμμα, να συλλέγει τα αποτελέσματα που προκύπτουν χωρίς την εκτέλεση του κώδικα (Program Non-Runtime Results) και να παράγει τρία διαφορετικά είδη γράφων που προκύπτουν από την ανάλυση του προγράμματος του πηγαίου κώδικα. Οι γράφοι αυτοί είναι ο Γράφος Ροής Ελέγχου του προγράμματος (Control Flow Graph) , ο Γράφος Ροής Δεδομένων (Data Flow Graph) και τέλος ο Γράφος Εξαρτήσεων του προγράμματος (Dependence Graph) όπως αυτός ισχύει για προγράμματα αντικειμενοστραφούς κώδικα γραμμένα σε java. Όταν η ανάλυση του προγράμματος ολοκληρωθεί σχηματίζεται ο γράφος, ανάλογα με το είδος/είδη γράφων που έχει επιλέξει ο χρήστης να δημιουργήσει, και εμφανίζεται σε ένα πάνελ (JPanel),
στην κεντρική φόρμα (JForm) της εφαρμογής. Αν επιλεχθούν περισσότερα από ένα είδη γράφων, εμφανίζεται ένα από αυτά και τα υπόλοιπα αποθηκεύονται αυτόματα και είναι διαθέσιμα σε ένα μενού κύλισης (Combobox) στην κύρια φόρμα.
Στη συνέχεια μπορεί ο χρήστης, ανάλογα με τον γράφο που έχει δημιουργήσει, να δει πληροφορίες που αφορούν το γράφο, του κόμβους του, τις ακμές και τα μονοπάτια που δημιουργούνται και να επιλέξει τη λειτουργία που του επιτρέπει την αυτόματη παραγωγή περιπτώσεων ελέγχου για τον πηγαίο κώδικα που επίλεξε. Η παραγωγή περιπτώσεων ελέγχου γίνεται με τη χρήση γενετικών αλγορίθμων όπως θα δούμε αναλυτικότερα στη συνέχεια. Ο χρήστης μπορεί με τη χρήση μενού να επιλέξει τις ρυθμίσεις που επιθυμεί όσον αφορά την παραγωγή περιπτώσεων ελέγχου, ρυθμίσεις που αφορούν τους γενετικούς αλγορίθμους, τον τρόπο επιλογής χρωμοσωμάτων, το μέγεθος του πληθυσμού, τον αριθμό των εξελίξεων (evolutions) , κοκ. Τέλος, μπορεί να δει τα αποτελέσματα των περιπτώσεων ελέγχου, το ποσοστό κάλυψης του κώδικα με τα επιλεγμένα κριτήρια, το χρόνο εκτέλεσης των γενετικών αλγορίθμων και γραφική αναπαράσταση των καλυμμένων μονοπατιών στον γράφο κατ' αντιστοιχία με τις περιπτώσεις-ελέγχου που επιλέγει ο χρήστης.

4.6 Υλοποίηση

Η εφαρμογή αυτή, όπως προαναφέρθηκε έχει σκοπό την δημιουργία ενός ενιαίου συστήματος το οποίο να προσφέρει αναλυτές στο χρήστη τρία διαφορετικά είδη γράφων

δεδομένου πηγαίου κώδικα γραμμένο σε java και εμφάνιση του γράφου και δυνατότητα αλληλεπίδρασης του χρήστη με τον γράφο, όπως επίσης και η αυτοματοποιημένη παραγωγή περιπτώσεων ελέγχου με τη βοήθεια γενετικών αλγορίθμων και η παροχή στον χρήστη επιπρόσθετων πληροφοριών σχετικά με τον πηγαίο κώδικα, τον γράφο, και τα αποτελέσματα τις εκτέλεσης των γενετικών αλγορίθμων, όπως το ποσοστό κάλυψης του γράφου, σκιαγράφηση των μονοπατιών που έχουν καλυφθεί και δυναμική αλληλεπίδραση με τον γράφο και τα μονοπάτια αυτά.
Για να επιτευχθεί κάτι τέτοιο, χρειάστηκε να γίνει συνένωση προηγούμενων εργασιών, που εκπονήθηκαν στο παρελθόν από συμφοιτητές και καθηγητές του τμήματος πληροφορικής. Ο κώδικας αυτός χρησιμοποιήθηκε είτε μερικώς είτε εξ' ολοκλήρου κατόπιν αρκετών αλλαγών, ανάλογα με τη λειτουργία που χρειαζόταν να ενσωματωθεί.

Η υλοποίηση ωστόσο του εργαλείου ξεκίνησε σε μια νέα βάση, με νέα διεπιφάνεια, μενού , κουμπιά και εικονίδια, διαφορετική προσέγγιση και αρχιτεκτονική. Ένας από τους στόχους ήταν η απλοποίηση των προηγούμενων εργαλείων όσον αφορά την χρήση, η παροχή περισσότερων πληροφοριών, η κατάργηση κλάσεων που δεν χρησιμοποιούνταν, η αναβάθμιση παλαιών βιβλιοθηκών (jars) με νεότερες που παρέχουν περισσότερες δυνατότητες και βελτιωμένη αίσθηση χρήσης της εφαρμογής (Look and Feel). Παράλληλα, δόθηκε έμφαση στην υποστήριξης της εφαρμογής μέσω διαδικτύου και η εξασφάλιση ότι αυτή τρέχει χωρίς προβλήματα, εξίσου αποτελεσματικά και αποδοτικά σε διάφορα λειτουργικά συστήματα όπως Linux (Ubuntu), Microsoft Windows XP και Microsoft Windows Vista (αφού κατά το μεγαλύτερο μέρος της ανάπτυξης του συστήματος εκπονήθηκε με την χρήση και τον τριών λειτουργικών συστημάτων).

Σε πρώτο στάδιο δημιουργήθηκε ένας σκελετός της εφαρμογής, με μια βασική κύρια φόρμα, την κεντρική φόρμα της εφαρμογής που είχε τρεις επιλογές (κουμπιά), ένα για την δημιουργία του κάθε είδους γράφου, ο οποίος σχηματίζονταν και απεικονίζονταν απευθείας στο πάνελ. Στο σημείο αυτό είχανε προστεθεί αρχικά οι αναλυτές (Parsers) που δημιουργούν τους γράφους ροής δεδομένων, στη συνέχεια οι αναλυτές για τη δημιουργία γράφου εξαρτήσεων και τέλος αυτοί για τη δημιουργία γράφων ροής δεδομένων. Στο πρώιμο αυτό στάδιο, έγινε οργάνωση των πακέτων(packages) της εφαρμογής με βάση τη

χρήση των κλάσεων (classes) και οι κλάσεις ανακατανεμήθηκαν στα πακέτα αυτά με βάση συνάφειάς τους ως προς τις υπόλοιπες κλάσεις των πακέτων αυτών. Έτσι η οργάνωση των κλάσεων είχε ως εξής :
(ονόματα πακέτων)
controlflowanalysis
dataflowanalysis
dependenceanalyis
gui
settings
graph
Όπου τα τρία πρώτα πακέτα αφορούν την ανάλυση του πηγαίου κώδικα (parsing), την δημιουργία του γράφου και τη συλλογή των αποτελεσμάτων που προκύπτουν χωρίς εκτέλεση του κώδικα (program non-runtime results) και πληροφορίες για κόμβους που περιέχουν δομές ελέγχου. Για αυτόν τον λόγο περιέχουν κλάσεις όπως ο ParseVisitor, ProgramNonTimeResults, StmtStack, VertexWithCondition. Πρέπεινασημειωθεί στο σημείο αυτό ότι κάθε είδους ανάλυση (control flow, data flow και dependence) χρησιμοποιεί διαφορετικούς αναλυτές και συλλέγει διαφορετικά non-runtime αποτελέσματα, για το λόγο αυτό δεν συγκεντρώθηκαν οι κλάσεις αυτές σε ένα πακέτο. Επίσης κατά την ανάλυση ροής δεδομένων, χρειάζονται πρόσθετες κλάσεις όπως αυτής του HelpPathForKDr που διευκολύνει την δημιουργία μονοπατιών για κάποια συγκεκριμένη κ-dr αλληλεπίδραση (k-dr interaction).

Το πακέτο gui (Graphical User Interface) περιέχει όλες τις κλάσεις που αποτελούν κάποιο είδος αλληλεπίδρασης του χρήστη με το γραφικό περιβάλλον της εφαρμογής, όπως φόρμες, πλαίσια (JFrames), πάνελ (JPanels), μενού , παράθυρα διαλόγου (Dialogs) και άλλες βοηθητικές κλάσεις που χρησιμοποιούνται από τις προαναφερθείσες.

Το πακέτο settings περιέχει τον πυρήνα τις εφαρμογής (kernel) που θεωρείται ‘ακρογωνιαίος λίθος’ της εφαρμογής αφού περιέχει όλες τις ρυθμίσεις του εργαλείου. Ρυθμίσεις που αφορούν την ανάλυση, τους γράφους, τα αποτελέσματα της ανάλυσης, τις επιλογές και ρυθμίσεις του χρήστη, όπως ρυθμίσεις για τη συμβατότητα του λειτουργικού

συστήματος, τα μονοπάτια των αρχείων (test files path), τους αλγόριθμους εμφάνισης των γράφων (layout algorithms), ρυθμίσεις σχετικές με τη σύνδεση, κλπ.
Η κλάση που είναι υπεύθυνη για τα παραπάνω είναι η κλάση Kernel.
Ανά πάσα στιγμή οι κλάσεις επικοινωνούν με τον kernel μέσω του instance του. Δηλαδή δεν έχουμε δημιουργία νέου αντικειμένου (object) τύπου Kernel κάθε φορά που χρειαζόμαστε την κλάση αυτή αλλά χρησιμοποιούμε το συγκεκριμένο στιγμιότυπο (instance) με τις τρέχουσες πληροφορίες που έχει η κλάση Kernel.

Το πακέτο graph, αρχικά χωρισμένο σε 3 διαφορετικά πακέτα, ένα για κάθε είδος γράφου που χρησιμοποιούνταν, λόγω του ότι κάθε προϋπάρχουσα εργασία χρησιμοποιούσε διαφορετικά τη βιβλιοθήκη γράφων που ήταν η ibmgraph.jar , με διαφορετικές μεθόδους, διαφορετικούς τρόπους απεικόνισης των κόμβων του γράφου και των ακμών (χρώματα, σχήματα, κ.λπ.). Όταν σε μεταγενέστερο στάδιο αντικατέστησα την πεπαλαιωμένη βιβλιοθήκη γράφων που χρησιμοποιούνταν με μια νέα πολύ πιο βελτιωμένη οπτικά αλλά και με πολλές περισσότερες δυνατότητες και αλγορίθμους βιβλιοθήκη, την jung.jar (version 1.7.6), τότε αντικαταστάθηκαν όλες οι κλάσεις του πακέτου graph στις εξής τρεις: pGraph, pVertex, pEdge. Οι οποίες αφορούν αντίστοιχα τον γράφο, τους κόμβους και τις ακμές. Οι μέθοδοι τροποποιήθηκαν ώστε να μπορεί ο κάθε αναλυτής, ανεξάρτητα με το είδος γράφου που θέλει να χτίσει, να χρησιμοποιεί την κλάση graph. Ομοίως αλλαγές έγιναν και για τις κλάσεις των κόμβων και των ακμών. Έτσι, τα τρία πακέτα γράφων αντικαταστάθηκαν με ένα ενιαίο πακέτο γράφων που με τις τροποποιήσεις μπορούσε να καλύψει τις ανάγκες όλων των αναλυτών (parsers), μειώνοντας ταυτόχρονα αχρείαστο κώδικα και απλοποιώντας έτσι τη σχεδίαση.
Τέλος, στα τελευταία στάδια της ανάπτυξης της εφαρμογής προστέθηκαν δυο επιπλέον κλάσεις, που αφορούν διακοσμητές (decorators) των κλάσεων των ακμών και των κόμβων, οι EdgeLabelViewer και VertexLabelViewer. Αυτές οι κλάσεις επιτρέπουν στους αναλυτές να προσθέτουν επιπρόσθετες πληροφορίες στις ακμές και στους κόμβους αντίστοιχα, όπως για παράδειγμα ενδείξεις με ‘ετικέτες’ (labels) true ή false σε ακμές που απορρέουν από ένα if-statement. Επίσης, τροποποιήθηκαν οι προεπιλεγμένες επιλογές εμφάνισης και αλληλεπίδρασης με τον γράφο. Έτσι με επιπρόσθετο κώδικα στις κλάσεις των Decorators επιτεύχθηκε η στοίχιση των ονομάτων των κόμβων (statements-nodes) του γράφου στο

κέντρο των κόμβων, και όχι έξω από αυτούς, όπως και άλλες προσθήκες για την οπτική βελτίωση του γράφου, όπως το χρώμα των κόμβων, οι αντιθέσεις, το περίγραμμα των κόμβων, κλπ.

Το επόμενο βήμα ήτανε η δημιουργία πάνελ (JPanel) το οποίο περιέχει μια κλάση τύπου 'editor' (JEditorPane), στην οποία, σύμφωνα με τις προδιαγραφές του εργαλείου θα ήταν να διαβάζει και να εμφανίζει το τρέχον επιλεγμένο αρχείο που επιθυμεί ο χρήστης να ελέγξει και να αναλύσει. Το αρχείο αυτό εμφανίζεται στην κύρια οθόνη της εφαρμογής, αν βέβαια ο χρήστης έχει επιλέξει να εμφανίζεται αυτός ο editor, αφού πρώτα αναλύσει τον κώδικά του (java parsing) και στη συνέχεια με τη χρήση ειδικού πακέτου , το editor και το advancededitor πακέτα που θα τα δούμε σε λεπτομέρεια αργότερα, εμφανίζει τον πηγαίο κώδικα με τη χρήση υπογράμμισης-έμφασης σύνταξης Java (Java-syntax highlighting). Τα πακέτα editor και advancededitor, τα οποία προϋπαρχαν από τις προηγούμενες εργασίες, ενσωματώθηκαν στην εφαρμογή για να δώσουν την ευχέρεια τον χρήστη να μπορεί να επεξεργάζεται τον γράφο, να ελέγχει τις πληροφορίες του κώδικα που επιλέγει, κ.ο.κ. ενώ ταυτόχρονα να έχει τη δυνατότητα τα βλέπει τον κώδικα αυτόν στα δεξιά της κύριας οθόνης (φόρμας) σε μια βελτιωμένη οπτικά αποτύπωση, με τη χρήση του συντακτικού της γλώσσας Java. Αυτού που υλοποιείται με άλλα λόγια, είναι να τρέχει ένας αναλυτής (parser) και να συλλέξει τις πληροφορίες του πηγαίου κώδικα, όπως λέξεις κλειδιά και δεσμευμένες λέξεις της αντικειμενοστραφούς γλώσσας Java, (πχ if, return, package, public, int, class, κλπ.) και να τις χρωματίζει με μπλε. Αλλάζοντας το το χρώμα του caret (caret color) όταν αυτό χρειάζεται και το foreground, έχοντας ο γραφέας (editor) στις ιδιότητές του τον τύπο περιεχομένου (content type) ως κείμενο/java (text/java), με τη βοήθεια των προαναφερθέντων πακέτων και θέτοντας τα σχόλια (comments) του κώδικα, είτε αυτά που παρατίθενται μεταξύ '/*' … '*/' είτε σχόλια μονής γραμμής που αρχίζουν με '//' σε σκούρο πράσινο χρώμα, ενώ το υπόλοιπο τμήμα του κώδικα αποτυπώνεται σε μαύρο χρώμα. Ο editor ανανεώνεται κάθε φορά που ο χρήστης μεταφορτώνει ένα νέο πηγαίο κώδικα για προβολή και κάθε φορά που επιλέγει να φορτώσει ένα πρόγραμμα από τα είδη υπάρχοντα (δημιουργηθέντες γράφοι) που είχε δημιουργήσει προηγουμένως και βρίσκονται στο μενού κυλιόμενης μπάρας (combobox).

Λόγω της αντικατάστασης του πακέτου δημιουργίας γράφου της ibm, το ibmgraph, με το jung 1.7.6 δημιουργήθηκαν αρκετά προβλήματα αναφορικά με την ανάλυση των προγραμμάτων από του αναλυτές (parsers) και την δημιουργία των γράφων. Έτσι οι γράφοι λόγω των διαφορετικών και συχνά αυστηρών περιορισμών (constraints) που έθετε η βιβλιοθήκη jung.jar , αποτύγχαναν να εμφανιστούν ή δημιουργούσαν διαφορετικών ειδών εξαιρέσεις (runtime Exceptions) και πολλές φορές δεν εμφανίζονταν καθόλου στο πάνελ, ή εμφανίζονταν κατακερματισμένοι και με ελλιπείς ακμές ή και κόμβους , χωρισμένοι σε υπό-γράφους, αντί για έναν ενιαίο γράφο. Αυτό συνέβαινε λόγω των περιορισμών της νέας βιβλιοθήκης και έπρεπε να γίνουν αλλαγές στους αναλυτές και των τριών ειδών γράφων (ροής δεδομένων, ροής ελέγχου και εξαρτήσεων) για να αντιμετωπισθεί αυτό το πρόβλημα. Για παράδειγμα ενώ με την παλαιά βιβλιοθήκη επιτρεπόταν να προσθέσουμε μια ακμή μεταξύ δυο κόμβων η οποία ήδη ανήκε σε έναν γράφο, σε κάποιον άλλο, ανεξάρτητο με τον αρχικό γράφο, η νέα βιβλιοθήκη δεν επέτρεπε κάτι τέτοιο και δημιουργούσε μέσω των προαπαιτούμενών της (predicates) προβλήματα ή/και εξαιρέσεις. Σε κάποιες άλλες περιπτώσεις, χρειαζόταν η διαγραφή ενός βοηθητικού κόμβου από τον γράφο (όπως οι βοηθητικοί κόμβου 'If_Exit<hash_number>' οι οποίοι δημιουργούνται κατά την ανάλυση για να βοηθήσουν στην συνένωση μεταξύ πολλαπλών και εμφωλευμένων (ή μη) if-statements και να γίνει η σωστή σύνδεση μεταξύ τους και με το σώμα του 'else' του ελέγχου. Στο τέλος της ανάλυσης, αυτοί οι κόμβοι διαγράφονται από τον γράφο). Ο τρόπος που γινόταν αυτή η διαγραφή με τους παλιούς αναλυτές και την παλιά βιβλιοθήκη δούλευε αποδίδοντας τα επιθυμητά αποτελέσματα. Κάτι τέτοιο όμως δεν γινόταν με την καινούρια βιβλιοθήκη γράφων jung, αφού σε πολλές περιπτώσεις η διαγραφή ενός βοηθητικού κόμβου σημαίνει, με βάση τα προαπαιτούμενα της βιβλιοθήκης και τη διαγραφή των ακμών που τον συνέδεαν με το υπόλοιπο τμήμα του γράφου (που προκαλούσε και τον κατακερματισμό του γράφου). Έτσι, για την ορθότητα των γράφων και την αποτελεσματικότητα της εφαρμογής επιβαλλόταν να γίνουν αλλαγές στους αναλυτές και να υλοποιηθούν διαφορετικά ορισμένες μεθόδους επεξεργασίας του γράφου. Μέθοδοι οι οποίες αφορούσαν προσθήκη ή διαγραφή νέου κόμβου ή ακμής στο γράφο ή μετακίνησης μιας ακμής ή κόμβου από έναν γράφο σε κάποιον άλλο. Οι αλλαγές αυτές συνοπτικά περιλαμβάνουν προσθήκες μεθόδων οι οποίες 'κλωνοποιούν' (clone) επιθυμητές ακμές που θέλουμε να ανήκουν σε περισσότερου από έναν γράφους, όπως αυτές έχουν και χωρίς την

απώλεια των πληροφοριών που περιέχουν τα αντικείμενα των ακμών. Αντιγράφονται δηλαδή όλες οι πληροφορίες της ακμής σε μια νέα ακμή με τις ίδιες πληροφορίες, εκτός από τον μοναδικό αριθμό της ακμής του γράφου, πληροφορίες που υποδεικνύουν ότι η ακμή ανήκει σε περισσότερου από έναν γράφους. Με παρόμοιο τρόπο αντιμετωπίστηκαν και τα προβλήματα διαγραφής βοηθητικών κόμβων και αντιγραφής κόμβων σε άλλο γράφο, συλλογή πληροφοριών των κόμβων για τον γράφο ροής δεδομένων και τη δημιουργία μονοπατιών για τον ίδιο γράφο. Πρέπει να σημειωθεί εδώ ότι δεν τροποποιήθηκαν οι αλγόριθμοι για τη δημιουργία των μονοπατιών κατά την k-dr αλληλεπίδραση (k-dr interaction) ή αλγόριθμοι δημιουργίας των γράφων (κάτι που ίσως αλλοίωνε τα αποτελέσματα προηγούμενων εργασιών), καθ' όλη τη διάρκεια της ανάλυσης, απλά δημιουργήθηκαν ή τροποποιήθηκαν βοηθητικές μέθοδοι επεξεργασίας των κόμβων και τον ακμών του γράφου για να υπερπηδηθούν τα εμπόδια των περιορισμών (constraints) και προαπαιτουμένων (predicates) που έθετε η νέα βιβλιοθήκη γράφων. Τέλος, χρειάστηκε να δημιουργηθεί ένα ακόμη πακέτο , το dataflowanalysisalgorithms που περιέχει την κλάση BreadthFirstSearch.java, μια κλάση που έγραψα η οποία υλοποιεί την έρευνα – διάβαση κατά πλάτος ενός γράφου. Η κλάση αυτή χρησιμοποιούνταν από τον παλιό αναλυτή της δημιουργίας γράφου ροής δεδομένων (data flow parser) και η υλοποίησή της ήταν μέρος της βιβλιοθήκης ibmgraph.jar. Σύμφωνα με τις προδιαγραφές του εργαλείου όμως, η βιβλιοθήκη αυτή έπρεπε να αντικατασταθεί από την νεότερη έκδοση του jung (1.7.6) και στη συνέχεια να διαγραφεί από το project. Έτσι, υλοποιήθηκε η κλάση αυτή που υλοποιεί την ίδια εργασία με την χρήση του αλγορίθμου έρευνας κατά πλάτος. Σε αυτήν την φάση η παλιά βιβλιοθήκη γράφων είχε αντικατασταθεί επιτυχώς και διαγράφηκε από την εφαρμογή.

4.7 Σύγκριση με παρόμοια εργαλεία

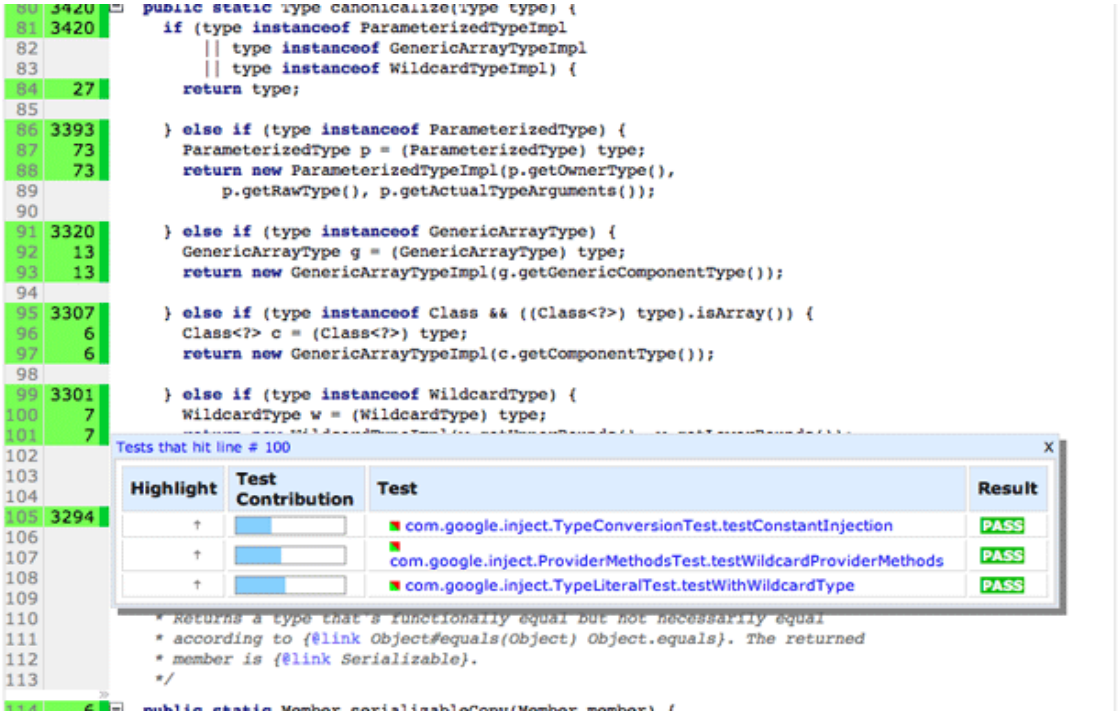
Κατόπιν έρευνας στο διαδίκτυο και ελέγχου πληροφοριών που πλαισιώνουν εργαλεία γνωστά στην αγορά του ελέγχου λογισμικού αλλά και στον ακαδημαϊκό χώρο έρευνας και

ανάπτυξης παρόμοιων προγραμμάτων, επιτεύχθηκε να συλλεχθούν αρκετές πληροφορίες αναφορικά με τις δυνατότητες που παρέχουν τα εργαλεία αυτά στους χρήστες. Παρά τις δυσκολίες που παρουσιάστηκαν, αναφορικά με την χρήση των προγραμμάτων αυτών για την διαπίστωση εξ ιδίων των δυνατοτήτων και τις αποτελεσματικότητας των εργαλείων αυτών αλλά και της απόδοσής τους σε διάφορες μηχανές, αφού τα περισσότερα από αυτά κοστίζουν αρκετά έως πολύ για την απόκτηση τους και για τις άδειες χρήσης ενώ οι διαδικασίες για την απόκτηση δοκιμαστικών εκδόσεων (evaluations) ήταν σε αρκετές περιπτώσεις πολύπλοκες. Για το λόγο αυτό σε αρκετές περιπτώσεις η έρευνα περιορίστηκε σε πληροφορίες από τους οργανισμούς και τις εταιρίες που ανέπτυξαν τα εργαλεία αυτά, την γραπτή τεκμηρίωσή τους (documentation), ζωντανές παρουσιάσεις και χρήση των εργαλείων σε βίντεο, τα φόρουμ και κατόπιν επικοινωνίας με email. Έτσι κατέστη δυνατό να συγκεντρωθούν τα απαραίτητα στοιχεία και πληροφορίες που να επιτρέπουν τη σύγκριση των εργαλείων αυτών με το παρόν εργαλείο, το οποίο εκπονήθηκε στα πλαίσια της προπτυχιακής διπλωματικής μου εργασίας. Θα πρέπει να σημειωθεί εδώ ότι θα αναφέρεται πια από τα περιγραφόμενα εργαλεία είναι πνευματική ιδιοκτησία κάποιων οργανισμών/εταιρειών, ποια υπόκεινται σε πνευματικά δικαιώματα ομάδας ανθρώπων η μεμονομένων φυσικών προσώπων που διατίθενται χωρίς χρέωση στα πλαίσια του ανοιχτού κώδικα (open source) και πια είναι δωρεάν.

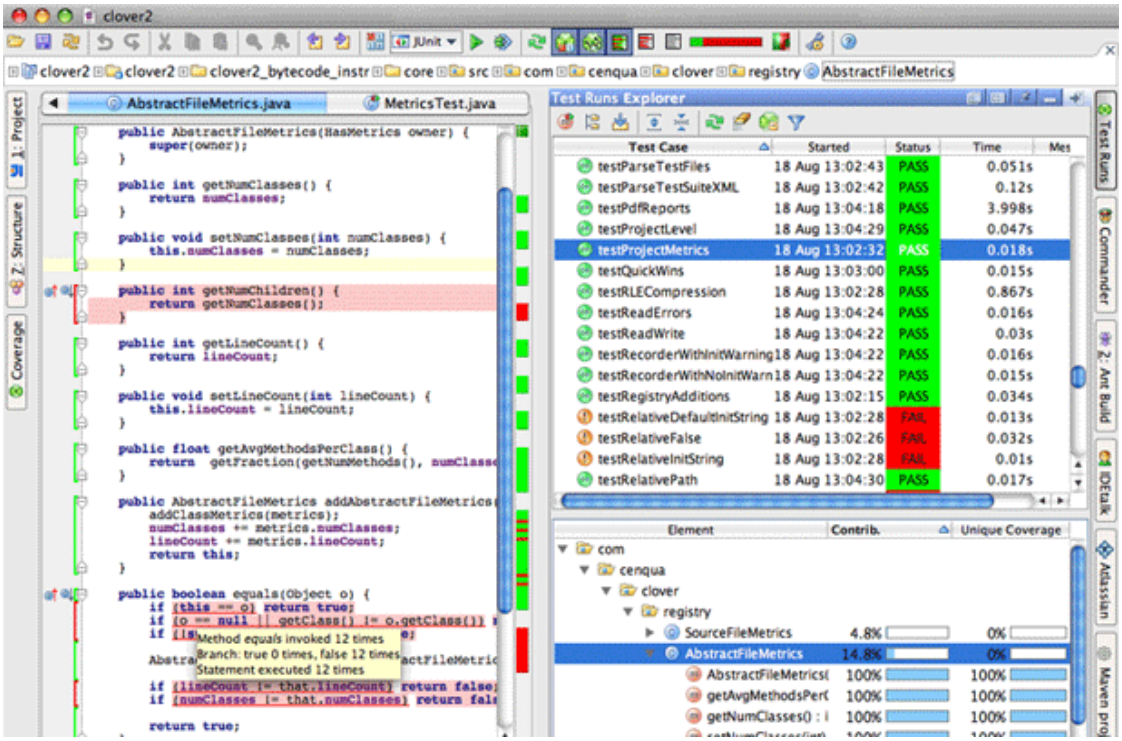
Το πρώτο εργαλεία ονομάζεται Clover , της εταιρίας Atlassean [29]. Εν συντομία το εργαλείο αυτό , μεταξύ άλλων, παράγει περιπτώσεις ελέγχου (test cases) και να ενημερώνει τον χρήση όσον αφορά το πιο τμήμα του κώδικα (αντικειμενοστραφής κώδικας γραμμένος σε Java) που θέλει να ελέγξει έχει καλυφθεί. Οι πληροφορίες που συλλέγονται ανατροφοδοτούνται στην ομάδα ανάπτυξης με στόχο τη βελτίωση της ποιότητας του λογισμικού. Το εργαλείο αυτό δέχεται μια ή περισσότερες κλάσεις του πρότζεκτ υπό έλεγχο και ενημερώνει πιο ποσοστό επί τοις εκατό του κώδικα έχει καλυφθεί από τις περιπτώσεις ελέγχου και επιτρέπει στο χρήστη να δει συγκεκριμένα πιο από τα test cases έχει ελέγξει πιο στοιχείο (element). Μπορεί παράλληλα να αναγνωρίζει κομμάτια ‘νεκρού κώδικα’ (dead code), κώδικα δηλαδή που δεν χρησιμοποιείται σε καμία περίπτωση εκτέλεσης. Τα είδη κάλυψης κώδικα που παρέχει είναι κάλυψη μεθόδων (method), δήλωσης (statement), και κάλυψη κλάδων (branch). Τα αποτελέσματα των ελέγχων

γίνονται διαθέσιμα στον χρήστη μέσω αναφορών (reports) που παράγονται από το εργαλείο σε μορφή HTML, XML ή PDF. Ακόμη είναι συμβατό και μπορεί να ενσωματωθεί (integrated) με γνωστά εργαλεία ανάπτυξης λογισμικού σε Java και ολοκληρωμένα γραφικά περιβάλλοντα, όπως το IDEA, το Eclipse, Maven, Ant και μπορεί να χρησιμοποιηθεί και από τη γραμμή εντολών (command-line). Στη χρήση με το εργαλείο Eclipse, μπορεί ο χρήστης να επιλέξει (select) τον κώδικα που επιθυμεί να ελέγξει μέσα σε μια κλάση, και όταν ολοκληρωθεί η εκτέλεση των περιπτώσεων ελέγχου, χρωματίζονται μέσα στον κώδικα του εργαλείου με πράσινο χρώμα οι γραμμές του κώδικα που κατάφεραν να καλυφθούν από τις περιπτώσεις ελέγχου και με κόκκινο χρώμα οι γραμμές που δεν έχουν καλυφθεί. Εν κατακλείδι, είναι ένα αποδοτικό εργαλείο, χρήσιμο για μεγάλες ομάδες ανάπτυξης λογισμικού πολλαπλών κλάσεων που χρειάζονται μαζικό και όχι τόσο σε βάθος έλεγχο του κώδικα, για την εύρεση της μεγάλης πλειοψηφίας σφαλμάτων και τμημάτων που δεν μπορούν να καλυφθούν, υπολογισμό της πολυπλοκότητας των κλάσεων. Το γραφικό περιβάλλον που παρέχει (Σχήμα 4.15, Σχήμα 4.16, Σχήμα 4.17) και οι αναφορές είναι βοηθητικές αλλά τα είδη ελέγχων περιορισμένα και, χωρίς βέβαια να έχω γνώση του τρόπου υλοποίησης, φαίνονται να είναι βασισμένα σε μαζική παραγωγή περιπτώσεων ελέγχου χωρίς κάποια συγκεκριμένη έξυπνη στρατηγική για την επιλογή τους. Για την απόκτηση άδειας χρήσης του εργαλείου αυτού σε μια μόνο μηχανή, για εμπορικούς σκοπούς, η χρέωση είναι $1,200 ενώ για ακαδημαϊκούς σκοπούς $600. Ομοίως, για 10 μηχανές οι άδειες για εμπορικούς σκοπούς είναι $2,200 και για ακαδημαϊκούς σκοπούς $1,100 και για 100 μηχανές $8,000 και $4,000 αντίστοιχα.

Σχήμα 4.15 : Στιγμιότυπο από εκτέλεση προγράμματος Clover



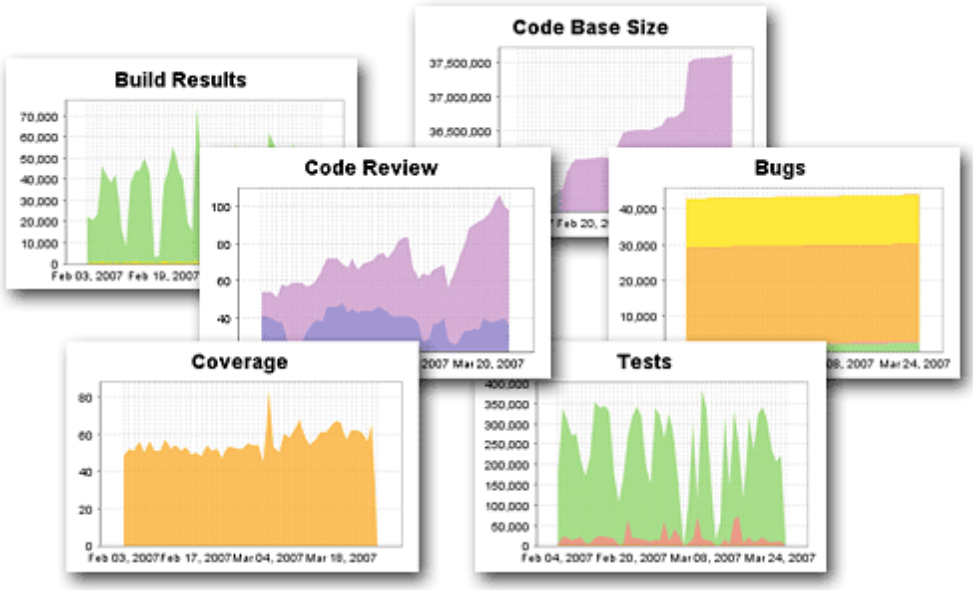
Σχήμα 4.16 : Παρουσίαση καλυμμένων γραμμών προγράμματος Clover



Σχήμα 4.17 : Παρουσίαση test-cases και κάλυψη προγράμματος Clover

Το εργαλείο Parasoft JTest της εταιρείας Parasoft [32] , είναι ένα άλλο εργαλείο που δημιουργήθηκε για να εξυπηρετήσει προγραμματιστές με την ανάλυση του κώδικα (code analysis), επίβλεψη κώδικα, αυτοματοποιημένο έλεγχο unit και component, ανάλυση κάλυψης του κώδικα, και regression έλεγχο. Παρέχει έλεγχο προγραμμάτων γραμμένα σε Java, ανάπτυξη λογισμικού σε Java SOA και εφαρμογές βασισμένες στο διαδίκτυο (Web applications). Το εργαλείο αυτό σκοπεύει να πιστοποιήσει ότι ο κώδικας μιας εφαρμογής δουλεύει με τον τρόπο που αναμένεται να λειτουργεί. Δίνει στον χρήστη τη δυνατότητα να βλέπει τον τρέχοντα κώδικα και να μπορεί να τον τροποποιήσει. Αποκαλύπτει λάθη όσο το δυνατό πιο σύντομα γίνεται, καθώς όσο πιο γρηγορότερα ανακαλυφθούν τα λάθη τόσο γρηγορότερο και λιγότερο κόστος έχει για να διορθωθούν. Δοκιμάζει πιθανά μονοπάτια χρήσης της εφαρμογής με στόχο να βρει δύσκολα στον εντοπισμό προβλήματα. Ο τρόπος με τον οποί εργάζεται και αυτό που παρουσιάζεται στον χρήστη, μεταξύ άλλων, είναι να συμβουλεύεται περισσότερους από 1000 ενσωματωμένους (built-in) κανόνες που αναπτύχθηκαν από την εφαρμογή. Οι κανόνες αυτοί αφορούν την στατική ανάλυση (static analysis), τις περιπτώσεις ελέγχου Junit (Junit test case), συμβάσεις καλού προγραμματισμού (πχ αν λείπει ο όρος ‘finally’ από ένα try-catch statement), τη συνδεσιμότητα με βάση δεδομένων (database connectivity), κανόνες αναφορικά με αντικειμενοστραφή προγραμματισμό, κ.ο.κ. Διορθώνει καταπατήσεις από 250 κανόνες, και μπορεί να εντοπίσει προβλήματα με την νηματικότητα (threading problems), βρίσκει σφάλματα σε μονοπάτια εκτέλεσης και παράγει λειτουργικές (functional) Junit περιπτώσεις ελέγχου που αποτυπώνουν την πραγματική συμπεριφορά του κώδικα. Εκτελεί την test suit για να εντοπίσει οπισθοδρομήσεις (regressions) και μη αναμενόμενα αποτελέσματα. Καταγράφει την κάλυψη των τεστ και επιτυγχάνει υψηλά ποσοστά κάλυψης χρησιμοποιώντας την ανάλυση κάλυψης κλάδων (branch coverage analysis). Επιπρόσθετα, μπορεί να εντοπίσει διαρροές μνήμης (memory leaks) και υπολογίζει διάφορες μετρικές όπως το βάθος της κληρονομικότητας, έλλειψη συνοχής, κυκλωματική πολυπλοκότητα, βάθος εμφωλευμένων μπλοκ, κλπ. Τέλος, πρέπει να αναφερθεί ότι το εργαλείο αυτό ενσωματώνεται με γνωστά περιβάλλοντα ανάπτυξης λογισμικού σε Java , όπως το Eclipse, RAD, JBuilder, και παράγει αναφορές (reports) σε XML, HTML και μπορεί ο χρήστης να τρέξει το εργαλείο είτε σε γραφικό περιβάλλον, είτε από την γραμμή εντολών (command-line).

Το Java Quality Solution (Σχήμα 4.18) είναι ένα άλλα εργαλείο της εταιρείας Parasoft [32], αρκετά παρόμοιο με το προαναφερθέν, το οποίο επικεντρώνεται στην βελτίωση της ποιότητας του παραγόμενου λογισμικού. Είναι ένα πλήρως ενσωματωμένο εργαλείο με αυτοματοποιημένη μια ευρεία σειρά από καλές πρακτικές που αποδεδειγμένα βελτιώνουν την ανάπτυξη λογισμικού, την παραγωγικότητα της προγραμματιστικής ομάδας και την ποιότητα του λογισμικού. Μερικά από τα χαρακτηριστικά αυτού του εργαλείου είναι κάποια προγραμματιστικά στάνταρ στατικής ανάλυσης, ανάλυση διαφόρων μετρικών και ανάλυση κάλυψης κώδικα. Να σημειωθεί εδώ ότι ένα μεγάλο κοινό του προγράμματος αυτού με την παρούσα εφαρμογή διπλωματικής εργασίας είναι ότι κάνει στατική ανάλυση της ροής δεδομένων. Παράλληλα, παράγει unit tests και τα εκτελεί ενώ διαθέτει αυτοματοποιημένο έλεγχο οπισθοδρόμησης.



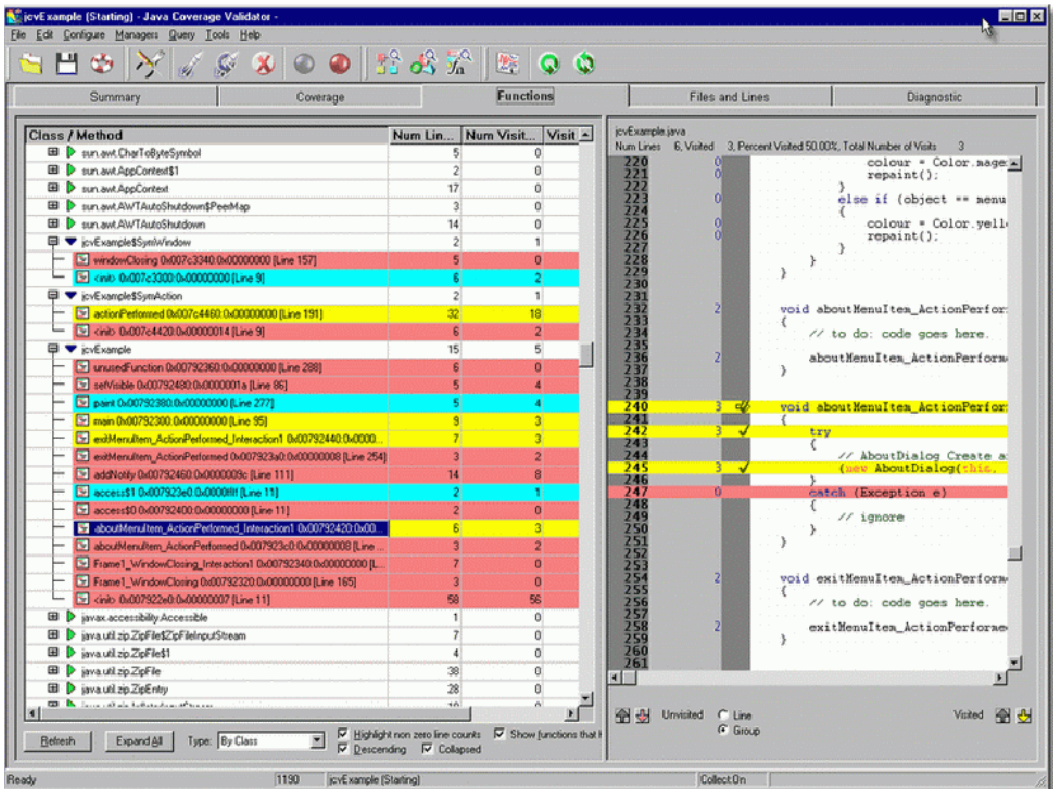
Σχήμα 4.18 : Τρόποι παρουσίασης αποτελεσμάτων εργαλείου Java Quality Solution

Το εργαλείο Software Verification [30], είναι ένα ακόμα προϊόν το οποίο προσφέρει εργαλεία τεχνολογίας λογισμικού για εύρεση διαρροών μνήμης, κάλυψη κώδικα, ανάλυση πολυνηματικότητας και καταγραφή της ροής της εφαρμογής που τρέχει σε αρκετές

πλατφόρμες, ανάμεσά τους Windows XP, Windows Vista, NT, 2000, 2003 και NT πλατφόρμες, αλλά όχι σε Linux. Υποστηρίζει τα ορισμένα από τα παραπάνω εργαλεία (σε μεγαλύτερο ή μικρότερο βαθμό) στις γλώσσες προγραμματισμού C++, C#, Visual Basic, VB.Net, Delphi, Fortran 95, Java, JavaScript, Lua, Perl, PHP, Python και Ruby.

Το εργαλείο που μας ενδιαφέρει περισσότερο, ως προς τη συνάφειά του με την παρούσα εφαρμογή είναι το Java Coverage Validator. Αυτό παρέχει αυτόματη ανάλυση κάλυψης πηγαίου κώδικα εφαρμογών καθώς αυτές τρέχουν. Όμοια με προαναφερθέντα εργαλεία, έτσι και αυτό λειτουργεί με πληροφορίες που αντλούνται από το Java Virtual Machine Debugger Interface. Σαν αποτέλεσμα που παρουσιάζεται στον χρήστη (output) είναι ο πηγαίος κώδικας που επιθυμεί ο χρήστης να ελέγξει και χρωματισμένες οι γραμμές οι οποίες έχουνε καλυφθεί και με διαφορετικό χρώμα αυτές που δεν έχουν επιτευχθεί να καλυφθούν με τις δοκιμές. Αντιλαμβανόμαστε δηλαδή ότι η κάλυψη αναφέρεται σε γραμμές κώδικα, αυτό είναι δηλαδή το μέτρο επιτυχίας των τεστ και όχι κάλυψη κλάδων, κάλυψη ελέγχων ή μονοπατιών, κλπ.

Είναι χρήσιμη εφαρμογή για έλεγχο κάλυψης τμημάτων κώδικα, όπου μπορεί ο χρήστης να θέσει ένα σημείο αρχής στον κώδικα και ένα σημείο τέλους, και να ελέγξει όλα τα πιθανά μονοπάτια κώδικα και αν έχουν εκτελεστεί. Παίρνει ως είσοδο ένα εκτελέσιμο πρόγραμμα γραμμένο σε java (Σχήμα 4.19, Σχήμα 4.20) και το εργαλείο θα συνδεθεί με το πρόγραμμα αυτό. Σύμφωνα με την τεκμηρίωση του εργαλείου, δεν επιφέρει σημαντική επιβάρυνση της απόδοσης της μηχανής αλλά επιβραδύνει το εκτελέσιμο πρόγραμμα που έχει επιλέξει ο χρήστης για έλεγχο. Για να το προμηθευτεί το εργαλείο αυτό (μόνο το Java Coverage Validator) ένας χρήστης θα πρέπει να καταβάλει το ποσό των $199 (άδεια χρήσης για έναν χρήστη).



Σχήμα 4.19 : Παρουσίαση κάλυψης κώδικα στο εργαλείο το Java Coverage Validator

Summary	Coverage
cvExample.exe	
Number of files	587	-
Number of visited files	91	15.50%
Number of unvisited files	496	84.50%
Number of 100% coverage files	9	1.53%
Number of functions	6,704	-
Number of visited functions	494	7.37%
Number of unvisited functions	6,210	92.63%
Number of 100% coverage functions	317	4.73%
Number of lines	53,700	-
Number of visited lines	3,397	6.33% (6.33%)
Number of unvisited lines	50,293 (50,303)	93.67% (93.6...
Number of unhookable lines	10	0.02%
Number of line visits	1,044,211	-

Σχήμα 4.20 : Παρουσίαση κάλυψης κώδικα στο εργαλείο το Java Coverage Validator

Το Quilt [31], ένα εργαλείο open source, πνευματικά δικαιώματα του οποίου διατηρεί η The Apache Software Foundation, έχει δημιουργηθεί για έλεγχο προϊόντων τεχνολογίας λογισμικού γραμμένου σε Java. Το Quilt μετράει την κάλυψη του κώδικα με βάση των αριθμό των statements που καλύπτονται από τον πηγαίο κώδικα από τα τεστ. Παρέχει με άλλα λόγια statement-coverage του κώδικα που θέλει να ελέγξει ο χρήστης. Έχει υλοποιηθεί για να λειτουργεί βέλτιστα με το πακέτο ελέγχου Junit, το Ant, και το project Maven. Η κάλυψη τύπου statement είναι το πιο απλό είδος κάλυψης από άποψη ελέγχου λογισμικού. Όπως και αρκετά από τα προαναφερθέντα εργαλεία, μεσολαβεί ανάμεσα στον κώδικα και τον τροποποιεί. Δεν δουλεύει με βάση τον πηγαίο κώδικα, κάτι που το εργαλείο της παρούσας διπλωματικής εργασίας πραγματοποιεί, χωρίς να τροποποιεί τον κώδικα, αλλά με παραγωγή περιπτώσεων ελέγχων επιδιώκει να επιτύχει το μέγιστο επίπεδο κάλυψης του κώδικα με τη βοήθεια γενετικών αλγορίθμων. Ανάμεσα στις δυνατότητες που παρέχει το Quilt είναι η μετατροπείς φορτωτών κλάσεων (transforming class loaders), συνθέτη κλάσεων (class synthesizer) που χτίζει μια κλάση κατά τη διάρκεια εκτέλεσης (runtime), εργαλεία για παραγωγή και τροποποίηση γράφων ροής ελέγχου (control flow graphs) των μεθόδων, και τέλος διευκολύνσεις για την παραγωγή αναφορών για την παρουσίαση των αποτελεσμάτων του ελέγχου που πραγματοποιήθηκαν. Το Quilt, με το λογισμικό που διαθέτει για τον έλεγχο κάλυψης δεν μπορεί να βοηθήσει τον προγραμματιστή να βελτιώσει τον τρόπο που γράφει περιπτώσεις ελέγχου (τεστ) ούτε πιο ποιοτικό κώδικα. Το μόνο που μπορεί να παρέχει στον χρήστη είναι να τον ενημερώνει για το αν και κατά πόσο ο κώδικας που έχει γράψει έχει ελεγχθεί. Όπως σημειώθηκε στην αρχή, το συγκεκριμένο εργαλείο υπόκειται κάτω από δυο άδειες ανοιχτού κώδικα, την Apache License και την Artistic License .

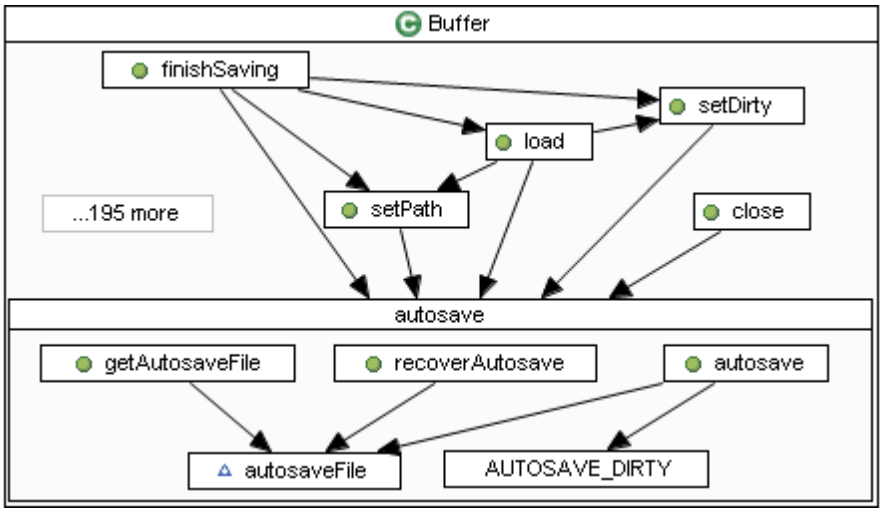
Ένα ακόμη εργαλείο με παρόμοιο σκοπό ύπαρξης, το JCover [33], της εταιρίας Codework Solutions. Το JCover, στα πλαίσια του ελέγχου white-box, βοηθά την προγραμματιστική ομάδα να διαπιστώσει αν έχει επιτευχθεί επαρκής έλεγχος σε ένα δεδομένο πρόγραμμα. Η ανάλυση κάλυψης του κώδικα αποκαλύπτει τις περιοχές του κώδικα που δεν χρησιμοποιήθηκαν κατά τη διάρκεια του ελέγχου. Το εργαλείο αυτό χρησιμοποιεί μια εύκολη και πολύ φιλική προς τον χρήστη διεπιφάνεια, διευκολύνοντας τον να ξεκινήσει

γρήγορα και να κατατοπιστεί χωρίς μεγάλη δυσκολία στις λειτουργίες του εργαλείου. Παρέχει στον χρήστη δυο είδη κάλυψης του κώδικα. Αυτά τα είδη κάλυψης είναι κάλυψη τύπου statement και κάλυψη τύπου branch, που είναι και από τα πιο δημοφιλή είδη κάλυψης στον χώρο της βιομηχανίας. Παράλληλα, υπολογίζει την κάλυψη ανά πακέτο (package), κλάση (class) και αρχείο (file). Επιπρόσθετα, δίνει τη δυνατότητα στον χρήστη να επιλέξει αν θέλει οι μετρικές του ελέγχου να συλλεχθούν κάνοντας χρήση εφαρμογών που ο πηγαίος του κώδικας είναι διαθέσιμος ή χρησιμοποιώντας τα μεταγλωττισμένα εκτελέσιμα αρχεία java (class files). Παρατηρούμε λοιπόν ότι σε αντίθεση με τα ανωτέρω περιγραφόμενα εργαλεία, δεν κάνει χρήση μόνο των εκτελέσιμων αρχείων αλλά μπορεί να ελέγξει και αρχεία πηγαίου κώδικα java, κατ'επιλογήν του χρήστη, ή να κάνει χρήση και των δυο. Προσφέρει ακόμα έλεγχο για απλές εφαρμογές τύπου πελάτη (client) αλλά και εξυπηρετητών (servers) εφαρμογές. Με χρήση της δυνατότητας ανάλυσης δεδομένων κάλυψης, μπορεί να γίνει ορατός ο τρόπος με τον οποίο η κάλυψη βελτιώνεται όσο τα τεστ εκτελούνται. Ακόμη μπορεί να παρουσιάσει στον προγραμματιστή το πώς κάθε σύνολο τεστ διαφέρει από ένα άλλο, την κάλυψη που παρέχει το ένα τεστ και το άλλο στον κώδικα, κατά πόσο επικαλύπτονται κάποια τμήματα ή αν είναι συμπληρωματικά με απώτερο στόχο να κατανοηθεί πιο από τα σύνολα τεστ είναι καλύτερο από κάποιο άλλο και να μειωθεί η αχρείαστη παραγωγή τεστ. Όπως και άλλα παρόμοια εργαλεία που αναλύθηκαν, παρέχει αναφορές και διαγράμματα για μια οπτικά πιο ευχάριστη αποτύπωση των αποτελεσμάτων. Αναφορές αυτές είναι σε HTML, XML, και CVS και τα αποτελέσματα μπορούν να εξαχθούν και σε μορφή MDB για περαιτέρω ανάλυση. Για να αποκτηθεί το εργαλείο αυτό απαιτείται η καταβολή του ποσού των $495. Πρέπει να σημειωθεί εδώ ότι είναι συμβατό μόνο σε λειτουργικό σύστημα Windows 32bit.

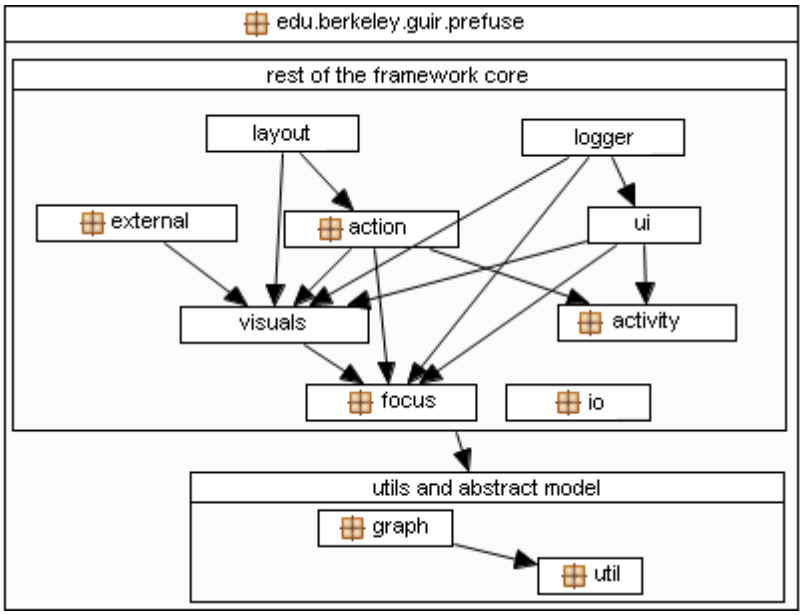
Στο σημείο αυτό θα δούμε μερικά εργαλεία τα οποία επικεντρώνονται στην ανάλυση του κώδικα και την παραγωγή γράφων του κώδικα, ένα υποσύνολο δηλαδή από τις δυνατότητες που παρέχει το BPAS & ATCGS , εργαλείο που παρουσιάζεται.
Αρχικά να σημειωθεί ότι παρότι υπάρχουν αρκετά εργαλεία για παραγωγή γράφων από πηγαίο κώδικα ή bytecodes Java, τα οποία ως επί το πλείστον είναι επιπρόσθετα plugins

για ευρέως χρησιμοποιούμενα εργαλεία όπως το Eclipse, δεν βρέθηκαν κατόπιν έρευνας εργαλεία που να είναι ανεξάρτητα και να παράγουν πολλά είδη γράφων και να είναι διαθέσιμα για χρήση μέσω διαδικτύου. Πέραν τις έλλειψης εργαλείων που ενσωματώνουν την παραγωγή γράφων εξαρτήσεων και ταυτόχρονα γράφων ροής δεδομένων και ελέγχου, όλα βάση πηγαίου κώδικα java, δεν βρέθηκαν να υπάρχουν αυτόνομα εργαλεία που να χρησιμοποιούν αυτούς τους γράφους για παραγωγή περιπτώσεων ελέγχου που να καλύπτουν ακμές και εξαγόμενα μονοπάτια, βασισμένα σε γνωστές τεχνικές κάλυψης κώδικα στα πλαίσια του ελέγχου λογισμικού, καθιστώντας το παρόν εργαλείο ως ένα μοναδικό στο είδος του πολυ-εργαλείο. Λαμβάνοντας ακόμη υπόψη μας ότι για την παραγωγή περιπτώσεων ελέγχου γίνεται χρήση γενετικών αλγορίθμων, που κάνουν την κάλυψη κώδικα μια πιο αποτελεσματική εργασία από άποψη χρόνου και πόρων που καταναλώνονται, μπορεί με ασφάλεια να εκτιμηθεί η μοναδικότητα του παρόντος εργαλείου και να κατανοηθεί ότι με περαιτέρω ανάπτυξη θα γίνει ένας πανίσχυρος σύμμαχος του προγραμματιστή στην παραγωγή ποιοτικά ελεγμένου κώδικα.

Ξεκινώντας με την εφαρμογή ispace [35], πνευματικά δικαιώματα του οποίου διατηρεί η ispace Team με την Eclipse Public License (EPL) άδεια χρήσης, είναι ένα εργαλείο βασισμένο σε γράφο (graph-based tool) για την απεικόνιση, ανάλυση και πειραματική αναδιοργάνωση γράφων εξαρτήσεων Java. Αυτό το εργαλείο παρέχει απλούς αλλά ευέλικτους τρόπους επεξεργασίας του εικονιζόμενου γράφου αναφορικά με τις ανάγκες εξαγωγής πληροφοριών του χρήστη. Επιπρόσθετα, μπορεί να ενσωματωθεί με το Eclipse. Μπορούμε να δούμε στα πιο κάτω σχήματα ότι οι παραγόμενοι γράφοι είναι απλοί οπτικά και παρέχουν απεικονίσεις εξαρτήσεων τόσο μεταξύ πακέτων (Σχήμα 4.22) όσο και κλάσεων (Σχήμα 4.21).

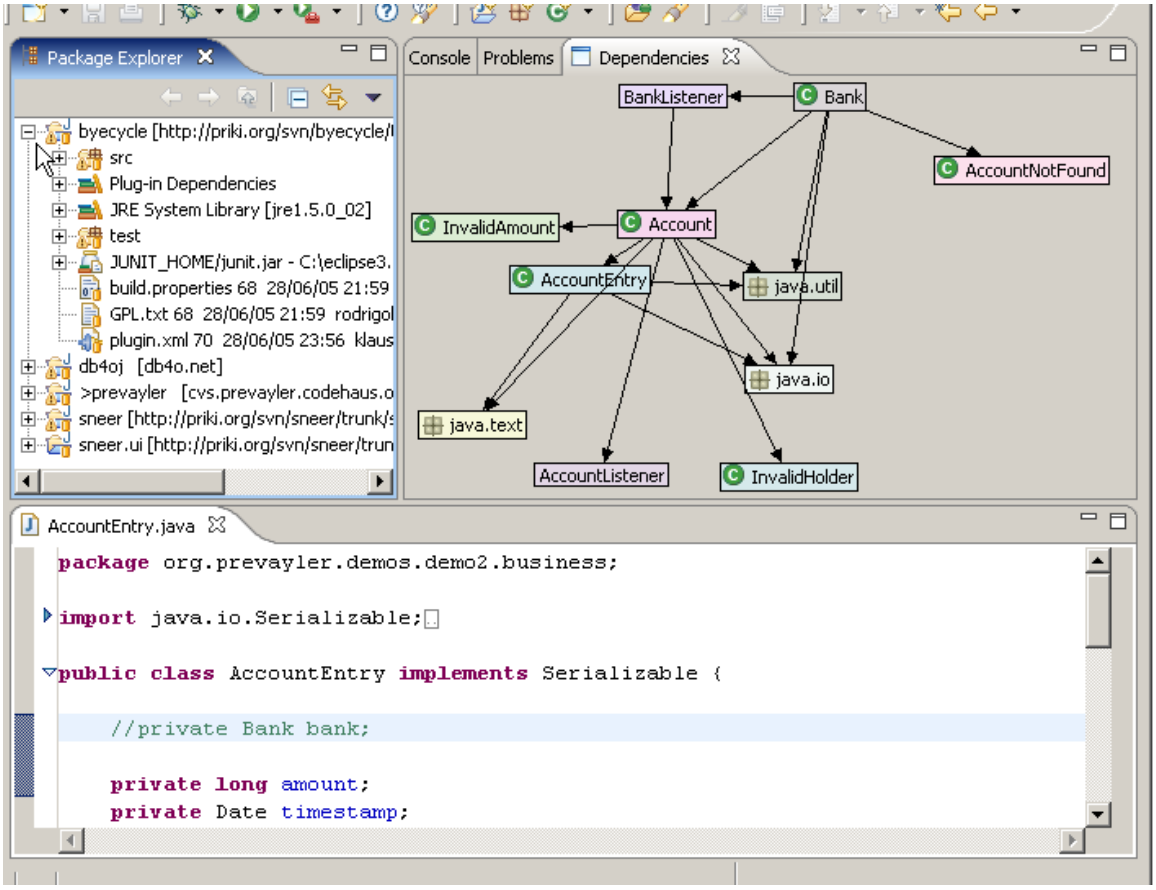


Σχήμα 4.21 : Παρουσίαση εξαρτήσεων σε μια κλάση από το εργαλείο ispace.



Σχήμα 4.22 : Παρουσίαση εξαρτήσεων μεταξύ πακέτων από το εργαλείο ispace.

Ένα ακόμη εργαλείο που στοχεύει στην ανάλυση προγραμμάτων γραμμένων σε java και την δημιουργία και απεικόνιση των γράφων εξαρτήσεών τους είναι το Byecycle [34]. Η ομάδα ανάπτυξης του εργαλείου, Klaus Wuestefeld, Rodrigo B. de Oliveira, Kent Beck αυτού δίνει μεγάλη έμφαση στον γράφο εξαρτήσεων ενός προγράμματος, όπως χαρακτηριστικά αναφέρουν : "Μετά από έναν μεταγλωττιστή και ένα πρόγραμμα σύνταξης κειμένου, το πρώτο πράγμα που χρειάζεται είναι έναν αναλυτή που να εμφανίζει τις εξαρτήσεις". Για να τρέξει αυτό το εργαλείο απαιτεί το Eclipse 3.1, στο οποίο ενσωματώνεται για να λειτουργήσει (plugged-in) ή νεότερο και τουλάχιστον Java 5. Μετά από εκτέλεσή του μπορεί να ειπωθεί ότι παράγει χρήσιμους γράφους εξαρτήσεων , όχι μόνο μεταξύ πακέτων ενός πρότζεκτ ή μόνο μεταξύ μεθόδων και κλάσεων αλλά και εξαρτήσεις ανάμεσα σε πακέτα και κλάσεις που χρειάζονται τα πακέτα για την εκτέλεσή τους. Οπτικά, ο γράφος είναι απλός και υστερεί στους τρόπους αλληλεπίδρασης που μπορεί να έχει ο χρήστης πάνω στον γράφο έναντι στο παρόν εργαλείο (BPAS & ATCGS) ενώ υπάρχουν γνωστά bugs που αφορούν την διεπιφάνεια κυρίως και τον τρόπο ενεργοποίησής του στο Eclipse. Παρέχει ωστόσο αυτοματοποιημένο αλγόριθμο αναπαράστασης του γράφου (auto-layout algorithm) για την βελτίωση της απεικόνισης του που μπορεί να καταναλώσει μέχρι και το 20% της κεντρική μονάδας επεξεργασίας (CPU). Το BPAS & ATCGS παρέχει 4 αυτοματοποιημένους αλγόριθμους αναπαράστασης (FRLayout, Circle Layout, ISOM Layout και τον συνεχώς κινούμενο (animated) αλγόριθμο Spring Layout) και ο χρήστης μπορεί να επιλέξει ανάμεσα σε αυτούς για την αναπαράσταση των γράφων του. Οι αλγόριθμοι αυτοί απαιτούν ένα μηδαμινό ποσοστό του CPU, και αυτό κατά τη δημιουργία τους μόνο και στη συνέχεια αν επιλέξει ο χρήστης την αλληλεπίδραση με τον γράφο και η επιβράδυνση που μπορεί να προκληθεί δεν είναι ορατή. Εξαιρέσεις υπάρχουν όταν ο πηγαίος κώδικας είναι εκτενής και όταν επιλεχθεί ο τελευταίος από τους προαναφερθέντες αλγορίθμους. Πιο κάτω παρουσιάζονται στιγμιότυπα από την εκτέλεση του Byecycle στο Eclipse (Σχήμα 4.23).



Σχήμα 4.23 : Παρουσίαση εξαρτήσεων από το εργαλείο Byecycle.

ΠΙΝΑΚΑΣ ΣΥΓΚΡΙΣΕΩΝ

Χαρακτηριστικά / Εργαλεία	Clover	Java Quality Solution	Parasoft JTest	Java Coverage Validator	Quilt	JCover	ispace	Byecycle	BPAS & ATCGS
CFG					✓				✓
DFG		✓							✓
DPG							✓	✓	✓
BC			✓			✓			✓
SC	✓	✓		✓	✓	✓			✓
E/CC									✓
Ntafos Coverage Reports	✓	✓	✓			✓	✓		✓
Interactive Reports	✓					✓	✓		✓
Src-based Analysis						✓	✓	✓	✓
Bytecode – based analysis	✓	✓	✓	✓	✓	✓			
Microsoft Windows XP Compatible	✓	✓	✓	✓	✓	✓			✓
Microsoft Windows Vista Compatible	✓	✓	✓	✓	✓	✓			✓
Linux Compatible (Ubuntu)									✓
Web-Enabled									✓
ATCG With GA									✓
Save graph in JPG / PNG format									✓
Integration with	IDEA, Eclipse, Maven, Ant		Eclipse, RAD, JBuilder		Junit Ant, Maven		Eclipse	Eclipse	
Price	$600 - $4000	$199/χρήστη				$495			

Επεξήγηση συμβόλων πίνακα

CFG = Control Flow Graph , Γράφος Ροής Ελέγχου
DFG = Data Flow Graph , Γράφος Ροής Δεδομένων
DPG = Dependence Graph, Γράφος Εξαρτήσεων
BC = Branch Coverage, Κάλυψη τύπου Κλάδου
SC = Statement Coverage, Κάλυψη τύπου Δήλωσης
E/CC = Edge/Condition Coverage, Κάλυψη τύπου Ακμής/Συνθήκης
ATCG = Automated Test Case Generation, Αυτοματοποιημένη παραγωγή περιπτώσεων ελέγχου
GA = Genetic Algorithms, (ΓΑ) Γενετικοί Αλγόριθμοι

4.8 Πλοήγηση στο εργαλείο

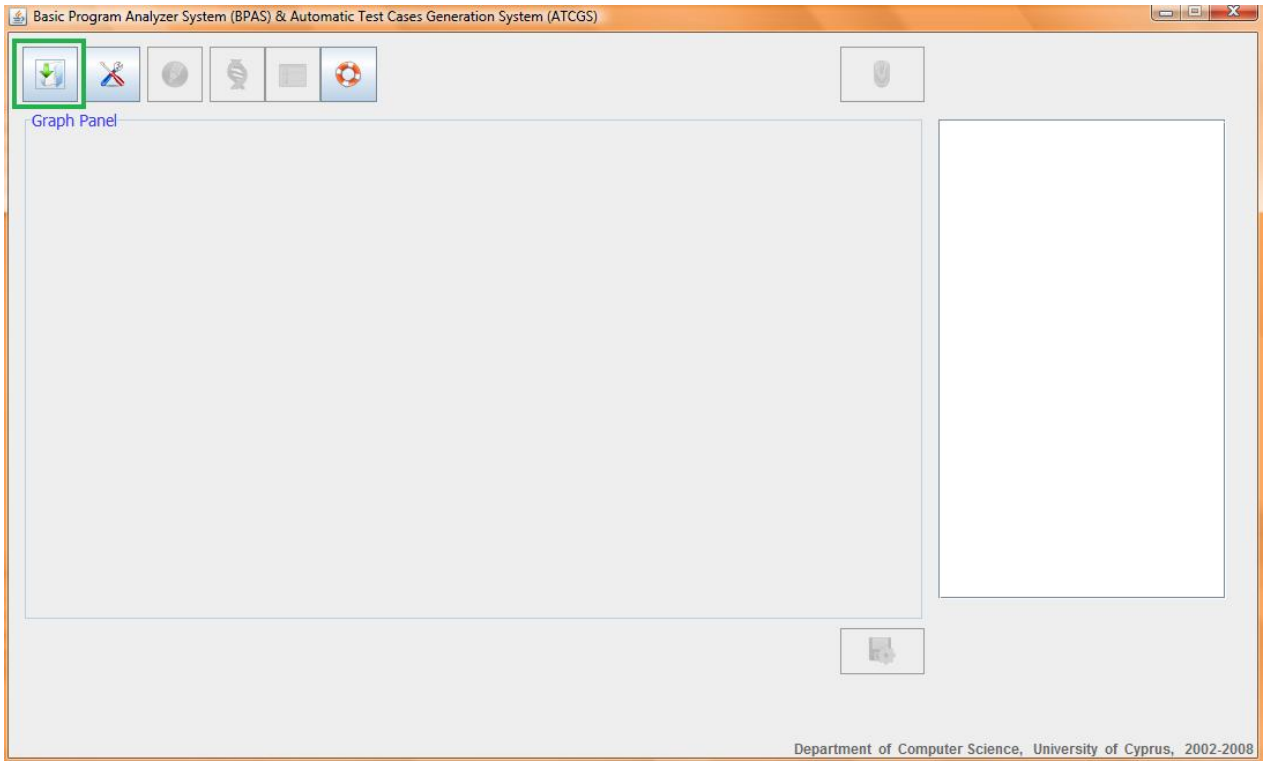
Θα ακολουθήσει στο σημείο αυτό μια ξενάγηση στο εργαλείο και στις δυνατότητές του δομημένη σε μια σειρά από σενάρια χρήσης (Use Case Scenarios).

I. Σενάριο πρώτο : Δημιουργία γράφου ροής ελέγχου για το πρόγραμμα BinarySearch.java (που κάνει δυαδική αναζήτηση) , αλλαγή του αλγορίθμου διάταξης του γράφου σε κυκλική διάταξη (Circle Layout) και στη συνέχεια αλλαγή του τρόπου αλληλεπίδρασης με τον γράφο και η μετακίνηση μερικών κόμβων.

- Ανοίγουμε την εφαρμογή μέσω του Java Web Start από τον σύνδεσμο όπως βλέπουμε πιο κάτω (Σχήμα 4.24).

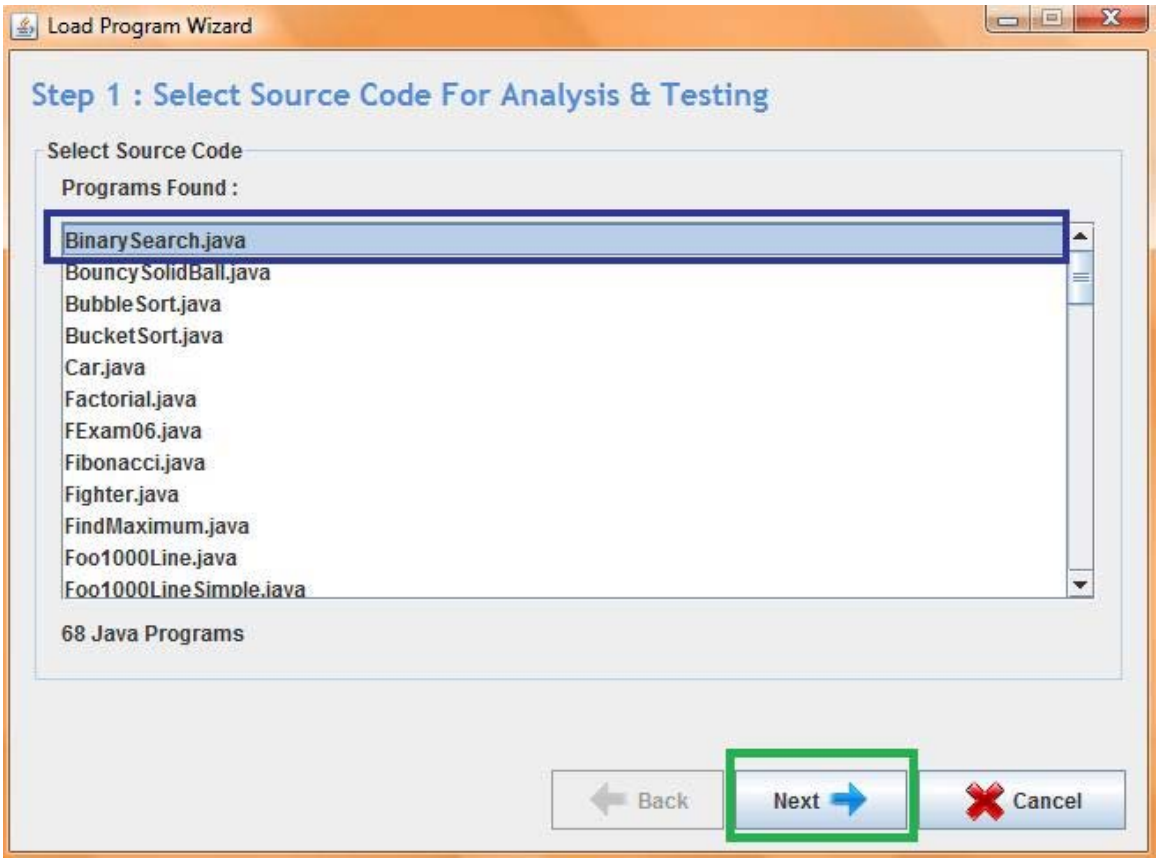
Σχήμα 4.24 : Σύνδεσμος για άνοιγμα εφαρμογής.

- Έχοντας ανοίξει την εφαρμογή, κάνουμε κλικ στο κουμπί του Οδηγού Ανοίγματος-Φόρτωσης προγραμμάτων, όπως βλέπουμε στην επόμενη εικόνα (Σχήμα 4.25)



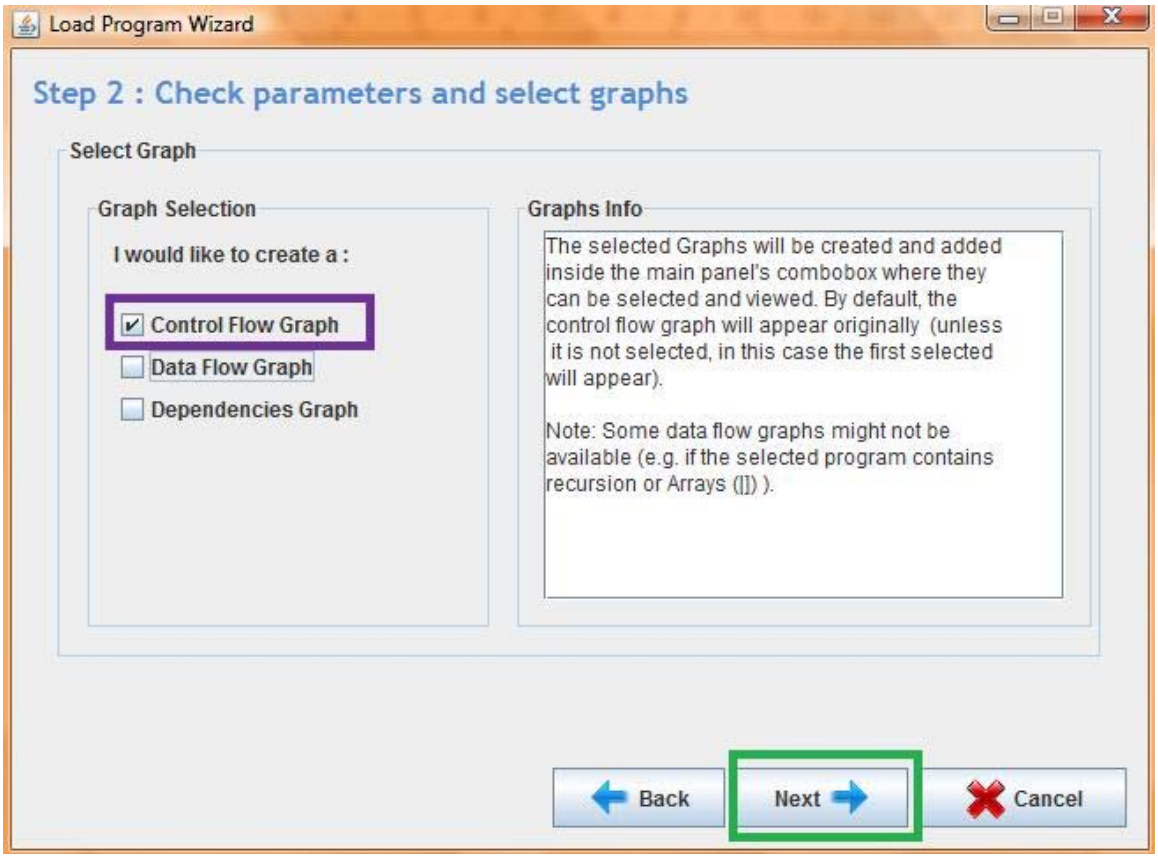
Σχήμα 4.25 : Κουμπί ανοίγματος Οδηγού Ανοίγματος-Φόρτωσης Αρχείου

- Από το πρώτο βήμα του οδηγού επιλέγουμε το αρχείο BinarySearch.java και πατάμε το πλήκτρο "Next" για να προχωρήσουμε στο επόμενο βήμα, όπως βλέπουμε παρακάτω (Σχήμα 4.26).



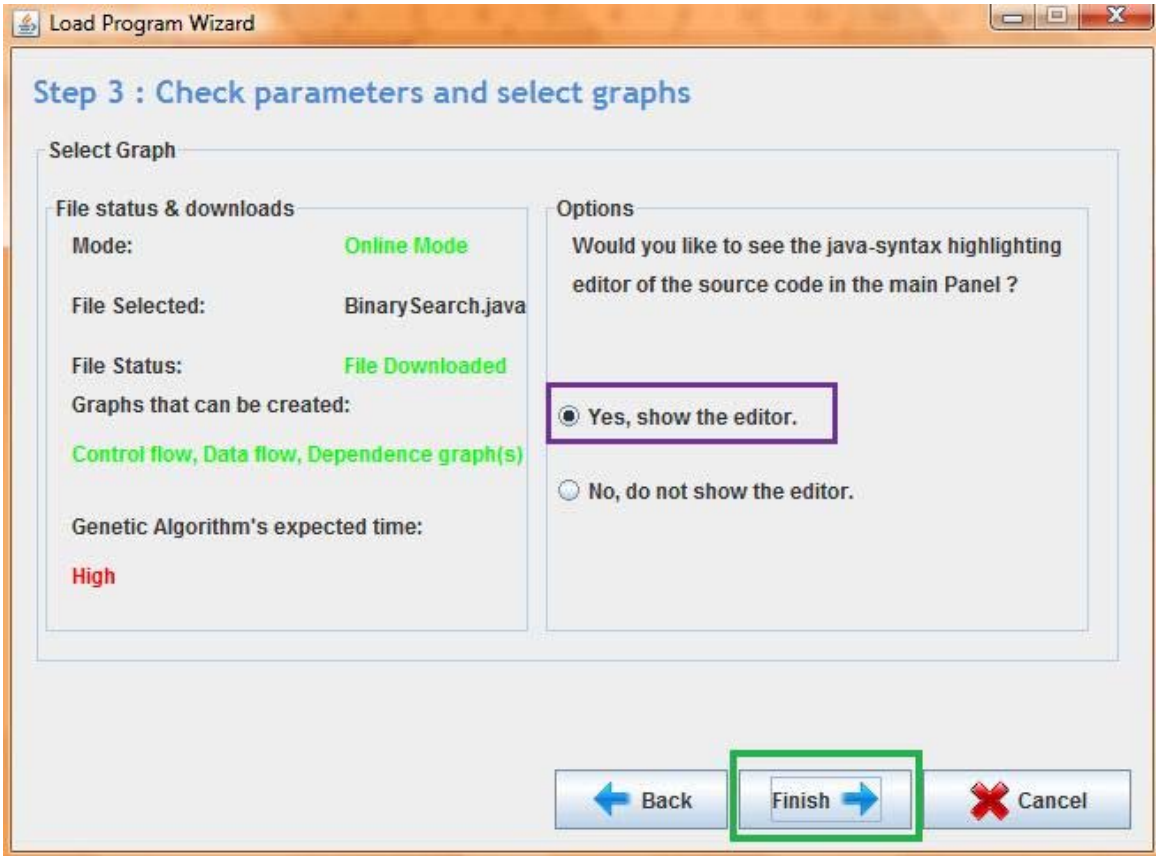
Σχήμα 4.26 : Επιλογή προγράμματος πηγαίου κώδικα

- Στο επόμενο , δεύτερο βήμα του Load Wizard, επιλέγουμε μεταξύ των διαθέσιμων ειδών γράφων το πρώτο, μόνο το γράφο ροής δεδομένων (Control Flow Graph) και πατάμε το κουμπί “Next” για να προχωρήσουμε στο επόμενο βήμα, όπως βλέπουμε παρακάτω (Σχήμα 4.27).



Σχήμα 4.27 : Επιλογή τύπου γράφου που θα δημιουργηθεί

- Στο τρίτο και τελευταίο βήμα του Οδηγού βλέπουμε την κατάσταση του επιλεγμένου αρχείου, δεδομένου ότι εργαζόμαστε με σύνδεση (online), βλέπουμε λοιπόν ότι το αρχείο έχει βρεθεί και θα μεταφορτωθεί, και ότι για το συγκεκριμένο αρχείο μπορούμε να δημιουργήσουμε γράφο ροής ελέγχου. Επιλέγουμε να εμφανιστεί ο editor (java syntax-highlighted editor) που θα παρουσιάζει τον πηγαίο κώδικα του επιλεγμένου BinarySearch.java , όπως είναι προεπιλεγμένο και πατάμε το κουμπί "Finish" για να ξεκινήσει η ανάλυση του κώδικα και η δημιουργία και παρουσίαση του γράφου. Βλέπουμε πιο κάτω την εικόνα με το βήμα αυτό (Σχήμα 4.28).

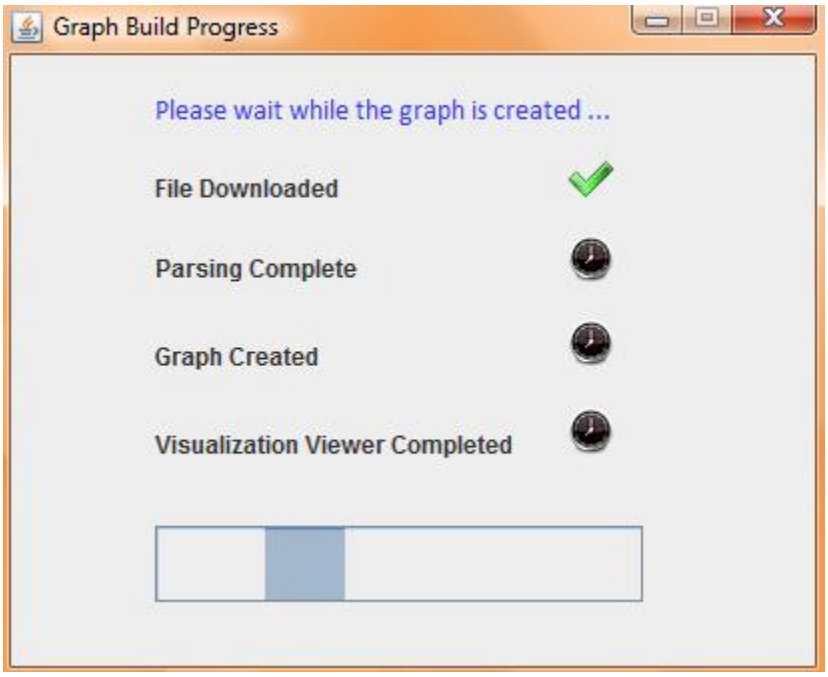


Σχήμα 4.28 : Πληροφορίες σχετικές με το αρχείο, σύνδεση και editor

- Στη συνέχεια θα δούμε ένα αναδυόμενο παράθυρο το οποίο παρουσιάζει την πρόοδο της εργασίας ανάλυσης του κώδικα και παραγωγής του γράφου, πιο αναλυτικά, μας παρέχει πληροφορίες για :

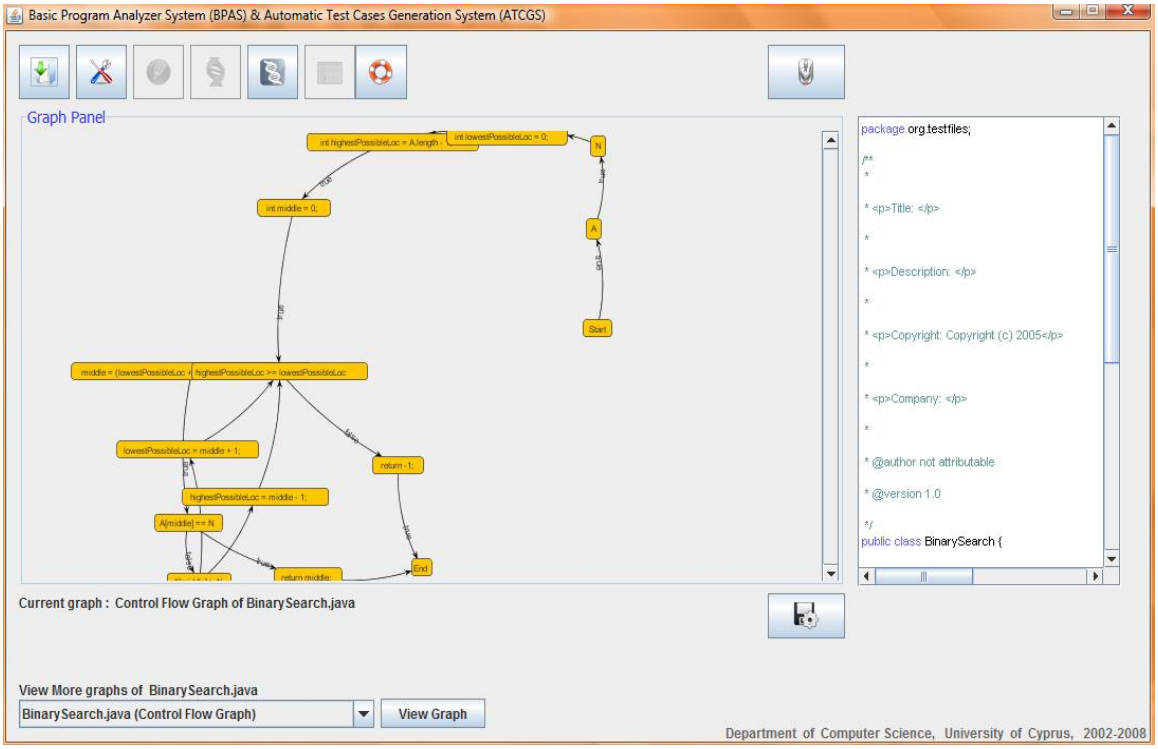
1. Αν το αρχείο πηγαίου κώδικα έχει μεταφορτωθεί
2. Αν η ανάλυση (parsing) του πηγαίου κώδικα έχει ολοκληρωθεί
3. Αν ο γράφος έχει κατασκευαστεί
4. Αν η οπτική-γραφική απεικόνιση του γράφου (Graph Visualization Viewer) έχει ολοκληρωθεί και τοποθετηθεί στο πάνελ των γράφων (Graph Panel).

Πιο κάτω (Σχήμα 4.29) βλέπουμε ένα στιγμιότυπο αυτού του πληροφοριακού παραθύρου. Το εικονίδιο με το πράσινο τικ σημαίνει ότι η αντίστοιχη λειτουργία, που περιγράφεται στα αριστερά του έχει ολοκληρωθεί, ενώ το εικονίδιο με το ρολόι υποδεικνύει ότι η αντίστοιχη λειτουργία εκκρεμεί.



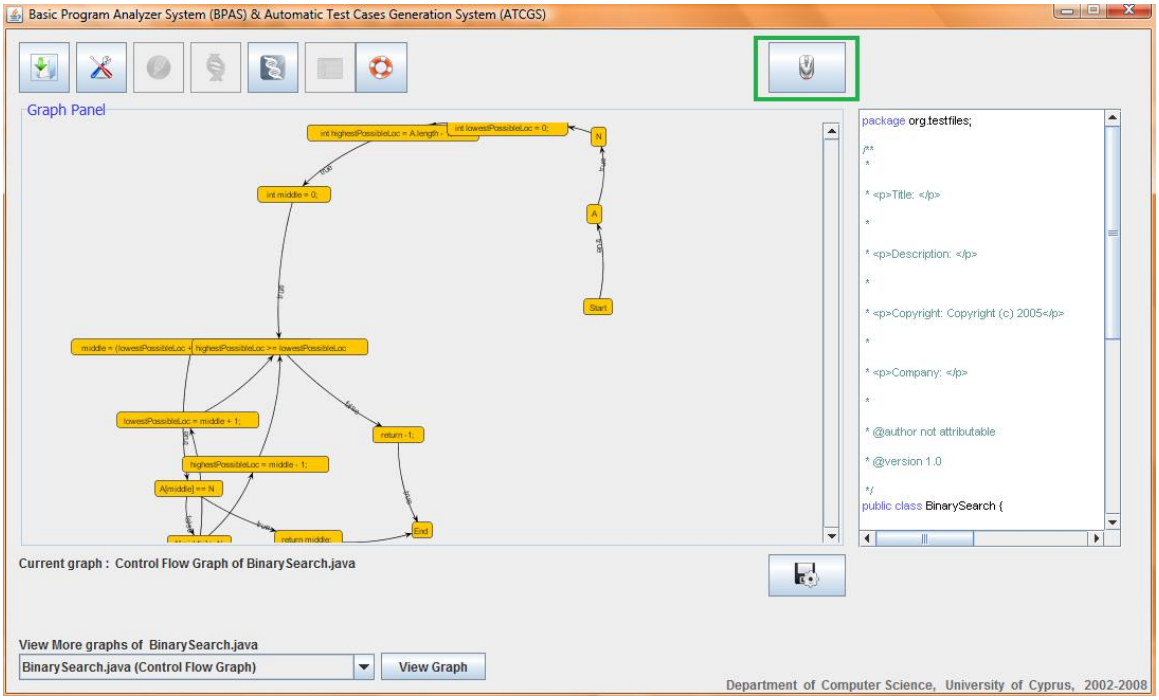
Σχήμα 4.29 : Πληροφορίες προόδου δημιουργίας γράφου

- Όταν όλες οι απαιτούμενες εργασίες ολοκληρωθούν μπορούμε να δούμε τον γράφο να απεικονίζεται στο πάνελ γράφων, όπως βλέπουμε και πιο κάτω (Σχήμα 4.30).



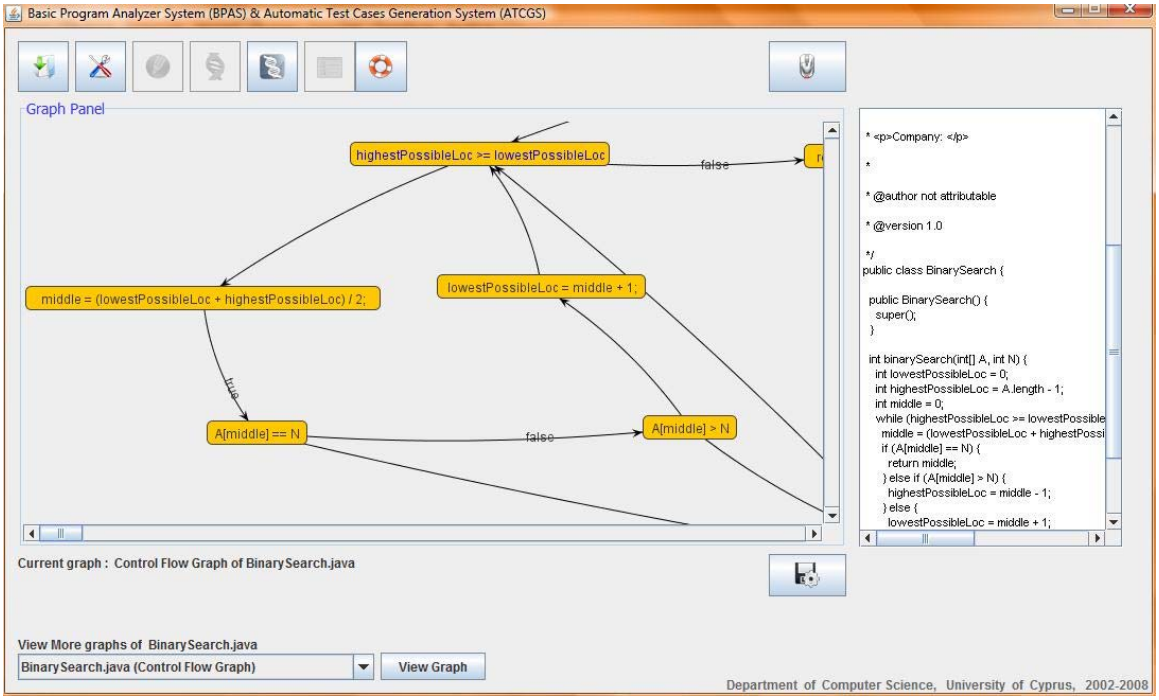
Σχήμα 4.30 : Παρουσίαση κατασκευασμένου γράφου

- Σε περίπτωση που ο γράφος είναι αρκετά μεγάλος ώστε να μην χωράει εξολοκλήρου στο πάνελ, τότε μπορούμε να χρησιμοποιήσουμε τις λειτουργίες μεγέθυνσης και σμίκρυνσης (zoom-in/out) με τη χρήση της ροδέλας του ποντικιού για να επικεντρωθούμε στα τμήματα του γράφου που μας ενδιαφέρουν. Σε επόμενο στάδιο, θα επιλέξουμε να αλλάξουμε τον τρόπο αλληλεπίδρασης με τον γράφο, σε graph-mouse αλληλεπίδραση (graph-mouse interaction), που μας επιτρέπει να επιλέγουμε μεμονωμένους κόμβους ή σύνολα κόμβων και να τα μετακινήσουμε ανεξάρτητα από τον υπόλοιπο γράφο για να μπορούμε να ξεκαθαρίσουμε πυκνούς γράφους και να μπορέσουμε να επικεντρωθούμε σε περιοχές ενδιαφέροντος. Η λειτουργία αυτή γίνεται από το κουμπί που φαίνεται πιο κάτω, στην επόμενη εικόνα (Σχήμα 4.31).



Σχήμα 4.31 : Κουμπί αλλαγής τρόπου αλληλεπίδρασης με τον γράφο

- Έχοντας αλλάξει τον τρόπο αλληλεπίδρασης, μπορούμε να μετακινήσουμε τους κόμβους που μας ενδιαφέρουν. Στο σενάριο αυτό θεωρούμε ότι μας ενδιαφέρουν οι κόμβοι με τον έλεγχο και την ανάθεση που φαίνονται πιο κάτω :
  highestPossibleLoc >= lowestPossibleLoc)
  και
  middle = (lowestPossibleLoc + highestPossibleLoc) / 2;
  τους κόμβους των οποίων μετακινούμε πάνω αριστερά στο πάνελ όπως φαίνεται στην πιο κάτω εικόνα (Σχήμα 4.32), και κάνουμε μεγέθυνση στο σημείο αυτό.



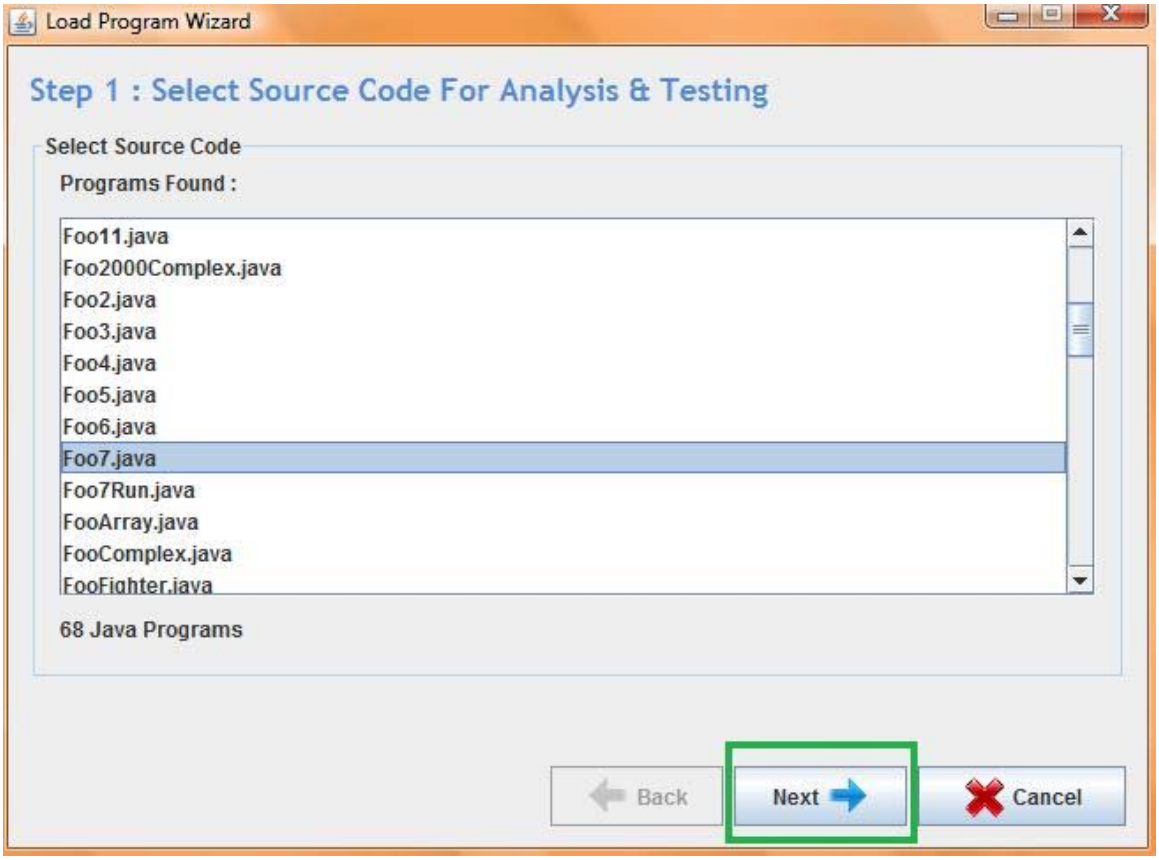
Σχήμα 4.32 : Αποτέλεσμα μετακίνησης κόμβων και μεγέθυνσης

- Στο σημείο αυτό έχει ολοκληρωθεί το παρόν σενάριο.

II. Σενάριο δεύτερο : Στο σενάριο χρήσης αυτό, θέτουμε ως στόχο τη δημιουργία γράφου ροής δεδομένων (Data Flow Graph) για τον πηγαίο κώδικα Foo7.java, την προβολή πληροφοριών που αφορούν τους κόμβους και τις ακμές του γράφου, καθώς και πληροφορίες σχετικές με τα μονοπάτια που παράγονται από τον γράφο ροής δεδομένων με βάση το k-tuples κριτήριο. Στη συνέχεια θα δούμε τις ρυθμίσεις που αφορούν τους γενετικούς αλγορίθμους για την παραγωγή περιπτώσεων ελέγχου, θα τροποποιήσουμε μερικές ρυθμίσεις και τέλος θα τρέξουμε τους γενετικούς αλγορίθμους και θα δούμε τα αποτελέσματα της παραγωγής περιπτώσεων ελέγχου.

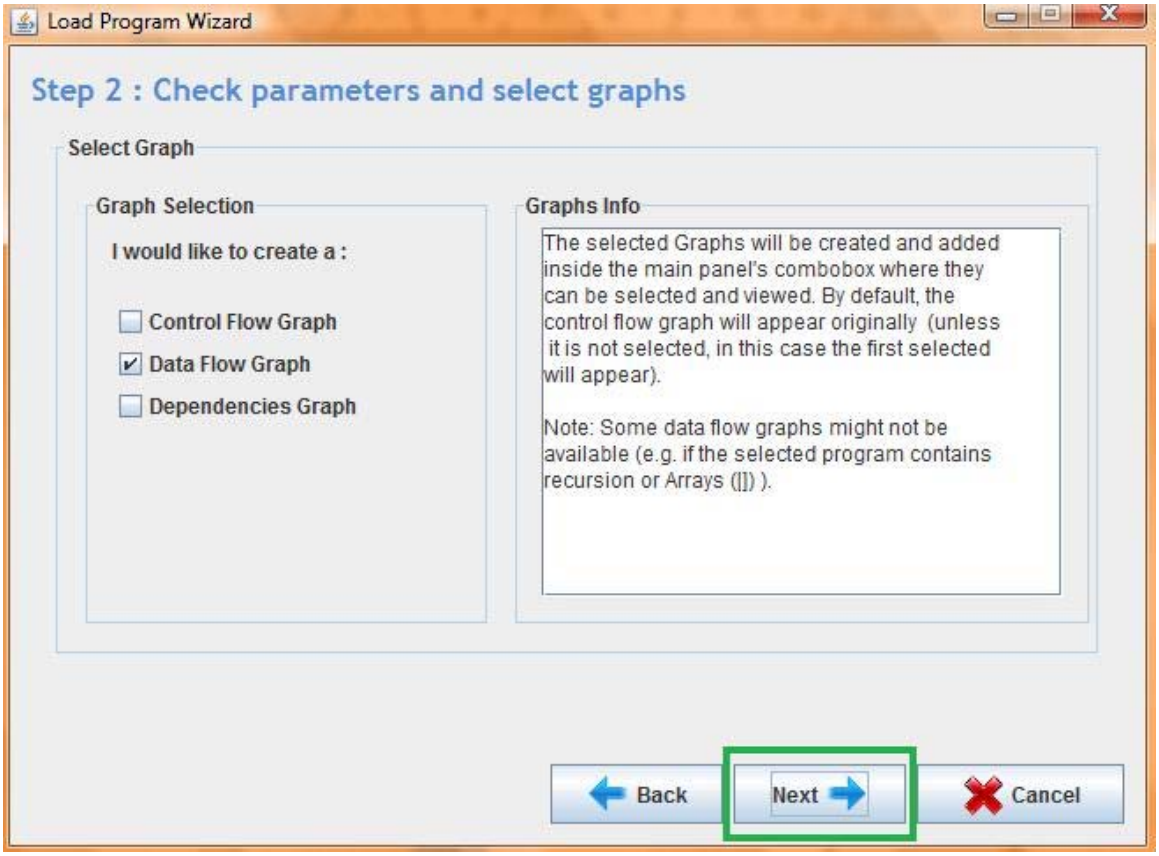
- Πρώτο βήμα είναι να ανοίξουμε τον Οδηγό Φόρτωσης-Ανοίγματος αρχείου, όπως και στο προηγούμενο σενάριο και να επιλέξουμε το αρχείο Foo7.java, όπως

μπορούμε να δούμε στην πιο κάτω εικόνα (Σχήμα 4.33). Πατάμε το κουμπί "Next" για να προχωρήσουμε στο δεύτερο βήμα του Οδηγού.



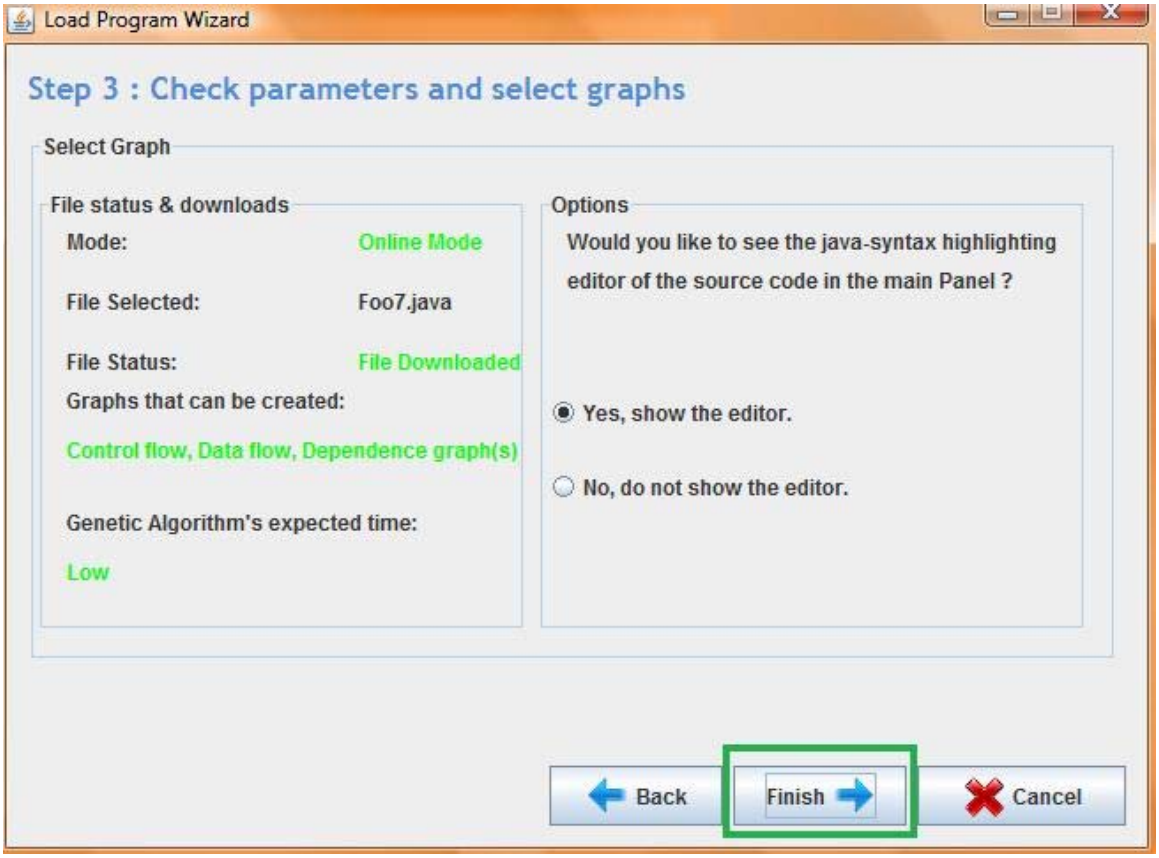
Σχήμα 4.33 : Επιλογή προγράμματος πηγαίου κώδικα

- Στο δεύτερο βήμα, επιλέγουμε την δημιουργία γράφου ροής δεδομένου μόνο, και πατάμε το κουμπί "Next" για να συνεχίσουμε στο τρίτο βήμα, όπως φαίνεται και πιο κάτω (Σχήμα 4.34).



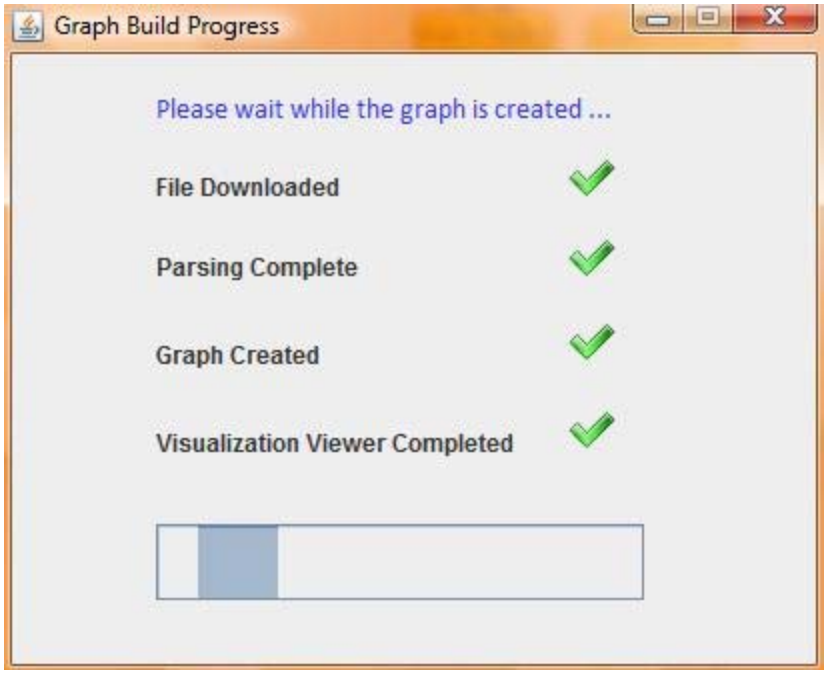
Σχήμα 4.34 : Επιλογή γράφου ροής δεδομένων

- Στο τρίτο βήμα του Οδηγού, βλέπουμε την κατάσταση του επιλεγμένου αρχείου, Foo7.java, όπως και την κατάσταση της σύνδεσης. Έχοντας ελέγξει ότι όλες οι παράμετροι είναι σωστές (Σχήμα 4.35), πατάμε το κουμπί “Finish”.



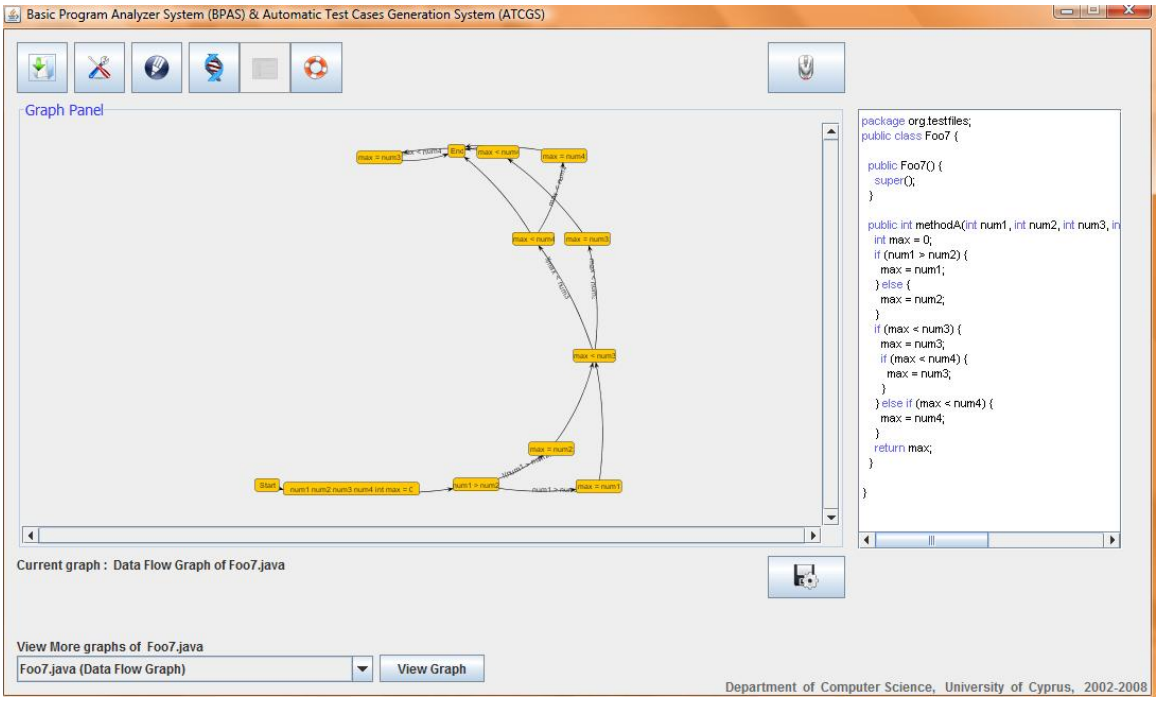
Σχήμα 4.35 : Έλεγχος παραμέτρων και επιλογή δημιουργίας του γράφου

- Όταν δούμε ότι το παράθυρο προόδου της ανάλυσης του κώδικα και δημιουργίας γράφου υποδεικνύει ότι όλες οι εργασίες έχουν ολοκληρωθεί (Σχήμα 4.36), και το παράθυρο αυτό κλείσει αυτόματα, όπως βλέπουμε πιο κάτω, θα έχει φορτωθεί ο γράφος στο πάνελ



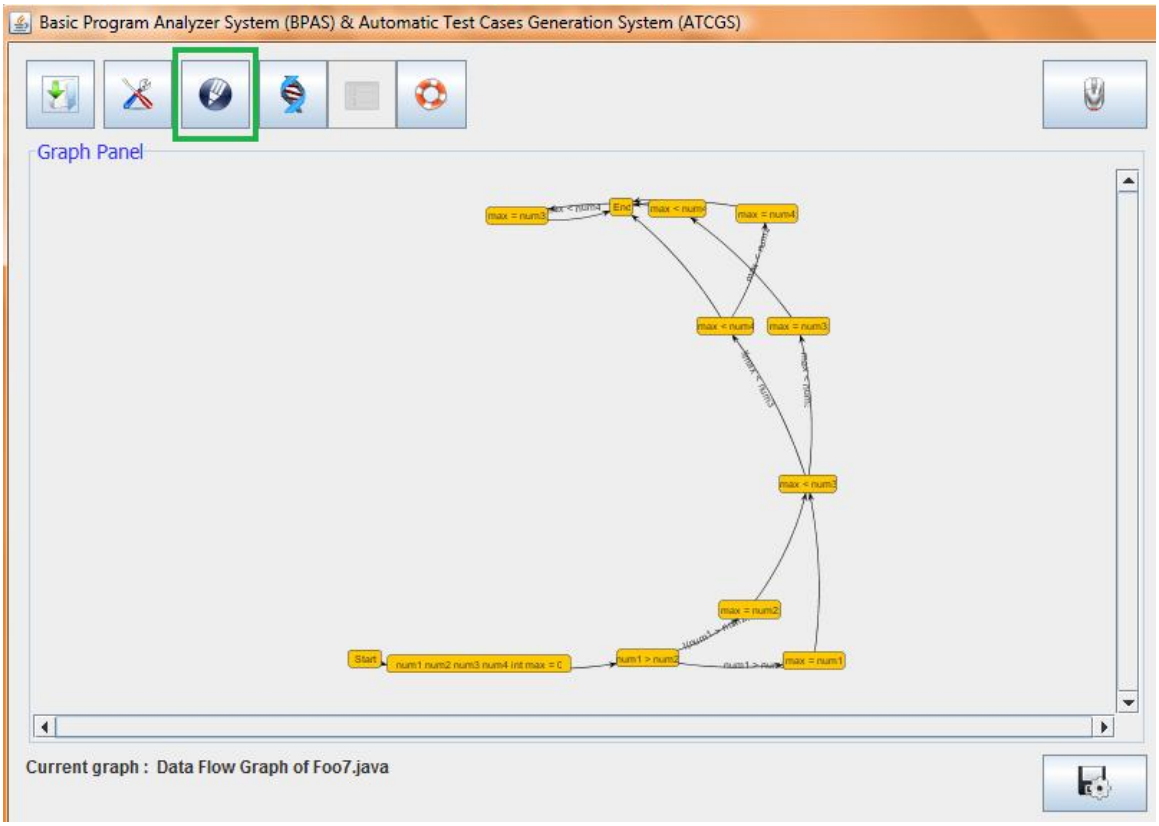
Σχήμα 4.36 : Πρόοδος δημιουργίας γράφου

- Στο σημείο αυτό μπορούμε να δούμε στο πάνελ τον γράφο ροής δεδομένων για το πρόγραμμα Foo7.java (Σχήμα 4.37).



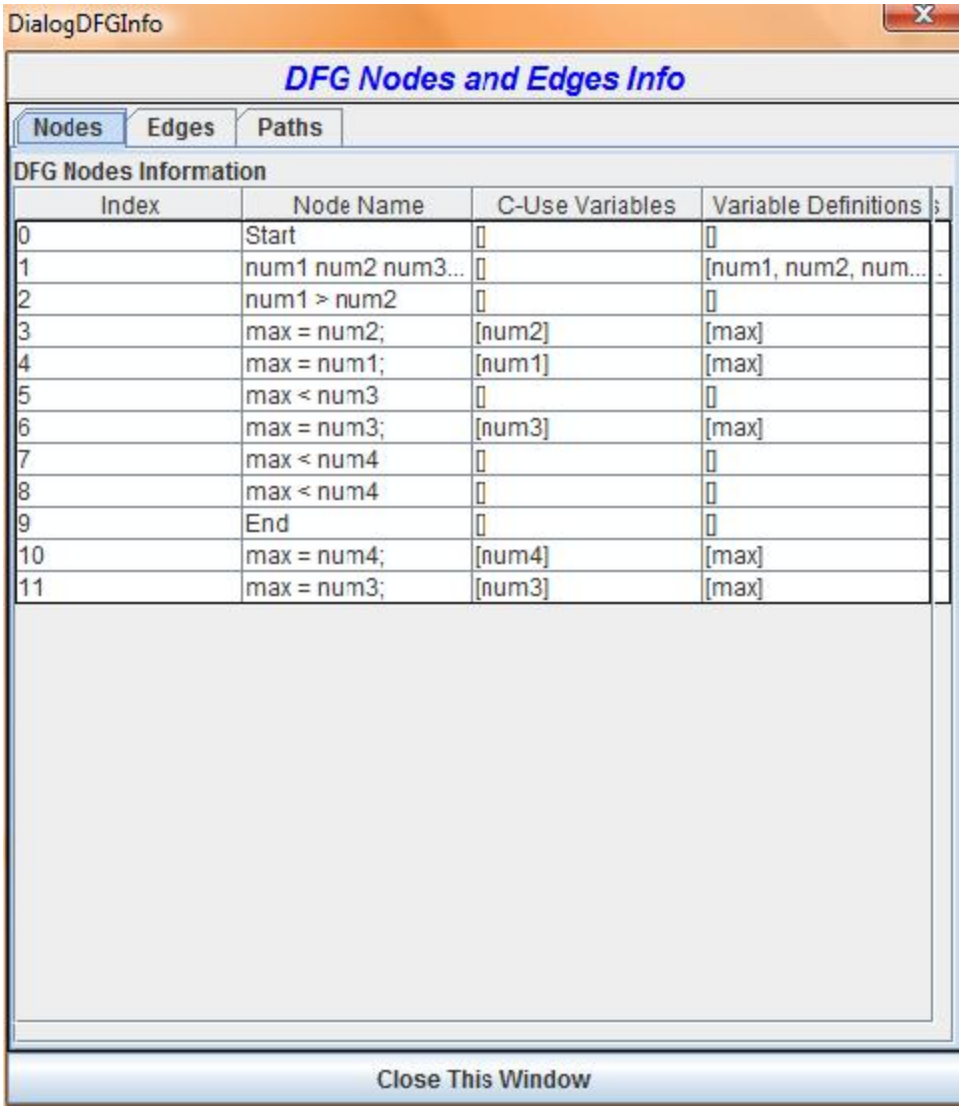
Σχήμα 4.37 : Παρουσίαση γράφου ροής δεδομένων

- Στο βήμα αυτό, με βάση το σενάριο χρήσης, θα ανοίξουμε το παράθυρο με τις πληροφορίες που αφορούν τους κόμβους, τις ακμές και τα μονοπάτια του γράφου όπως αυτά εξάγονται με βάση το k-tuples κριτήριο. Οι πληροφορίες αυτές είναι διαθέσιμες στο κουμπί που φαίνεται στην πιο κάτω εικόνα (Σχήμα 4.38).



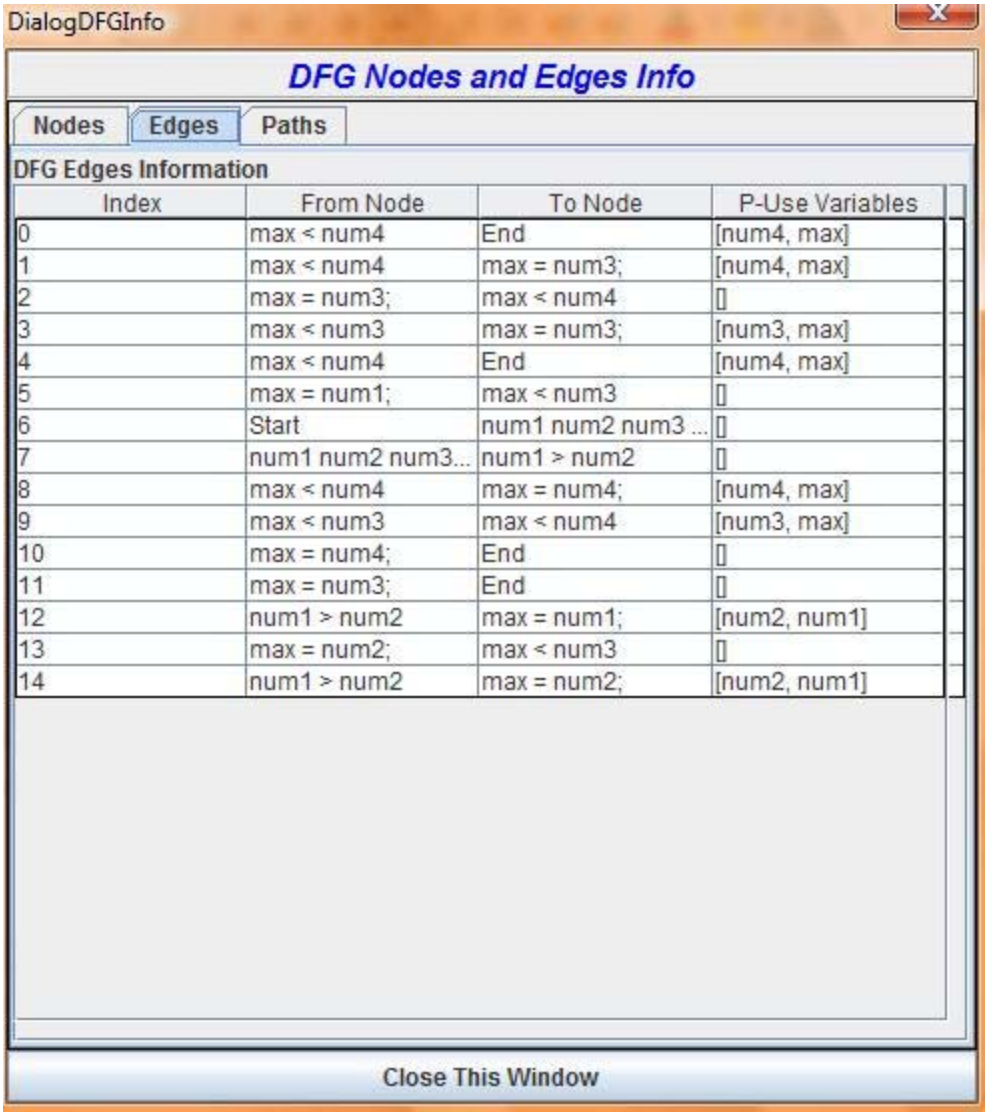
Σχήμα 4.38 : Κουμπί για παρουσίαση πληροφοριών γράφου ροής δεδομένων

- Πατώντας το κουμπί αυτό, θα δούμε το παράθυρο με τις πληροφορίες να αναδύεται στην οθόνη, με πρώτο πλάνο να φαίνονται οι πληροφορίες των κόμβων, όπως μπορούμε να δούμε στη συνέχεια (Σχήμα 4.39). Στις πληροφορίες των κόμβων περιλαμβάνονται το όνομα του κόμβου, τις C-Use μεταβλητές και τις δηλώσεις μεταβλητών (Variable Definitions). Ακόμη, κάθε κόμβος έχει ανατεθεί σε έναν μοναδικό αριθμό, όπως φαίνεται στην πρώτη στήλη (index). Αυτός ο μοναδικός αριθμός είναι χρήσιμος για τον προσδιορισμό κόμβων και ακμών αντί της χρήσης του ονόματός τους το οποίο μπορεί να είναι μια μακροσκελής δήλωση.



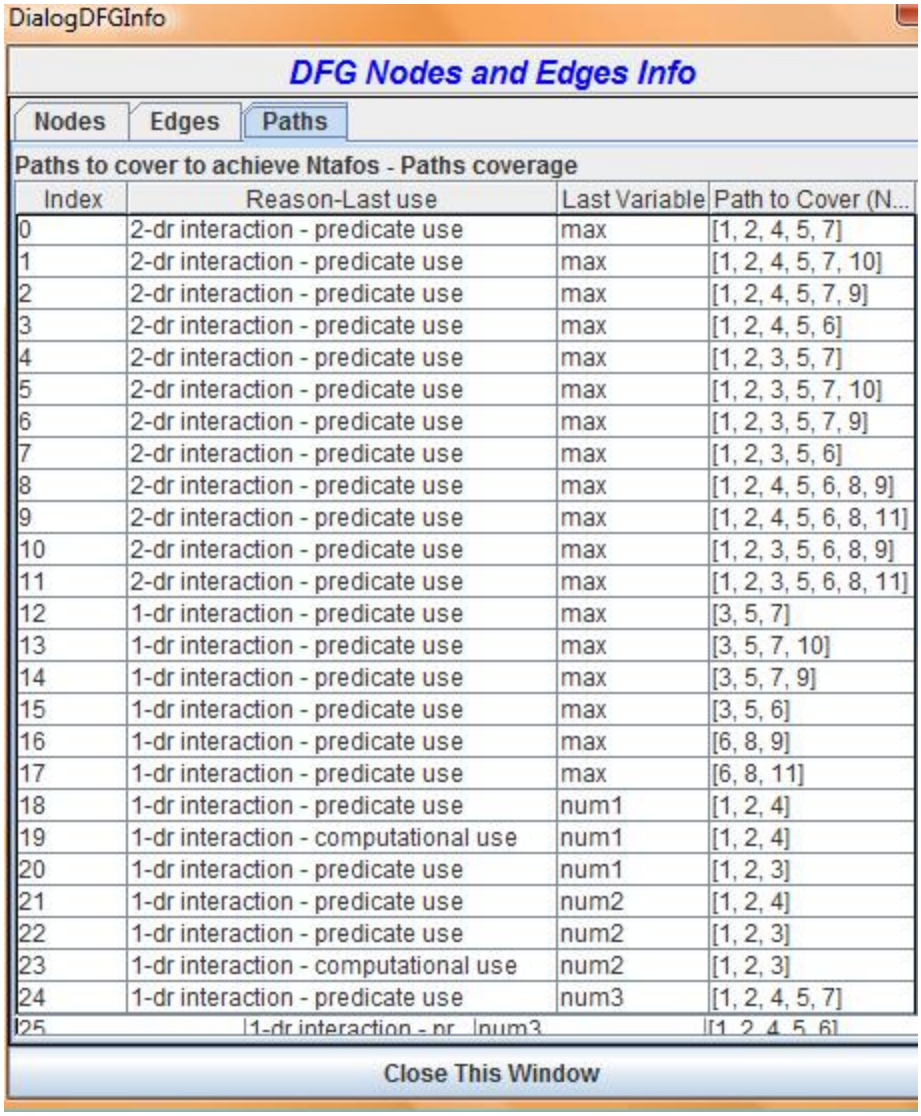
Σχήμα 4.39 : Πληροφορίες κόμβων DFG

- Στη συνέχεια θα δούμε τις πληροφορίες που αφορούν τις ακμές (Σχήμα 4.40). Στις πληροφορίες αυτές περιλαμβάνονται ο μοναδικός αριθμός που αντιστοιχεί σε κάθε ακμή (index) , ο κόμβος από τον οποίο πηγάζει η ακμή, ο κόμβος προορισμού της ακμής και οι P-Use μεταβλητές.



Σχήμα 4.40 : Πληροφορίες ακμών DFG

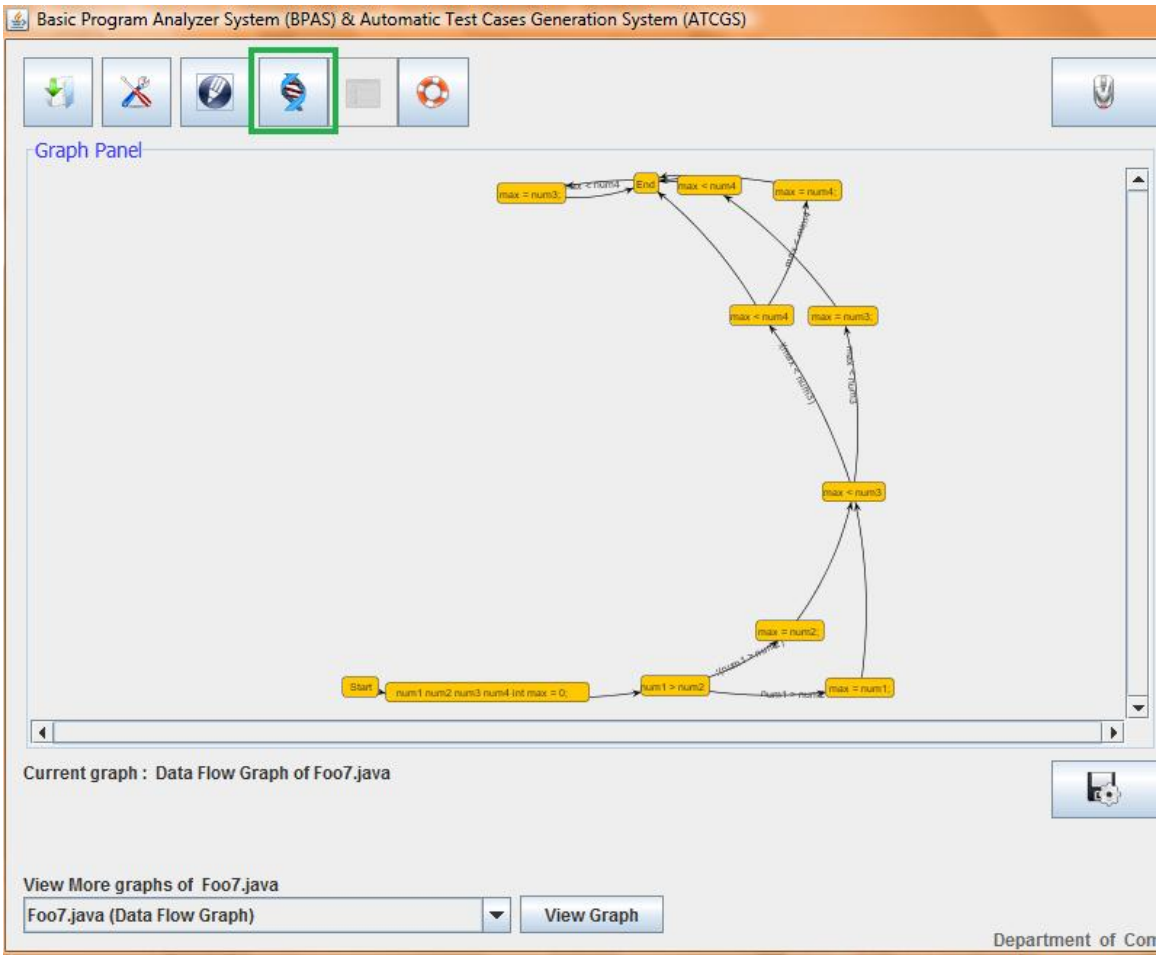
- Τέλος, θα δούμε τις πληροφορίες για τα μονοπάτια που εξάγονται από το k-tuples criterion (Σχήμα 4.41). Οι πληροφορίες αυτές αφορούν το λόγο τελευταίας χρήσης και την αλληλεπίδραση, την τελευταία μεταβλητή, και τη σειρά κόμβων που συνθέτουν το μονοπάτι που θα πρέπει να καλυφθεί.



Index	Reason-Last use	Last Variable	Path to Cover (N...
0	2-dr interaction - predicate use	max	[1, 2, 4, 5, 7]
1	2-dr interaction - predicate use	max	[1, 2, 4, 5, 7, 10]
2	2-dr interaction - predicate use	max	[1, 2, 4, 5, 7, 9]
3	2-dr interaction - predicate use	max	[1, 2, 4, 5, 6]
4	2-dr interaction - predicate use	max	[1, 2, 3, 5, 7]
5	2-dr interaction - predicate use	max	[1, 2, 3, 5, 7, 10]
6	2-dr interaction - predicate use	max	[1, 2, 3, 5, 7, 9]
7	2-dr interaction - predicate use	max	[1, 2, 3, 5, 6]
8	2-dr interaction - predicate use	max	[1, 2, 4, 5, 6, 8, 9]
9	2-dr interaction - predicate use	max	[1, 2, 4, 5, 6, 8, 11]
10	2-dr interaction - predicate use	max	[1, 2, 3, 5, 6, 8, 9]
11	2-dr interaction - predicate use	max	[1, 2, 3, 5, 6, 8, 11]
12	1-dr interaction - predicate use	max	[3, 5, 7]
13	1-dr interaction - predicate use	max	[3, 5, 7, 10]
14	1-dr interaction - predicate use	max	[3, 5, 7, 9]
15	1-dr interaction - predicate use	max	[3, 5, 6]
16	1-dr interaction - predicate use	max	[6, 8, 9]
17	1-dr interaction - predicate use	max	[6, 8, 11]
18	1-dr interaction - predicate use	num1	[1, 2, 4]
19	1-dr interaction - computational use	num1	[1, 2, 4]
20	1-dr interaction - predicate use	num1	[1, 2, 3]
21	1-dr interaction - predicate use	num2	[1, 2, 4]
22	1-dr interaction - predicate use	num2	[1, 2, 3]
23	1-dr interaction - computational use	num2	[1, 2, 3]
24	1-dr interaction - predicate use	num3	[1, 2, 4, 5, 7]
25	1-dr interaction - pr...	num3	[1, 2, 4, 5, 6]

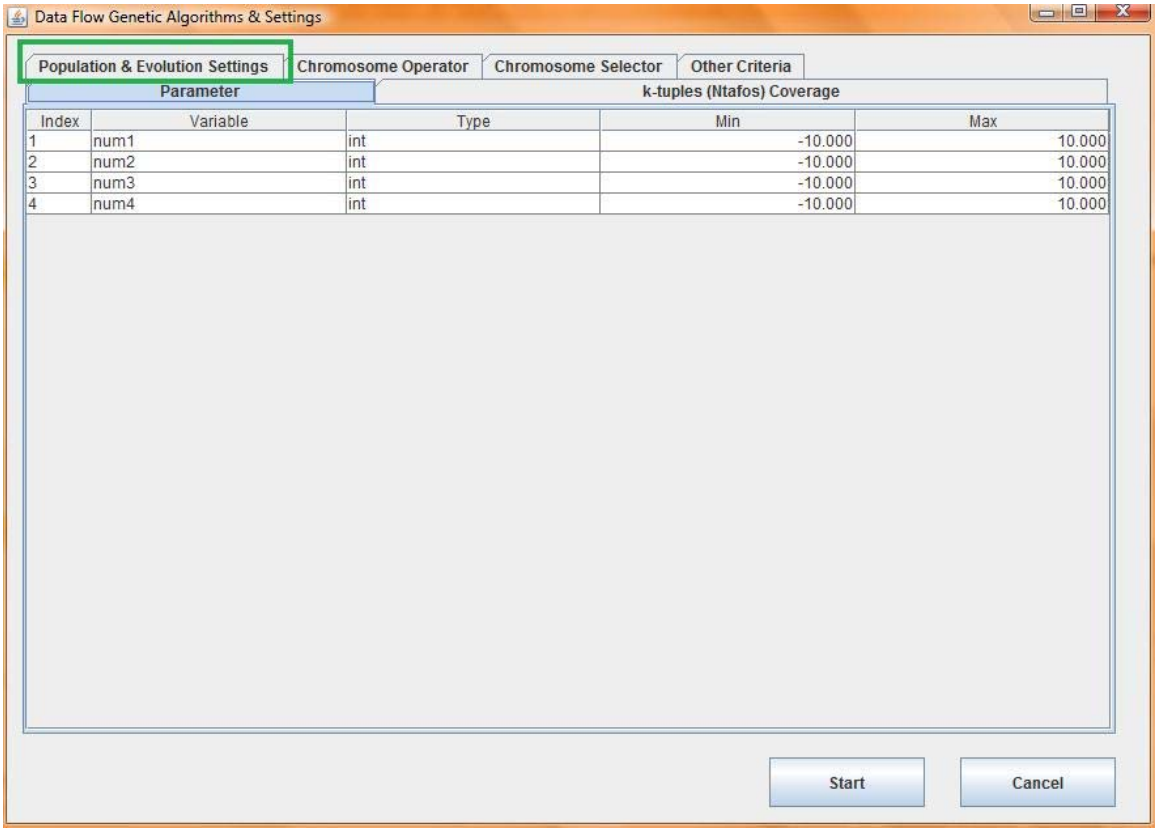
Σχήμα 4.41 : Πληροφορίες μονοπατιών DFG (k-tuples criterion)

- Στη συνέχεια στο σενάριο, ανοίγουμε τις ρυθμίσεις που αφορούν την παραγωγή περιπτώσεων ελέγχου με τους γενετικούς αλγορίθμους (Σχήμα 4.42). Οι ρυθμίσεις αυτές μπορούν να ανοίξουν και να τροποποιηθούν από το κουμπί που φαίνεται πιο κάτω



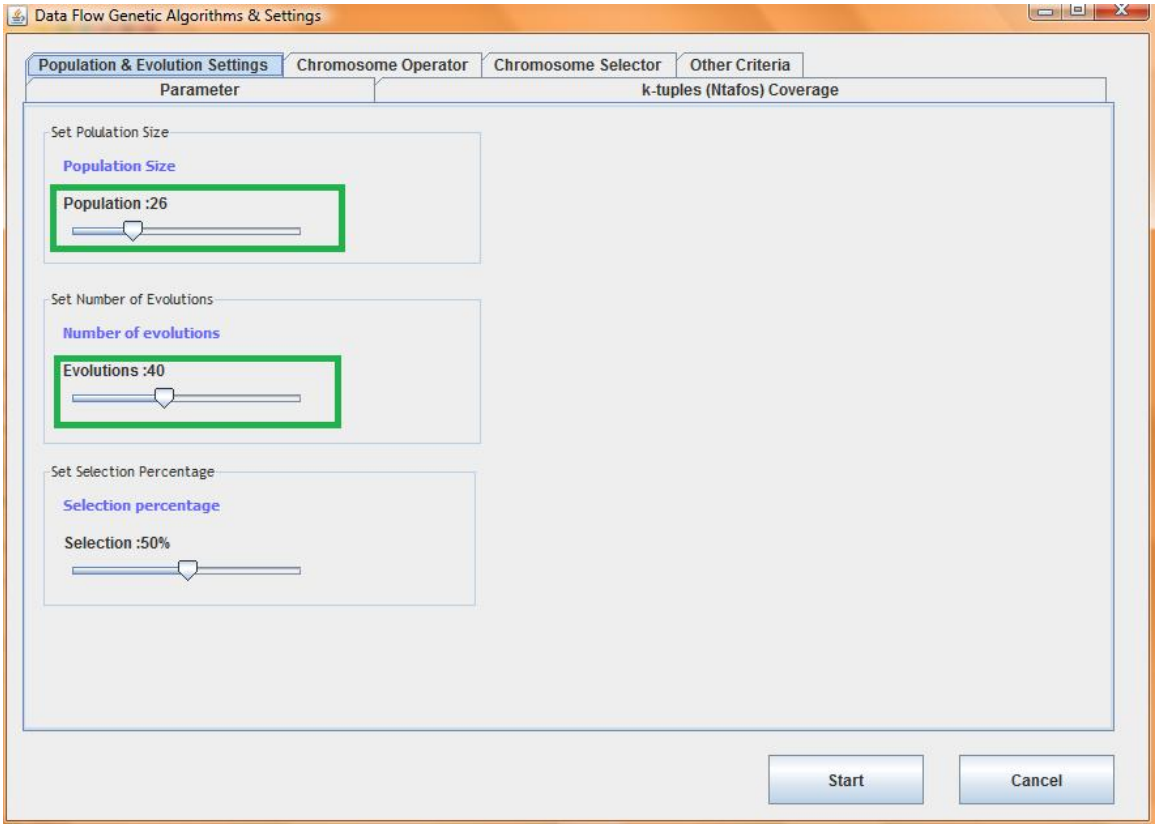
Σχήμα 4.42 : Κουμπί για εκτέλεση GA με βάση DFG

- Μπορούμε να δούμε στην επόμενη εικόνα το παράθυρο με όλες τις ρυθμίσεις που αφορούν τον τρόπο που θα τρέξουν οι γενετικοί αλγόριθμοι και τις παραμέτρους που μπορούν να τροποποιηθούν από τον χρήστη. Σε αυτό το σενάριο, μας ενδιαφέρουν οι ρυθμίσεις που βρίσκονται κάτω από την κατηγορία “Population & Evolution Settings” και σε αυτήν θα μεταβούμε όπως φαίνεται πιο κάτω (Σχήμα 4.43).



Σχήμα 4.43 : Πληροφορίες και παράμετροι GA

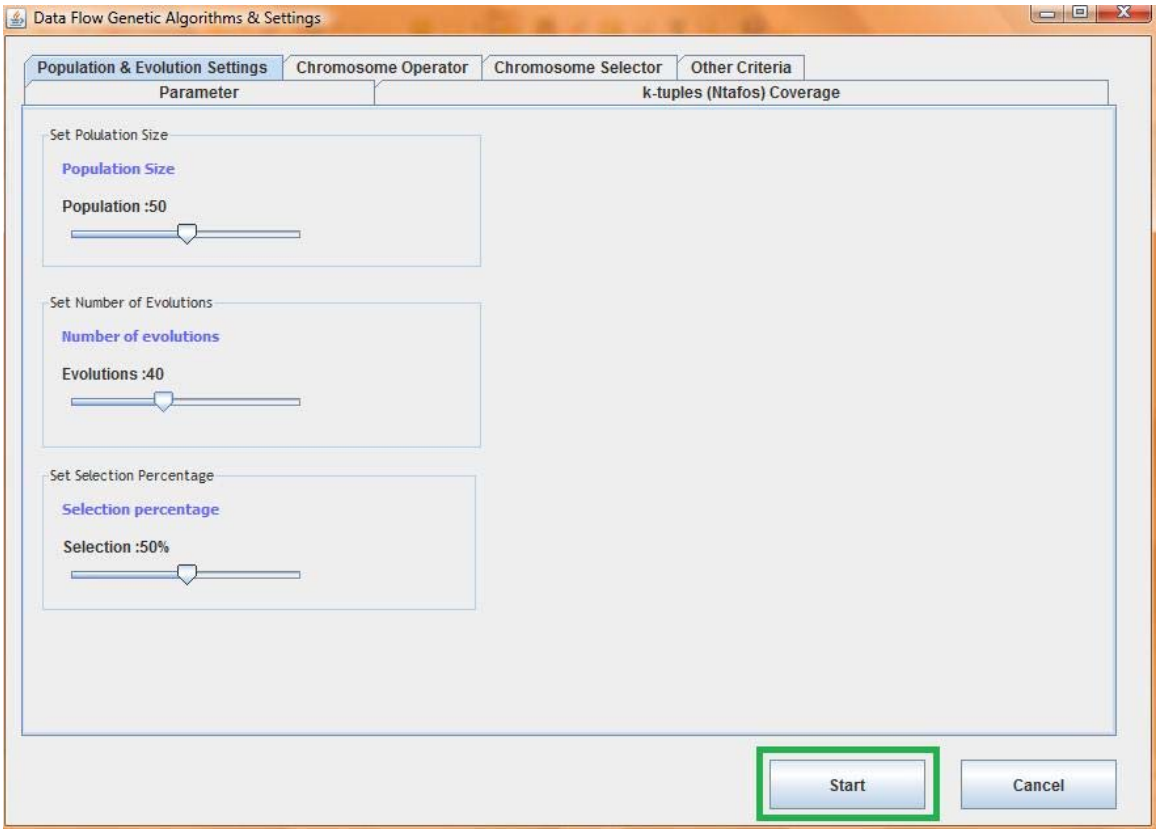
- Στην κατηγορία με τις ρυθμίσεις αυτές μπορούμε να βρούμε παραμέτρους όπως το μέγεθος του πληθυσμού, τον αριθμό των εξελίξεων (evolutions) και το ποσοστό της φυσικής επιλογής. Το συγκεκριμένο σενάριο χρήσης θέλει να αλλάξει ο αριθμός των εξελίξεων από 500 που είναι η προεπιλογή σε 40. Αυτό θα γίνει μετακινώντας την αντίστοιχη μπάρα (scrollbar) μέχρι τον αριθμό 40, όπως φαίνεται πιο κάτω (Σχήμα 4.44). Παράλληλα, μειώνουμε το μέγεθος του πληθυσμού από 50 σε 26.



Σχήμα 4.44 : Αλλαγή μεγέθους πληθυσμού και αριθμού εξελίξεων

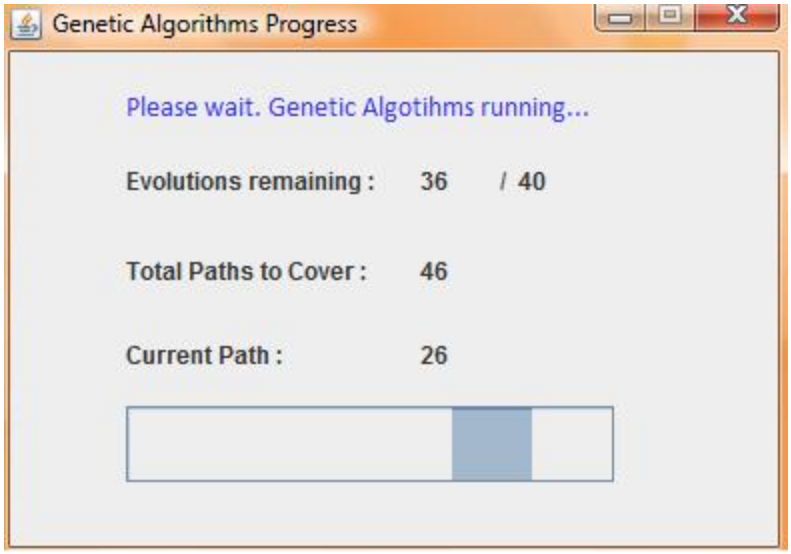
- Έχοντας αλλάξει τον αριθμό των εξελίξεων σε 40, κάνοντας την παραγωγή περιπτώσεων ελέγχου με αυτό τον τρόπο συντομότερη καθώς οι γενετικοί αλγόριθμοι αναμένονται να τερματίσουν συντομότερα, αποδεχόμενοι το ρίσκο το ποσοστό κάλυψης του κώδικα να είναι μειωμένο. Έπειτα πατάμε το κουμπί “Start” για να ξεκινήσει η παραγωγή περιπτώσεων ελέγχου, όπως θα δούμε πιο κάτω (Σχήμα 4.45).

.



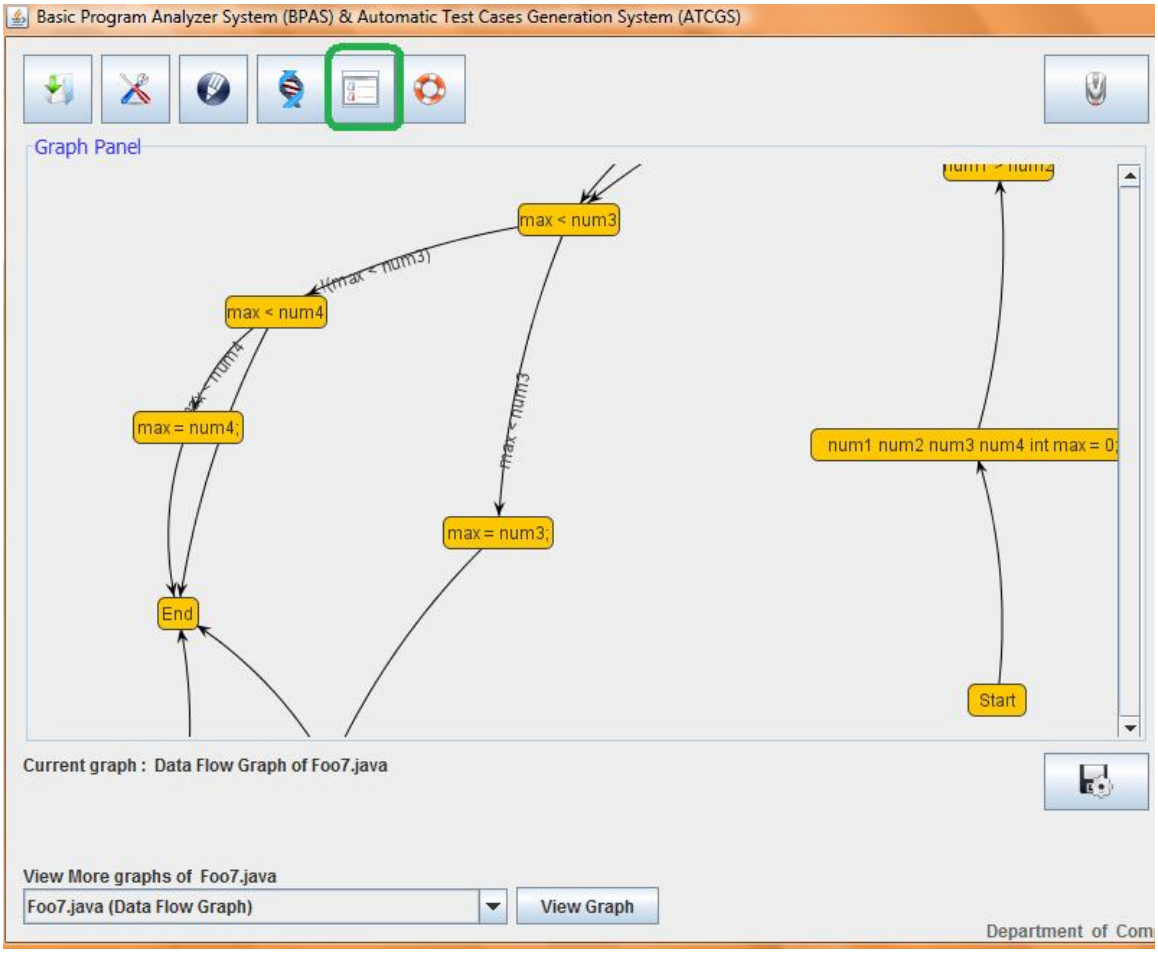
Σχήμα 4.45 : Κουμπί έναρξης εκτέλεσης GA

- Όταν οι γενετικοί αλγόριθμοι ξεκινήσουν την εκτέλεση, θα δούμε ένα παράθυρο διαλόγου με πληροφορίες για την τρέχουσα κατάσταση τους, όπως αποτυπώνεται και στην επόμενη εικόνα (Σχήμα 4.46).
- Οι πληροφορίες αυτές όπως μπορούμε να δούμε περιλαμβάνουν τον αριθμό των συνολικών μονοπατιών που θα πρέπει να καλυφθούν, τον αριθμό του τρέχοντος μονοπατιού και τον αριθμό της εξέλιξης στην οποία βρίσκονται ανά πάσα στιγμή. Πληροφορίες χρήσιμες, ειδικά όταν τα μονοπάτια που πρέπει να καλυφθούν είναι πολλά, κάτι που δεν ισχύει στο σενάριο αυτό.



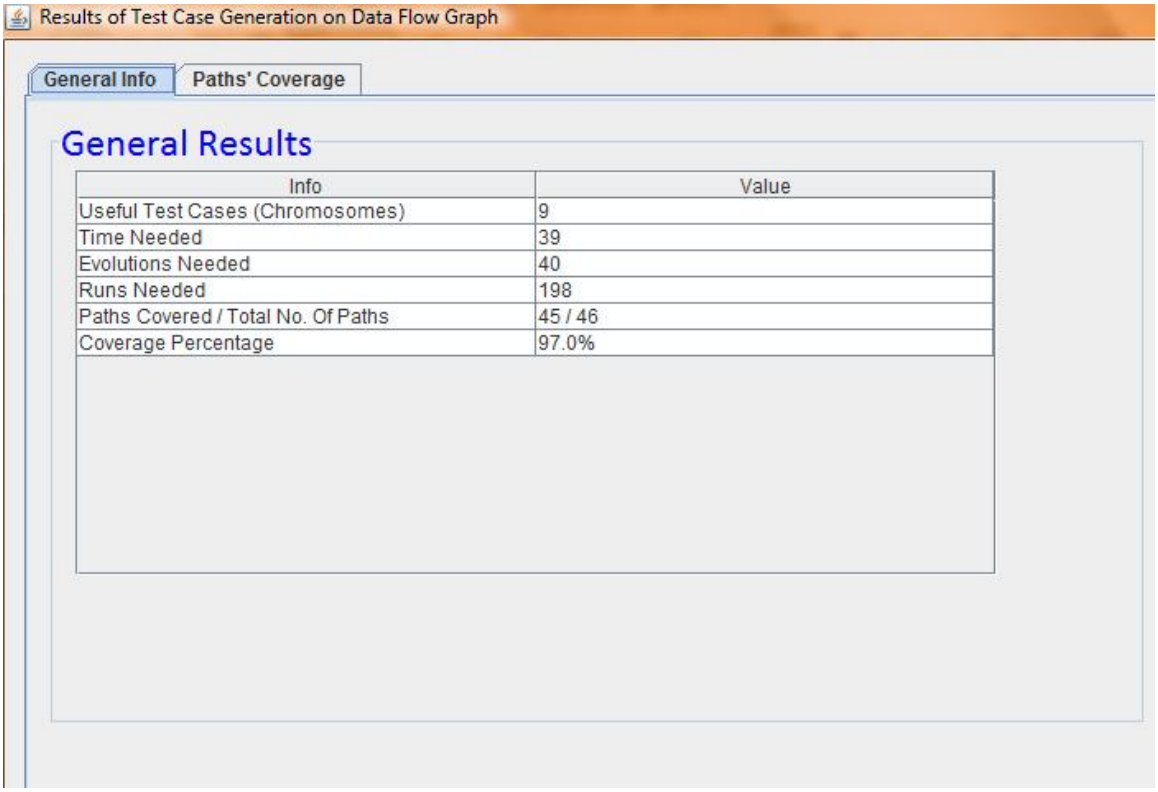
Σχήμα 4.46 : Παράθυρο προόδου εκτέλεσης GA

- Όταν η παραγωγή περιπτώσεων ελέγχου έχει ολοκληρωθεί, θα ακουστεί ένας σύντομος ενημερωτικός ήχος (System Beep), υποδηλώνοντας ότι οι γενετικοί αλγόριθμοι τελείωσαν την εκτέλεσή τους και τα αποτελέσματα είναι διαθέσιμα, και μπορούμε να τα δούμε πατώντας το κουμπί που φαίνεται πιο κάτω (Σχήμα 4.47).



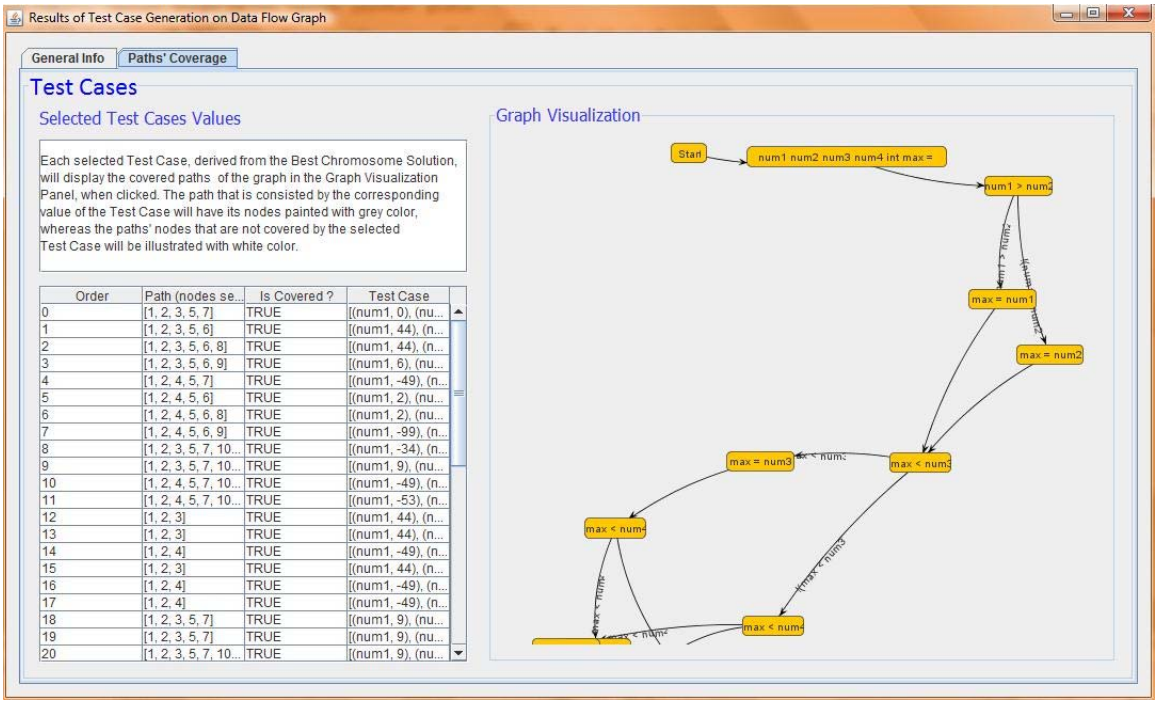
Σχήμα 4.47 : Κουμπί παρουσίασης αποτελεσμάτων GA για DFG

Πατώντας το κουμπί που φαίνεται πιο πάνω, θα ανοίξει το παράθυρο με τα συνοπτικά και αναλυτικά αποτελέσματα της παραγωγής περιπτώσεων ελέγχου. Όπως θα δούμε στην επόμενη εικόνα (Σχήμα 4.48), στα συνοπτικά αποτελέσματα βλέπουμε ότι οι χρήσιμες περιπτώσεις ελέγχου (χρωμοσώματα) ήταν 9, ο χρόνος εκτέλεσης των αλγορίθμων ήταν 39 δευτερόλεπτα, ο αριθμός των απαιτούμενων εξελίξεων ήταν 40, οι εκτελέσεις που απαιτήθηκαν 198, και ότι καλύφθηκαν 45 από τα 46 μονοπάτια, με αποτέλεσμα να έχουμε ποσοστό κάλυψης 97%.



Σχήμα 4.48 : Γενικές πληροφορίες - αποτελέσματα GA

- Στην επόμενη καρτέλα, με το όνομα “Paths' Info” μπορούμε να δούμε αναλυτικά τις χρήσιμες περιπτώσεις ελέγχου που παράχθηκαν και τα μονοπάτια πάνω στον γράφο τα οποία χρωματίζονται κατ' αντιστοιχία με την επιλεγμένη περίπτωση ελέγχου, όπως βλέπουμε πιο κάτω (Σχήμα 4.49).



Σχήμα 4.49 : Παράθυρο με αναλυτικές πληροφορίες περιπτώσεων ελέγχου

- Σύμφωνα με το σενάριο αυτό, θα πρέπει να επιλέξουμε κάποια περίπτωση ελέγχου από τη λίστα που βρίσκεται στα αριστερά του γράφου, και να δούμε τις τιμές που έλαβε ως είσοδο και το μονοπάτι που καλύφθηκε. Τυχαία λοιπόν επιλέγουμε το μονοπάτι με αριθμό 13. Το μονοπάτι αυτό αποτελείται από τους κόμβους [1, 2, 3]. Δηλαδή όσον αφορά τον κώδικα τους τρεις παρακάτω κόμβους:
    - *int num1, int num2, int num3, int num4, int max = 0*
    - *num1 > num2*
    - *max = num1*

  που αντιστοιχούν φυσικά στους μοναδικούς αριθμούς 1,2,3, όπως μπορούμε να δούμε από το κουμπί με τις πληροφορίες για τους κόμβους-ακμές-μονοπάτια. Η περίπτωση ελέγχου 13 αναθέτει τις παρακάτω τιμές :

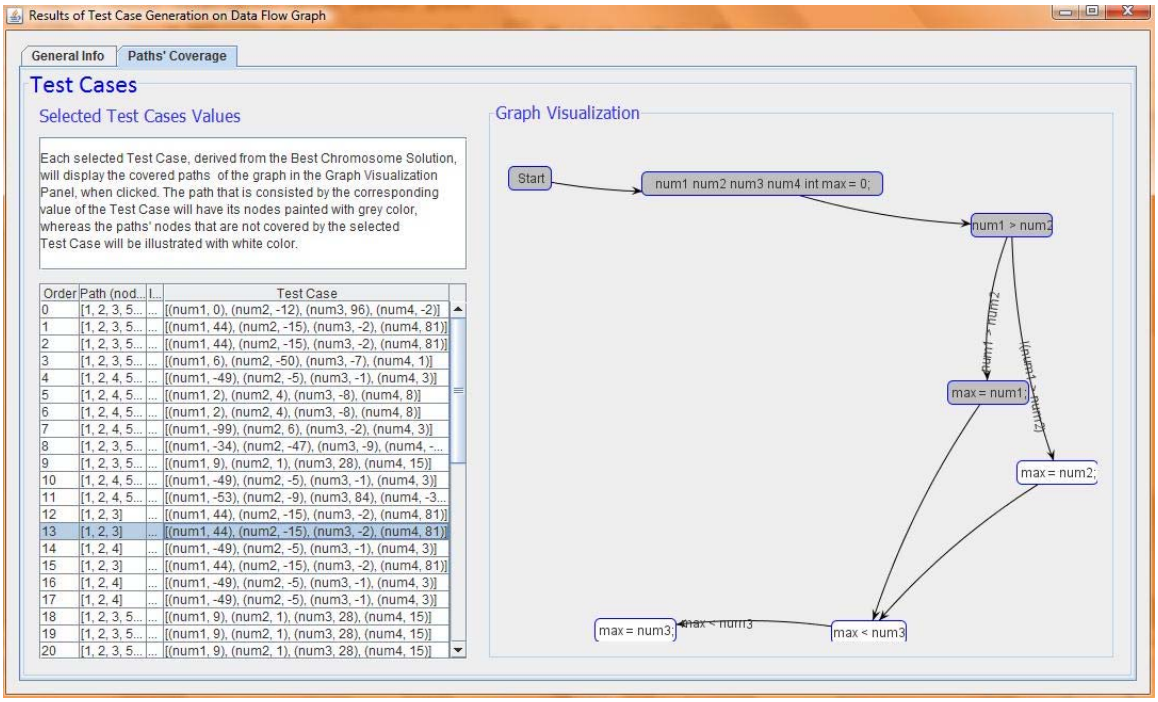
i. num1 = 44

ii. num2 = -15

iii. num3 = -2

iv. num4 = 81

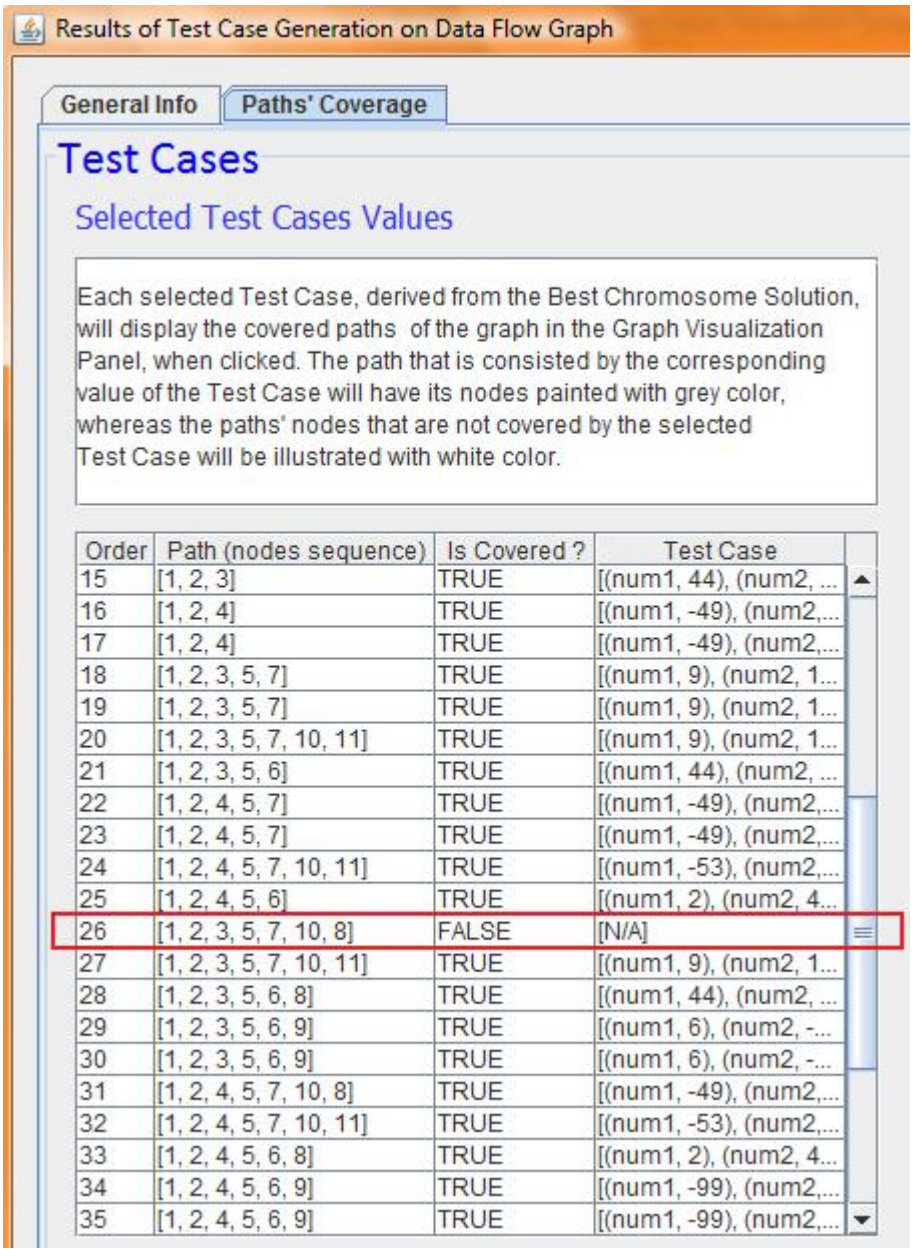
έτσι, όπως μπορούμε να δούμε και στην επόμενη εικόνα(Σχήμα 4.50), ο κόμβος 1 έχει καλυφθεί, αφού περιέχει τις αρχικοποιήσεις, η δεύτερη επίσης γιατί είναι έλεγχος που ισχύει, εφόσον num1 = 44 > num2 = -15. Τέλος, ο κόμβος 3 , του μονοπατιού θα καλυφθεί ο οποίος κάνει την ανάθεση max = num1. Στα δεξιά του παραθύρου, στο πάνελ που βρίσκεται ο γράφος βλέπουμε ότι όταν επιλέξουμε το test case 13, το αντίστοιχο μονοπάτι ([1,2,3]) θα χρωματιστεί με γκρι, αφού έχει καλυφθεί, ενώ ο υπόλοιπος γράφος θα έχει χρώμα άσπρο. Κατά σύμβαση οι κόμβοι αρχής και τέλους (Start, End) χρωματίζονται γκρι όταν στο σύνολο κόμβων που αποτελούν το μονοπάτι υπάρχει τουλάχιστον ένας κόμβος που να είναι γειτονικός με τον κόμβο Start ή με τον κόμβο End αντίστοιχα



Σχήμα 4.50 : Επιλογή περίπτωσης ελέγχου 13 και αποτύπωση στο γράφο

- Αξίζει να παρατηρήσουμε ότι το μοναδικό μονοπάτι που δεν καλύφθηκε, ήταν το μονοπάτι με αριθμό 26 (Σχήμα 4.51), αποτελούμενο από τους κόμβους [1,2,3,5,7,10,8].

Για το λόγο αυτό έχουμε 45 / 46 μονοπάτια καλυμμένα και ποσοστό κάλυψης 97%. Βέβαια αυτό έγινε σκόπιμα για να επεξηγηθεί ο τρόπος που δουλεύει το εργαλείο. Αυξάνοντας το μέγεθος του πληθυσμού και τον αριθμό των εξελίξεων θα πετύχουμε κάλυψη 100%.



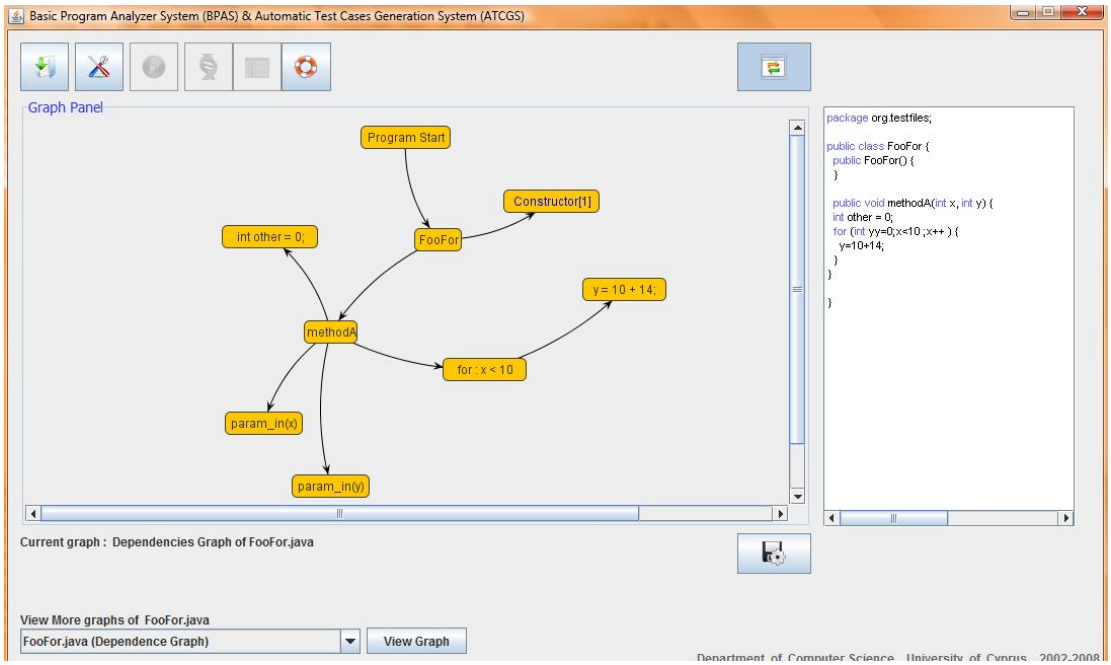
Order	Path (nodes sequence)	Is Covered ?	Test Case
15	[1, 2, 3]	TRUE	[(num1, 44), (num2, ...
16	[1, 2, 4]	TRUE	[(num1, -49), (num2,...
17	[1, 2, 4]	TRUE	[(num1, -49), (num2,...
18	[1, 2, 3, 5, 7]	TRUE	[(num1, 9), (num2, 1...
19	[1, 2, 3, 5, 7]	TRUE	[(num1, 9), (num2, 1...
20	[1, 2, 3, 5, 7, 10, 11]	TRUE	[(num1, 9), (num2, 1...
21	[1, 2, 3, 5, 6]	TRUE	[(num1, 44), (num2, ...
22	[1, 2, 4, 5, 7]	TRUE	[(num1, -49), (num2,...
23	[1, 2, 4, 5, 7]	TRUE	[(num1, -49), (num2,...
24	[1, 2, 4, 5, 7, 10, 11]	TRUE	[(num1, -53), (num2,...
25	[1, 2, 4, 5, 6]	TRUE	[(num1, 2), (num2, 4...
26	[1, 2, 3, 5, 7, 10, 8]	FALSE	[N/A]
27	[1, 2, 3, 5, 7, 10, 11]	TRUE	[(num1, 9), (num2, 1...
28	[1, 2, 3, 5, 6, 8]	TRUE	[(num1, 44), (num2, ...
29	[1, 2, 3, 5, 6, 9]	TRUE	[(num1, 6), (num2, -...
30	[1, 2, 3, 5, 6, 9]	TRUE	[(num1, 6), (num2, -...
31	[1, 2, 4, 5, 7, 10, 8]	TRUE	[(num1, -49), (num2,...
32	[1, 2, 4, 5, 7, 10, 11]	TRUE	[(num1, -53), (num2,...
33	[1, 2, 4, 5, 6, 8]	TRUE	[(num1, 2), (num2, 4...
34	[1, 2, 4, 5, 6, 9]	TRUE	[(num1, -99), (num2,...
35	[1, 2, 4, 5, 6, 9]	TRUE	[(num1, -99), (num2,...

Σχήμα 4.51 : Επιλογή μονοπατιού 26, το οποίο δεν καλύφθηκε

- Στο σημείο αυτό έχει ολοκληρωθεί το σενάριο χρήσης.

III. Σενάριο τρίτο : Στο σενάριο αυτό θα δημιουργήσουμε ένα γράφο απαιτήσεων (Dependence Graph) του προγράμματος FooFor.java. Στη συνέχεια θα αλλάξουμε τον αλγόριθμο διάταξης γράφου στον κινούμενο αλγόριθμο Spring και θα δημιουργήσουμε για το ίδιο πρόγραμμα γράφο ροής δεδομένων και γράφο ροής ελέγχου ταυτόχρονα. Τέλος, θα αλληλεπιδράσουμε με έναν από αυτούς έχοντας αλλάξει τον τρόπο αλληλεπίδρασης σε GraphMouse Interaction και θα τραβήξουμε μερικούς κινούμενους κόμβους.

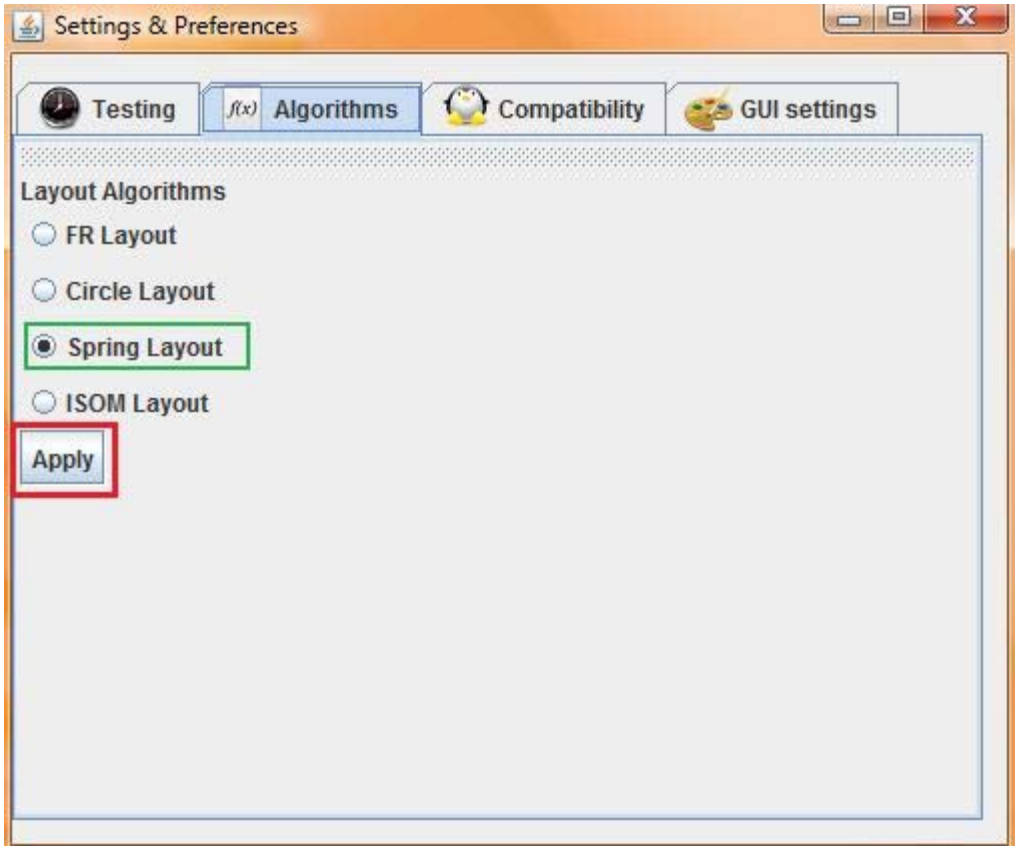
- Για λόγους συντομίας, δεν θα επαναλάβουμε τα προηγούμενα βήματα φόρτωσης αρχείου, καθώς είναι πανομοιότυπα με τα προηγούμενα σενάρια χρήσης. Έτσι, μεταπηδάμε στο σημείο που έχει δημιουργηθεί ο γράφος απαιτήσεων του FooFor.java και μπορούμε να το δούμε στην επόμενη εικόνα (Σχήμα 4.52).



Σχήμα 4.52 : Προβολή γράφου απαιτήσεων

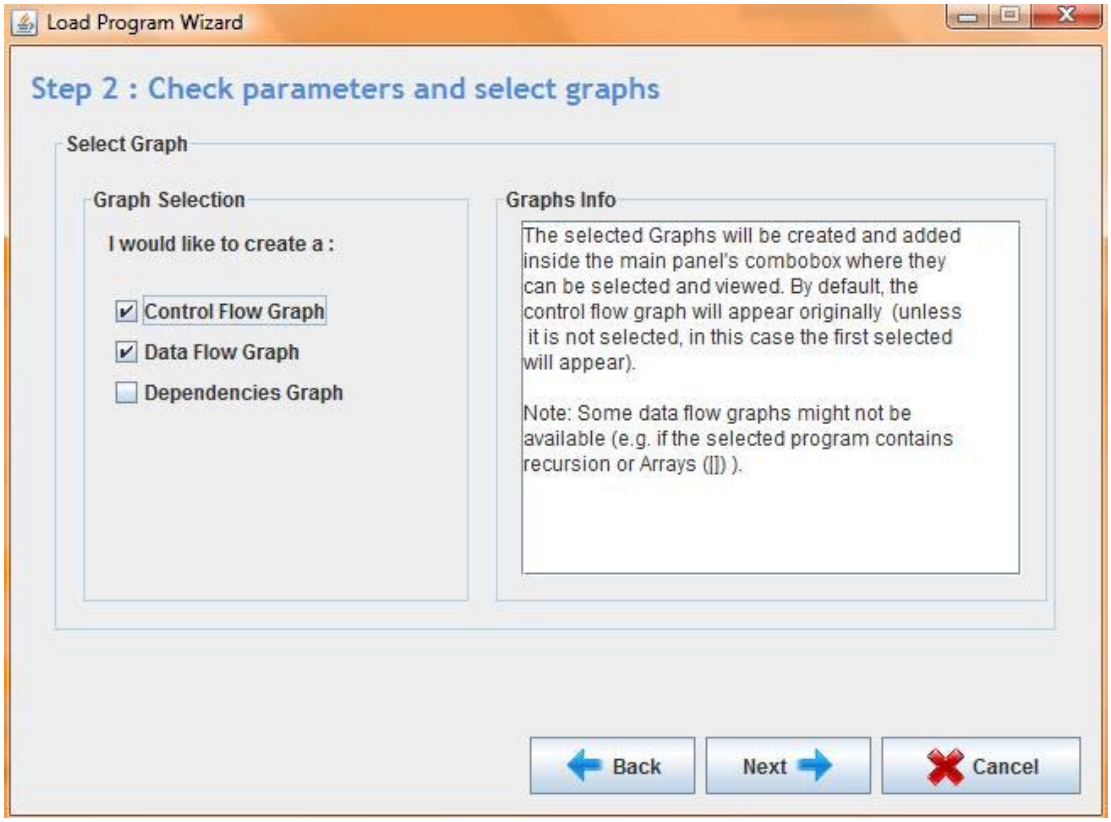
- Τώρα θα αλλάξουμε τον αλγόριθμο διάταξης γράφων από τις ρυθμίσεις, όπως κάναμε και στο σενάριο Ι . Αυτή τη φορά επιλέγουμε τον αλγόριθμο Spring και

πατάμε το πλήκτρο Apply για να αποθηκεύσουμε την αλλαγή, όπως δείχνει η επόμενη εικόνα (Σχήμα 4.53).



Σχήμα 4.53 : Καρτέλα αλλαγής αλγορίθμων διάταξης γράφων

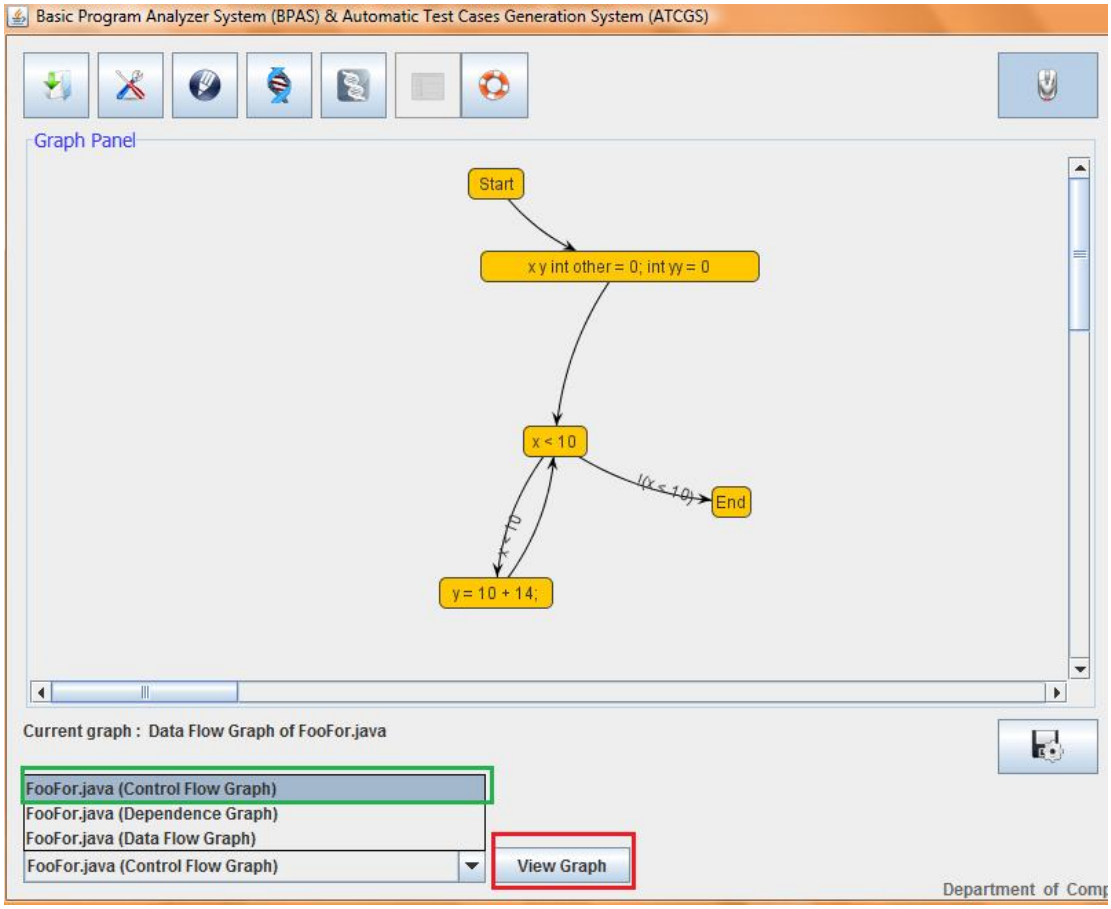
- Στη συνέχεια, θα ανοίξουμε ξανά τον Οδηγό Φόρτωσης-Ανοίγματος Αρχείου για να δημιουργήσουμε γράφο ροής ελέγχου και γράφο ροής δεδομένων ταυτόχρονα, και το μόνο που αξίζει να δούμε ως διαφορά με τις προηγούμενες διαδικασίες φόρτωσης αρχείου είναι ότι επιλέγουμε 2 είδη γράφων, όπως φαίνεται πιο κάτω (Σχήμα 4.54).



Σχήμα 4.54 : Επιλογή για ταυτόχρονη δημιουργία DFG και CFG

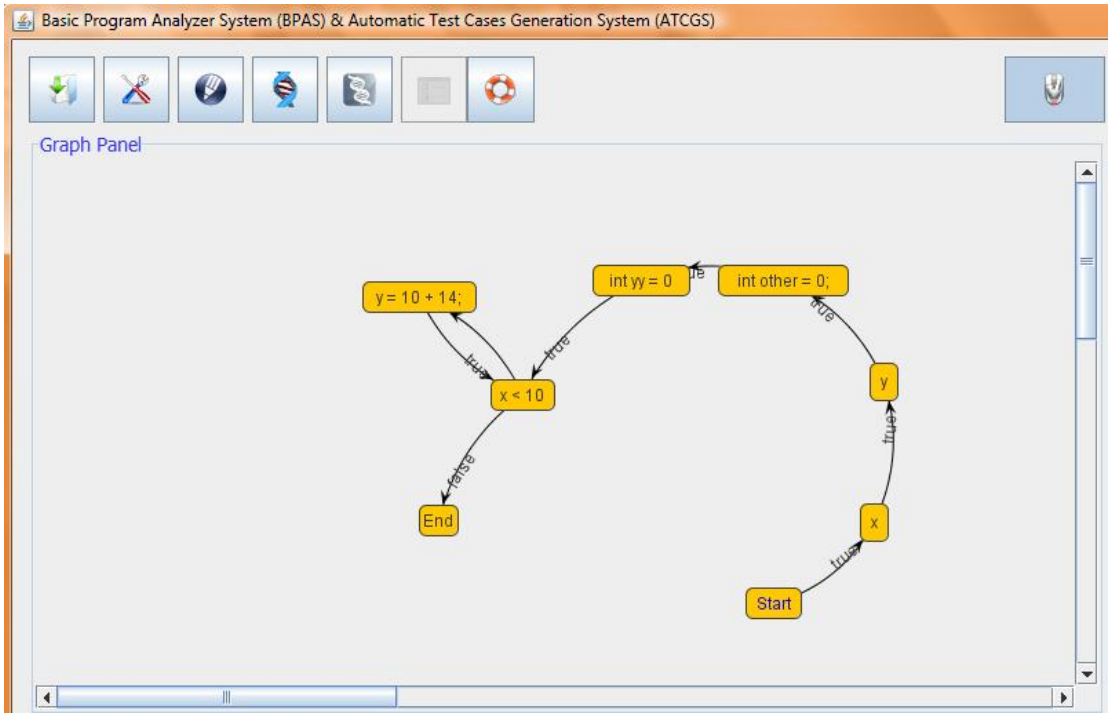
- Όταν οι γράφοι δημιουργηθούν, θα φορτωθούν με τη σειρά ολοκλήρωσής τους στο πάνελ των γράφων και άρα η γράφος που θα απεικονίζεται στο πάνελ θα είναι ο τελευταίος που κατασκευάστηκε. Όλοι οι γράφοι που δημιουργούνται ωστόσο είναι τοποθετημένοι στο graph combobox.

- Στη συνέχεια στο σενάριο θα επιλέξουμε (ανεξάρτητα από το ποιος γράφος απεικονίζεται στο πάνελ) τον γράφο ροής ελέγχου όπως φαίνεται στην πιο κάτω εικόνα (Σχήμα 4.55). Και πατάμε το κουμπί “View Graph” .



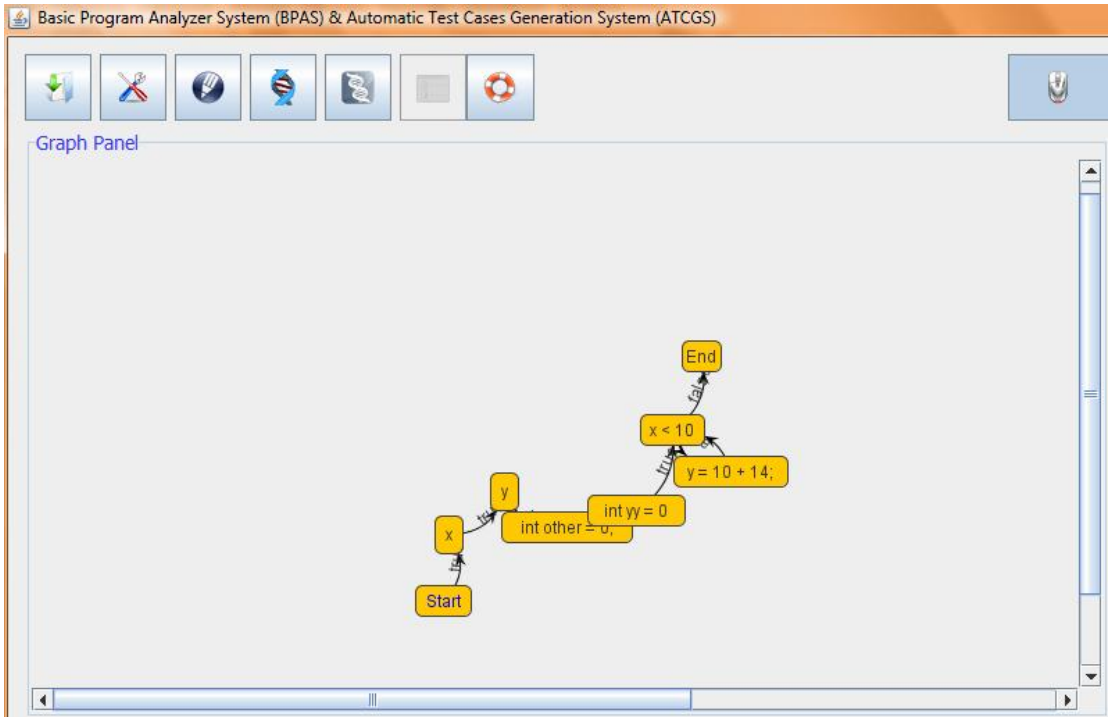
Σχήμα 4.55 : Επιλογή δημιουργημένου CFG για προβολή

- Μπορούμε να δούμε ότι ο γράφος που έχουμε επιλέξει φορτώνεται αμέσως στο πάνελ, χρησιμοποιώντας πάντα την έτοιμη γραφική απεικόνιση που είχε δημιουργηθεί. Άρα, όπως είναι αναμενόμενο, τη στιγμή δημιουργίας αυτού του γράφου είχαμε είδη αλλάξει τον αλγόριθμο διάταξης γράφου σε Spring Graph Layout από τις ρυθμίσεις και άρα αυτήν την απεικόνιση θα έχει. Να σημειωθεί ότι κάθε αλλαγή που επιφέρει ο χρήστης στην γραφική απεικόνιση του γράφου (πχ μεγέθυνση, μετακίνηση, περιστροφή, τράβηγμα κόμβου/κόμβων) αποθηκεύεται στην γραφική απεικόνιση του γράφου και χρησιμοποιείται η ίδια σε επόμενες επαναφορτώσεις του γράφου στο πάνελ.



Σχήμα 4.56 : Γραφική αναπαράσταση CFG με Spring Layout

- Στην πιο πάνω εικόνα (Σχήμα 4.56) βλέπουμε το Spring Layout για τον γράφο ροής του FooFor.java. Δυστυχώς σε ένα στιγμιότυπο δεν μπορεί να αποτυπωθεί η συνεχής κίνηση των κόμβων και ακμών του γράφου στα πρότυπα του αλγορίθμου. Καθώς ο γράφος κινείται διατηρώντας τις ισορροπίας που θέτει ο αλγόριθμος, θα αλλάξουμε την αλληλεπίδραση σε GraphMouse και θα τραβήξουμε προς τα κάτω έναν από τους κόμβους όπως ζητά το σενάριο, για παράδειγμα τον κόμβο Start. Το αποτέλεσμα της ενέργειας αυτής φαίνεται στην επόμενη εικόνα (Σχήμα 4.57).

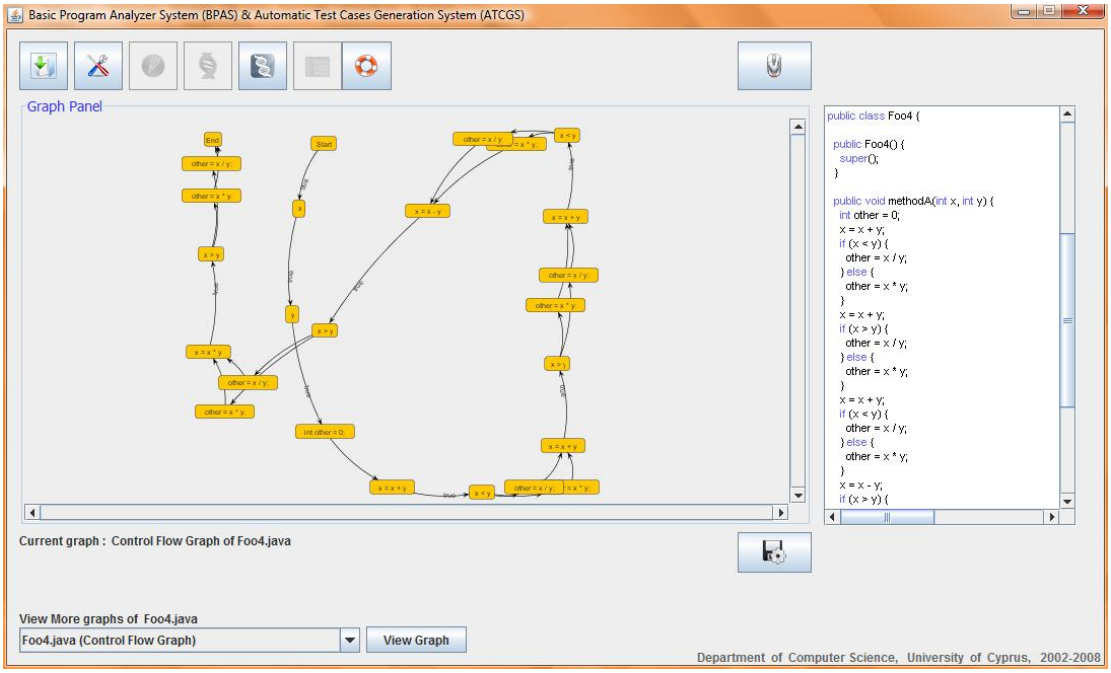


Σχήμα 4.57 : Μετακίνηση κόμβου Start.

- Και εδώ το σενάριο χρήσης αυτό έχει ολοκληρωθεί.

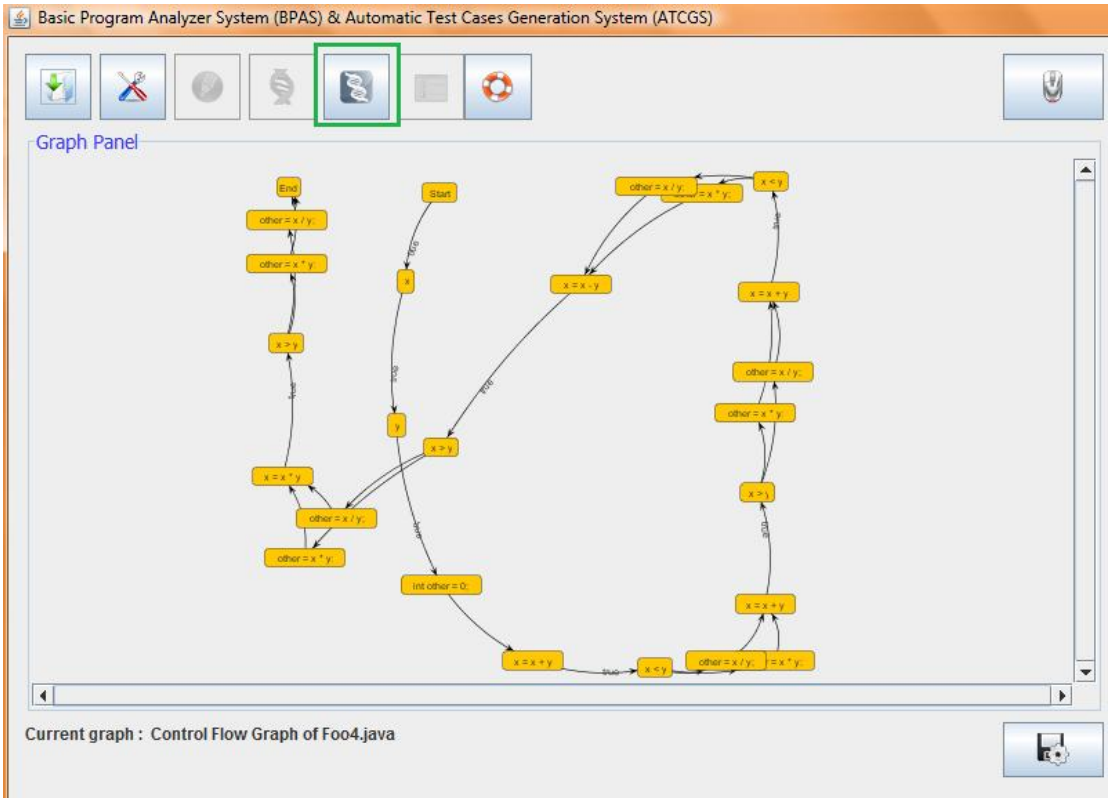
IV. Σενάριο τέταρτο : Δημιουργία γράφου ροής ελέγχου (Control Flow Graph) για το πρόγραμμα Foo4.java , εκτέλεση των γενετικών αλγορίθμων για την παραγωγή περιπτώσεων ελέγχου, και παρατήρηση των αποτελεσμάτων.

- Με το γνωστό τρόπο δημιουργούμε τον γράφο ροής για το πρόγραμμα Foo4.java, με τον Load Wizard. Κρίνεται σκόπιμο να μην επαναλάβουμε ξανά τα βήματα αυτά.
- Παρακάμπτοντας τα βήματα δημιουργίας του γράφου, φτάνουμε στο σημείο που ο γράφος έχει δημιουργηθεί και τοποθετηθεί στο πάνελ (Σχήμα 4.58).



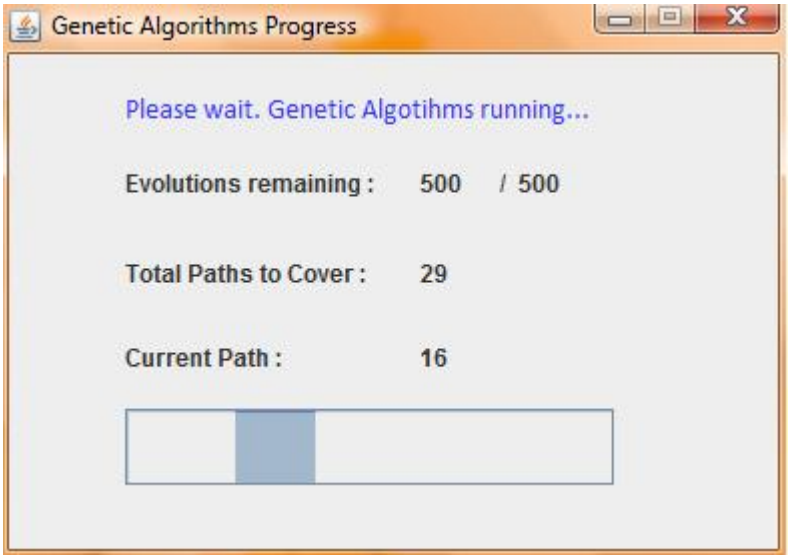
Σχήμα 4.58 : Δημιουργία και εμφάνιση γράφου CFG για Foo4.java

- Στη συνέχεια στο σενάριο χρήσης θα πρέπει να αρχίσουμε την παραγωγή περιπτώσεων χρήσης με του γενετικούς αλγορίθμους. Αυτό γίνεται με κλικ στο κουμπί για την παραγωγή περιπτώσεων χρήσεων με βάση τον γράφο ροής ελέγχου όπως φαίνεται πιο κάτω (Σχήμα 4.59).



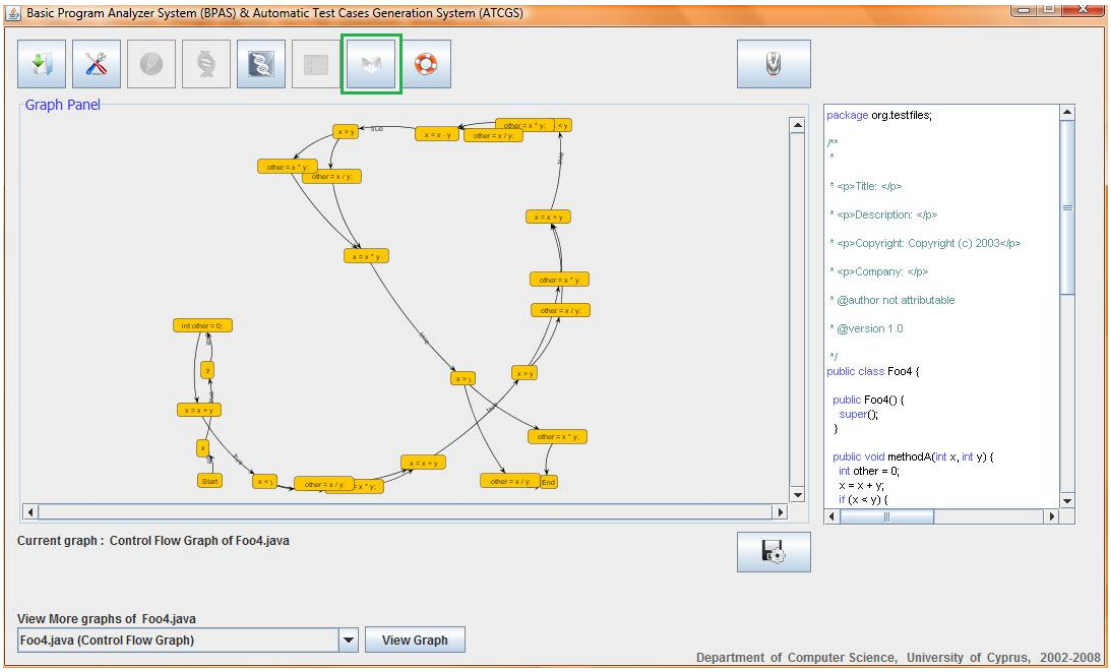
Σχήμα 4.59 : Κουμπί για εκτέλεση GA με βάση CFG

- Όταν πατήσουμε αυτό το κουμπί ξεκινά αμέσως η παραγωγή περιπτώσεων ελέγχου ενώ αναδύεται ένα ενημερωτικό παράθυρο με πληροφορίες για την κατάσταση της εκτέλεσης των γενετικών αλγορίθμων, παρόμοιο με αυτό για την κατάσταση των γενετικών αλγορίθμων που εμφανίστηκε σε προηγούμενο σενάριο χρήσης, που έγινε με βάση γράφο ροής δεδομένων. Η διαφορά είναι ότι στο ενημερωτικό παράθυρο που βλέπουμε πιο κάτω (Σχήμα 4.60), η αναφορά σε μονοπάτια γίνεται σε μεμονωμένες ακμές, στις οποίες επικεντρώνεται η παραγωγή περιπτώσεων, αφού το κριτήριο κάλυψης για τον γράφο ροής ελέγχου είναι edge/condition.



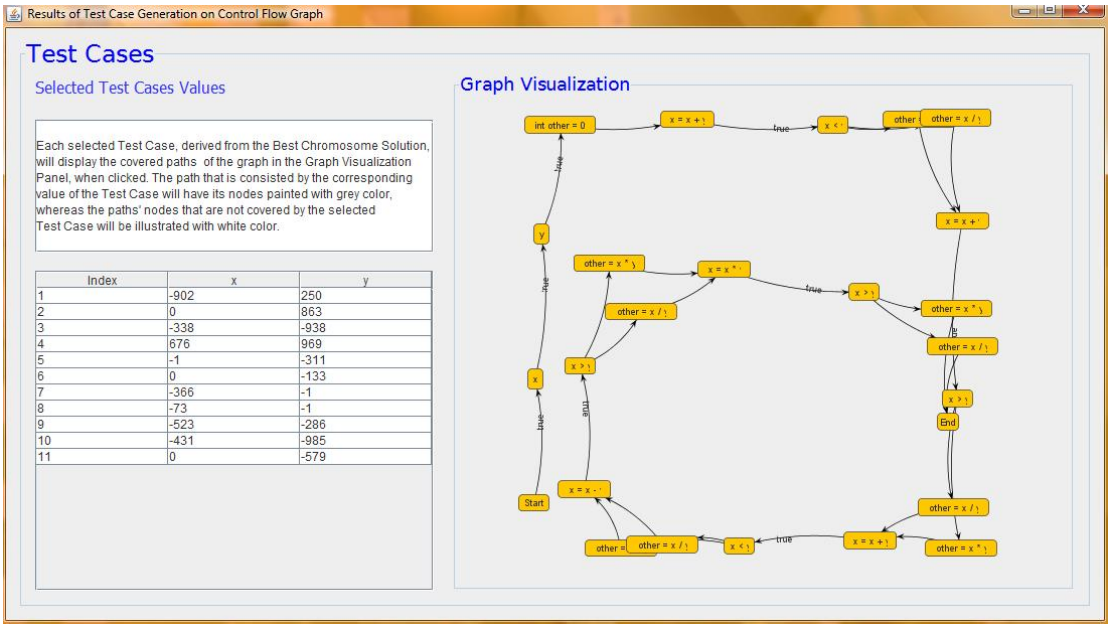
Σχήμα 4.60 : Πρόοδος εκτέλεσης GA με βάση CFG

- Όταν η παραγωγή περιπτώσεων ελέγχου ολοκληρωθεί, το παράθυρο αυτό θα κλείσει μόνο του, ενώ ήχος συστήματος (System Beep) θα ακουστεί, ειδοποιώντας τον χρήστη ότι οι γενετικοί αλγόριθμοι έχουν τερματίσει τη λειτουργία τους, ενώ παράλληλα ενεργοποιείται το κουμπί για την προβολή των σχετικών αποτελεσμάτων, όπως βλέπουμε στην επόμενη εικόνα (Σχήμα 4.61).



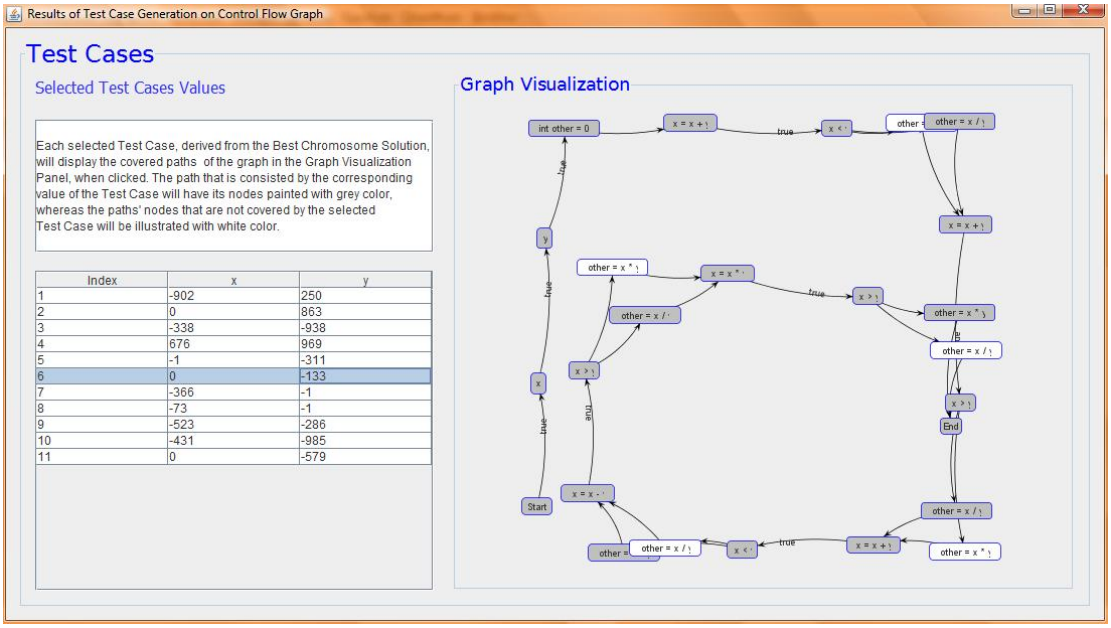
Σχήμα 4.61 : Κουμπί για εμφάνιση αποτελεσμάτων GA σε CFG

- Με κλικ στο κουμπί αυτό ανοίγει το παράθυρο, παρόμοιο με αυτό για την προβολή αποτελεσμάτων βασισμένα στον γράφο ροής δεδομένων, δείχνοντας αναλυτικά τις περιπτώσεις ελέγχου όπως βλέπουμε πιο κάτω (Σχήμα 4.62).



Σχήμα 4.62 : Παράθυρο παρουσίασης αποτελεσμάτων εκτέλεσης GA

- Βλέπουμε λοιπόν το σύνολο με τα χρήσιμα χρωμοσώματα, που μας δίνουν τις περιπτώσεις ελέγχου στην λίστα στα αριστερά του παραθύρου, ενώ επιλέγοντας κάποιο από αυτά, το αντίστοιχο μονοπάτι με τις τιμές που δίνονται στις παραμέτρους της περίπτωσης ελέγχου θα χρωματιστεί στον γράφο, στο πάνελ δεξιά στο παράθυρο. Πιο παραστατικά μπορούμε να το δούμε αυτό στην επόμενη εικόνα (Σχήμα 4.63).

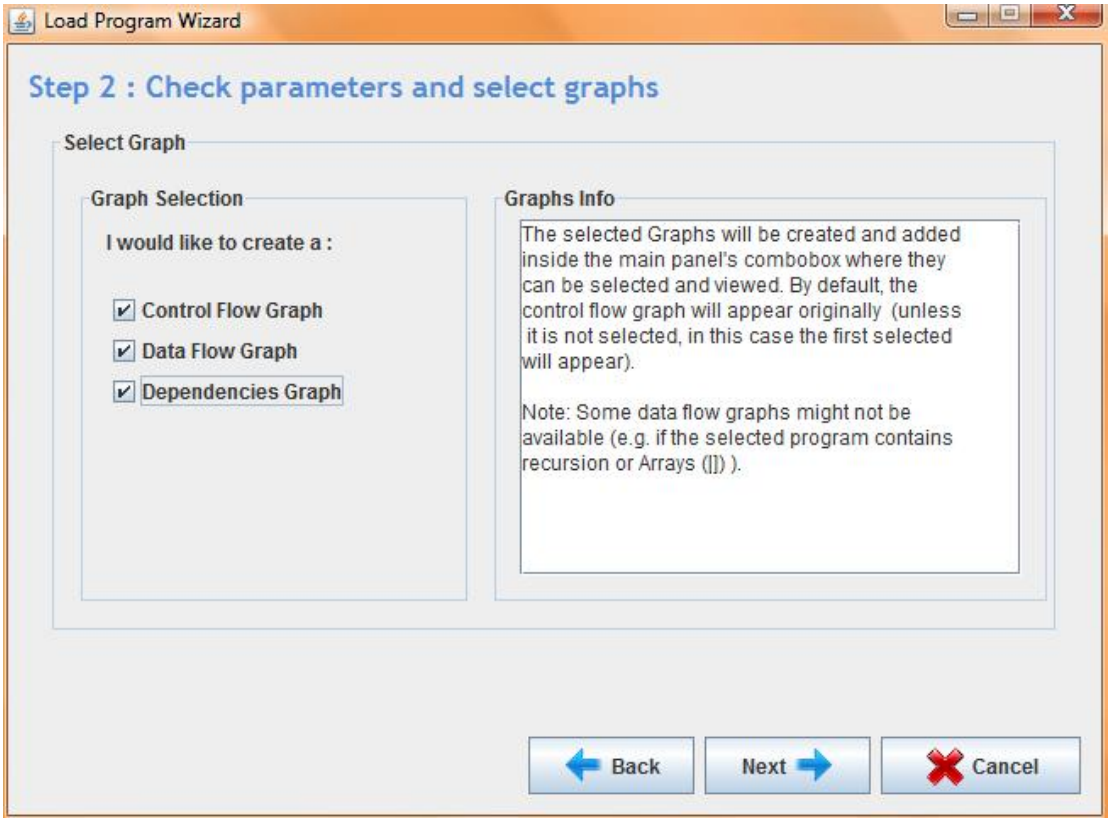


Σχήμα 4.63 : Επιλογή περίπτωσης ελέγχου και αποτύπωση του μονοπατιού που κάλυψε στον γράφο.

- Στο σημείο αυτό ολοκληρώνεται το τέταρτο σενάριο χρήσης.

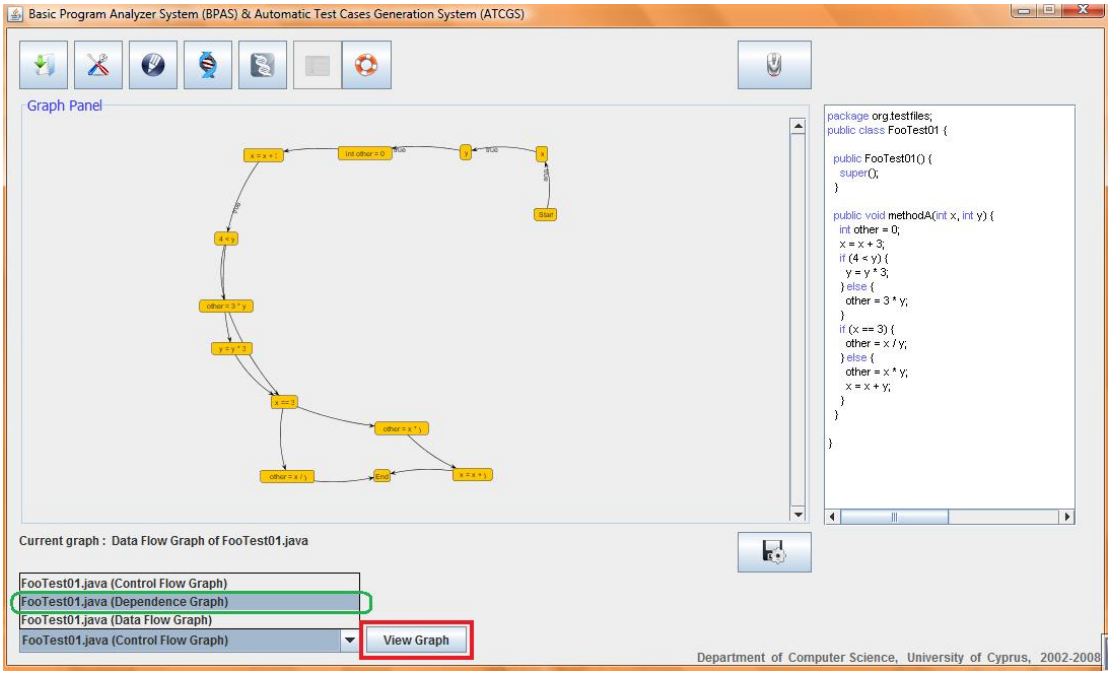
V. Σενάριο πέμπτο : Δημιουργία και των τριών γράφων για το πρόγραμμα FooTest01.java. Επιλογή γράφου εξαρτήσεων από το Combobox γράφων και στη συνέχεια σμίκρυνση του γράφου ώστε να χωρέσει εξ' ολοκλήρου στο πάνελ, και αποθήκευση της οπτικής απεικόνισης του γράφου σε μορφή εικόνας JPEG στον κατάλογο με εικόνες και άνοιγμα της εικόνας για προβολή. Τέλος, με βάση αυτό το σενάριο θα πρέπει να ανοίξουμε το Online Εγχειρίδιο Χρήσης του εργαλείου, και να πλοηγηθούμε στη σελίδα “How to interact with a graph” και μετά να ψάξουμε στην αναζήτηση με λέξεις κλειδιά τις “Operating System” και προβολή των πιο σχετικού αποτελέσματος.

- Έχοντας επιλέξει το πρόγραμμα FooTest01.java από το πρώτο βήμα του Load Wizard, στο δεύτερο βήμα επιλέγουμε όλους τους γράφους για δημιουργία, όπως βλέπουμε παρακάτω (Σχήμα 4.64).



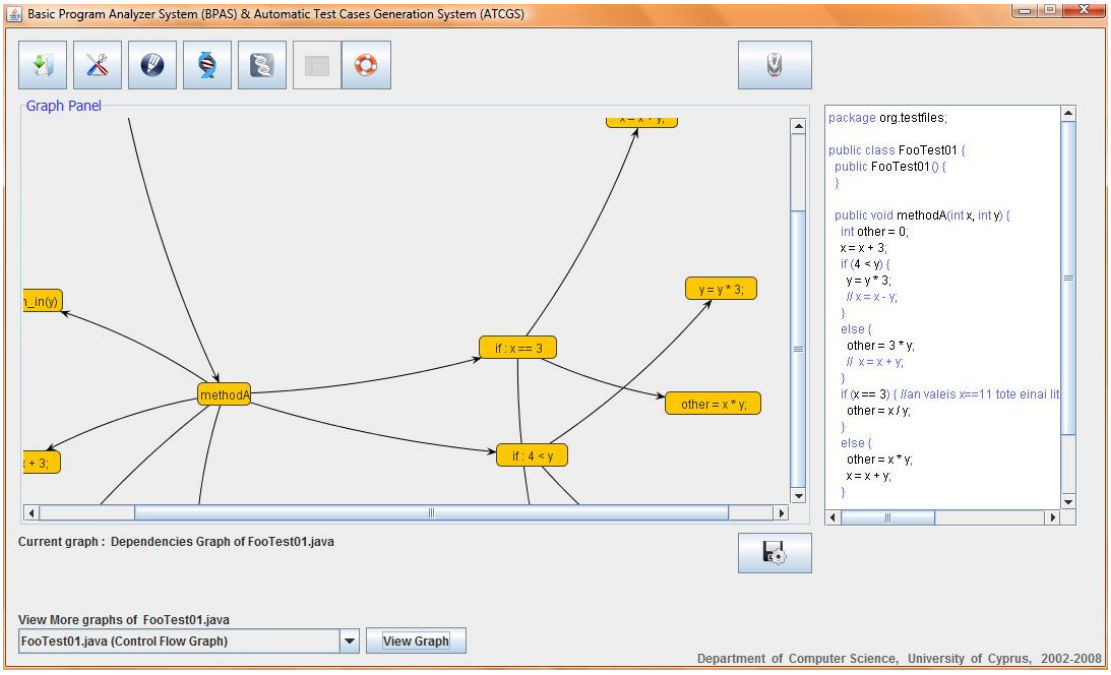
Σχήμα 4.64 : Επιλογή δημιουργίας τριών τύπων γράφων (CFG, DFG, DPG)

- Μετά συνεχίζουμε κανονικά με το Load Wizard και δημιουργούμε τους γράφους. Από το combobox των δημιουργημένων γράφων μπορούμε να δούμε ότι υπάρχουν οι 3 γράφοι που δημιουργήσαμε. Σύμφωνα με το σενάριο, φορτώνουμε τον γράφο εξαρτήσεων και πατάμε το κουμπί "View Graph" όπως φαίνεται στην εικόνα (Σχήμα 4.65).

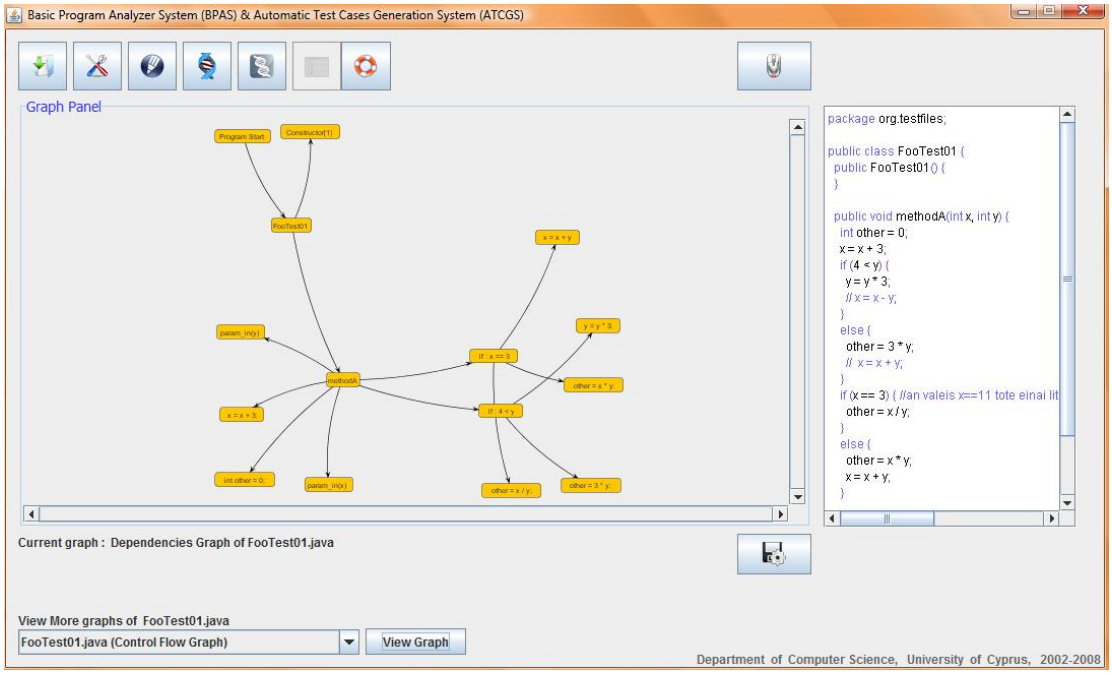


Σχήμα 4.65 : Επιλογή για εμφάνιση του DPG του κώδικα FooTest01.java

- Όταν ο γράφος φορτωθεί, όπως βλέπουμε πιο κάτω, θα κάνουμε σμίκρυνση του γράφου (zoom-out) με χρήση της ροδέλας του ποντικιού, με την κύλιση προς τα επάνω. Όταν διαπιστώσουμε ότι ο γράφος αποτυπώνεται εξ' ολοκλήρου στο πάνελ, τότε σταματάμε τη σμίκρυνση. Στην πρώτη εικόνα (Σχήμα 4.66) που ακολουθεί μπορούμε να δούμε τον γράφο απαιτήσεων όπως τον βλέπουμε στην αρχή που φορτώνεται στο πάνελ, και στη δεύτερη εικόνα (Σχήμα 4.67) όπως προκύπτει μετά τη σμίκρυνση.

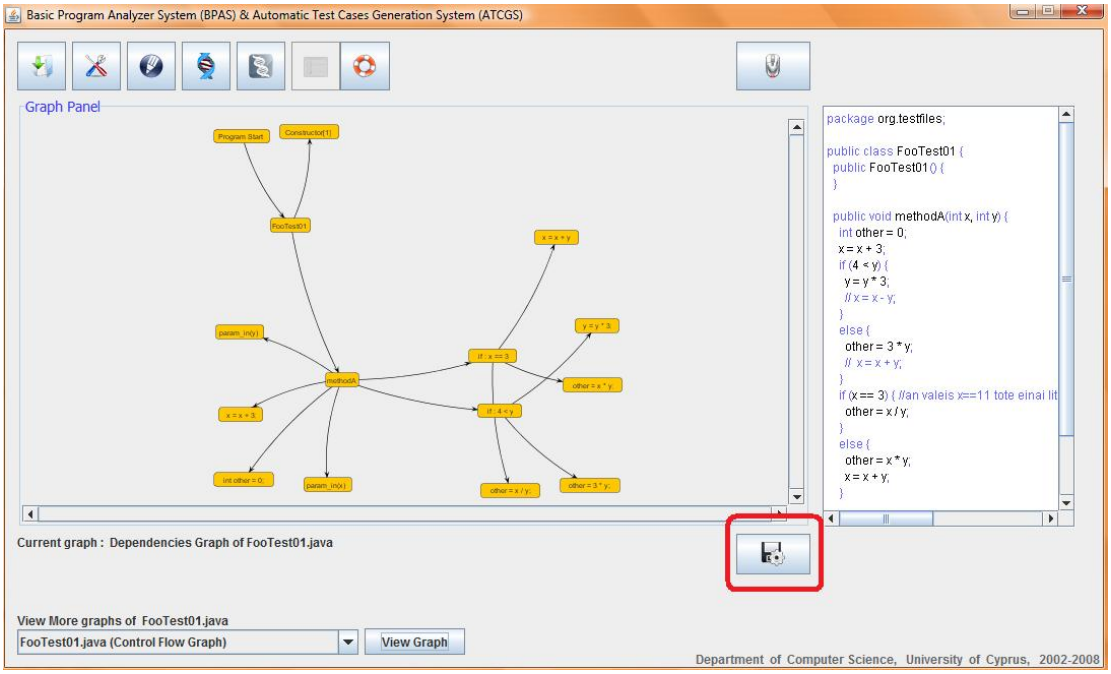


Σχήμα 4.66 : Αρχική γραφική απεικόνιση DPG



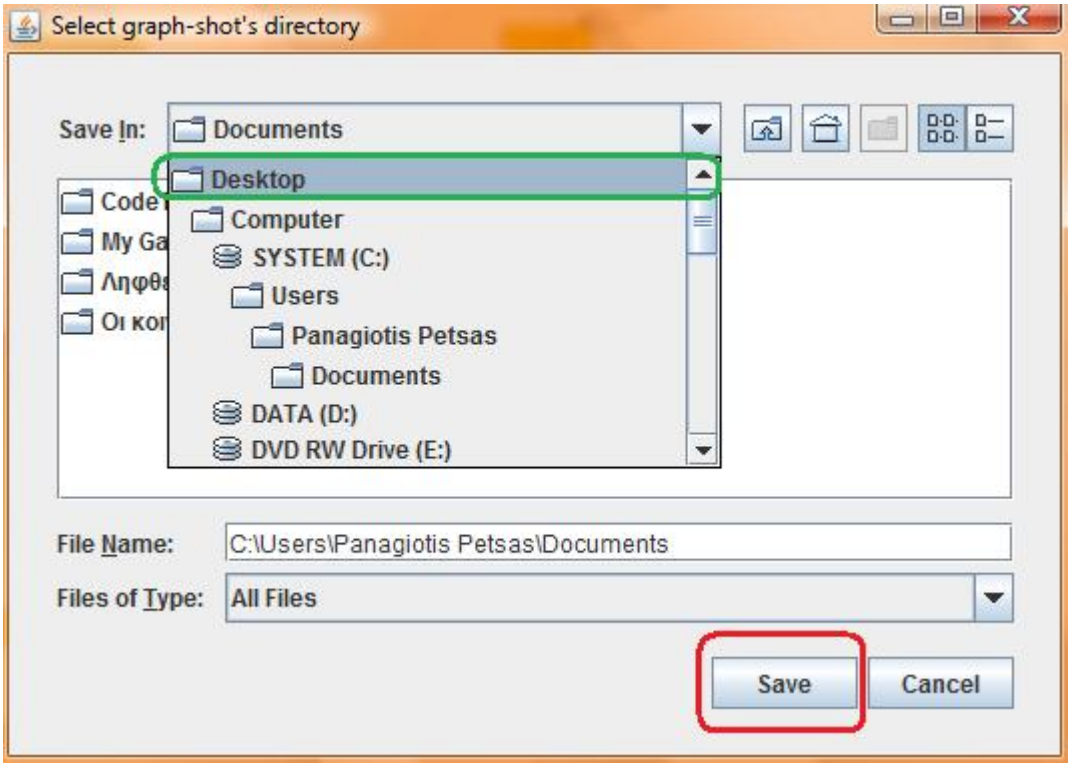
Σχήμα 4.67 : Απεικόνιση DPG μετά τη σμίκρυνση

- Συνεχίζοντας το σενάριο χρήσης, πρέπει να αποθηκεύσουμε τον γράφο σε μορφή εικόνας jpeg στον δίσκο, στην τοποθεσία της Επιφάνειας Εργασίας. Η αποθήκευση γίνεται με τη χρήση του πλήκτρου που βρίσκεται στο κάτω δεξιά μέρος του πάνελ με το εικονίδιο της δισκέτας. Η επόμενη εικόνα υποδεικνύει την θέση του κουμπιού (Σχήμα 4.68).



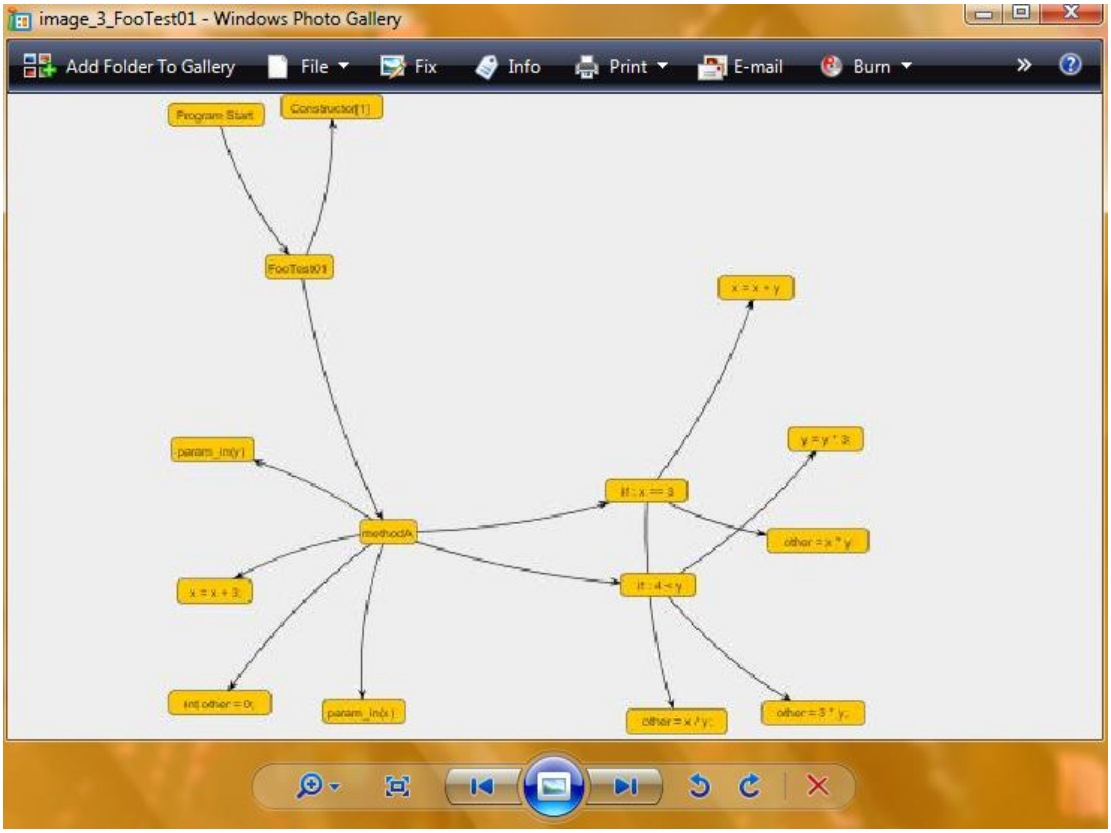
Σχήμα 4.68 : Κουμπί για αποθήκευση του γράφου ως εικόνα

- Με κλικ στο κουμπί αυτό θα ανοίξει ένα παράθυρο για διευκόλυνση του χρήστη στον εντοπισμό και επιλογή του καταλόγου στον οποίο θέλουμε να αποθηκευτεί η εικόνα του γράφου, σε γραφικό περιβάλλον. Από το παράθυρο αυτό θα επιλέξουμε τον κατάλογο που αναφέρει το σενάριο χρήσης, όπως βλέπουμε πιο κάτω (Σχήμα 4.69). Τέλος, κάνουμε κλικ στο κουμπί “Save”, και η εικόνα αποθηκεύτηκε.



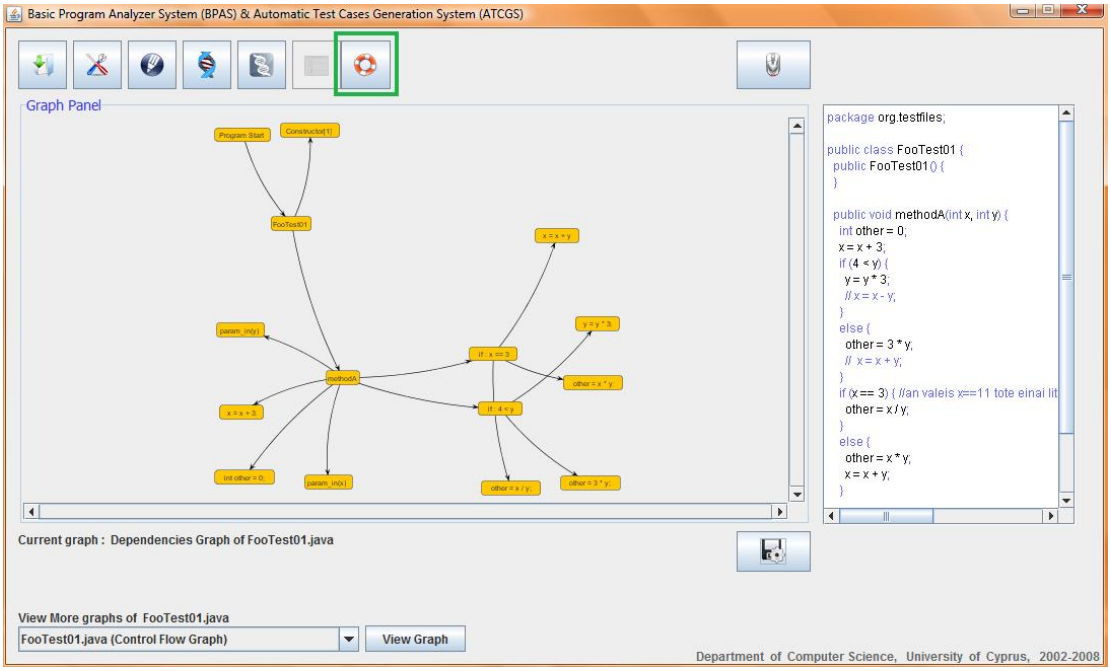
Σχήμα 4.69 : Επιλογή τοποθεσίας για αποθήκευση εικόνας

- Στο επόμενο βήμα του σεναρίου βρίσκουμε την εικόνα και την ανοίγουμε με ένα πρόγραμμα προβολής εικόνας, όπως δείχνει η εικόνα που ακολουθεί (Σχήμα 4.70).



Σχήμα 4.70 : Επιλογή εικόνας για άνοιγμα

- Στο επόμενο βήμα, θα ανοίξουμε το Εγχειρίδιο Χρήσης (Manual) του εργαλείου. Για να ανοίξει το manual, το οποίο είναι online, υλοποιήθηκε με τη βοήθεια του εργαλείου WINCHM [30] και βρίσκεται στην ίδια διαδικτυακή τοποθεσία με την εφαρμογή (στο Domain του τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου) κάνουμε κλικ στο κουμπί, όπως φαίνεται πιο κάτω (Σχήμα 4.71).



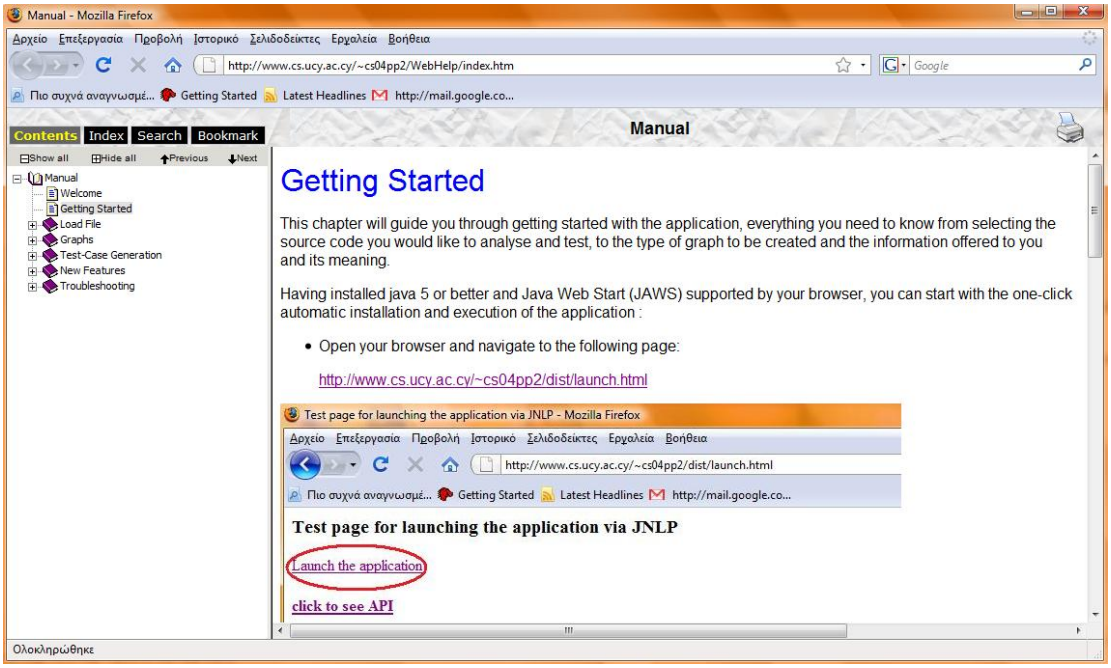
Σχήμα 4.71 : Κουμπί για άνοιγμα Online Εγχειριδίου χρήσης

- Όταν πατηθεί το κουμπί αυτό, η εφαρμογή ελαχιστοποιείται, ελέγχεται το Λειτουργικό Σύστημα στο οποίο χρησιμοποιείται η εφαρμογή και ανοίγει ανάλογα με το ΛΣ με την κατάλληλη εντολή ένας φυλλομετρητής (Browser). Για τα ΛΣ Microsoft Windows, Mac OS, Linux, θα ελεγχθούν ανάμεσα στους φυλλομετρητές οι

  - Epiphany,
  - Firefox,
  - Mozilla,
  - Konqueror,
  - Netscape,
  - Opera

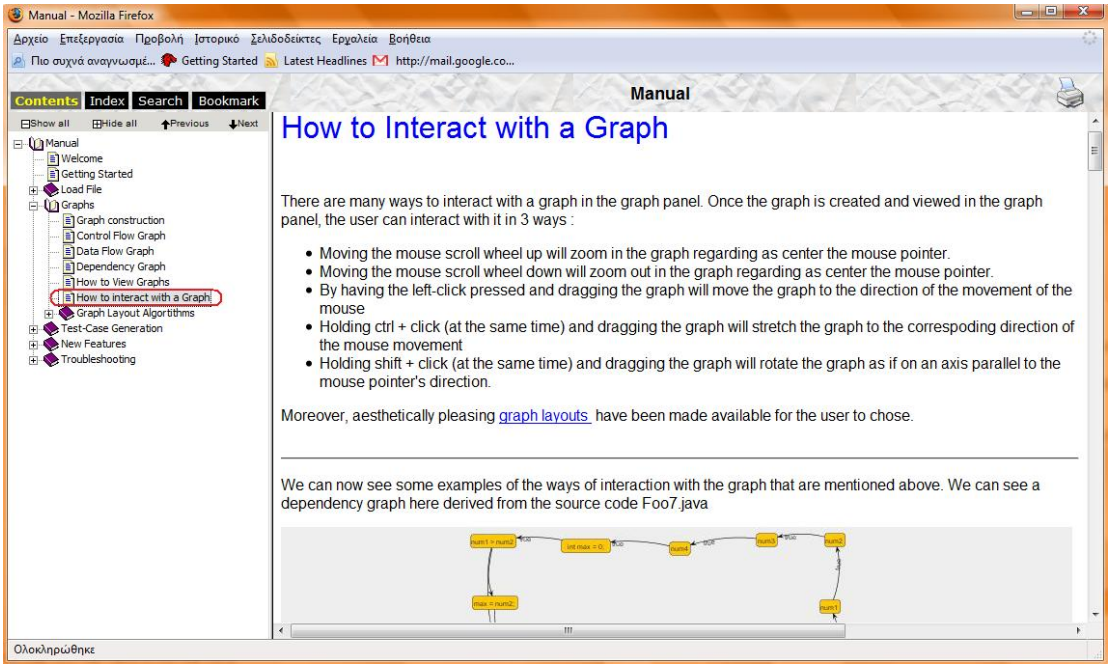
  Αν ο φυλλομετρητής είναι ήδη ανοιχτός, θα δημιουργηθεί νέα καρτέλα, στην οποία

ως διεύθυνση θα τοποθετηθεί η διεύθυνση του Εγχειριδίου Χρήσης, όπως μπορούμε να δούμε στην επόμενη εικόνα (Σχήμα 4.72).



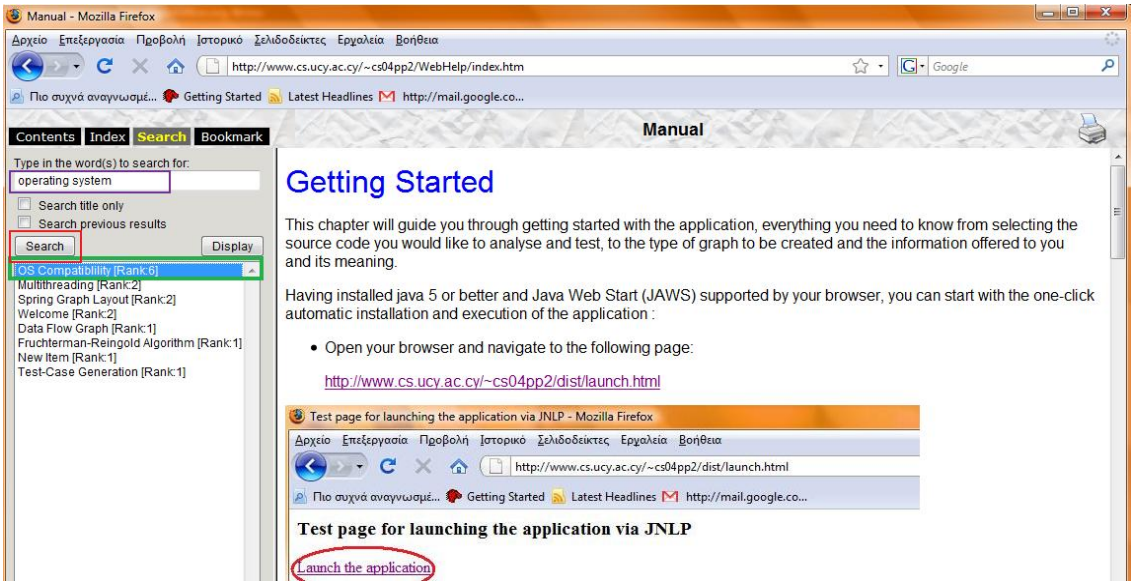
Σχήμα 4.72 : Αρχική σελίδα manual

- Στο σημείο αυτό μπορούμε να επιλέξουμε την επιλογή “Show All” από την καρτέλα “Contents” και να βρούμε τη σελίδα που υποδεικνύει το σενάριο, την σελίδα “How to Interact With a Graph”. Όταν βρούμε τη σελίδα, μπορούμε με κλικ πάνω σε αυτήν να την δούμε στο δεξί τμήμα του φυλλομετρητή, όπως δείχνουμε πιο κάτω (Σχήμα 4.73).



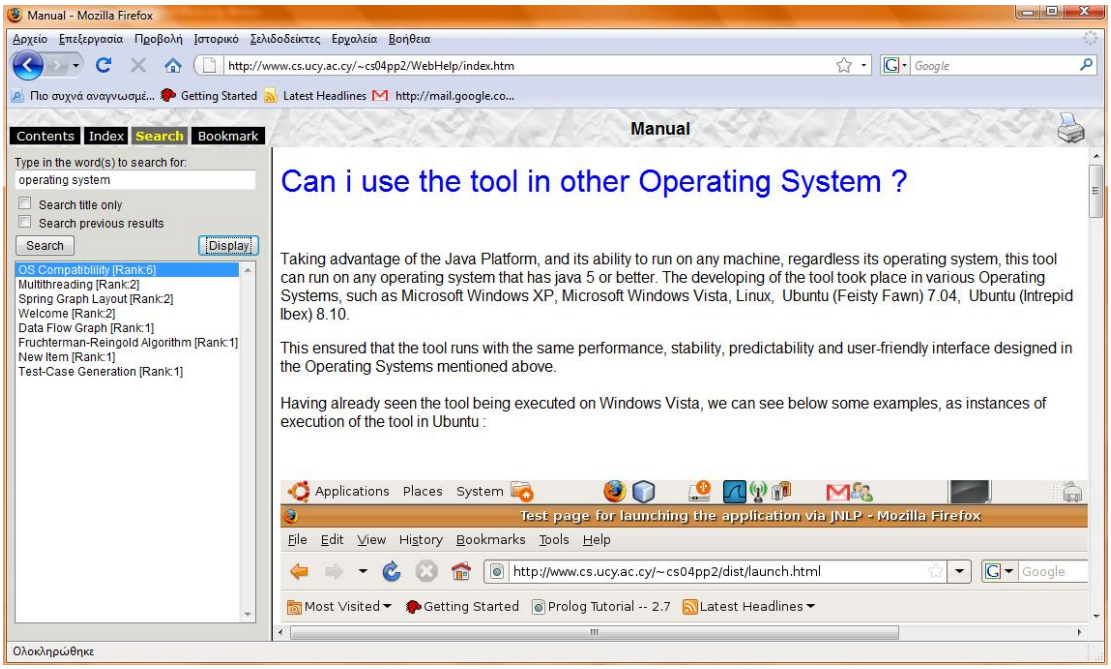
Σχήμα 4.73 : Εύρεση και άνοιγμα σελίδας “How to interact with a graph”

- Συνεχίζοντας στο επόμενο βήμα του σεναρίου χρήσης, πηγαίνουμε στην καρτέλα “Search” του Manual (στο αριστερό τμήμα) και αναζητούμε με λέξεις κλειδιά “Operating” και “System”. Θα δούμε τα παρακάτω αποτελέσματα, όπως φαίνονται στην εικόνα (Σχήμα 4.74), τα οποία είναι ταξινομημένα με σειρά φθίνουσας σχετικότητας του αποτελέσματος με τις λέξεις κλειδιά. Όταν γράψουμε τις λέξεις-κλειδιά, πατάμε Search.



Σχήμα 4.74 : Αναζήτηση με λέξεις κλειδιά “operating”, “system”.

- Πατώντας το κουμπί “Display” θα δούμε ότι θα εμφανιστεί η πιο σχετική σελίδα, που παρουσιάζει πληροφορίες σχετικές με την συμβατότητα της εφαρμογής με διάφορα λειτουργικά συστήματα. Βλέπουμε ένα κομμάτι της σελίδας αυτής πιο κάτω (Σχήμα 4.75).



Σχήμα 4.75 : Εμφάνιση σχετικότερου αποτελέσματος αναζήτησης

Κεφάλαιο 5

Συμπεράσματα

5.1 Συμπεράσματα

Η εργασία ξεκίνησε με μια σύντομη μελέτη σε προηγούμενες σχετικές εργασίες, εμβάθυνση στα σημεία του κώδικα που θα χρειαζόταν να ενσωματώσω στο παρόν εργαλείο, κατανόηση του τρόπου με τον οποίο τα προηγούμενα συστήματα δούλευαν και απομόνωση των λειτουργιών που χρειάζονταν. Έπειτα μια νέα βάση έπρεπε να δημιουργηθεί και η προσεκτική οργάνωση των πακέτων και κλάσεων της εφαρμογής ήταν απαραίτητη για το εγχείρημα της συνένωσης να πετύχει. Καθώς πολλές αλλαγές θα ακολουθούσαν, η αρχιτεκτονική του συστήματος θα έπρεπε να είναι πολύ-επίπεδη μεν, ενιαία δε, με την έννοια ότι οι κλάσεις και τα πακέτα θα πρέπει να έχουν μια ομοιομορφία που θα επιτρέπει τη διαχείριση των κλάσεων με τον ίδιο τρόπο. Για παράδειγμα, κάθε αναλυτής (είτε αυτός κατασκευάζει γράφο απαιτήσεων, είτε γράφο ροής ελέγχου, κ.ο.κ) θα πρέπει να έχει όμοιες μεθόδους για την ανάλυση του κώδικα, την ανάκτηση του γράφου, κ.λπ. Ομοίως, άλλες λειτουργίες, όπως τα Tasks για δημιουργία και εμφάνιση γράφων που σχεδίασα να εκτελούνται σε ανεξάρτητα νήματα και οι γενετικοί αλγόριθμοι, θα έπρεπε να διαθέτουν ομοιομορφία στην αλληλεπίδρασή τους , άσχετα από το αν κατασκευάζουν διαφορετικά είδη γράφων ή αν επιτυγχάνουν διαφορετικά είδη κάλυψης αντίστοιχα. Κάτι τέτοιο όπως είναι λογικό σήμαινε πολλές αλλαγές αφού τα κομμάτια κώδικα που απαιτούνται είχαν γραφτεί από διαφορετικά άτομα και με διαφορετική φιλοσοφία σχεδίασης και αυτό επιβράδυνε το έργο της συνένωσης. Όταν αυτό όμως

αποπερατώθηκε, διευκόλυνε την περαιτέρω ανάπτυξη του συστήματος, τις αλλαγές βιβλιοθηκών και την αναβάθμιση υπαρχουσών. Παράλληλα, το πιο σημαντικό είναι ότι παρά τη μεγάλη έκταση της εφαρμογής, η πολυπλοκότητα του κώδικα δεν αυξήθηκε σε αντίθεση με την ευκολία επεκτασιμότητας και συντηρισιμότητας. Στην πορεία, και όταν οι βασικές λειτουργίες είχαν ολοκληρωθεί και ελεγχθεί, δόθηκε έμφαση στη βελτίωση της αλληλεπίδρασης με τον χρήστη. Το πρώτο βήμα έγινε όταν κατά τη συνένωση, συστατικά (components) γραφικής αναπαράστασης στην java του awt αντικαταστήθηκαν με τα νεότερα, πιο «ελαφρά» συστατικά που προσφέρει το swing. Τα επόμενα βήματα σχεδιάστηκαν αργότερα με έμφαση στην παροχή πληροφοριών στον χρήστη, αποκόλληση των χρονοβόρων λειτουργιών σε ξεχωριστά νήματα, και προσεκτική καθοδήγηση του χρήστη στις λειτουργίες του εργαλείου. Παράλληλα με δομές όπως αυτή του Combobox γράφων γίνεται αποθήκευση γραφικών αναπαραστάσεων των γράφων, που ίσως χρειάστηκαν αρκετή ώρα για τη δημιουργία τους, και είναι διαθέσιμες στον χρήστη ανά πάσα στιγμή, χωρίς να χρειάζεται η εκ νέου ανάλυση του κώδικα και η κατασκευή του γράφου και της απεικόνισης του. Αλλαγές όπως η προσθήκη της δυνατότητας ανοίγματος της εφαρμογής διαδικτυακά με το JAWS συντέλεσαν στην αύξηση της μεταφερσιμότητας και συμβατότητας της εφαρμογής. Μεταφερσιμότητα, αφού μπορεί να λειτουργήσει σε κάθε μηχανή που διαθέτει φυλλομετρητή και java 5 ή νεότερη, και συμβατότητα, δεδομένου ότι εφαρμογές σε java μπορούν να τρέξουν ανεξάρτητου αρχιτεκτονικής, λειτουργικού συστήματος, κ.ο.κ. Πέραν τούτου, το μεγαλύτερο τμήμα της υλοποίησης έγινε σε Linux ΛΣ και μπορεί να εγγυηθεί ότι η συμβατότητα αυτή είναι πραγματική και όχι πλασματική (ακόμη και όσον αφορά σχεδιαστικές λεπτομέρειες, μεγέθη, fonts, κ.λπ.). Νέες λειτουργίες όπως το Online Εγχειρίδιο Χρήσης (Manual), η οθόνη Περί (About), ο Οδηγός Ανοίγματος-Φόρτωσης Αρχείου (Load Wizard), η λειτουργία εναλλαγής τρόπου αλληλεπίδρασης με τον γράφο, η λειτουργία Εκτός Σύνδεσης (Offline Mode), η αποθήκευση γράφου σε μορφή εικόνας (GraphShot) αύξησαν τη λειτουργικότητα του εργαλείου και τη διευκόλυνση του χρήστη.

Συνοψίζοντας, μπορεί να ειπωθεί ότι καρπός της εκπόνησης αυτής της εργασίας είναι μια ολοκληρωμένη διαδικτυακή εφαρμογή για ανάλυση και έλεγχο με παραγωγή περιπτώσεων ελέγχου προγραμμάτων γραμμένων σε java που παρέχει

i. Ευρύ φάσμα λειτουργιών στον χρήστη, στα πλαίσια μιας σύγχρονης,

καθοδηγητικής και ευχάριστης αισθητικά διεπιφάνειας.

ii. Εύκολη συντηρησιμότητα, επεκτασιμότητα, μεταφερσιμότητα, συμβατότητα, και τεκμηρίωση (documentation) του πηγαίου κώδικα στον προγραμματιστή που ενδεχομένως επιχειρήσει χρήση, τροποποίηση, ή επέκταση του εργαλείου.

iii. Ανεκτίμητη εμπειρία και επέκταση των προγραμματιστικών γνώσεων σε εμένα.

5.2 Μελλοντική Δουλειά

Στα πλαίσια της μελλοντικής δουλειάς μπορούν να προταθούν μερικές εισηγήσεις που θα βελτιώσουν το εργαλείο και θα το κάνουν ακόμη πιο λειτουργικό και πιο ανταγωνιστικό απέναντι σε παρόμοια εργαλεία που υπάρχουν στην αγορά.

- Επέκταση λειτουργιών ανάλυσης κώδικα. Με αυτό εννοείται η ολοκλήρωση της λειτουργικότητας των αναλυτών (parsers) με αποδοχή δομών όπως οι πίνακες (Arrays), της αναδρομής (recursion), αλλά και εντολές όπως : package, extends, κ.λπ., και κάλυψη δυνατοτήτων της java όπως interfaces, inheritance, polymorphism, object, classes, κ.λπ.

- Υποστήριξη και άλλων τύπων προγραμμάτων όπως Applets, JSP, Javascript.

- Υποστήριξη και άλλων προγραμματιστικών γλωσσών όπως για παράδειγμα τη C++, που έχει κάποια σχετική συνάφεια, ως προς την αντικειμενοστρέφεια.

- Προσθήκη νέων λειτουργιών όπως παραγωγή διαγραμμάτων UML, συμβολική εκτέλεση (symbolic execution).

- Ενσωμάτωση (integration) της παρούσας εφαρμογής με γνωστά εργαλεία και projects όπως το eclipse, netbeans, jbuilder, maven.

Βιβλιογραφία

[1] Patrice Godefroid1 and Sarfraz Khurshid. Exploring very large state spaces using genetic algorithms. In CPM ’01: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching, pages 266–280, Massachusetts Institute of Technology, USA, 2002.

.[2] B. F. Jones, H. H. Sthamer, and D. E. Eyres. Automatic structural testing using genetic algorithms. *Software Engineering Journal*, pages 299–306, Sep 1996.

[3] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[4] S. C. Ntafos, "On testing with required elements," in Proceedings of IEEE-CS COMPSAC, 1981, pp. 132-139.

[5] S. C. Ntafos, "On required element testing," *IEEE Transactions on Software Engineering,* vol. 10, No. 6, pp. 795-803, 1984.

[6] A. S. Andreou, K. A. Economides and A. A. Sofokleous, "An automatic software test-data generation scheme based on data flow criteria and genetic algorithms," in Proceedings of the 7th IEEE International Conference on Computer and Information Technology, Fukushima, Japan, 2007, pp. 867-872.

[7] A. Sofokleous and A. Andreou, "Dynamic search-based test data generation focused on data flow paths," in Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2006), Barcelona, Spain, 2008, pp. 27-35.

[8] Robert S.Boyer, Bernard Elspas , Karl N. Levitt, SELECT—a formal system for testing and debugging programs by symbolic execution, Proceedings of the international conference on Reliable software, p.234-245, April 21-23, 1975, Los Angeles, California

[9] Beizer, Boris, *Black-box Testing: techniques for functional testing of software and systems*. Publication info: New York : Wiley, c1995.

[10] Philip Koopman, John Sung, Christopher Dingman, Daniel Siewiorek, Ted Marz. Comparing Operating Systems Using Robustness Benchmarks. 16th IEEE Symposium on Reliable Distributed Systems, Durham, NC, October 22-24, 1997, pp.72-79.

[11] Myers, Glenford J., *The art of software testing,* Publication info: New York : Wiley, c1979.

[12] Michael R. Lyu , Handbook of Software Reliability Engineering. McGraw-Hill publishing, 1995, ISBN 0-07-039400-8

[13] http://www.ece.cmu.edu/~koopman/des_s99/sw_testing

[14] Κυριάκος Ιωάννου, Προπτυχιακή Ατομική Διπλωματική Εργασία, "Δημιουργία και Γραφική Απεικόνιση / Διαχείριση Γράφου Εξάρτησης Δεδομένων από κώδικα προγραμμάτων γραμμένα σε JAVA", 2007.

[15] Holland, J.H., "Adaptation in Natural and Artificial Systems", MIT Press, 1975.

[16] Deboeck, G. J. [Ed.]," Trading On The Edge", Wiley, 1994.

[17] Whitely, D., "A **Genetic** Algorithm Tutorial", Technical Report CS-93-103, Colorado State University,1993.

[18] http://www.acadjournal.com/2005/v15/part6/p4/

[19] Jeff Offutt, Aynur Abdurazik, October 1999, "Generating Tests from UML specifications", Second International Conference on the Unified Modelling Language (UML99), pp. 416-429, Fort Collins, CO.

[20] Clay E. Williams, November 1999, "Software testing and the UML", International Symposium on Software Reliability Engineering (ISSRE'99), Boca, Raton.

[21] Jeff Offutt, Aynur Abdurazik, October 2000, "Using UML Collaboration diagrams for static checking and test generation", Third International Conference on UML, York, UK.

[22] Roper, Marc, "Software Testing", London, McGraw-Hill Book Company, 1994

[23] Jeff Offutt, Shaoying Liu, Aynur Abdurazik, Paul Ammann, March 2003, "Generating Test data from State based Specifications", The Journal of Software Testing, Verification and Reliability, 13(1):25-53.

[24] Andras Toth, Daniel Varro, Andras Pataricca, 2003, "Model Level Automatic Test Generation for UML State-Charts", Sixth IEEE workshop on Design and Diagnostics of Electronic Circuits and System, (DDECS 2003).

[25] Marlon E. Vieira, Marcio S. Dias, Debra J. Richardson, "Object-Oriented Specification-Based Testing Using UML Statechart Diagrams", University of California

[26] Sami Baydeda, Volker Gruhn, 2003, "BINTEST – binary search-based test case generation", In Computer Software and Applications Conference (COMPSAC), IEEE Computer Society Press, 2003.

[27] http://en.wikipedia.org/wiki/

[28] http://jung.sourceforge.net/doc/api/

[29] http://www.atlassian.com/software/clover/

[30] http://www.softwareverify.com/java/coverage/feature.html

[31] http://quilt.sourceforge.net/

[32] http://www.parasoft.com/jsp/solutions/java_solution.jsp

[33] http://www.codework.com/JCover/product.html

[34] http://byecycle.sourceforge.net/

[35] http://ispace.stribor.de/index.php?n=Ispace.Home

[36] http://www.softpedia.com/

Παράρτημα Α.

Το Παράρτημα Α περιέχει το Εγχειρίδιο Χρήσης του BPAS & ATCGS, αυτούσιο, όπως αυτό παρέχεται στους χρήστες μέσω του εργαλείου.

Welcome

Welcome to BPAS & ATCGS (Basic Program Analyser System & Automatic Test Case Generation System). This manual will guide you through everything that is available to you, aiming to cover your needs concerning the analysis, view and interaction of your source code with various graphs, auto-generation of test cases and view of the coverage results.

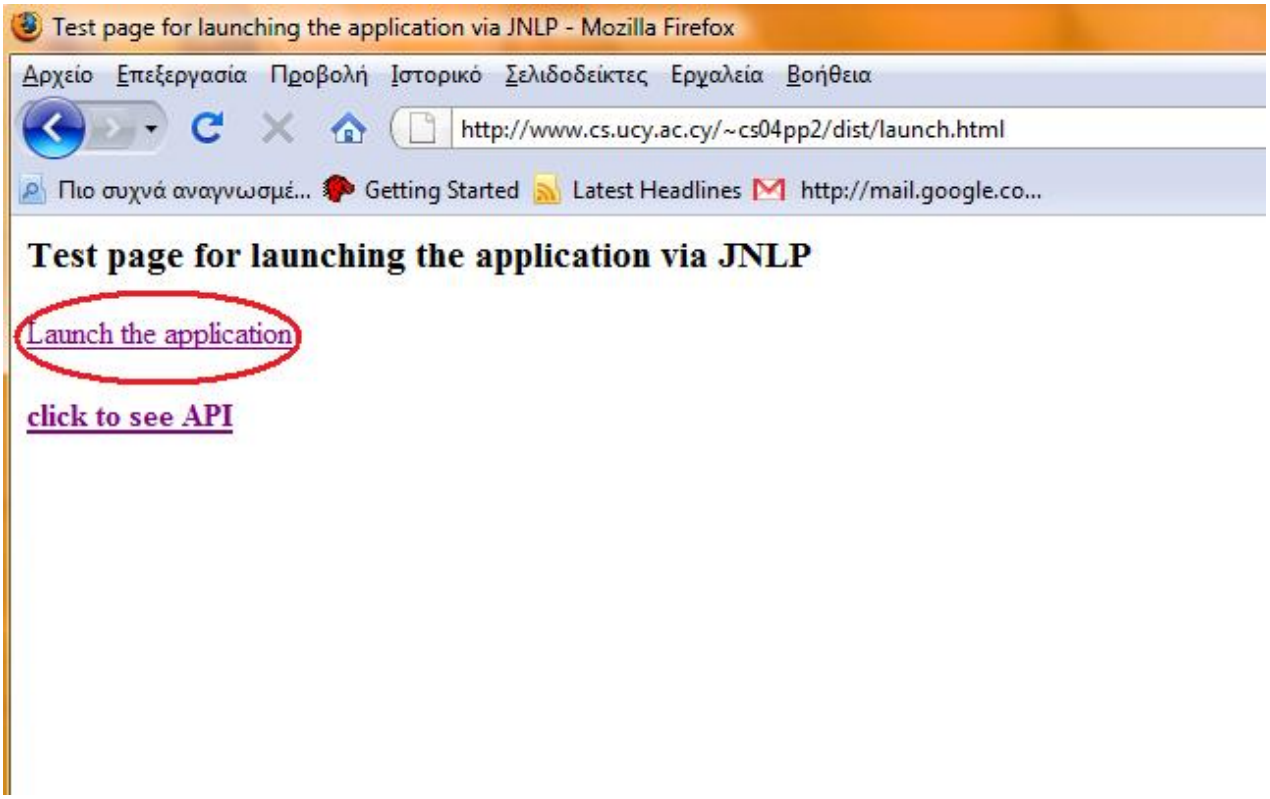
Getting Started

This chapter will guide you through getting started with the application, everything you need to know from selecting the source code you would like to analyse and test, to the type of graph to be created and the information offered to you and its meaning.

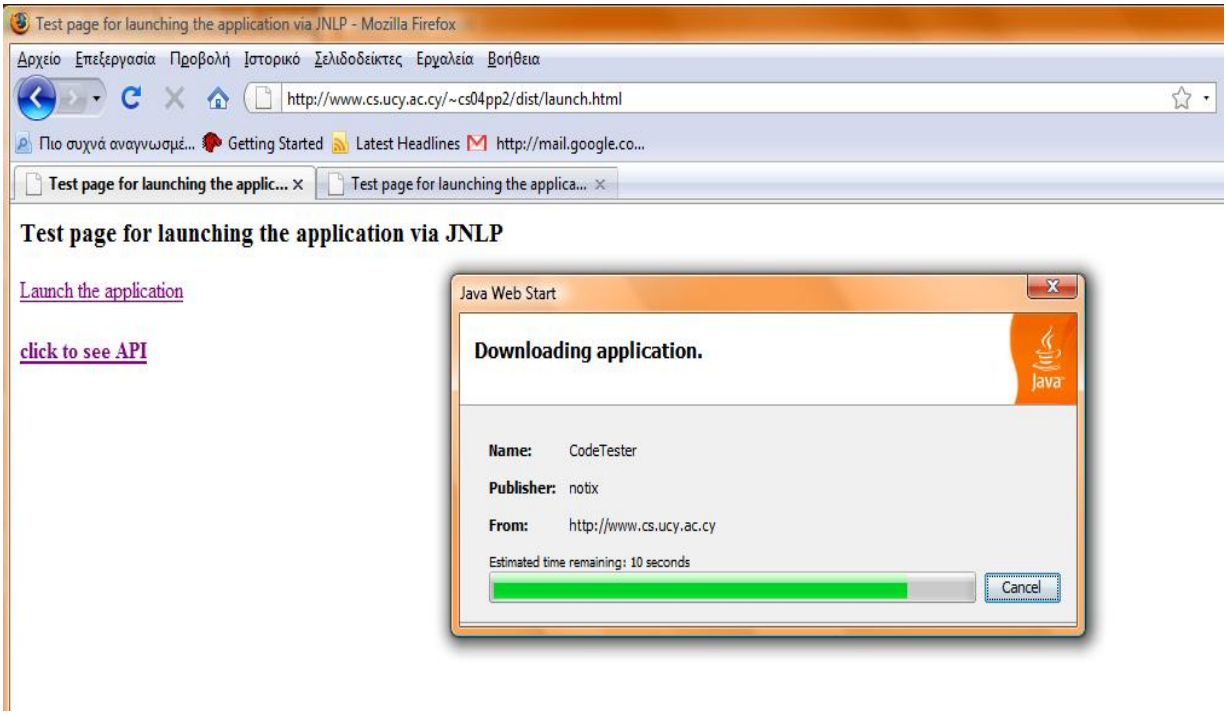
Having installed java 5 or better and Java Web Start (JAWS) supported by your browser, you can start with the one-click automatic installation and execution of the application :

- Open your browser and navigate to the following page:

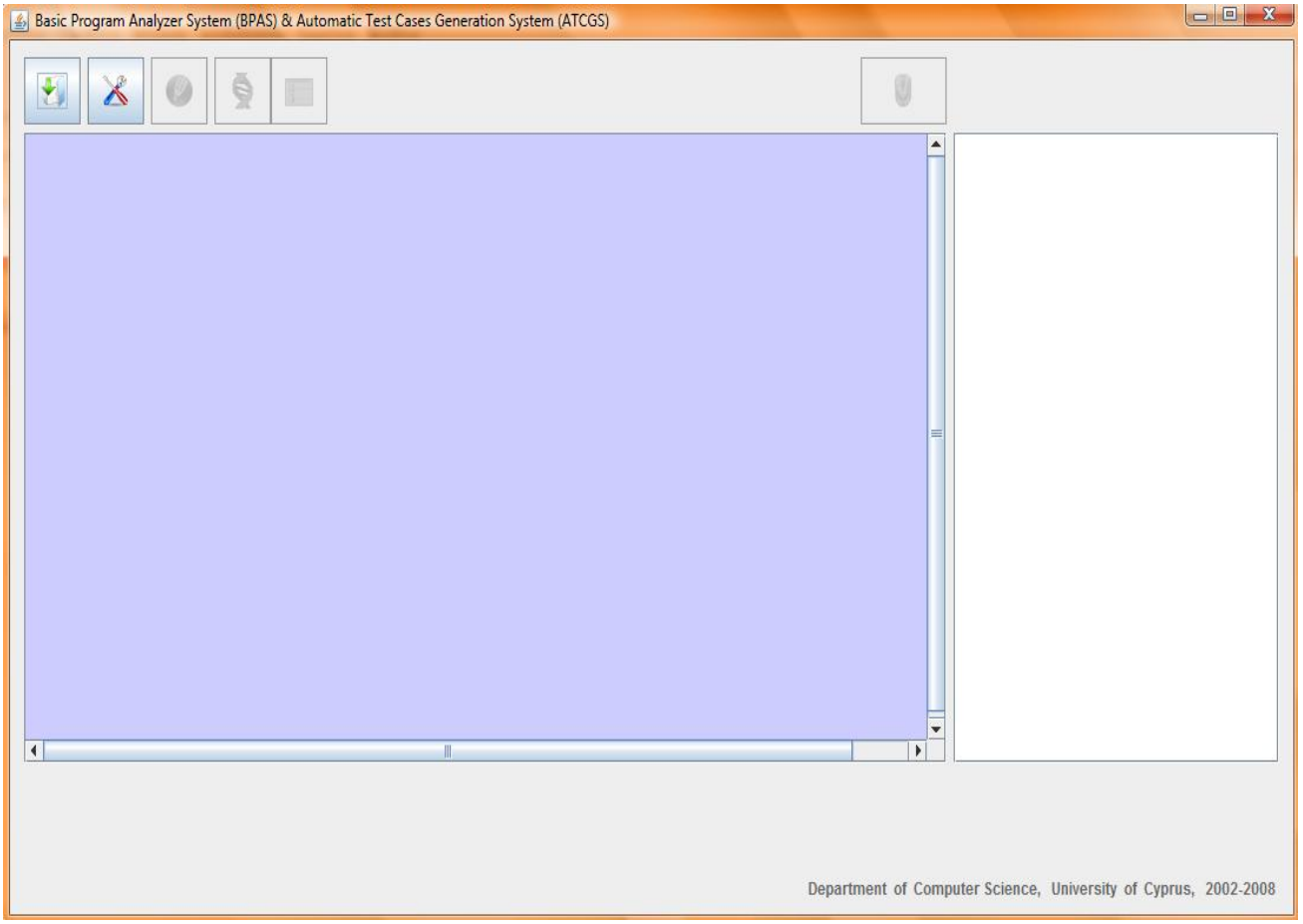
  http://www.cs.ucy.ac.cy/~cs04pp2/dist/launch.html



- The page should look like the one above.
- Click on the link "Launch the application", illustrated inside a red circle in the screenshot above



- As you can see above, the moment you click to the link, the download will begin. You can see that a dialog box will appear showing the progress of the download. At this point the application is being downloaded and verified. Depending on your connection and available download speed, the download might take a while, but almost never more than 3-4 minutes.
- When the download is complete, the application will start execution, as we can see below :



- And the application is ready to work ...

Load/Download a Java Source File

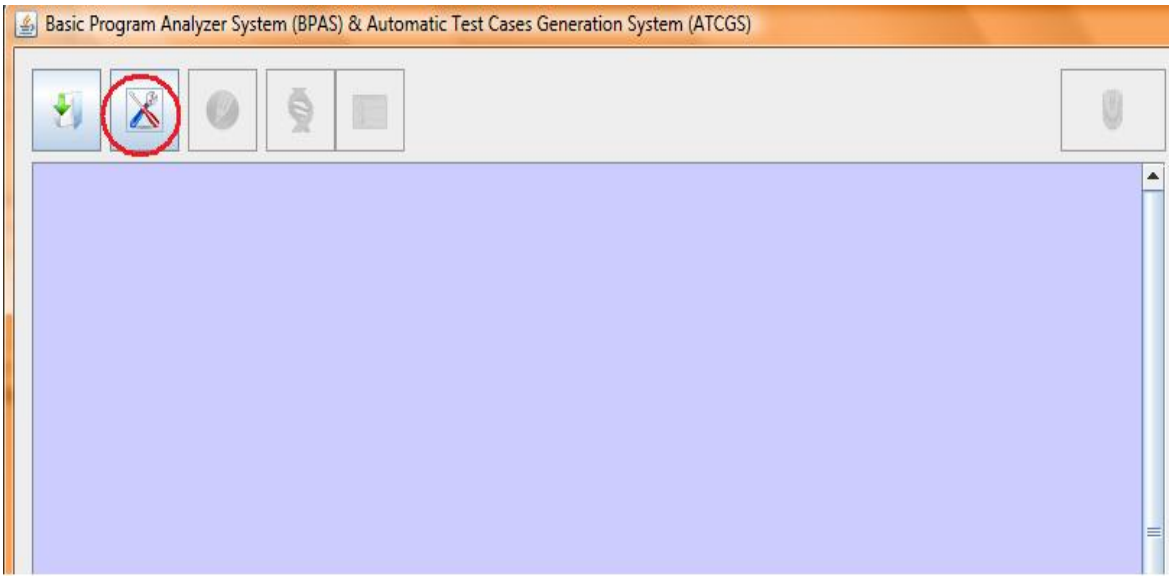
Once the application has started execution, you can choose to load a program that exists locally on the machine or download the sample programs available to you at the following link :

http://www.cs.ucy.ac.cy/~cs04pp2/testfiles/

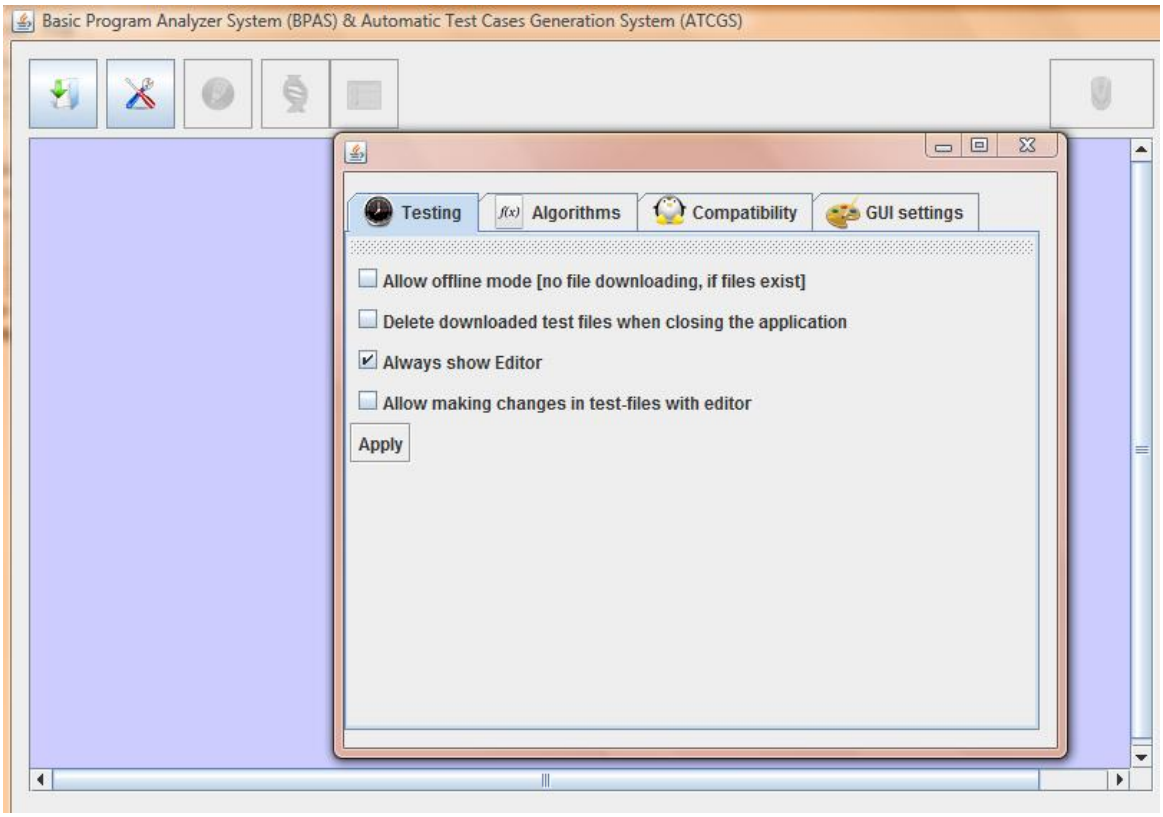
The second option is also the recommended one, since these programs have been checked and tested for compatibility with the parsers and the automatic test-cases generation system. In other words, loading source code written by the user might

contain syntax errors or data structures and algorithms (like arrays[], and recursion) which are not supported and therefore might lead to an exception or no results.

Before loading a program to the application, the user can check and/or modify some of the settings of the application by clicking the settings button, as shown below :



When the button illustrated above is clicked, the following settings menu will be available for the user :



It is important to note that the default option is to use internet connection in order to download the selected files from the remote directory using http protocol. Therefore, if the user knows that there will be no available connection during the execution, then it is recommended that the Offline Mode is selected.

The Offline mode can be found in the Testing Tab in the Settings menu. When selected and then "Apply" button is clicked, the application will no longer demand internet connection to use the selected files but will try to locate them in the local default folder were the application is stored when downloaded with JAWS. Of course, this scenario requests that the files needed actually exist. This means that they have been downloaded and stored during a previous execution or created by the user, using the same name as specified by the index file. Moreover, the index file, testfiles_index.txt, that contains the names of all the sample programs available in the remote location has to be stored in the same location on the machine. The index file can be found in the following link :

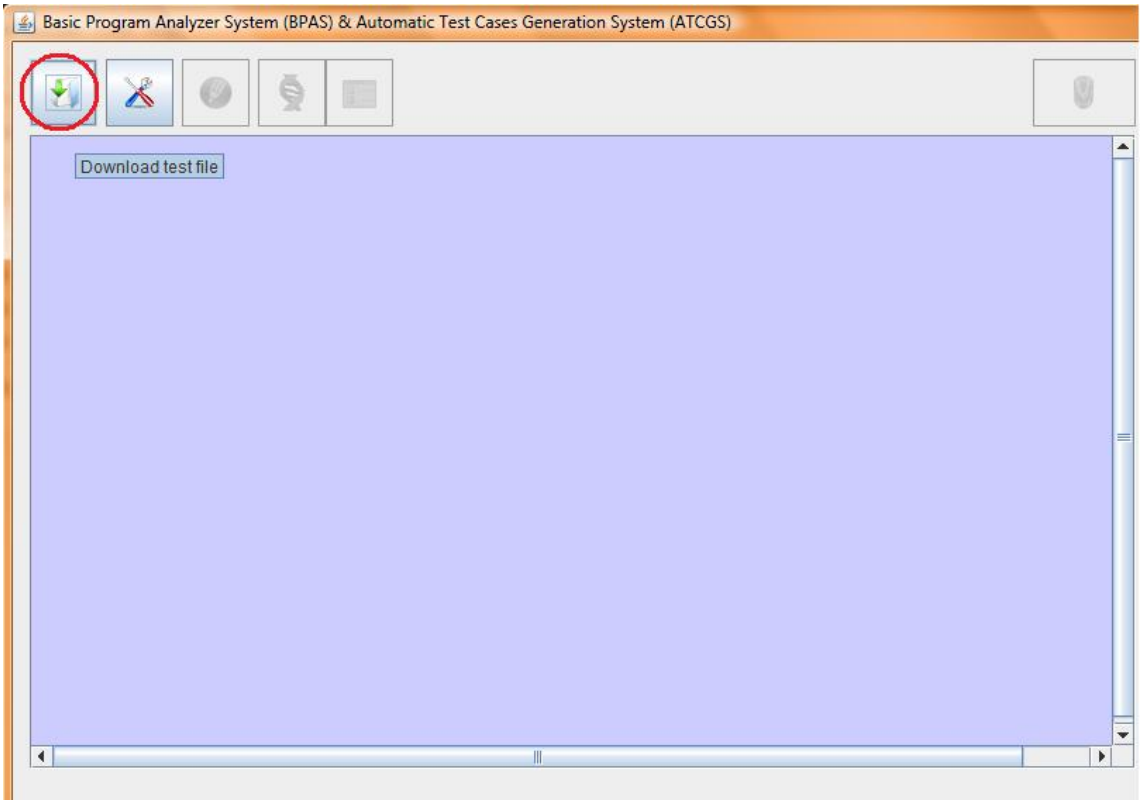
http://www.cs.ucy.ac.cy/~cs04pp2/testfiles_index.txt

At this point, the user can proceed with the Load Wizard, which will make the navigation through opening and loading a file with the assistance of a wizard consisting of 3 easy steps.

Load Wizard

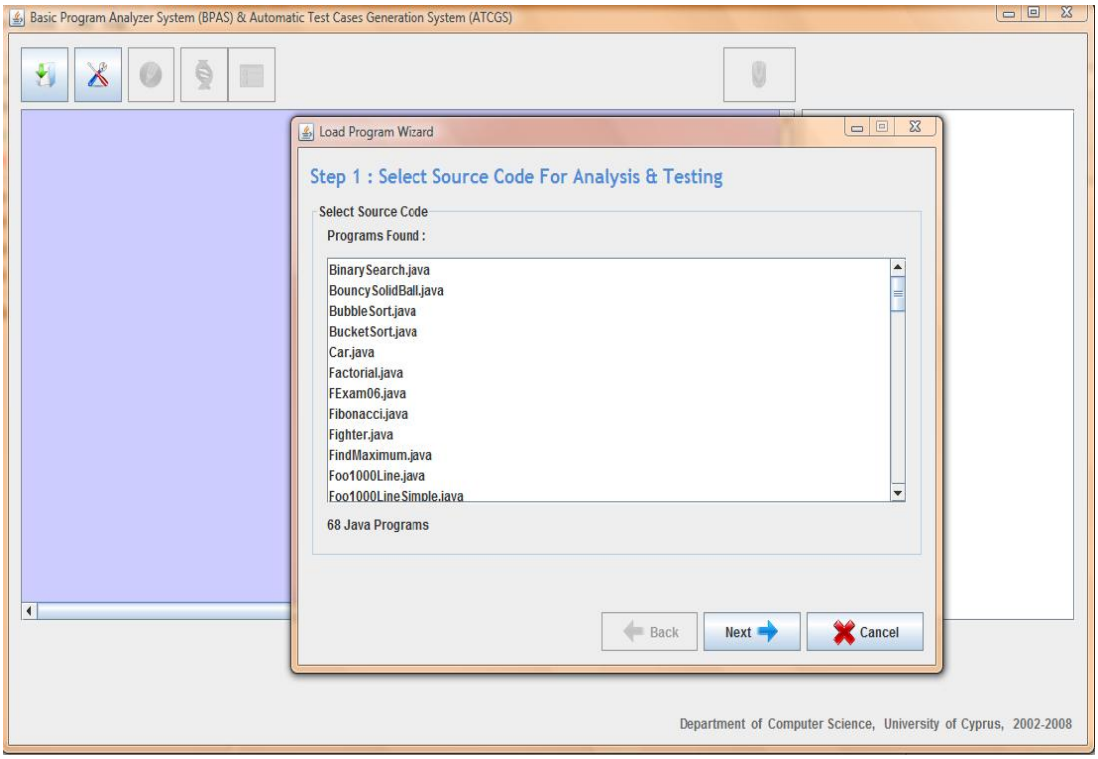
The Load Wizard is a simple wizard consisting of 3 steps, aiming to help the user navigate through the selection of the source code sample program, the type of graph(s) to be created, and provide the necessary information, regarding the status of the file, the availabilty of the selected graph of the selected source code, the expected duration of the genetic algorithms that will run during the test-cases generation, etc.

To view the Load Wizard, you have to click on the first button of the main frame of the application, as shown below :



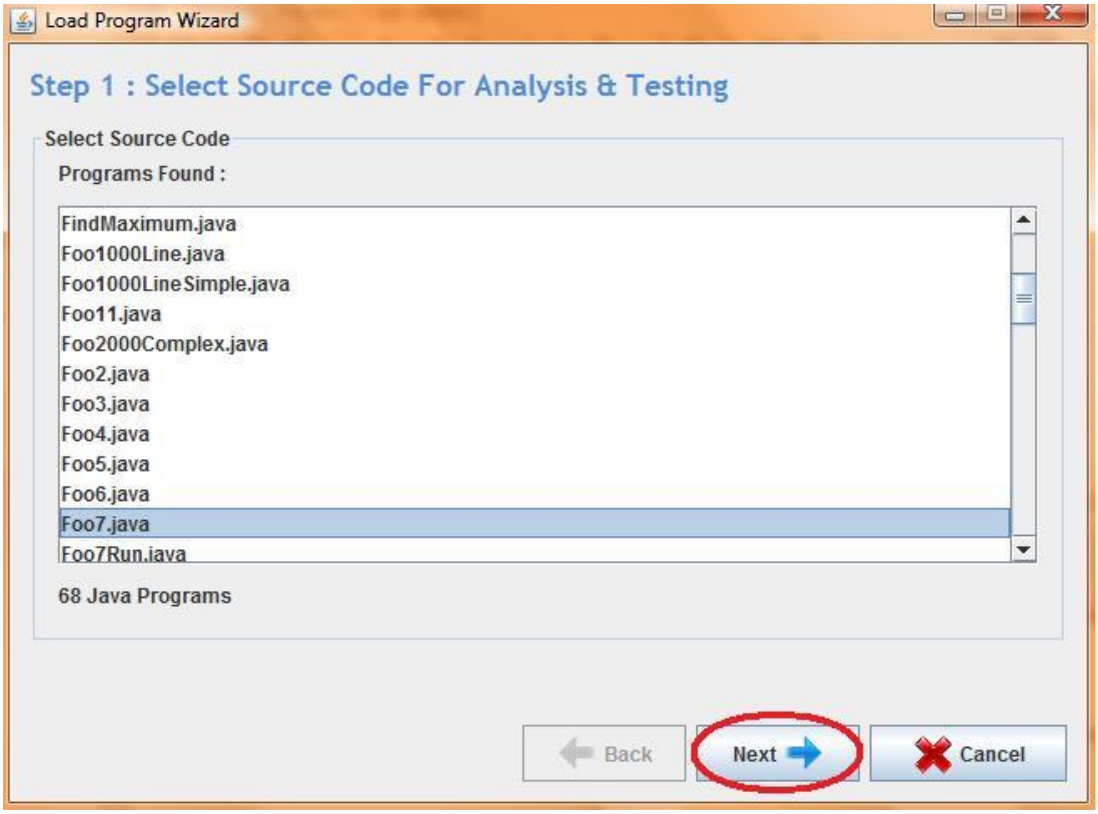
Load Wizard (step 1)

When the Load Wizard button is clicked, the first step of the Load Wizard will appear, as we can see bellow:



We can see that the first step of the wizard contains a list with all the available java source codes, downloaded from the remote directory or loaded from the local directory, depending on the user's choice on Offline Mode Allowed or not.

At this point, the necessary input from the user is selecting the program he wants to analyze and test and click the button "Next".

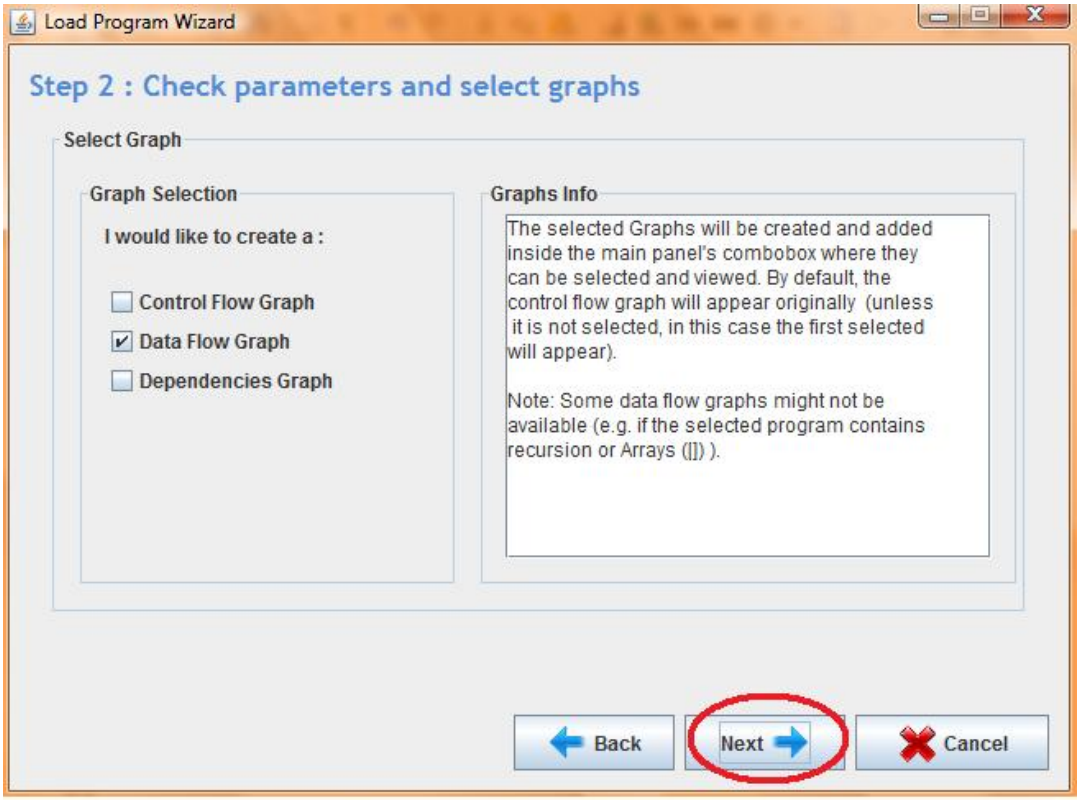


The second step of the wizard will follow.

Load Wizard (step 2)

At this step of the Load Wizard, the user is asked to select the type of graph, or graphs that would like to create based on the selected source code. The application can create a Control Flow graph, a Data Flow graph or a Dependence Graph or a combination of two or three of these graphs, depending on the availability of the graph (due to parsing constraints, like array structures, etc).

We can see below an example of the second step, assuming that we want to create a Data Flow graph for the program Foo7.java



When ready, we can click the button "Next" and continue with the third and last step, or click "Back" and go back to the first step to change the selected program, or "Cancel" to close the Load Wizard.

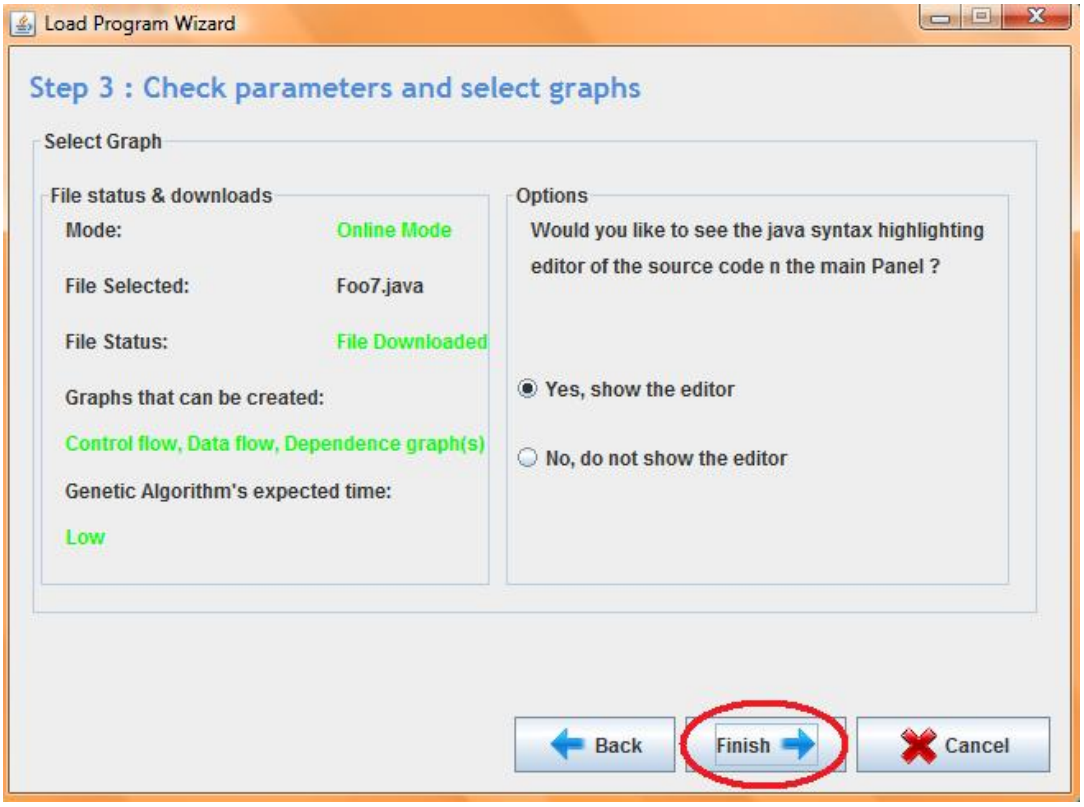
Load Wizard (step 3)

The third, last one, step of the Load Wizard offers information to the user, in order to make sure the parameters that have been chosen are correct. This information includes :

- Whether or not the file will be downloaded with http from the remote directory or loaded from the local disk, where it was saved
- The name of the file containing the source code
- The status of the file (downloaded if offline allowed is not selected and acquired if it is selected)

- The available graphs that can be created based on the source code (due to limitations and constraints mentioned later)
- The expected time until the genetic algorithms creating and evaluating the test-cases are completed (empirically measured)
- Furthermore, the user can chose whether or not the java-syntax highlighting editor will appear in the main frame, next to the graph.

We can see below the third step of the Load Wizard



When checked that everything is ready and the settings are according to the preferences, the user can proceed to the creation of the graph(s) by clicking the button "Finish", or go back and check or change some of the previous settings.

When "Finish" is clicked, the progress dialog(s) for the graph(s) will appear, indicating the progress of each graph and when the graphs are created and the visualization viewer is completed, the graph(s) will appear in the panel and the wizard will be disposed.

Graphs

Graphs are mathematical structures used to visualize relations between objects from a collection [27]. Depending on the semantics of each graph in the application, the set of nodes and edges can represent different concepts. For example, in the control flow graph the set of nodes represent statements whereas the edges represent the control flow of the program, or in other words the sequence of nodes (path) that can be traversed during the execution of the program. On the other hand, in a dependence graph, the set of nodes and the set of edges have different significance. The nodes in this case represent objects (like methods, constructors, statements,etc) and the directed edges the prerequisite objects; that is, the destination of the edge indicates the object on which the object of the source of the edge is dependent on.

Graphs can be a very powerful tool in the hands of an analyst, developer or tester. They have the ability to visualize various aspects of a program and help the viewer understand better the structure of a program, reveal specific characteristics of the source code (like existence of circles, dependencies, complexity, data flow, etc).

Graphs

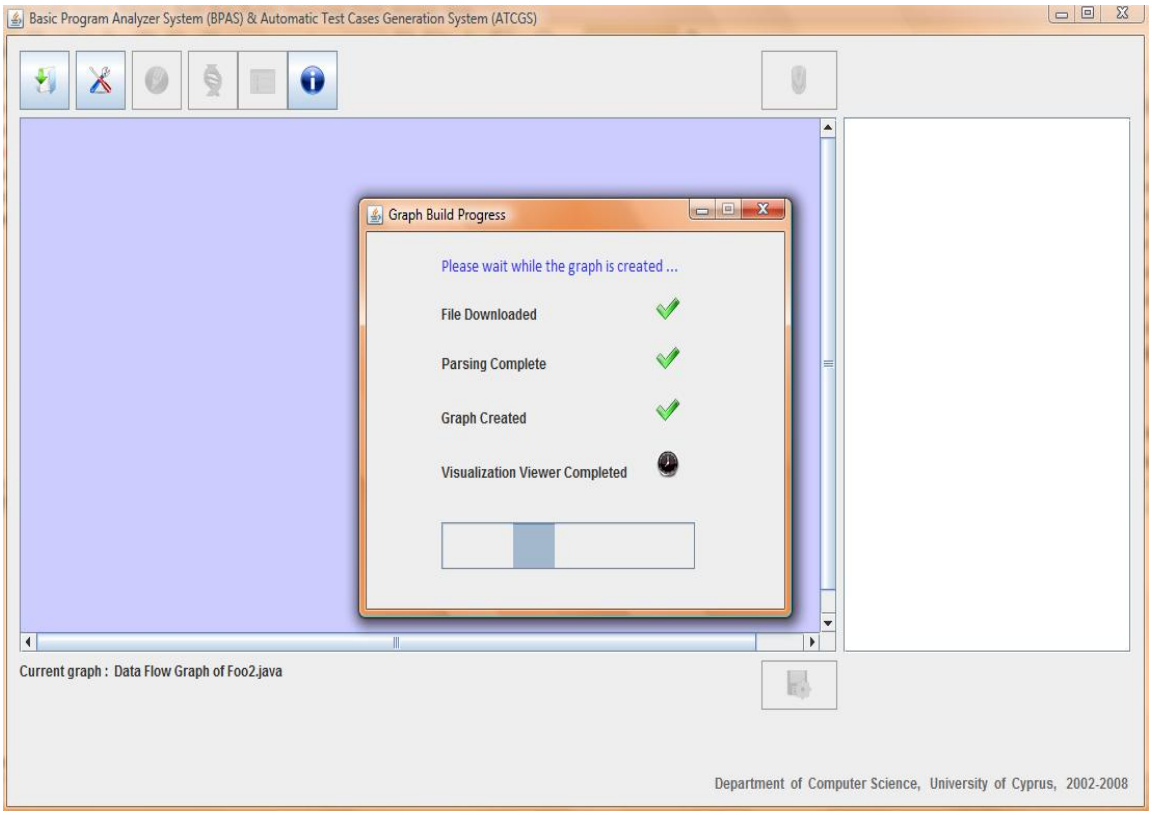
Once the settings and preferences for the graph to be created have been set and the last step of the Load Wizard is succesfully finished, then for every type of graph selected, a new thread begins execution, each thread corresponding to a type of graph. Moreover, for every thread created corresponds a dialog window with a progress bar, showing the current activity of the thread. These activities must take place in the order mentioned, although the same activities concerning the creation of different types of graphs can run at the same time. The activities mentioned are the following :

- File Download
- Parsing
- Graph Creation

- Visualization Viewer Creation

In this way the user can continue navigating in the tool and work on other tasks with no need to wait for a specific graph to be created. This proves to be very useful, especially when the complexity and lines of code increase and tend to consume a lot of time: time that in previous versions could not be used since the application was "frozen" until the completion of the time-consuming task. The user, clearly, can greatly benefit from these modifications and additions. The modifications of the way the Tasks of graph creation are executed, with multithreading allowing parallel threads to run at the same time constructing different graphs and storing them when finished, and additions of the informing dialog boxes, progress bars and display of the current activity of each task. Not only the user can save time and avoid the annoying application "freezes" but also benefits from the information offered to him, in accordance with Human-Computer Interaction Principles.

We can see below a dialog box, as described above, showing the current status of a thread processing a java source code and constructing its dataflow graph :



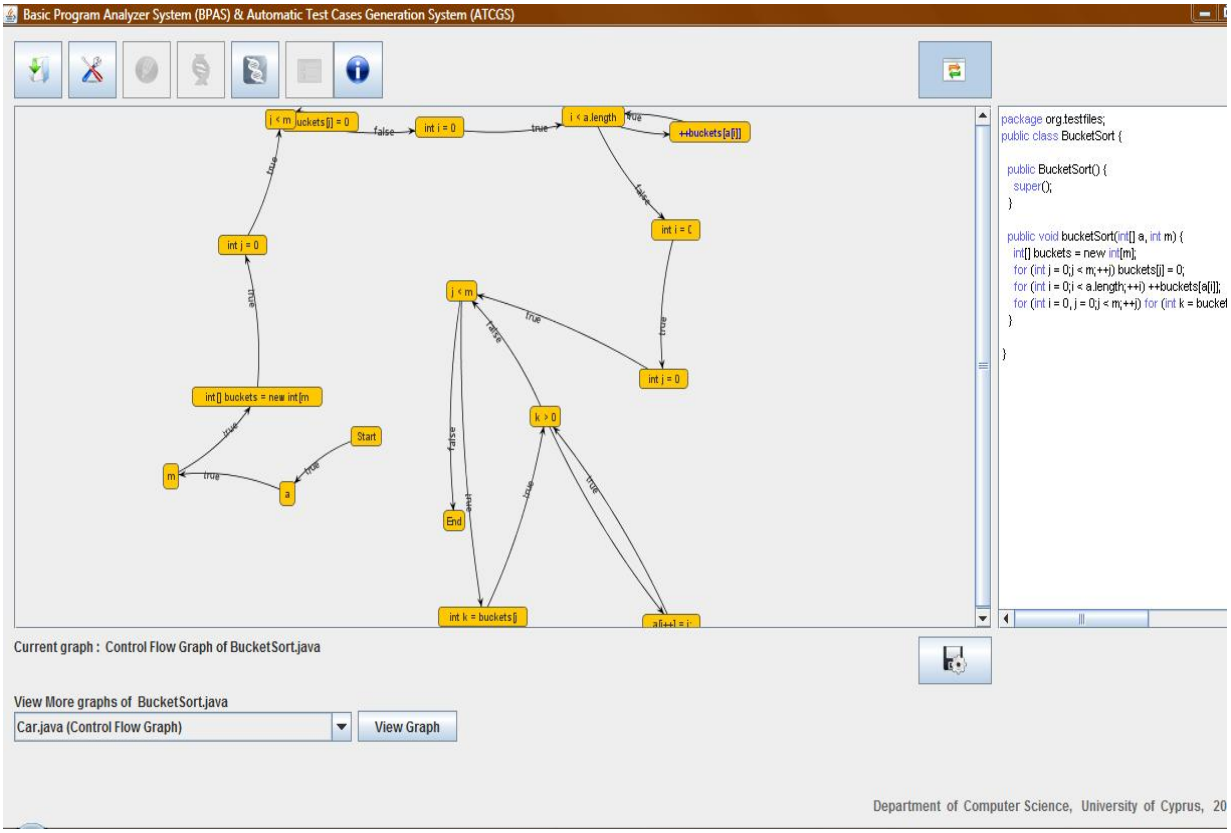
Control Flow Graph

In a **control flow** graph each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block. Directed edges are used to represent jumps in the control flow. By convention, the control flow graph start with an additional vertex called "Start" and all the leaves of the graph have their edges directed to an additional vertex called "End". The directed edges show the flow of control during the execution of the program, from a statement to another statement or from a statement to a control structure (if statement, while statement, etc), or from a control structure to a simple statement, or from a control structure to another control structure (e.g nested if statements, nested loops).

Edges that have as source node a control structure carry a label, indicating the flow of the program depending on the boolean value of the statement (e.g true/false). We can see below a simple program, implementing the bucket sort algorithm and the control flow that is constructing after parsing and creating the graph by using the tool.

```
package org.testfiles;
public class BubbleSort {

  public BubbleSort() {
    super();
  }

  public void bubblesort(int[] A) {
    for (int i = A.length;i >= 1;i++) {
      for (int j = 1;j < i - 1;j++) {
        if (A[j] > A[j + 1]) {
          int temp = 0;
          temp = A[j];
          A[j] = A[j + 1];
```

```
            A[j + 1] = temp;
          }
        }
      }
    }

}
```

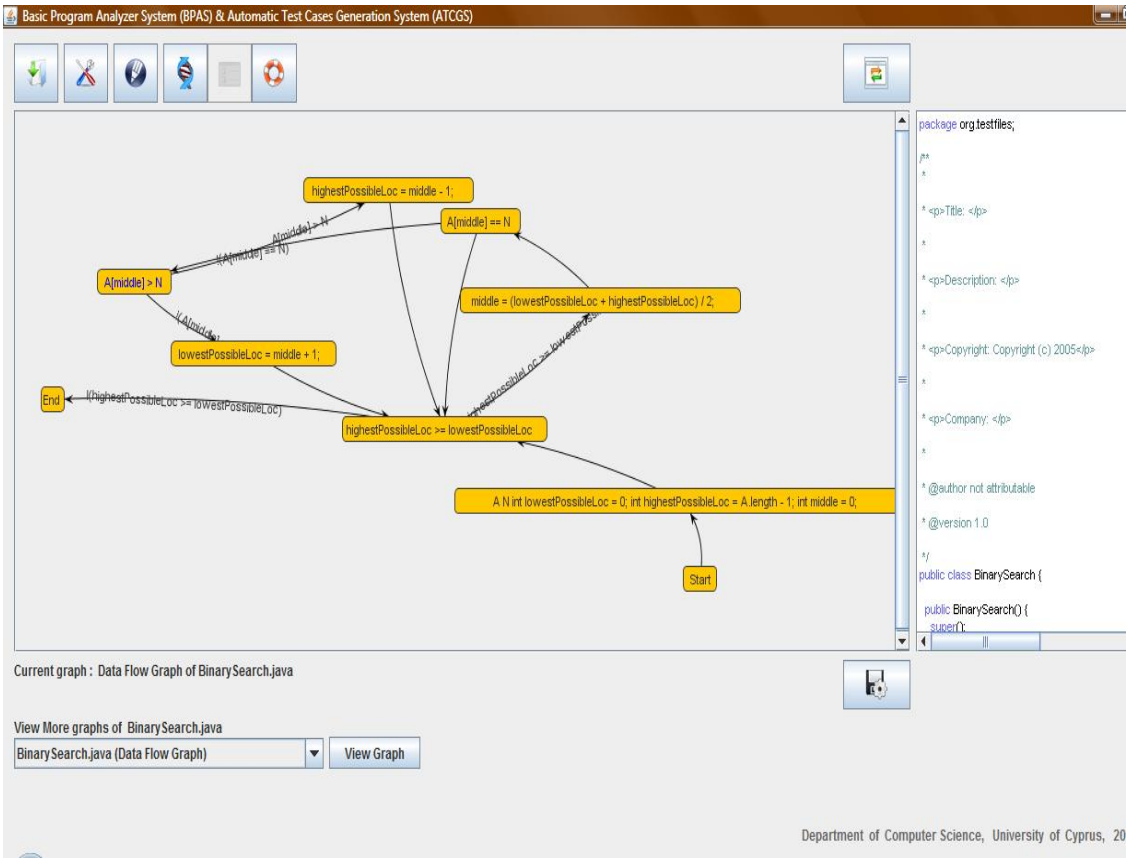


Data Flow Graph

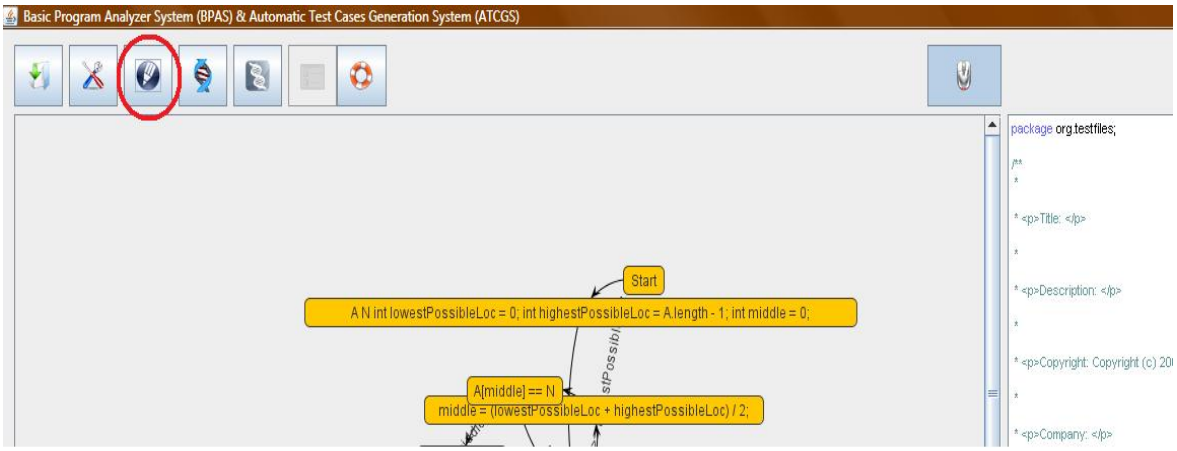
A **data-flow graph** (**DFG**) is a graphical representation of the "flow" of data through an information system [27]. It differs from the flowchart as it shows the *data* flow instead of the *control* flow of the program.

What is referred to as Data Flow Graph in this application, is actually a graph based on the control flow graph, joined with an identical graph and removing multiple declarations (by concatenating them in one node) and statements but leaving unchanged the nodes that have decision-making role. Moreover, the labels of the edges that have a decision-making role source node are changed from true/false boolean values to the condition string and if this holds or not (e.g x<y or !(x<y)).
Below we can see an example of a source code written in java and its correspondent dataflow graph :

```
public class BinarySearch {

  public BinarySearch() {
    super();
  }

  int binarySearch(int[] A, int N) {
    int lowestPossibleLoc = 0;
    int highestPossibleLoc = A.length - 1;
    int middle = 0;
    while (highestPossibleLoc >= lowestPossibleLoc) {
      middle = (lowestPossibleLoc + highestPossibleLoc) / 2;
      if (A[middle] == N) {
        return middle;
      } else if (A[middle] > N) {
        highestPossibleLoc = middle - 1;
      } else {
        lowestPossibleLoc = middle + 1;
      }
    }
    return -1;
  }

}
```

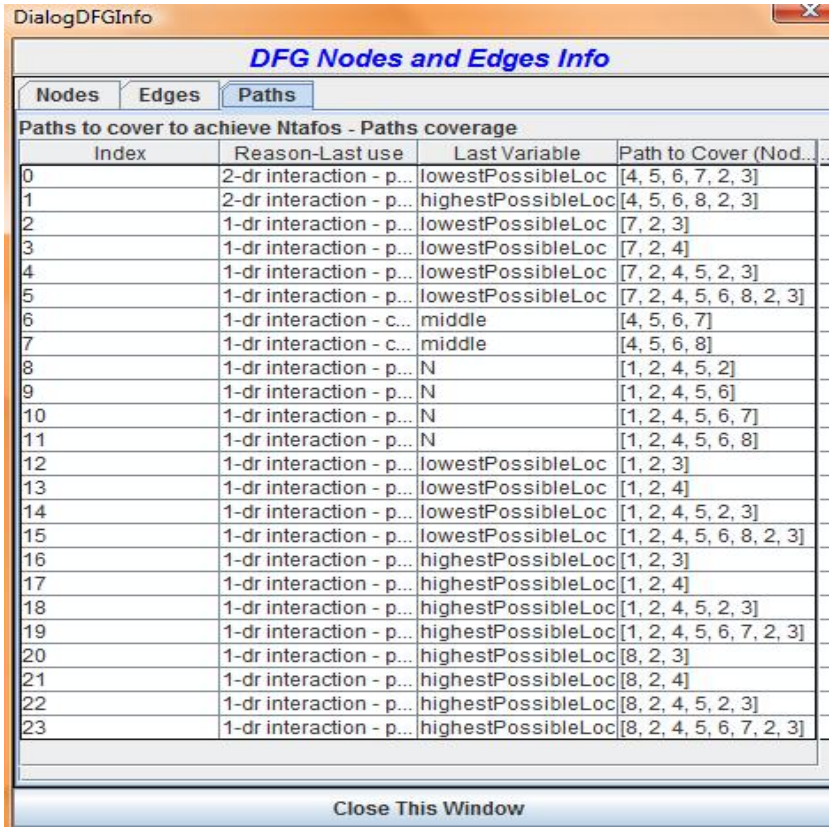


Having created the data flow graph, we can see the information extracted according to the k-tuples criterion in the button shown below :



The information window will open and we can see 3 tabs:

- Edges info
- Nodes info
- Paths info



Index	Reason-Last use	Last Variable	Path to Cover (Nod...
0	2-dr interaction - p...	lowestPossibleLoc	[4, 5, 6, 7, 2, 3]
1	2-dr interaction - p...	highestPossibleLoc	[4, 5, 6, 8, 2, 3]
2	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 3]
3	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 4]
4	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 4, 5, 2, 3]
5	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 4, 5, 6, 8, 2, 3]
6	1-dr interaction - c...	middle	[4, 5, 6, 7]
7	1-dr interaction - c...	middle	[4, 5, 6, 8]
8	1-dr interaction - p...	N	[1, 2, 4, 5, 2]
9	1-dr interaction - p...	N	[1, 2, 4, 5, 6]
10	1-dr interaction - p...	N	[1, 2, 4, 5, 6, 7]
11	1-dr interaction - p...	N	[1, 2, 4, 5, 6, 8]
12	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 3]
13	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 4]
14	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 4, 5, 2, 3]
15	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 4, 5, 6, 8, 2, 3]
16	1-dr interaction - p...	highestPossibleLoc	[1, 2, 3]
17	1-dr interaction - p...	highestPossibleLoc	[1, 2, 4]
18	1-dr interaction - p...	highestPossibleLoc	[1, 2, 4, 5, 2, 3]
19	1-dr interaction - p...	highestPossibleLoc	[1, 2, 4, 5, 6, 7, 2, 3]
20	1-dr interaction - p...	highestPossibleLoc	[8, 2, 3]
21	1-dr interaction - p...	highestPossibleLoc	[8, 2, 4]
22	1-dr interaction - p...	highestPossibleLoc	[8, 2, 4, 5, 2, 3]
23	1-dr interaction - p...	highestPossibleLoc	[8, 2, 4, 5, 6, 7, 2, 3]

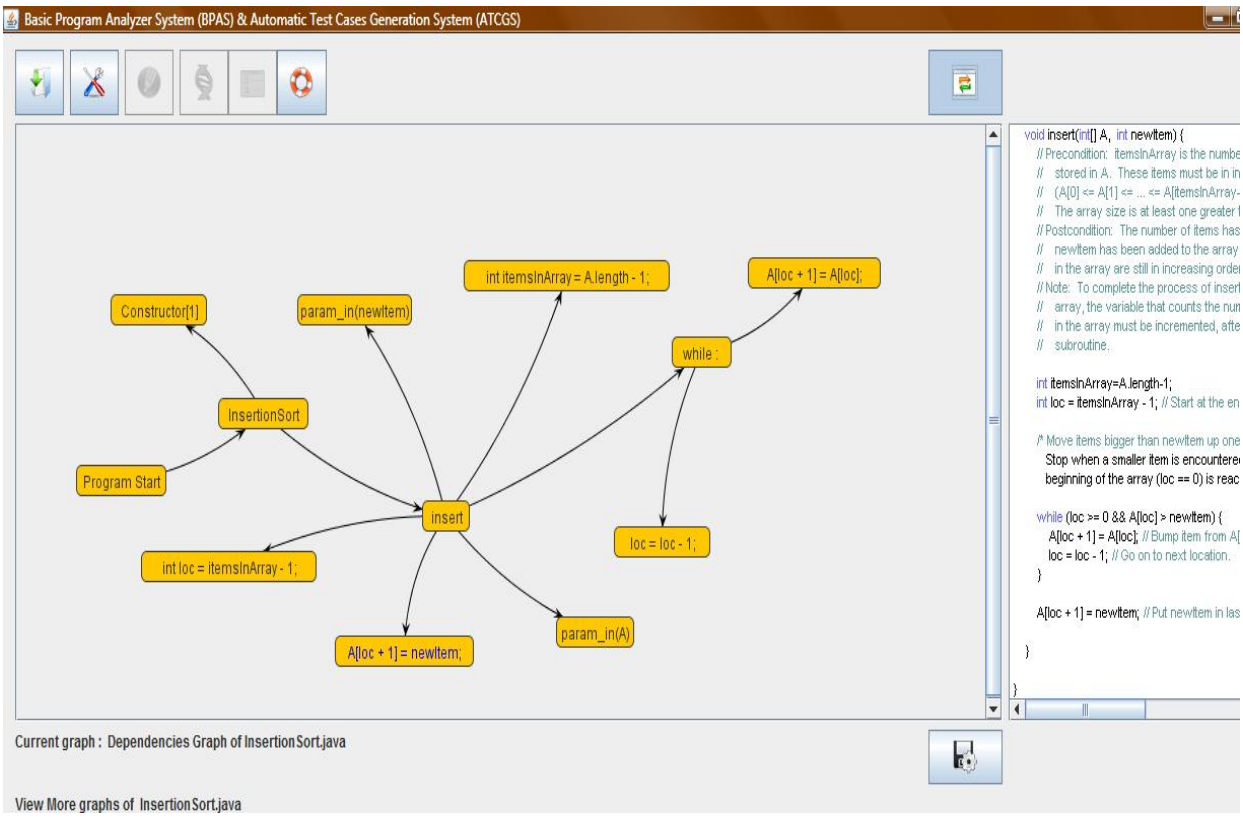
Dependence Graph

Dependence graph is a directed graph representing dependencies of several objects towards each other. Every directed edge, starting from a source node A

can have none, one or a set of destination nodes S, indicating that in order that the element of node A (statement/method/class/constructor/parameter, etc) can execute, it depends on the execution of the set S.
In this application the dependency graph illustrates the dependencies that occur only inside a class. This means that it doesn't show the dependencies between the elements of the class and the packages or dependencies between packages.

We can see below an example of a dependence graph created by the tool after parsing a java source code :

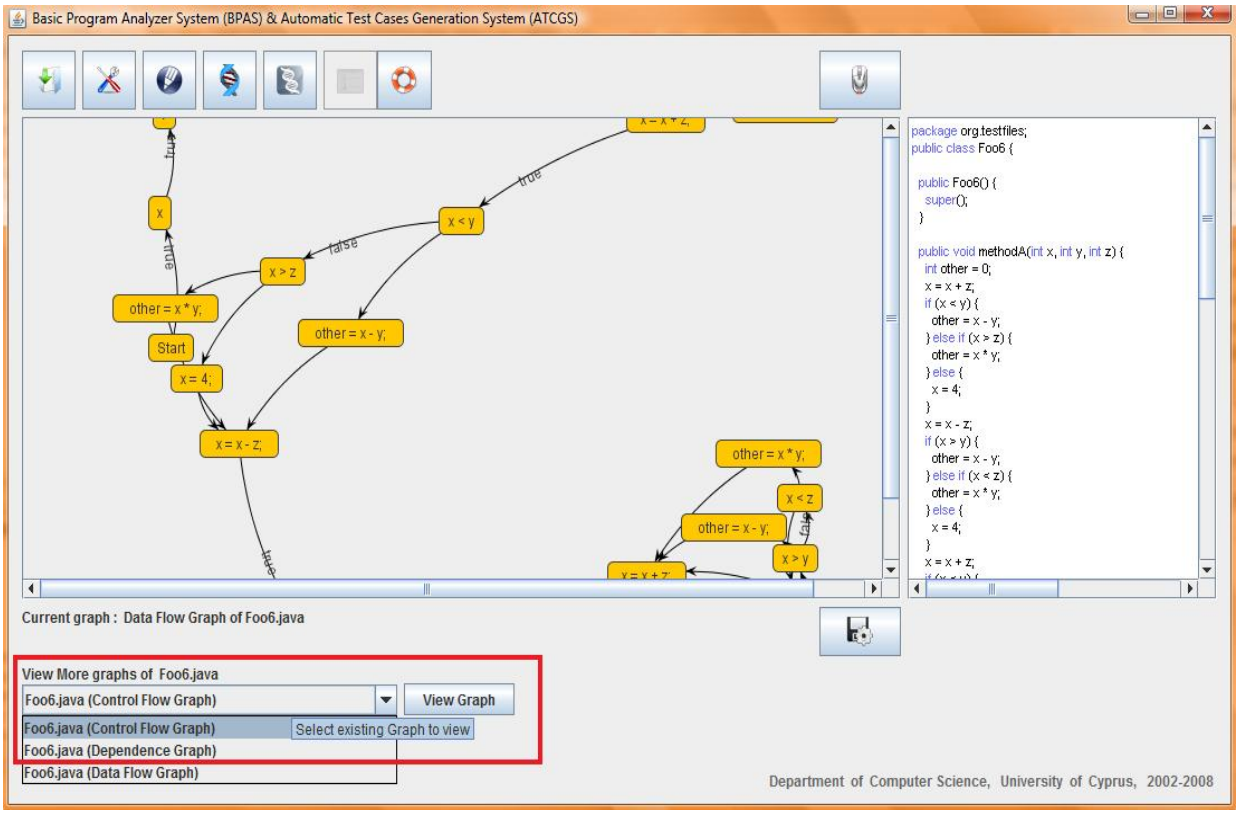
```
public class InsertionSort {
   void insert(int[] A,  int newItem) {
      // Precondition:  itemsInArray is the number of items that are
      //    stored in A.  These items must be in increasing order
      //    (A[0] <= A[1] <= ... <= A[itemsInArray-1]).
      //    The array size is at least one greater than itemsInArray.
      // Postcondition:  The number of items has increased by one,
      //    newItem has been added to the array, and all the items
      //    in the array are still in increasing order.
      // Note:  To complete the process of inserting an item in the
      //    array, the variable that counts the number of items
      //    in the array must be incremented, after calling this
      //    subroutine.

      int itemsInArray=A.length-1;
      int loc = itemsInArray - 1; // Start at the end of the array.

      /* Move items bigger than newItem up one space;
         Stop when a smaller item is encountered or when the
         beginning of the array (loc == 0) is reached. */

      while (loc >= 0 && A[loc] > newItem) {
         A[loc + 1] = A[loc]; // Bump item from A[loc] up to loc+1.
         loc = loc - 1; // Go on to next location.
      }

      A[loc + 1] = newItem; // Put newItem in last vacated space.

   }

}
```



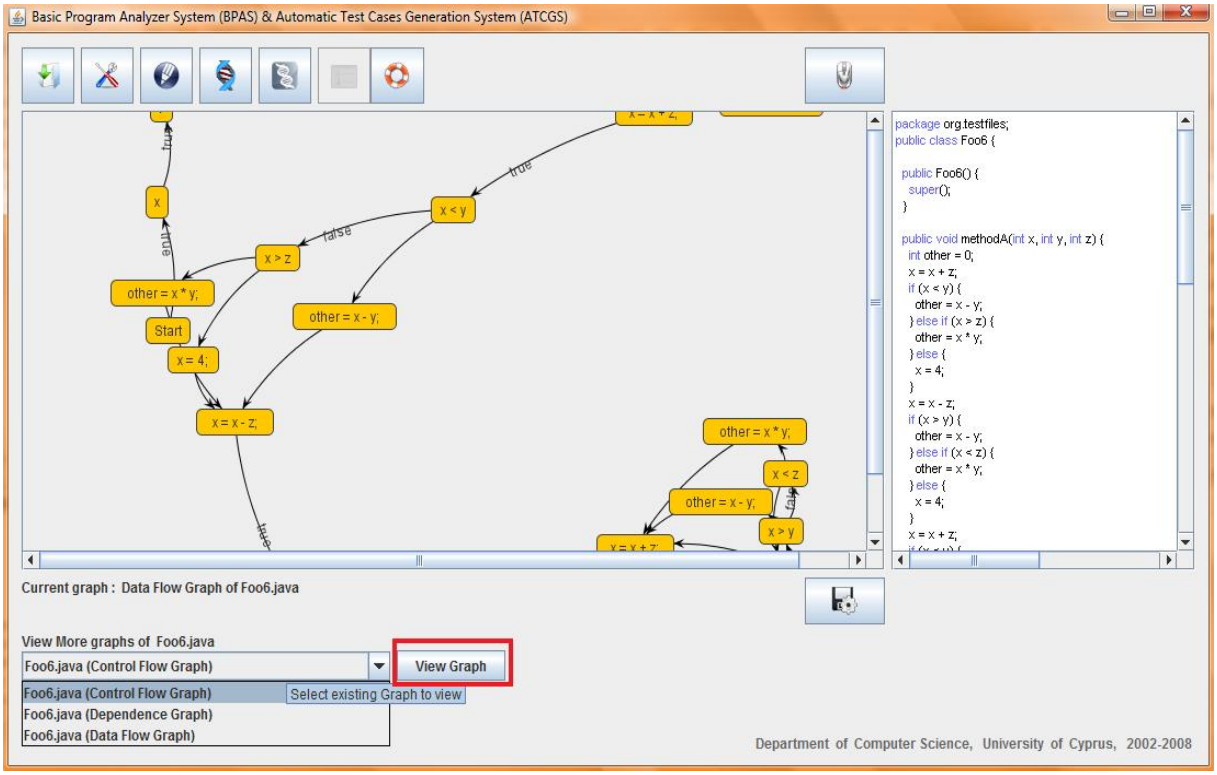
How to View Created Graphs

Graphs are designed to appear in the graph panel as soon as they are constructed and their visualization viewer is completed. Although, if the user selects to create multiple graphs (multiple types of graphs of the same program) then the first thread responsible for parsing the code and creating a type of graph to finish is the first that will add the visualization viewer of the created graph in the graph panel. If another thread is completed after that, then it replaces its visualization viewer of its type of graph with the existing one in the graph panel, and so on. Therefore, it is obvious that the graph panel will contain only the visualization of the type of graph created by the last thread to complete execution. In order that we don't 'lose' the previous visualizations of the graphs created, a combobox is designed for this

purpose. This combobox can be found in the south of the main frame, under the graph panel, as shown below :



As we can see above, the combobox stores all the visualization viewers (panels) of the graphs created, together with the names for the programs and the type of graphs created during the execution of the application. This feature gives the user the ability to see the graphs that have already been created, and quickly view each one of them, being able to compare them without waiting for the parsing of the code, the creation of the graph and the visualization all over again..
This can be done easily by simply clicking on the combobox and select among the source files that the user is interested in viewing and the specific type of graph and then clicking on the button "View Graph", as shown below :



After clicking on "View Graph", the corresponding visualization of the graph, the way the user left it (if it was modified) will appear on the graph panel and in the text editor the source code of that program will appear too (unless it is already in the editor).

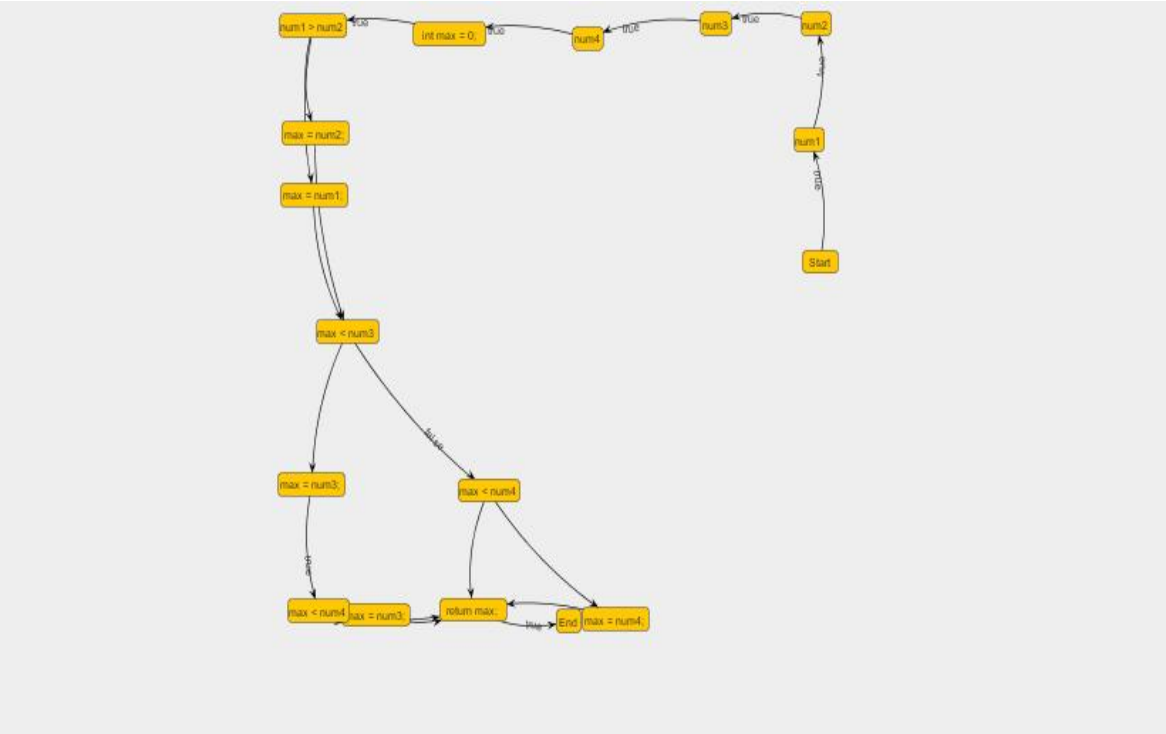
How to Interact with a Graph

There are many ways to interact with a graph in the graph panel. Once the graph is created and viewed in the graph panel, the user can interact with it in 3 ways :

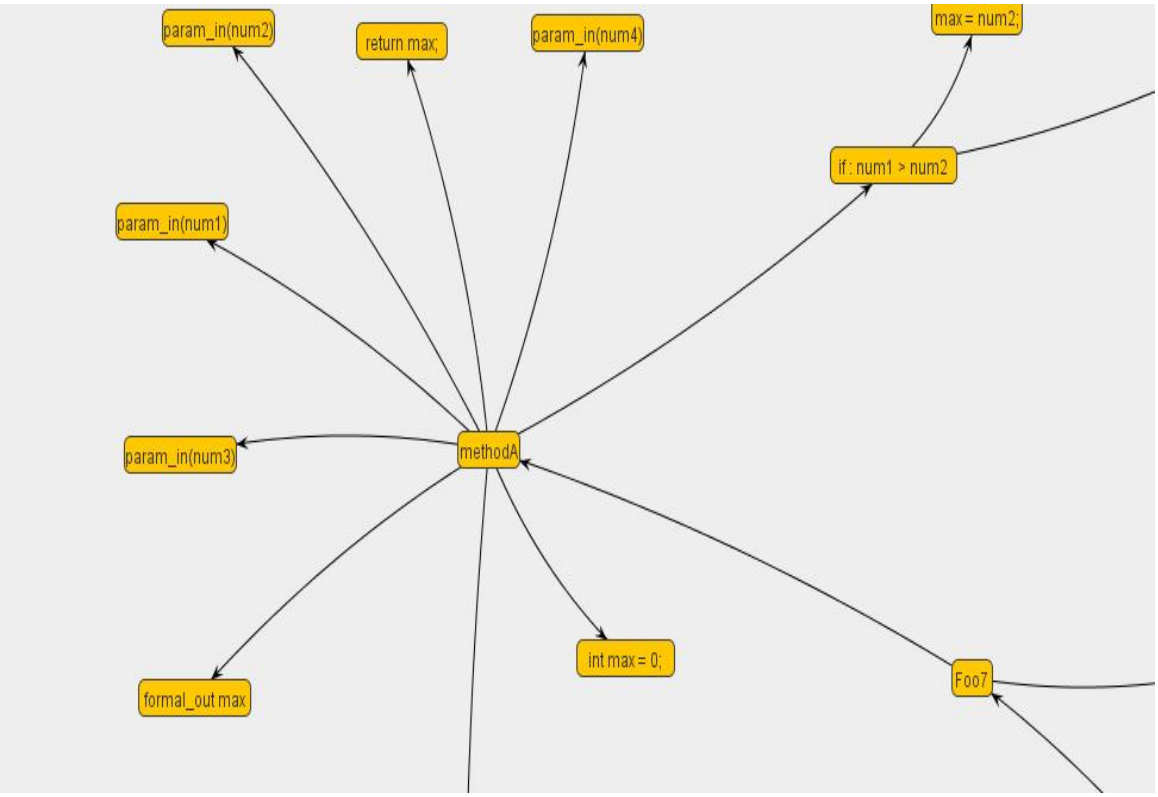
- Moving the mouse scroll wheel up will zoom in the graph regarding as center the mouse pointer.
- Moving the mouse scroll wheel down will zoom out in the graph regarding as center the mouse pointer.
- By having the left-click pressed and dragging the graph will move the graph to the direction of the movement of the mouse
- Holding ctrl + click (at the same time) and dragging the graph will stretch the graph to the correspoding direction of the mouse movement
- Holding shift + click (at the same time) and dragging the graph will rotate the graph as if on an axis parallel to the mouse pointer's direction.

Moreover, aesthetically pleasing graph layouts [28] have been made available for the user to chose.

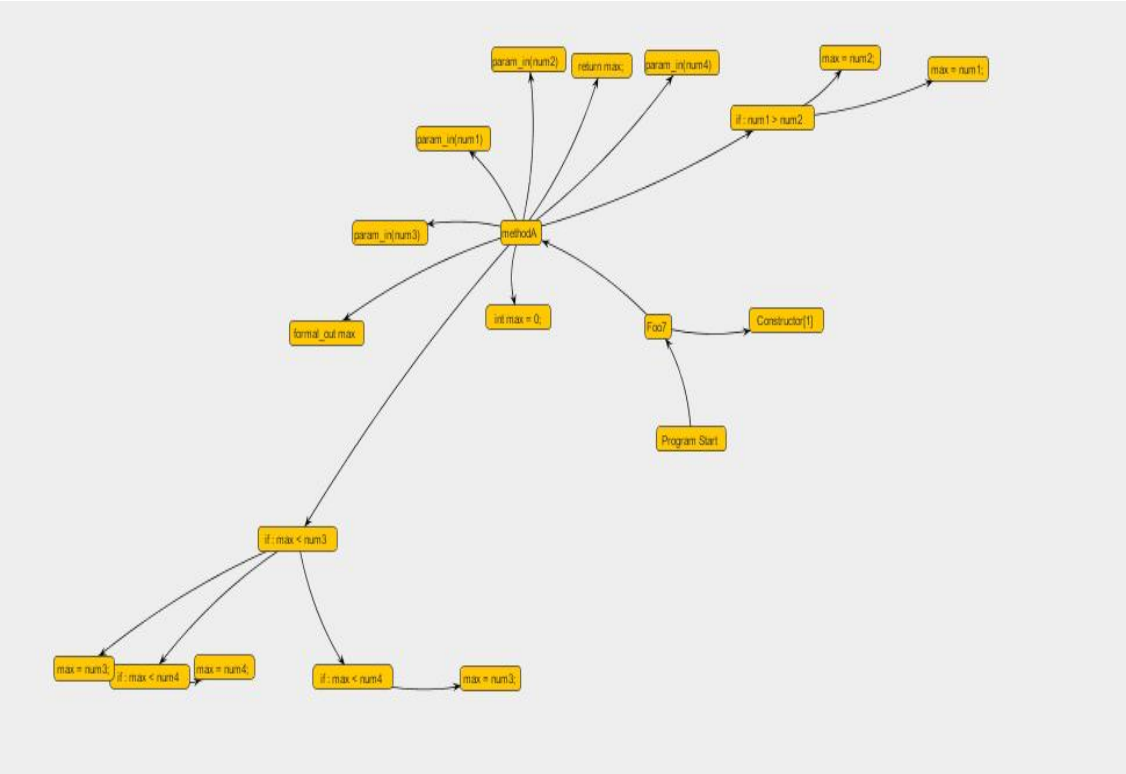
We can now see some examples of the ways of interaction with the graph that are mentioned above. We can see a dependency graph here derived from the source code Foo7.java



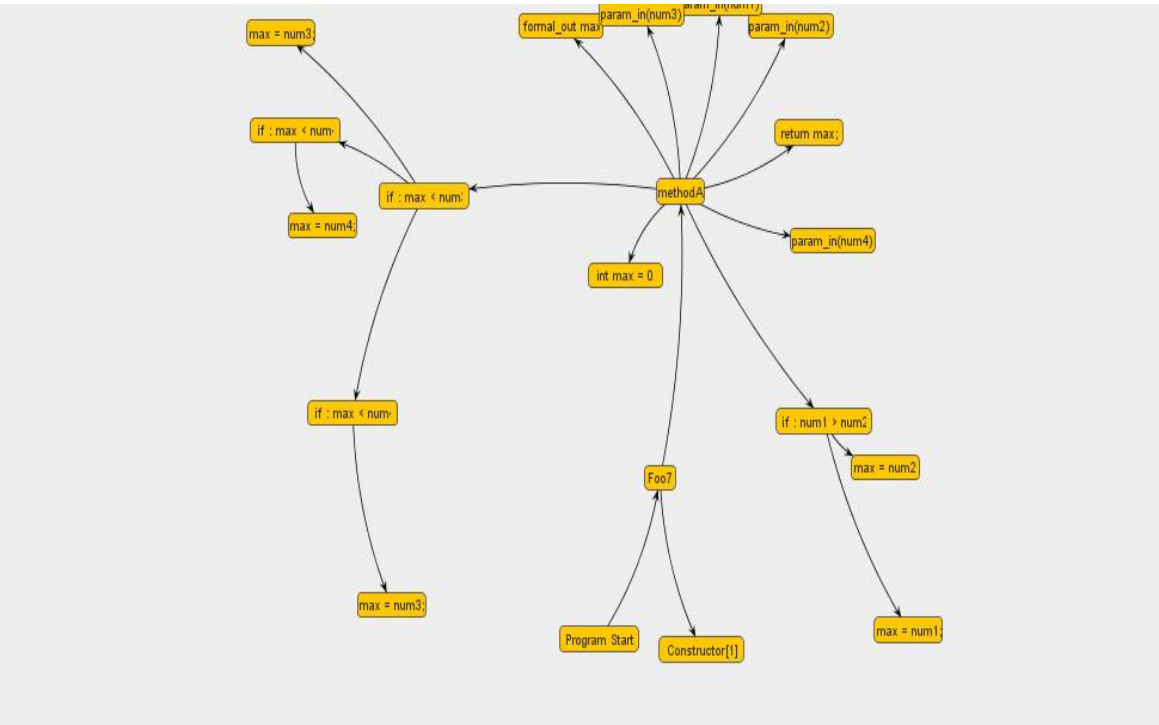
In the next scheme we can observe the graph's reaction to the mouse wheel scroller move, as we zoom in the node methodA :



Below, we can see the transformation that takes place by stretching the same graph to the right.

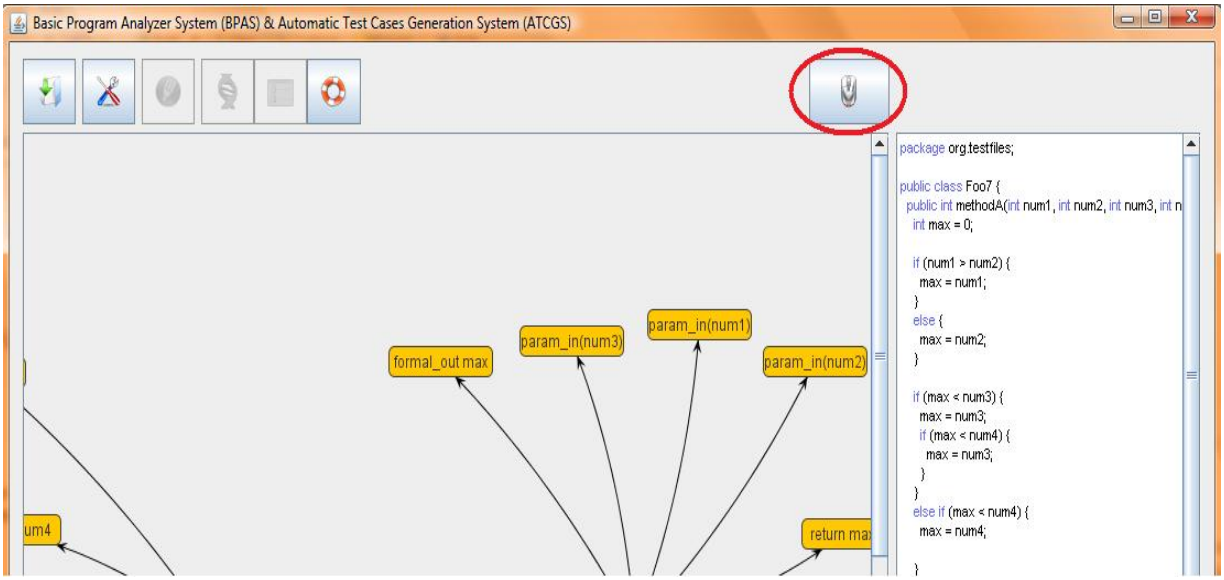


In the following scheme we can see the resut of the rotation (rotating about 90 degrees to clockwise) in the same visualization of the graph :

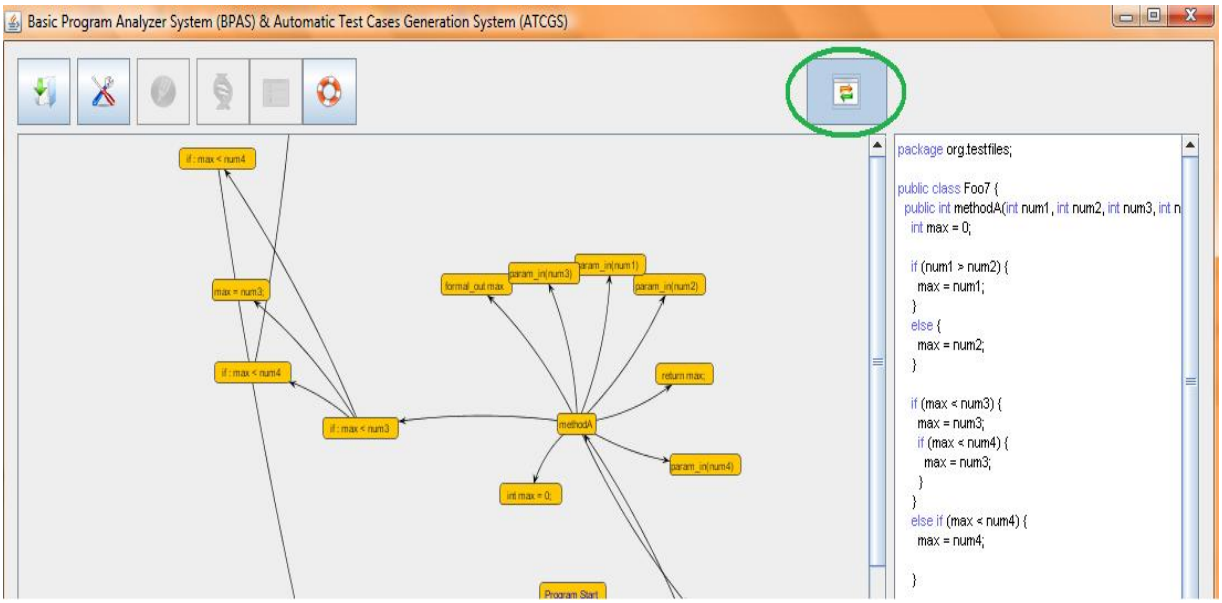


We are going to see now a new feature added to the application concerning the ways a user can interact with a graph. This is called mouse-graph interaction and is a collection of standard interaction with a graph and its components. The user can change this interaction from the default one, that is non-mouse-graph, and has been described in the examples above, to another kind of interaction. the mouse-graph. In few words, this interaction allows the user to pick one or more nodes with the mouse and to move them separately from the rest of the graph. This means that while he can change a node or a group of nodes to move, by selecting and dragging them, the rest of the graph will remain untouched. This is particularly useful when we have to deal with a dense graph for which the graph panel is not big enough and the nodes are very close to each other, making it impossible to distinct most of the edges. In this case, good solution would be for the user to zoom out the graph, making space around it and separate the nodes he is interested in viewing by picking them as a group or only one node and dragging them away from the rest of the graph. It is important to note here that the mouse-graph interaction still allows the user the zoom in/out capability but replaces the stretching and rotating abilities with picking - isolating groups of nodes, or individual ones.

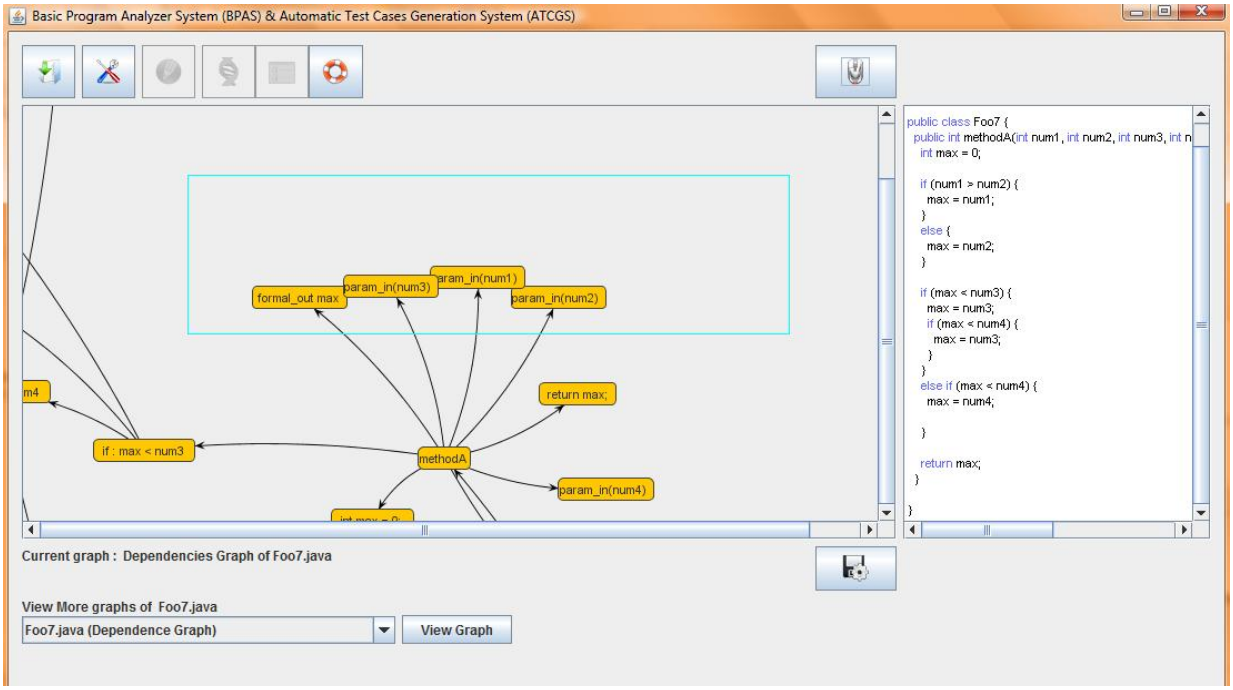
To activate this feature you need to click on the button shown below, in the application :



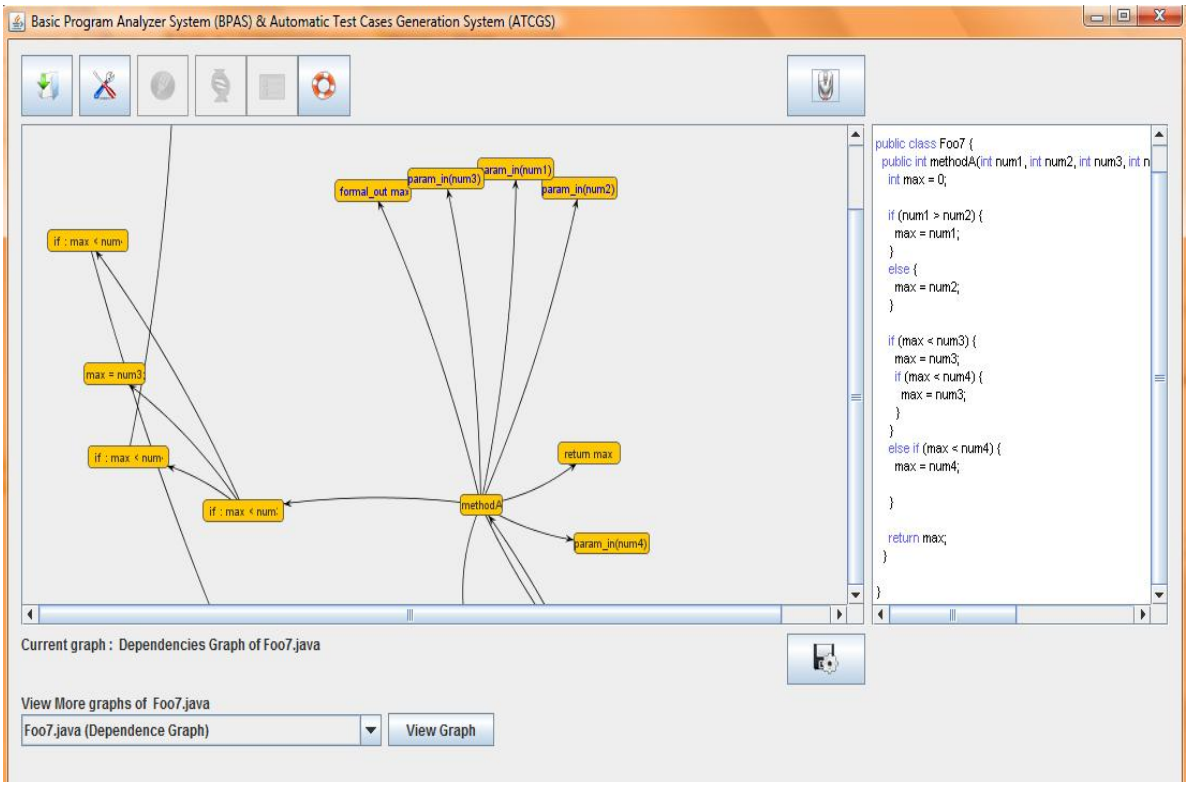
Once you press this button, you will notice the icon changing from a mouse icon, to an icon with two arrows, indicating that you have selected to move the graph as a whole, and therefore, to use the first kind of interaction. Let's take a look to the next scheme, illustrating this action :



When the interaction is changed to mouse-graph, picking individual nodes, the mouse pointer on mouse-over on the graph panel will be replaced by a hand. When we make sure this is the interaction we want, we can start picking edges by clicking on them and dragging them, or groups of nodes by holding shift and left-click, as we can notice a square get drawn on the graph panel, changing the nodes' names labels from black to blue, indicating we are selecting those nodes. We can understand this better looking at the scheme below :



Having selected those four nodes shown above (formal_out max, param_in(num3), param_in(num2), param_in(num1) that supposedly we are interested to focus on) we can try to move them up. As we can see in the scheme below, we managed to move them up, and therefore to isolate them a little but from the rest of the graph.



Graph Layout Algorithms

Graphs are usually represented pictorially using dots to represent vertices, and arcs to represent the edges between connected vertices. Arrows can be used to show the orientation of directed edges [27, 28]. Note that this graphical representation (a *graph layout* or an *embedding*) should not be confused with the graph itself (the abstract, non-graphical structure). Very different layouts can correspond to the same graph. In the abstract, all that matters is which vertices are connected to which others by how many edges. In the concrete, however, the arrangement of these vertices and edges impacts understandability, usability, fabrication cost, and aesthetics.

Based on these concepts and caveats, there are different graph layout strategies, such as:

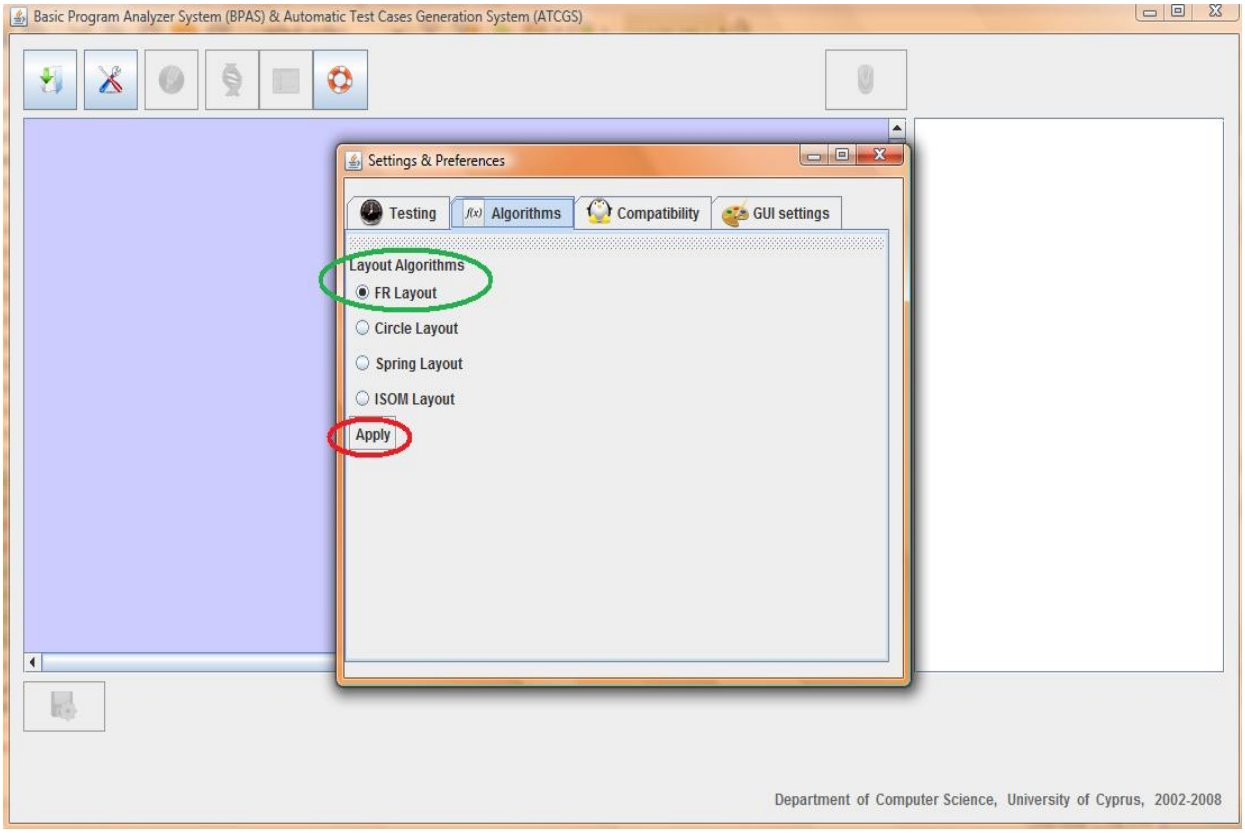
- force-based layout: gradient descent minimization of an energy function based on physical metaphors related to molecular mechanics.
- spectral layout: layout using as coordinates the eigenvectors of a matrix such as the Laplacian derived from the adjacency matrix of the graph.
- orthogonal layout: layout with edges running horizontally or vertically, with approaches that reduce the number of edge crossovers and area covered. These are of great interest in the areas of VLSI and PCB layout design
- symmetric layout: these attempt to find symmetry groups within the graph
- tree layout: these show a rooted tree-like formation, suitable for trees (i.e., graphs without cycles)
- hierarchical layouts: these attempt to find a source and sink within a directed graph and arrange the nodes in layers with most edges starting from the source and flowing in the direction of the sink

The tool offers the user a variety of graph layouts, aiming to serve different purposesaccording to the graph properties (sparse, dense,etc) and the user's preferences. These graph layouts are offered by the new graph library, jung 1.7.6. and are available to the user in the settings -> layouts menu. You can see in the next chapters some examples on how to select, activate and interact with the different kinds of graphs layouts.

Fruchterman-Reingold Graph Layout Algorithm

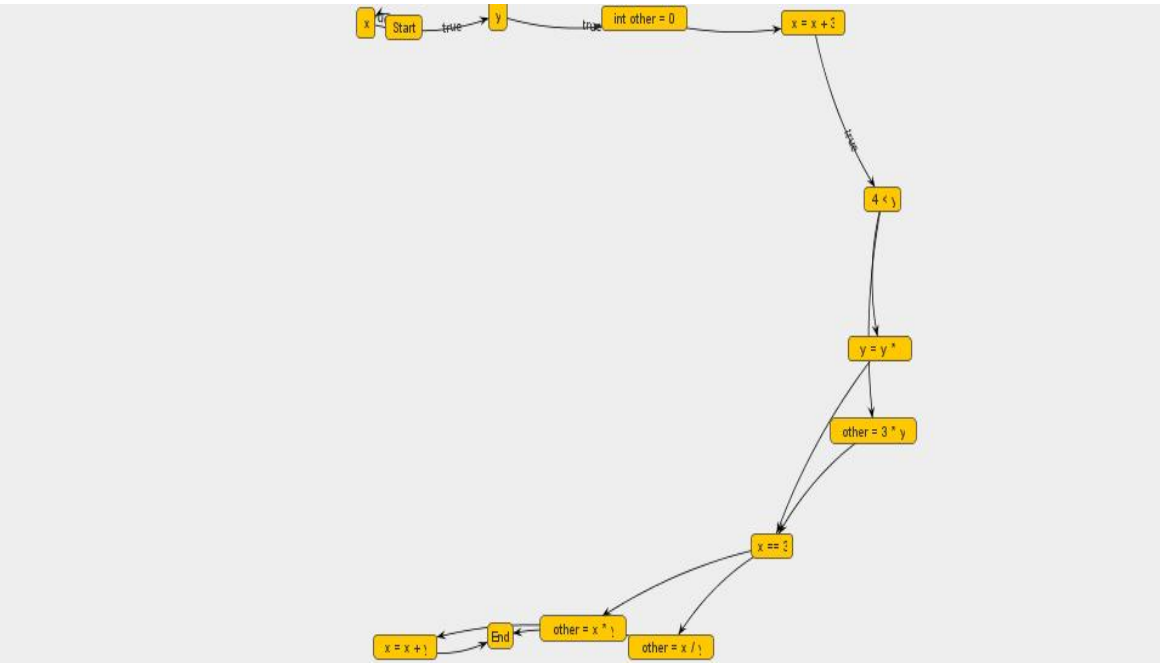
According to this layout algorithm, performs layout of unweighted graph. Unlike the Kamada-Kawai layout algorithm [27,28], this algorithm directly supports the layout of disconnected graphs (but see the `force_pairs` named parameter). It is a *force-directed* algorithm, meaning that vertex layout is determined by the forces pulling vertices together and pushing them apart. Attractive forces occur between adjacent vertices only, whereas repulsive forces occur between every pair of vertices. Each iteration computes the sum of the forces on each vertex, then moves the vertices to their new positions. The movement of vertices is mitigated by the *temperature* of the system for that iteration: as the algorithm progresses through successive iterations, the temperature should decrease so that vertices settle in place. In order to chose this algorithm in the tool, there are the following steps to follow :

- Open the Settings Menu and chose the Algorithms Tab.



- Select the FR Layout radio button
- Click on "Apply" button, and it's set.
- The next graph that will be created will appear in tha graph panel with the Fruchterman-Reingold algorithm.

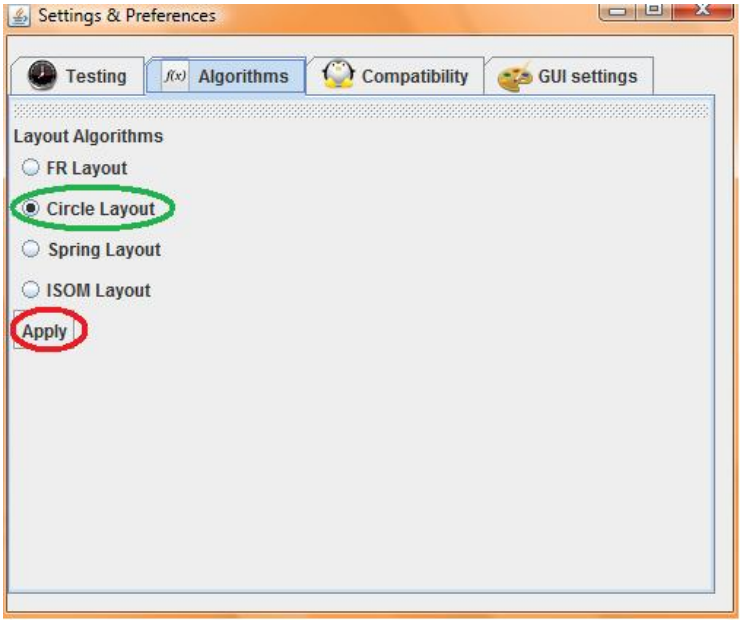
We can see in the next picture how the FR layout looks like in one of the programs



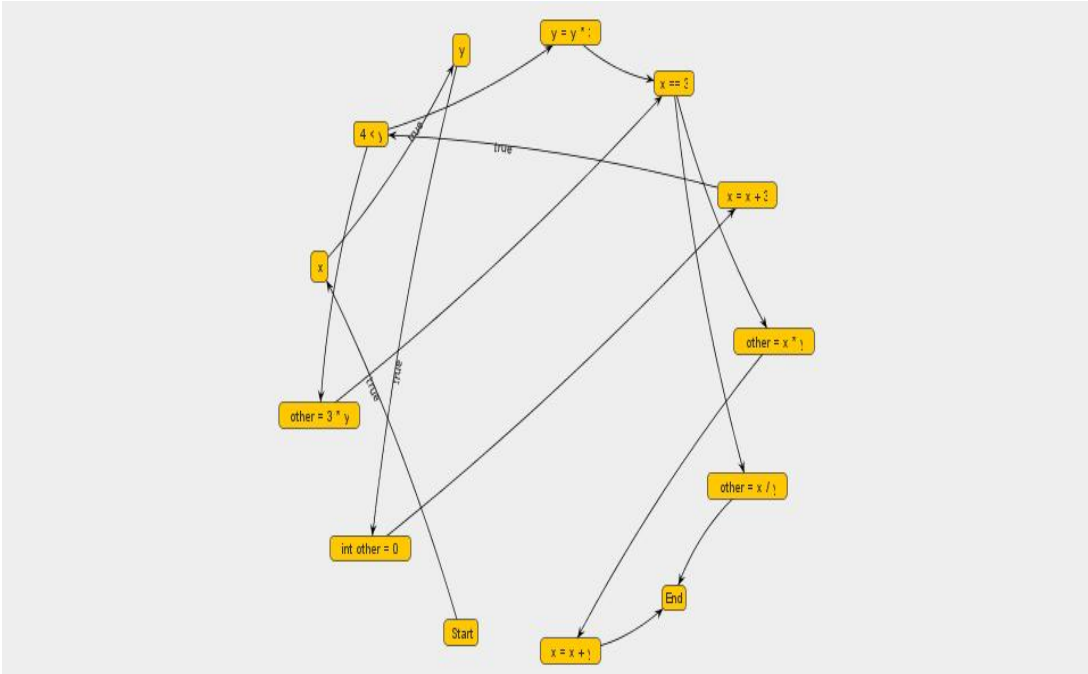
Circle Graph Layout Algorithm

According to this layout algorithm, the graph will expand, depending on the sixe of the set of the nodes, forming a circle [28]. The nodes of the graph will be placed around an imaginary center, will most of the edges will cross near that center. To avoid congestion of the edges, the neighboring nodes are placed as near as possible around the graph, therefore having their connecting edges far from the center of the circle. It is a practical and very useful layout algorithm, especially for graphs that are sparse.

We can see some examples on how to select and activate this layout algorithm on the following schemes :



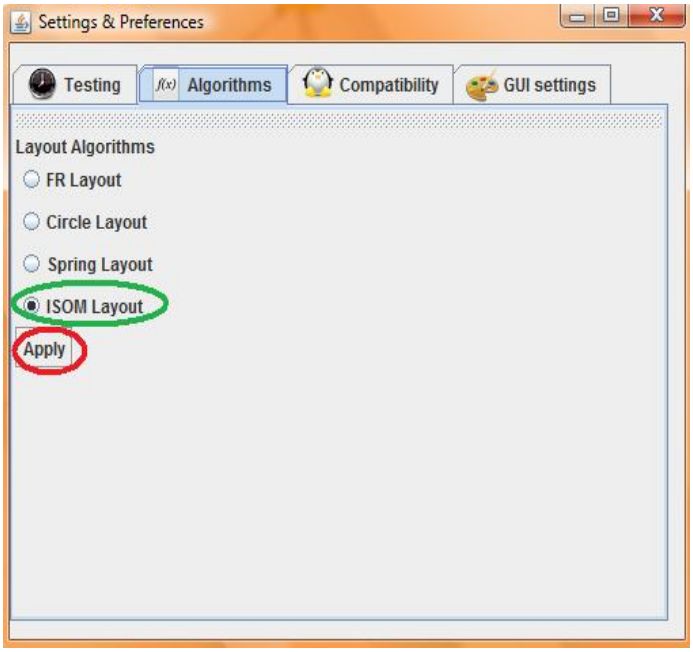
And the layout of the graph, having selected the circle layout and clicked on "Apply" button, will look like this :



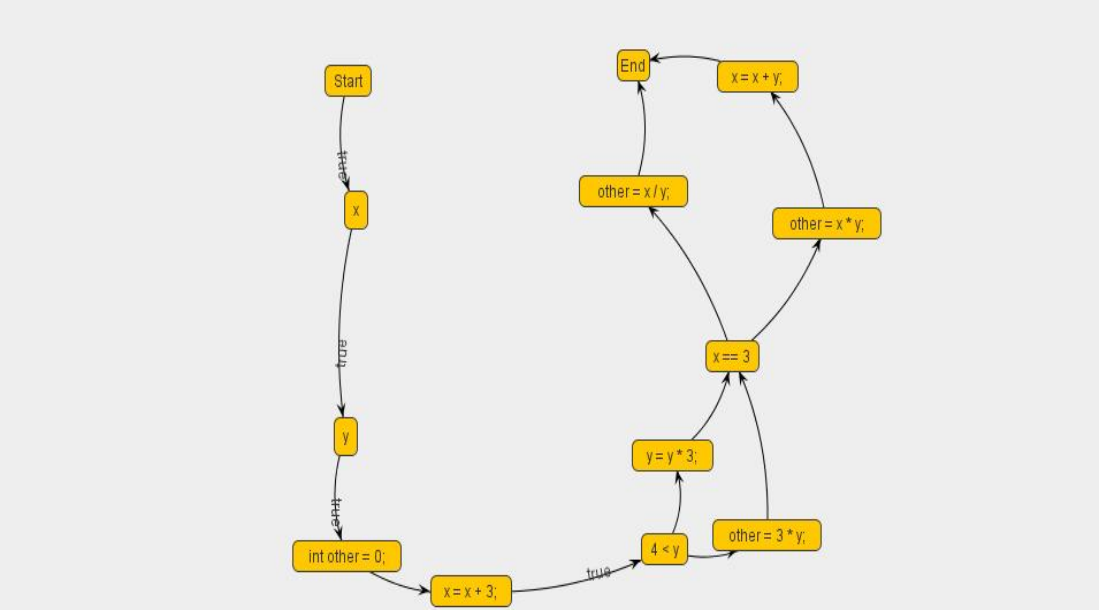
ISOM Graph Layout Algorithm

This layout is based on inverted self-organising maps. Self-organizing maps are similar with force-based layouts, as linked nodes have the tendency to form clusters [28]. The difference with the maps is that there is a uniform space filling distribution of nodes. This makes the bounds within which the layout takes place important to be calculated correctly at the beginning of the processing. ISOM layout algorithm is recommended to be used with well-connected graphs.

In the following schemes we can see how to set this layout algorithm and view an example of this layout applied on a graph of a sample source code :



And having selected the ISOM Layout radio button and clicked on the "Apply" button, we can see the effect on the layout of the graph on a sample program :



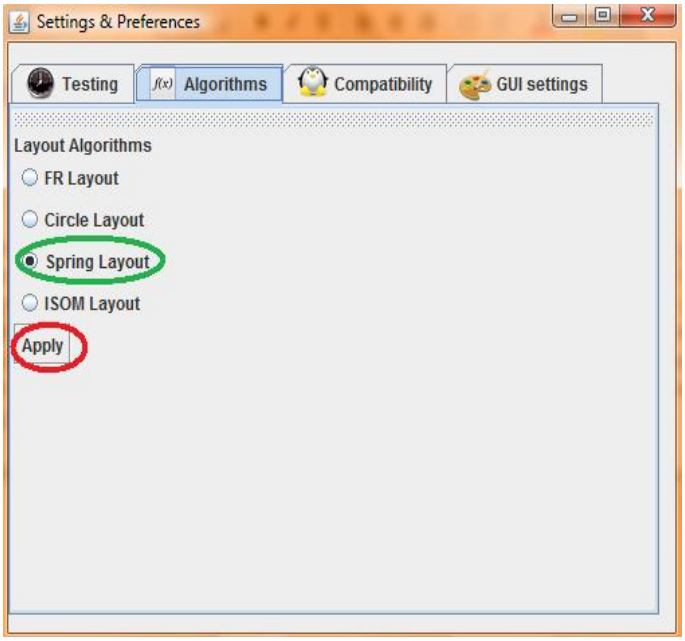
Spring Graph Layout Algorithm

This layout algorithm is on of the force-directed algorithms. Its purpose is to position the nodes of a graph in two dimensional or three dimensional space so that all the edges are of more or less equal length and there are as few crossing edges as possible. It can achieve this by assigning forces amongst the set of edges and the set of nodes; the most straightforward method is to assign forces as if the edges were springs (see Hooke's law) and the nodes were electrically charged particles (see Coulomb's law). The entire graph is then simulated as if it were a physical system. The forces are applied to the nodes, pulling them closer together or pushing them further apart. This is repeated iteratively until the system comes to an equilibrium state; i.e., their relative positions do not change anymore from one iteration to the next. At that moment, the graph is drawn. The physical interpretation of this equilibrium state is that all the forces are in mechanical equilibrium.

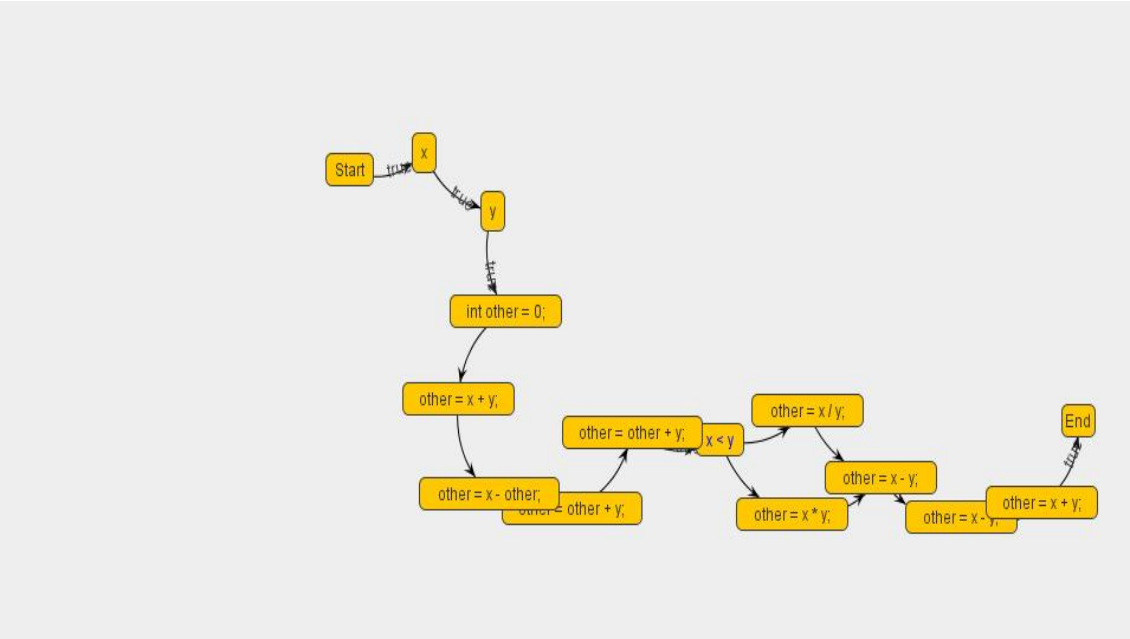
This graph is the only one among the 4 layout algorithms of the tool that is animated. This means that since the moment the graph is constructed and added in the graph panel, it will continue moving inside the panel according to the algorithm mentioned above. An aesthetically pleasing graph layout, that the user can interact by changing the interaction mode to graph-mouse interaction and

picking an individual node of the graph and dragging it. While dragging the node, the rest of the graph will follow to this direction, following the spring algorithm and continue moving.

In order to use this layout algorithm, it has to be selected from the algorithms tab in the settings menu and click on apply, as shown below :



As mentioned before, this is an animated algorithm. This means that the implementation of the graphs library is to continuously move, as electrically charges particles, like explained above, with the edges maintaining the nodes together, while the nodes, as particles charged with the same electrical charge, are trying to move away from each other. The graph can take various shapes and continue trasforming forever, according to the library's documentation so the following screenshot is an instance of this animated graph.



What is Test Case Generation

A **test case** in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system meets specifications.

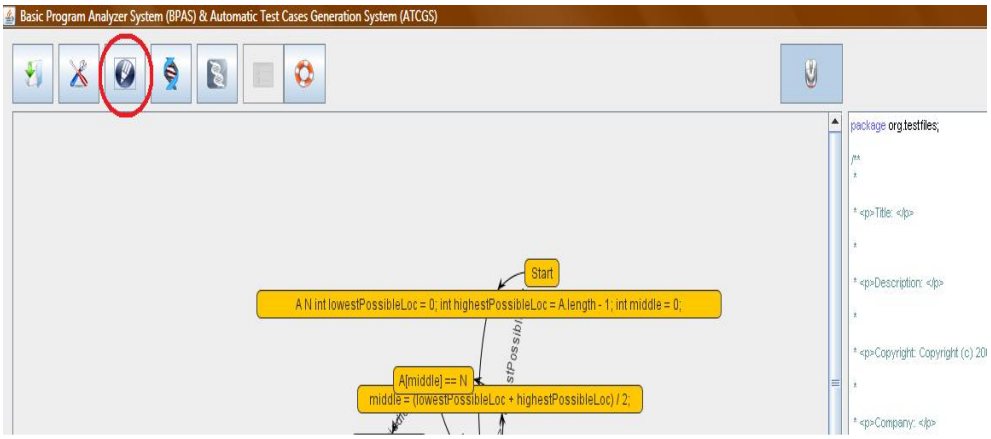
Depending on the graph creation, different metrics are used for the generation of test-cases, execution of the source code and determinig the level of coverage according to various criteria. Some of those criteria are mentioned here :

- **Statement coverage** - Has each line of the source code been executed?
- **Decision coverage** (also known as Branch coverage) - Has each control structure (such as an if statement) been evaluated both to true and false?
- **Condition coverage** - Has each boolean sub-expression been evaluated both to true and false (this does not necessarily imply decision coverage)?
- **Path coverage** - Has every possible route through a given part of the code been

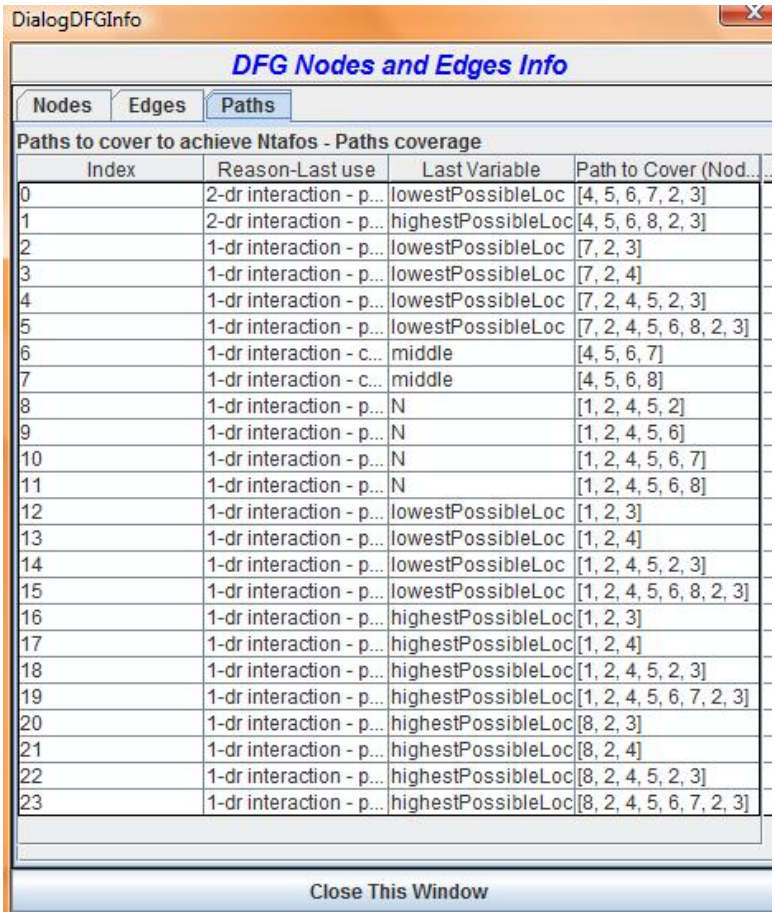
executed?

Test Case Generation based on Data Flow Graph

Having created the dataflow graph, we can see the paths that are extracted according to the k-tuples criterion by clicking the button with the pen icon in the main frame. When ready, we can start with the test case generation by clicking to the icon shown below :

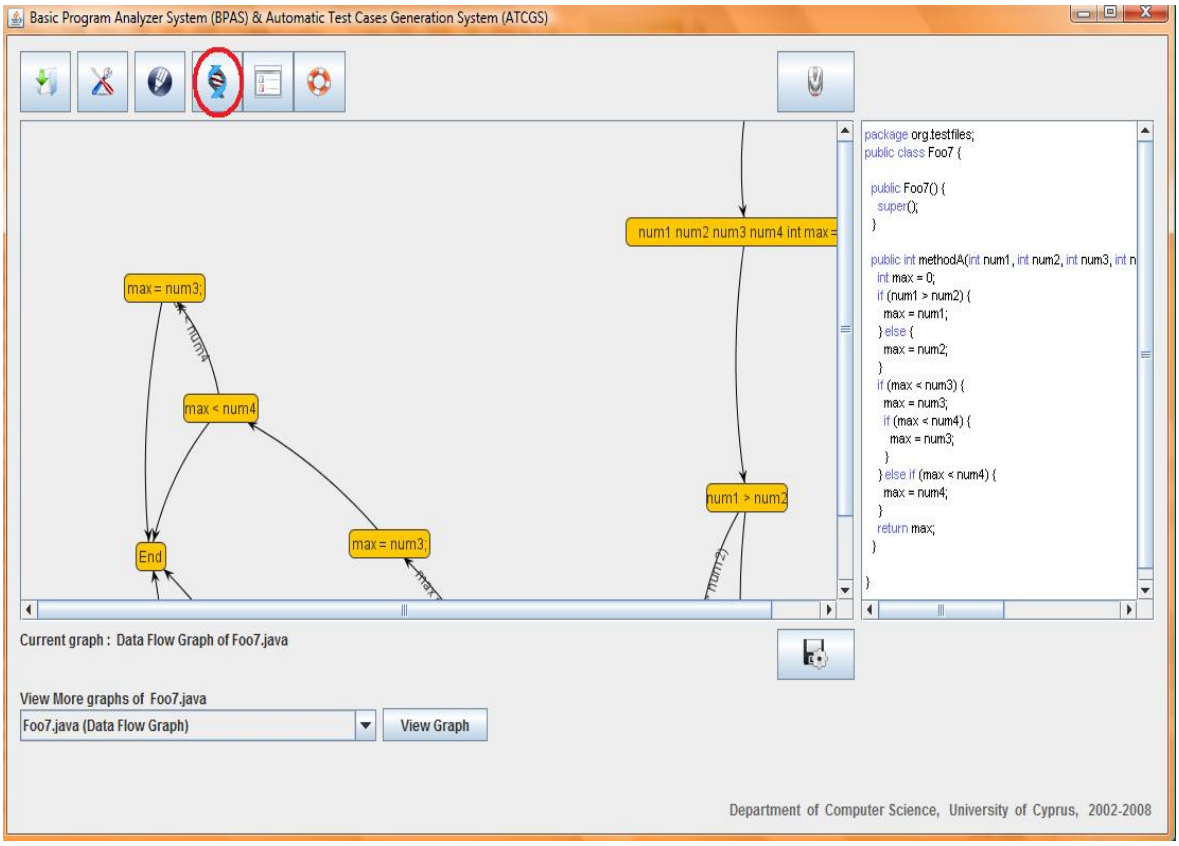


The following info window should open :

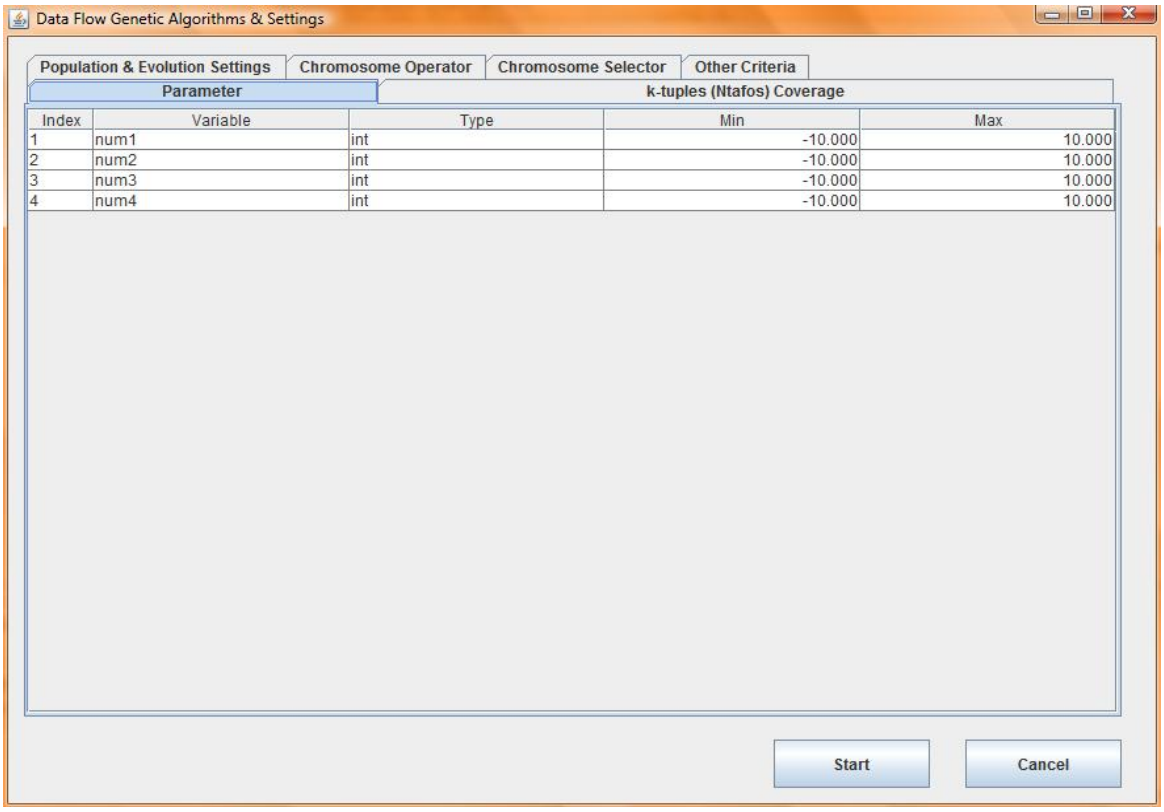


Index	Reason-Last use	Last Variable	Path to Cover (Nod...
0	2-dr interaction - p...	lowestPossibleLoc	[4, 5, 6, 7, 2, 3]
1	2-dr interaction - p...	highestPossibleLoc	[4, 5, 6, 8, 2, 3]
2	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 3]
3	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 4]
4	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 4, 5, 2, 3]
5	1-dr interaction - p...	lowestPossibleLoc	[7, 2, 4, 5, 6, 8, 2, 3]
6	1-dr interaction - c...	middle	[4, 5, 6, 7]
7	1-dr interaction - c...	middle	[4, 5, 6, 8]
8	1-dr interaction - p...	N	[1, 2, 4, 5, 2]
9	1-dr interaction - p...	N	[1, 2, 4, 5, 6]
10	1-dr interaction - p...	N	[1, 2, 4, 5, 6, 7]
11	1-dr interaction - p...	N	[1, 2, 4, 5, 6, 8]
12	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 3]
13	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 4]
14	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 4, 5, 2, 3]
15	1-dr interaction - p...	lowestPossibleLoc	[1, 2, 4, 5, 6, 8, 2, 3]
16	1-dr interaction - p...	highestPossibleLoc	[1, 2, 3]
17	1-dr interaction - p...	highestPossibleLoc	[1, 2, 4]
18	1-dr interaction - p...	highestPossibleLoc	[1, 2, 4, 5, 2, 3]
19	1-dr interaction - p...	highestPossibleLoc	[1, 2, 4, 5, 6, 7, 2, 3]
20	1-dr interaction - p...	highestPossibleLoc	[8, 2, 3]
21	1-dr interaction - p...	highestPossibleLoc	[8, 2, 4]
22	1-dr interaction - p...	highestPossibleLoc	[8, 2, 4, 5, 2, 3]
23	1-dr interaction - p...	highestPossibleLoc	[8, 2, 4, 5, 6, 7, 2, 3]

When ready to start with the test case generation with GA, we can click on the icon shown in the next scheme :



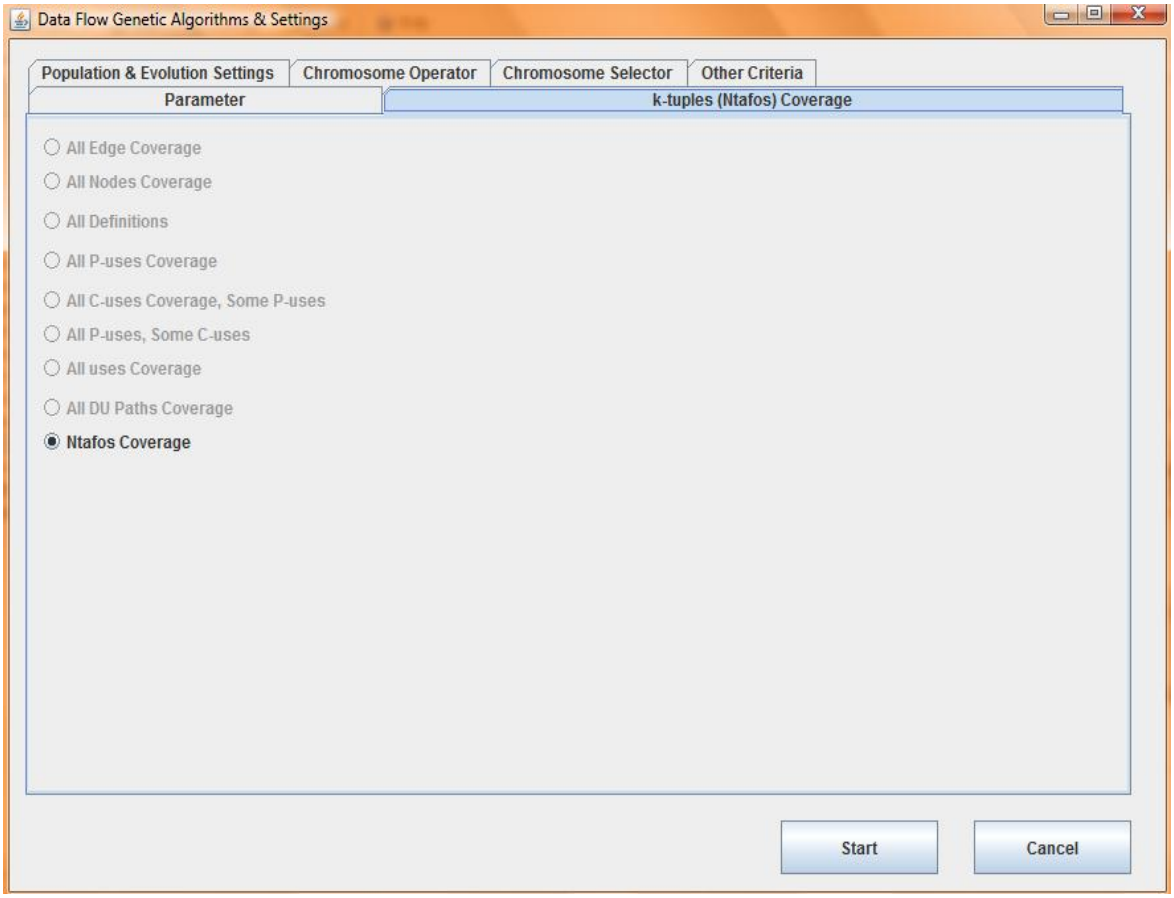
By clicking this button, we trigger the genetic algorithms frame to open, as we can see in the scheme below



We can see that this first tab shows tha parameters of the source code of the sample program Foo7.java.
We can see that there are 3 paramters : num1, num2, num3, num4 that are all integer numbers.
The values in the last 2 columns indicate the minimum and maximum values that can be assigned to these parameters during the test case generation process, Therefore, by default, those numbers can have values from -10.000 to 10.000.

The user can change the values of these boundaries and the changes will be saved when "Start" button is clicked and the Genetic Algorithms start the test case generation and testing.

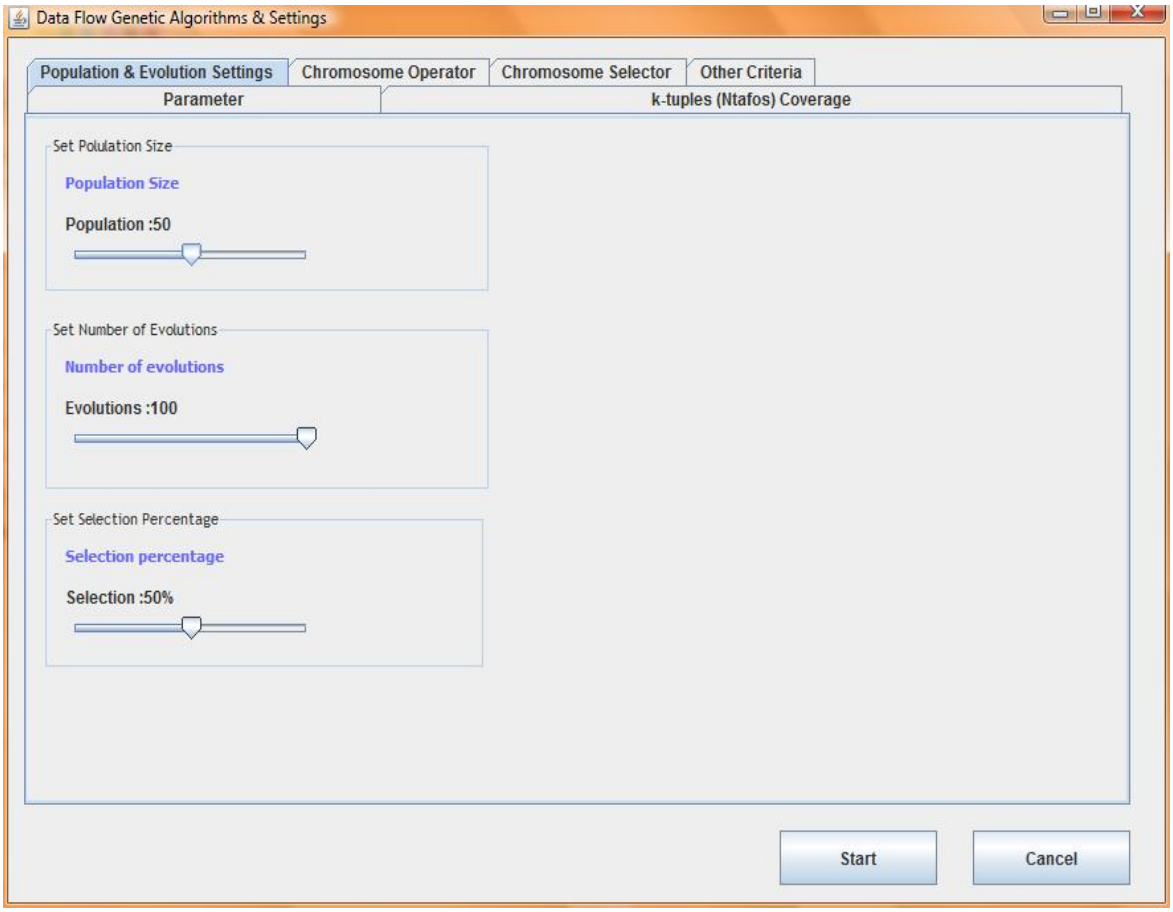
In the second tab we can see the coverage criteria that are available in the tool. By default, the Ntafos Coverage is used, as we can see below :



In the third tab we can see the Population & Evolution Settings. These setting are very important concerning the genetic algorithms that will run.

- The default size of the Population is 50, out of a maximum of 100 and a minimum of 0. The more we increase this number, the bigger the population will become and this will slightly increase the amount of time for the genetic algorithm to complete execution.
- The second parameter is the number of the evolutions. The number of the evolutions determines how many times the Chromosomes will evolve during the execution of the genetic algorithm. The default value is 100 and it is obvious that the more evolutions we chose to have, the more our fittest chromosomes tend to reach the optimal solution, if it exists. On the other hand increasing the number of the evolutions means increasing the time needed for the genetic algorithms to finish execution. Therefore, in programs with high complexity that the algorithms demand a lot of time to run, we could decrease the number of the evolutions, making execution faster, accepting as a trade-off the fact that coverage percentage might be lower.

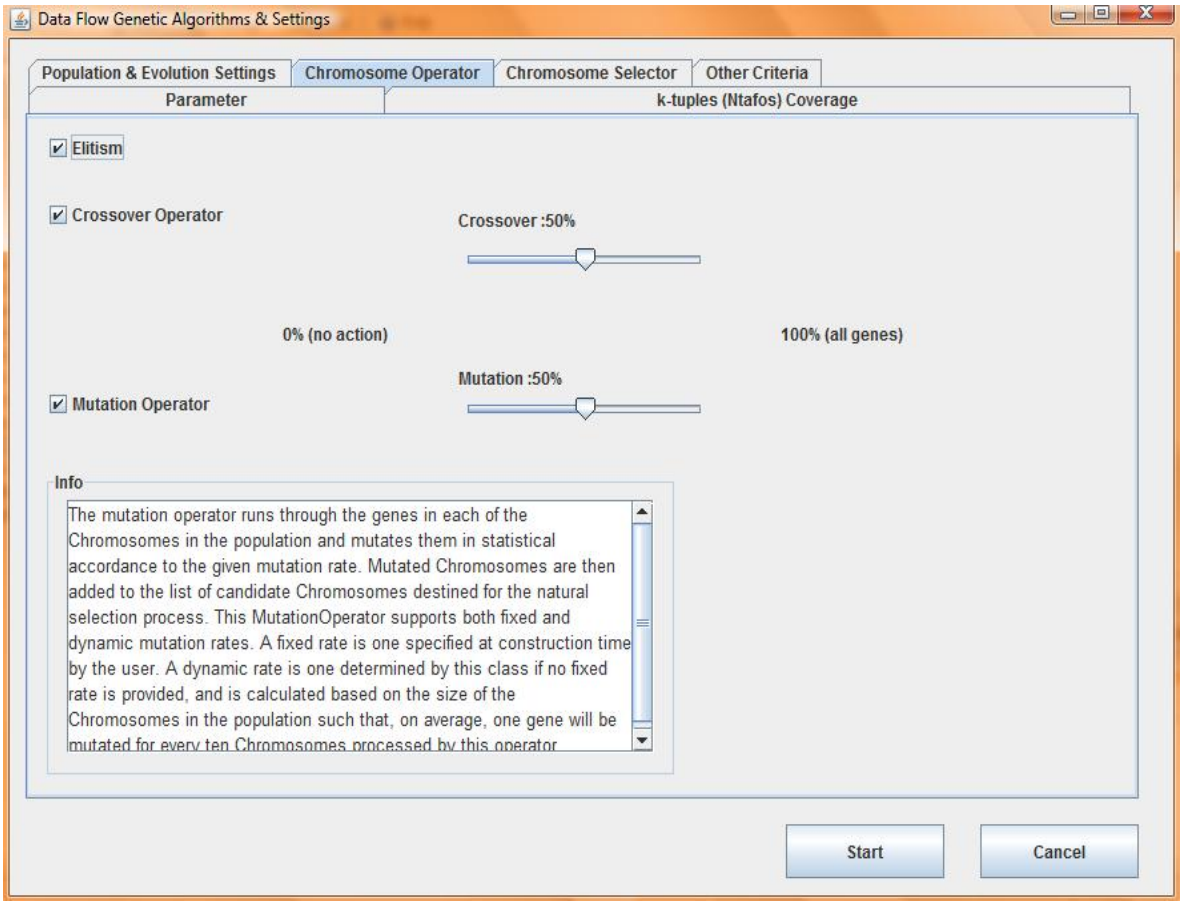
- The third parameter concerns selection. Changing the percentage of selection will affect the percantage of the chromosomes to be naturally selected. The default value is 50%.



The fourth tab contains more settings for the genetic algorithms, including elitism , the crossover operator and the mutation operator.

- **Elitism** means that the best chromosome(s) are copied to the
- population in the next generation. The rest are chosen in classical
- way. Elitism can very rapidly increase performance of GA, because it
- prevents losing the best found solution to date.
- **Crossover Operator** : The crossover operator randomly selects two Chromosomes from the population and "mates" them by randomly picking a gene and then swapping that gene and all subsequent genes between the two Chromosomes. The two modified Chromosomes
  are then added to the list of candidate Chromosomes. This operation is performed half as many times as there are Chromosomes in the population.

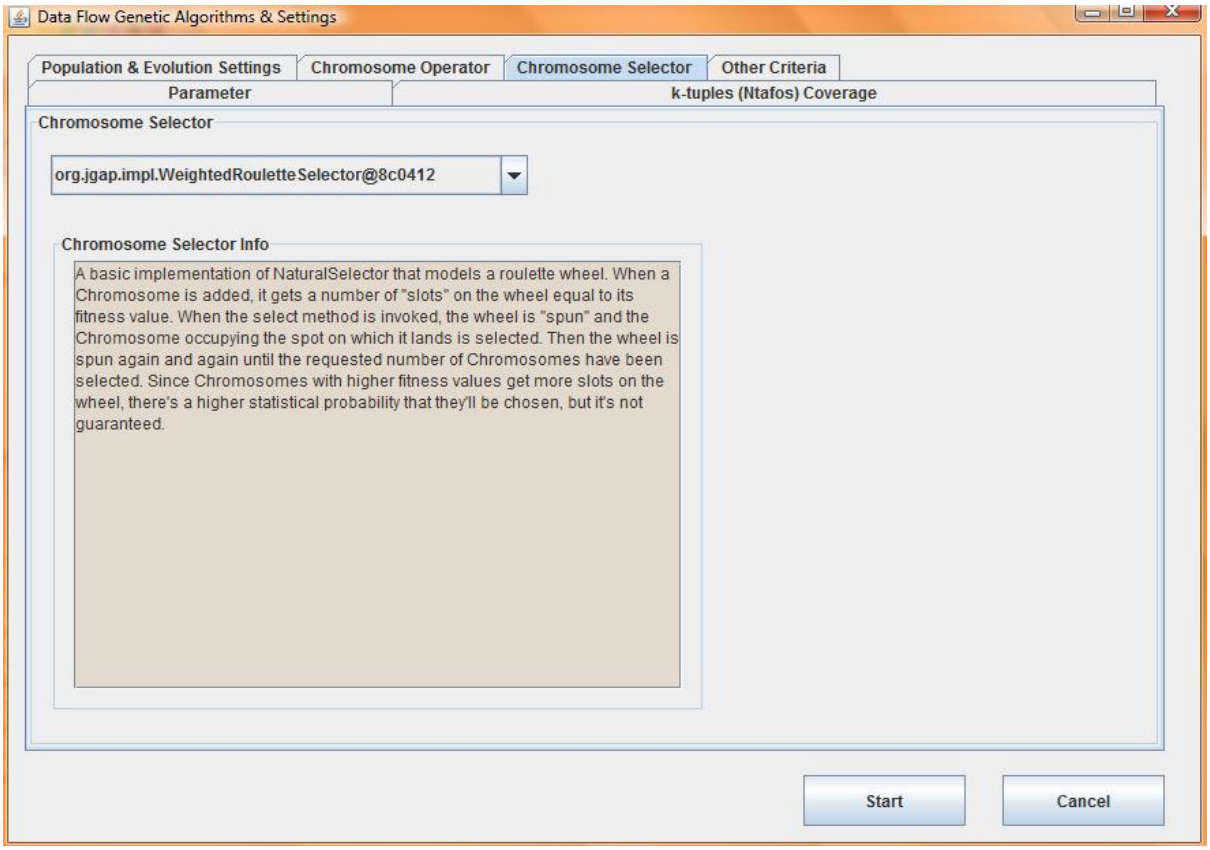
- **Mutation Operator** : The mutation operator runs through the genes in each of the Chromosomes
in the population and mutates them in statistical accordance to the given mutation rate. Mutated Chromosomes are then added to the list of candidate Chromosomes destined for the natural
selection process. This MutationOperator supports both fixed and dynamic mutation rates. A fixed
rate is one specified at construction time by the user. A dynamic rate is one determined by this
class if no fixed rate is provided, and is calculated based on the size of the Chromosomes in the
population such that, on average, one gene will be mutated for every ten Chromosomes processed
by this operator.



The fifth tab contains a combobox with an option of a chromosome selector as we can see in the scheme below.

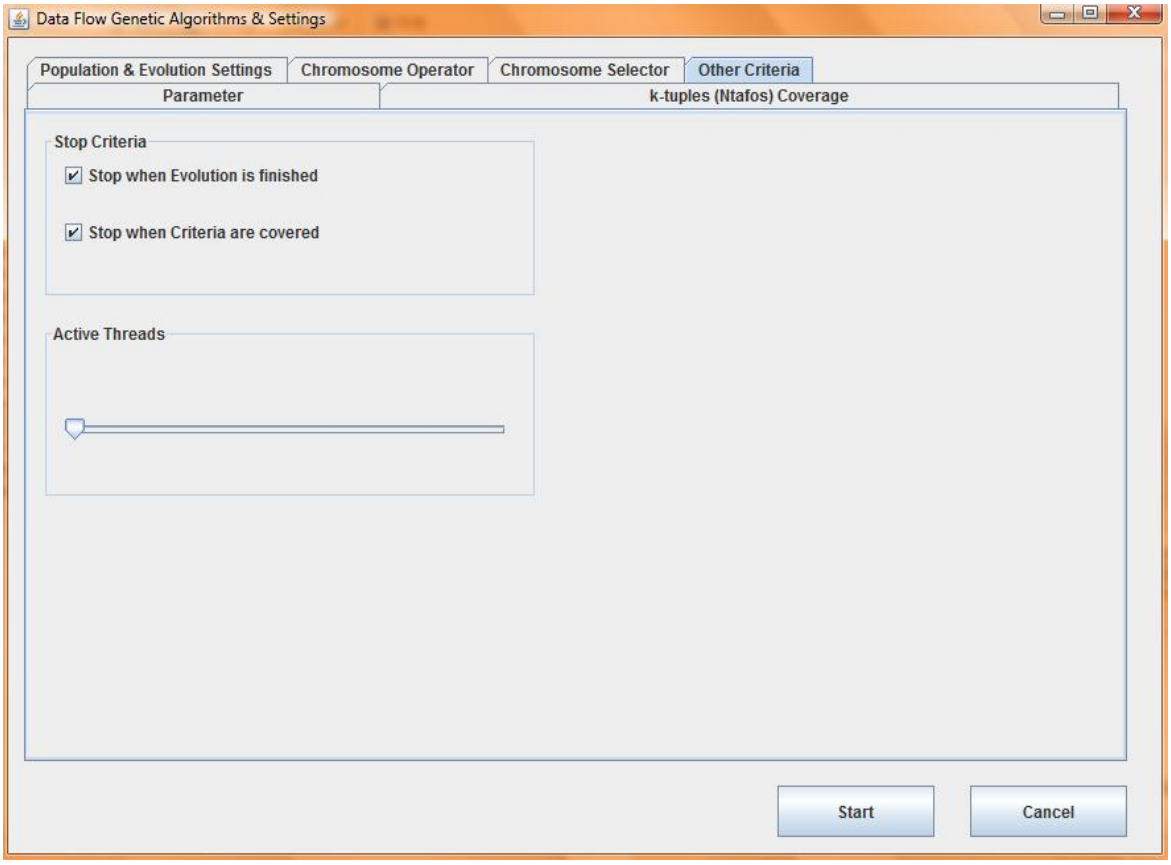
The available selector is a basic implementation of NaturalSelector that models a roulette wheel. When a Chromosome is added, it gets a number of "slots" on the wheel equal to its fitness value. When the select method is invoked, the wheel is

"spun" and the Chromosome occupying the spot on which it lands is selected. Then the wheel is spun again and again until the requested number of Chromosomes have been selected. Since Chromosomes with higher fitness values get more slots on the wheel, there's a higher statistical probability that they'll be chosen, but it's not guaranteed.



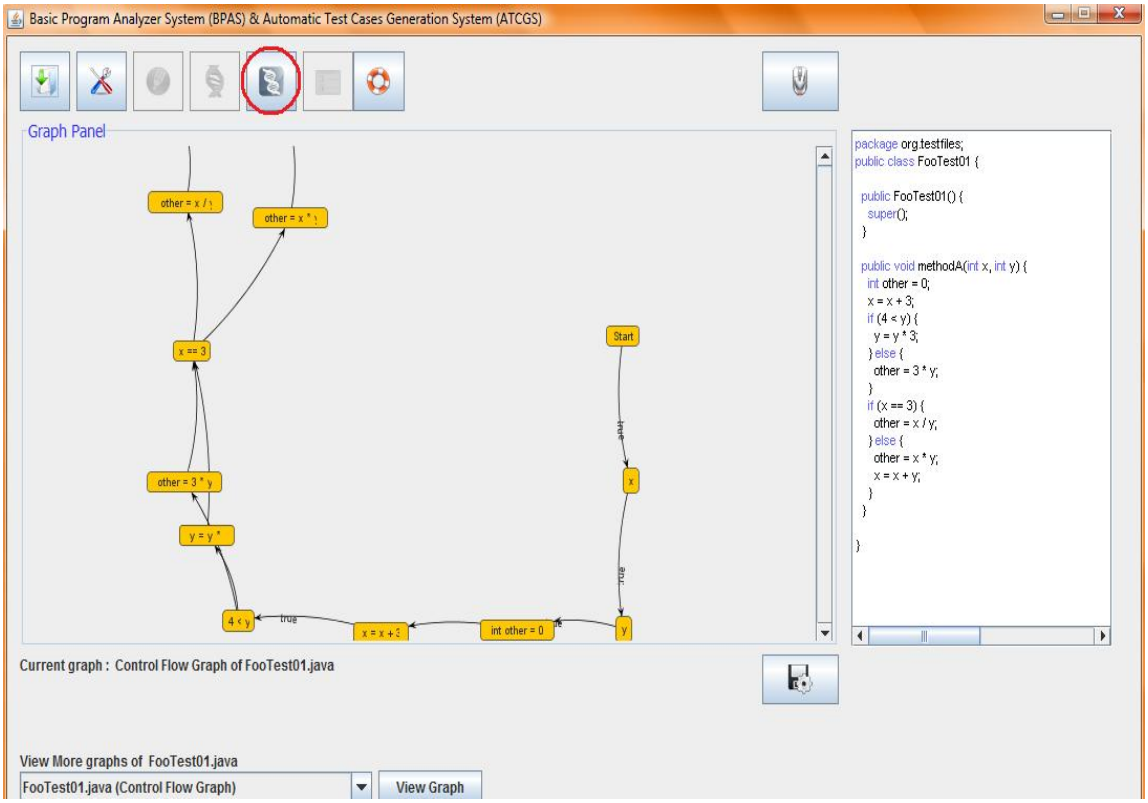
The sixth, last tab contains some other criteria, like the options :

- That the algorithms stop when the evolutions are finished
- That the algorithms stop when the criteria that the user has defined have been covered.
- Defining the number of additional threads, but that wont be necessary since the last modifications, making the GA running in their own separate thread.

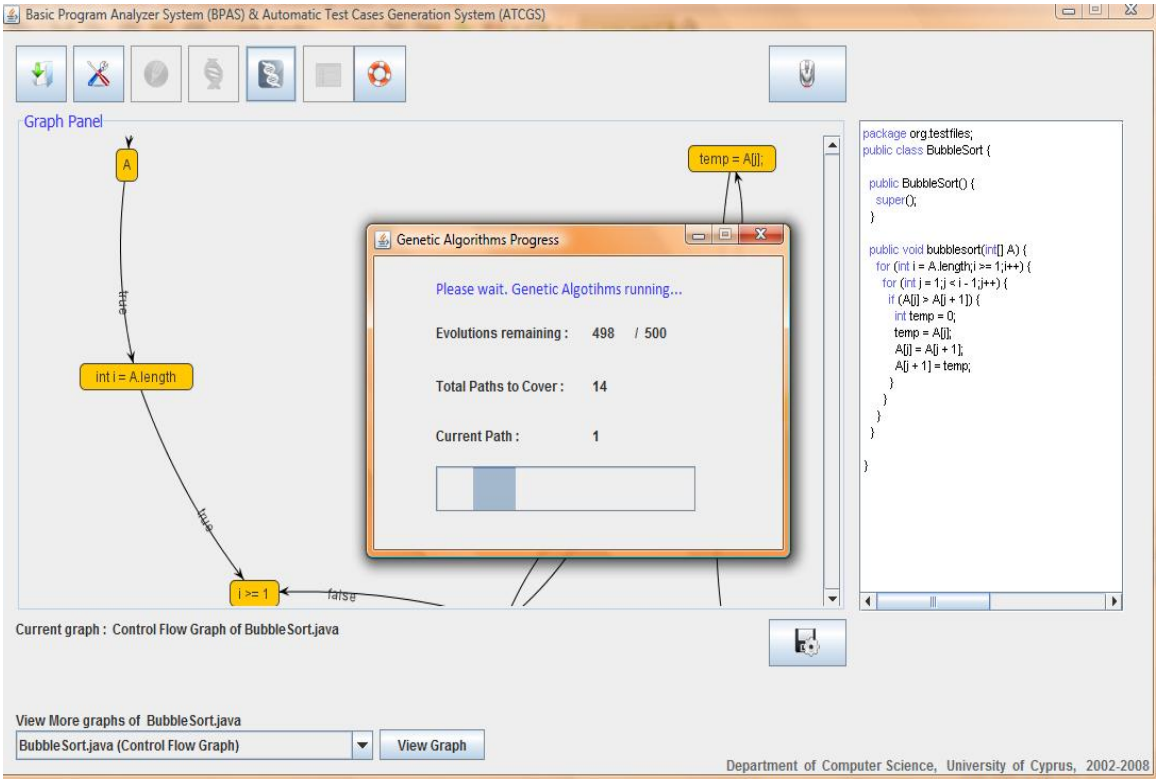


Test Case Generation based on Control Flow Graph

Test case generation based on Control Flow graph aims to satisfy edge/condition coverage. This is a simpler Task for the user to do, because most of the GA's parameters are used as defaults. So all the user needs to do is, having the control flow graph created, click on the button to start with the generation and testing of the test-cases, as shown below :



After that, we will see that the execution of the GA has started and we can monitor the progress from the informational dialog window, shown in the scheme below :



Genetic Algorithms

A **genetic algorithm (GA)** is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an

optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields.

A typical genetic algorithm requirements:

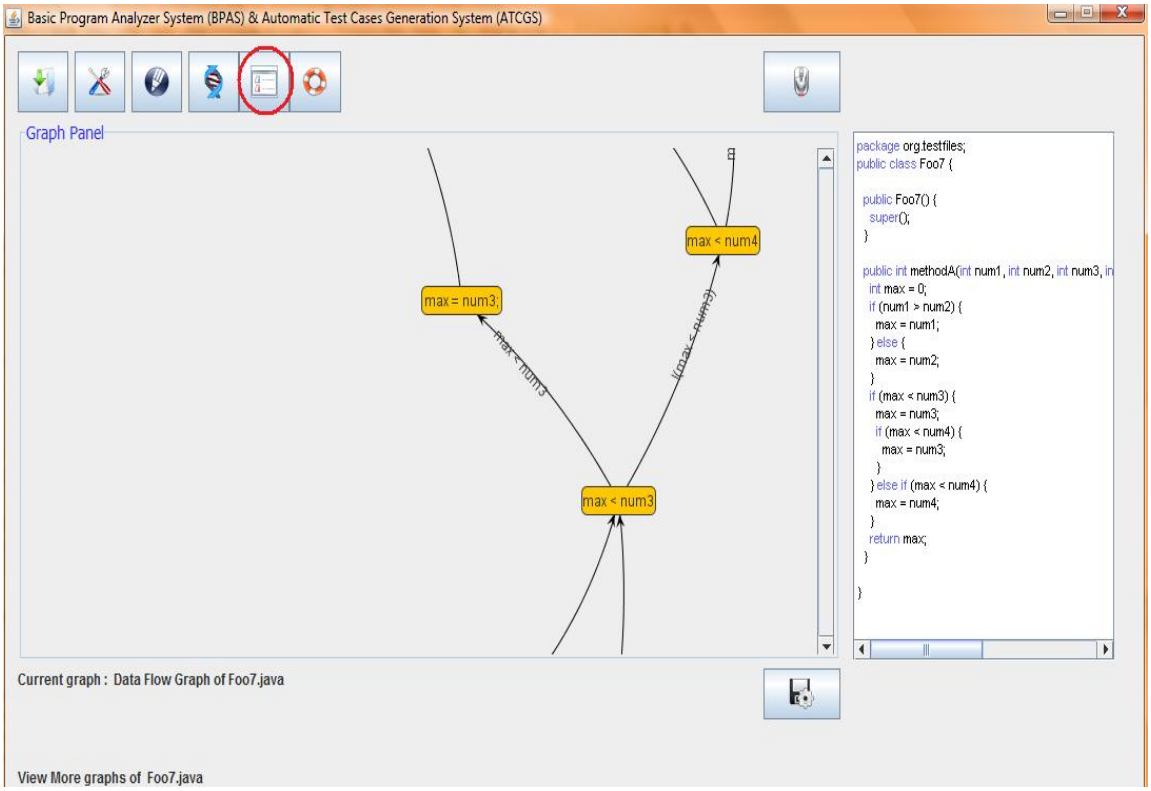
1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

How to view the test-case generation and coverage results ?

When the genetic algorithms complete the execution, the user will be notifies by a small "beep" sound, generated from the progress dialog window that opened in the beginning of the execution. This means that the test-case generation and coverage has been succesfully completed. By succesfully we don't necessarily mean that the coverage percentage is 100%. In order to view the results, depending on the graph based on which the automatic generation started the execution, we have to click on the corresponding button (that has just became enabled) in the interface of the main frame.

Scenario A : The generation was based on the Data Flow graph and has

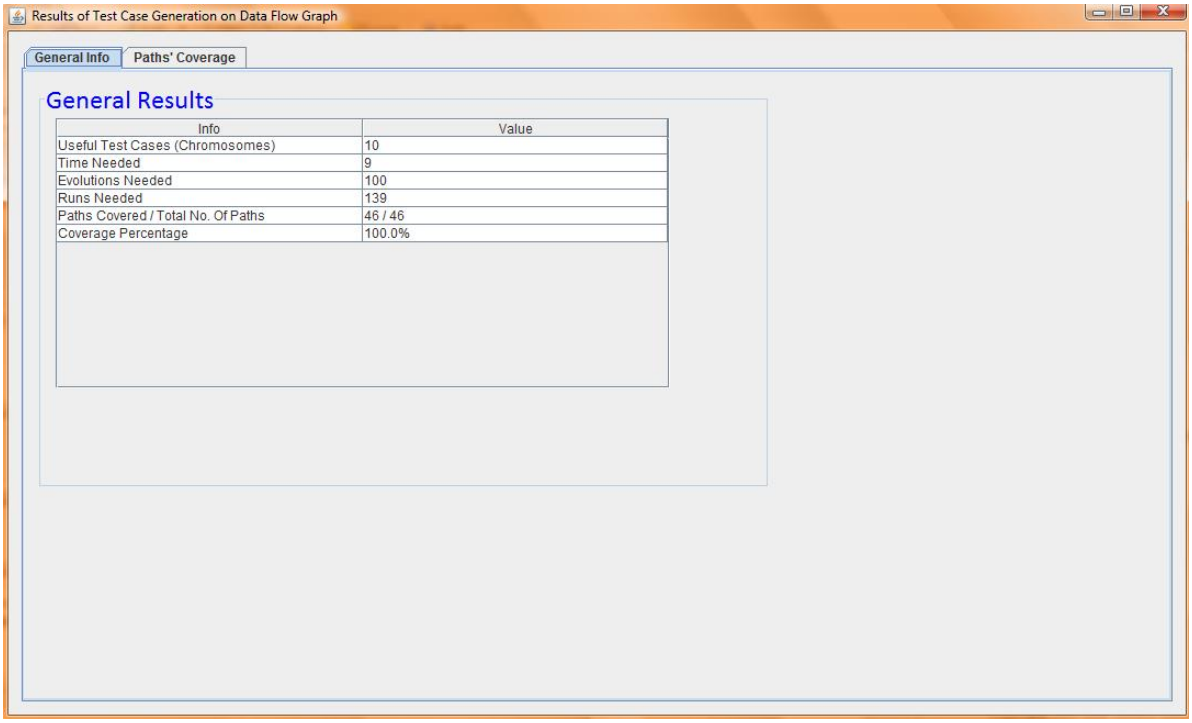
just finished
At this point we can click on the button shown below :



The results window will then open and we can see that it is consisted from 2 tabs. The first tab contains general info about the test-case generation procedure and the genetic algorithms, like

- The time needed until completion of the algorithms
- The number of useful chromosomes generated by the procedure
- The number of the evolutions needed
- The times the execution of the code had to run
- The number of the paths covered out of the total number of the paths that had to be covered
- The percentage representing the previous information

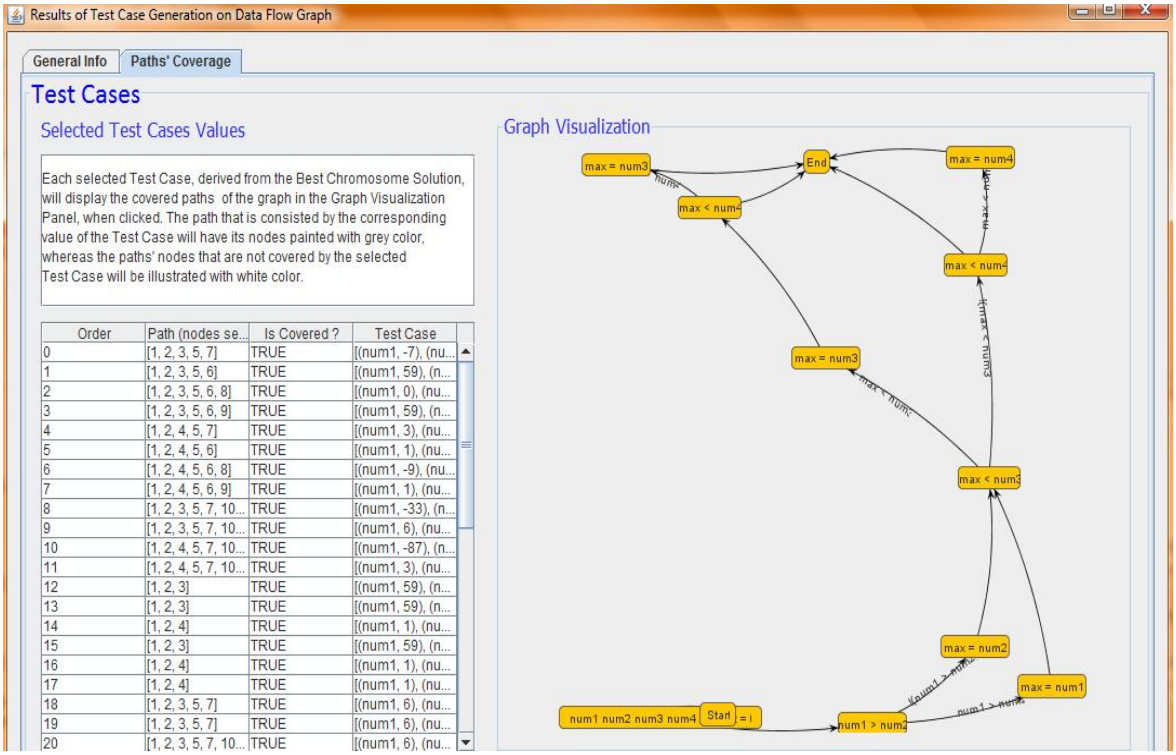
We can see below this first tab :



When we open the second tab we will see that there is more specific and more interactive information about the coverage and the test-cases.
We can see the paths extracted from the k-tuples critertion (as a sequence of numbers representing uniquely the nodes), the boolean value indicating whether or not the specific test-cases has succeeded to cover the path, and the test case used to test this path (the values given to the corresponding parameters).

On the right side of this frame we can see the data flow graph created. The graph is initially painted the same way as in graph panel of the main frame.
We can interact with the test cases by clicking on the ones we would like to examine and visually spot the on the right-hand graph.
Each selected Test Case, derived from the Best Chromosome Solution, will display the covered paths of the graph in the Graph Visualization Panel, when licked. The path that is consisted by the corresponding value of the Test Case will have its nodes painted with grey color, whereas the paths' nodes that are not covered by the selected Test Case will be illustrated with white color.

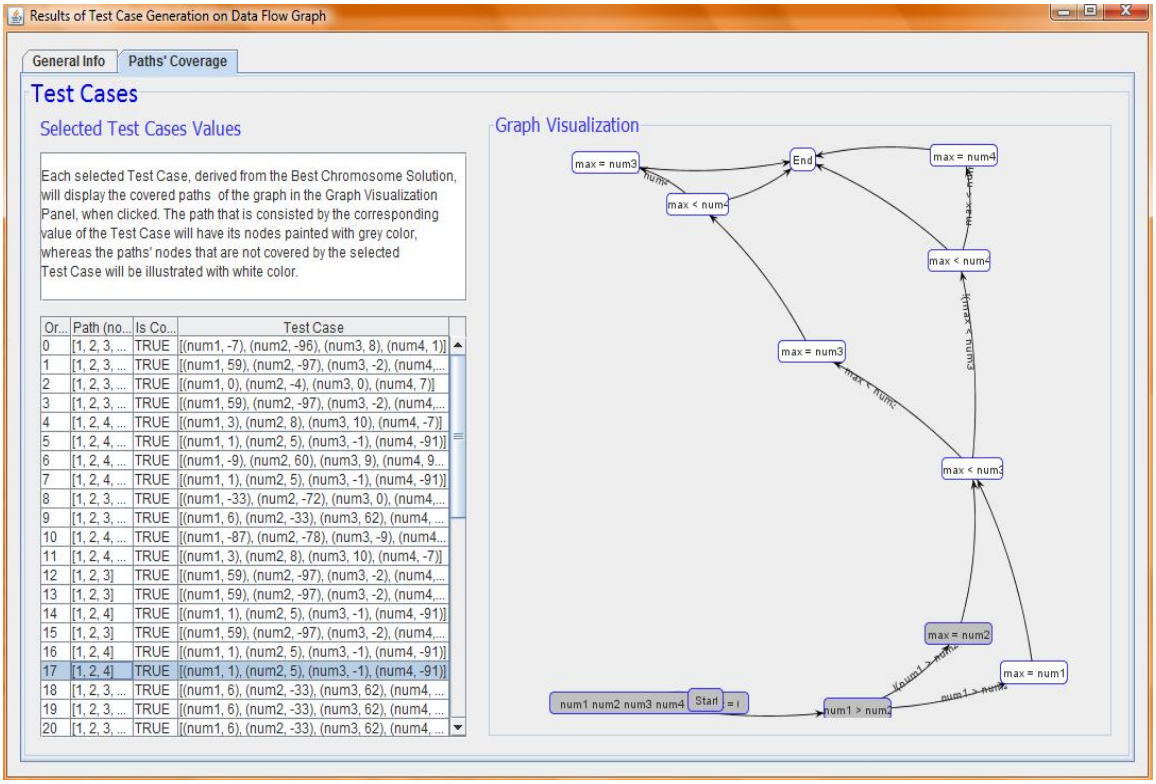
In the next scheme we can see the test-cases and the information mentioned earlier and the graph :



Suppose now that we are interested in examining the test-case no 17, attempting to cover the nodes 1,2,4 which correspond to the nodes (we can see the numbers of the nodes and the corresponding names in the data flow graph info window) :

- num1 num2 num3 num4 int max=0
- num1>num2
- max = num2

By clicking on the line with the 17th test-case, we can see that the graph will indicate the covered path with grey color, as mentioned before, adn the rest of the graph, that is not covered, will be painted white.



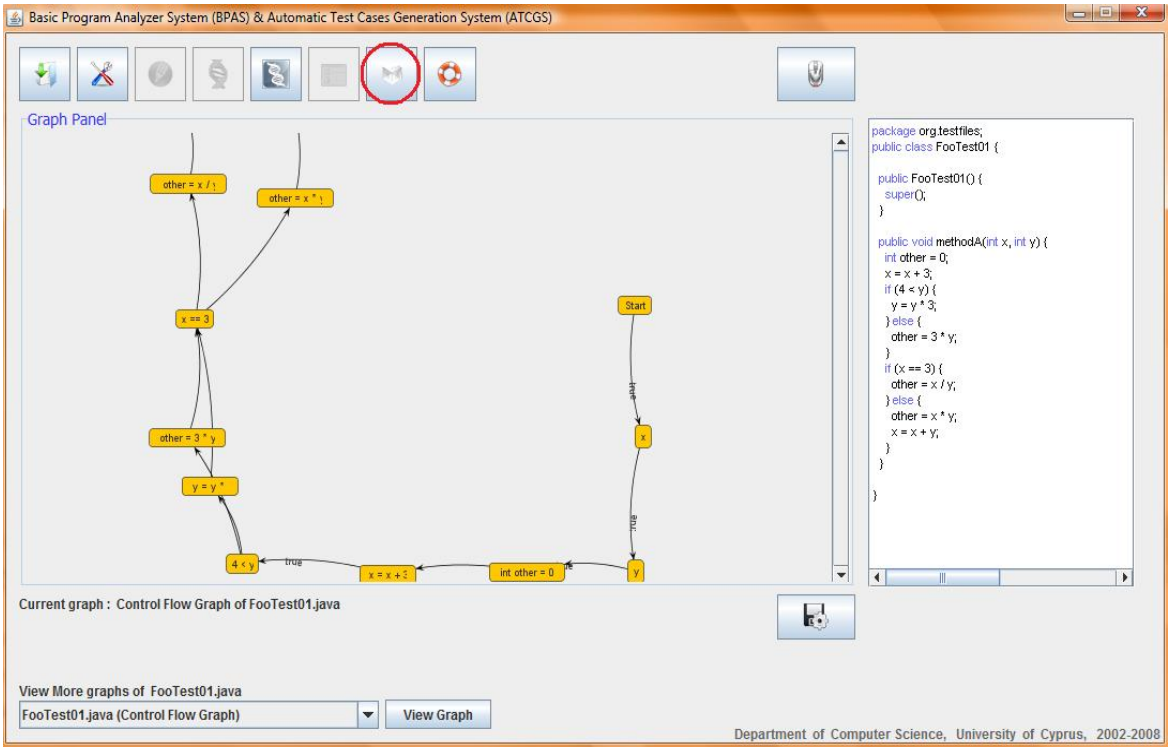
Indeed, if we examine carefully this test case, we will see that the values assigned to num1, num2, num3, num4 are :

- num1 = 1
- num2 = 5
- num3 = -1
- num4 = -91

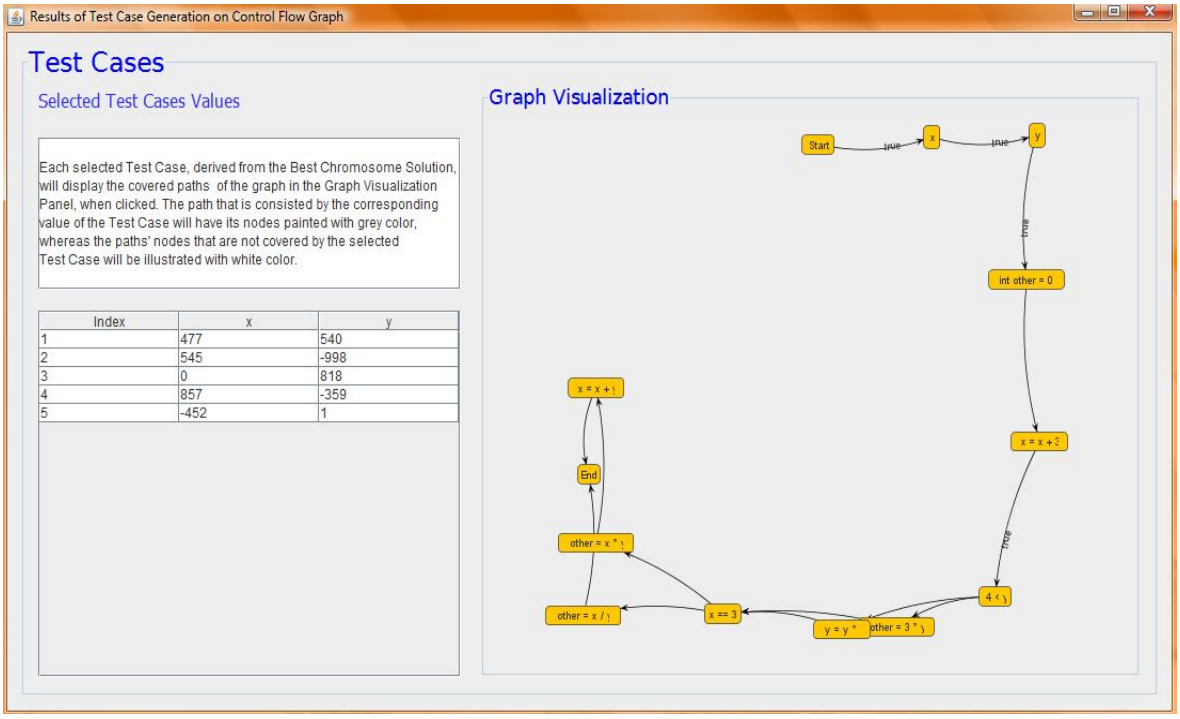
Therefore, we cover the node number 1, with the parameters and assignments, and when executing the if statement : if (num1> num2) , which is the node 2, the result of the comparison will be false, because num1 = 1 < num2 = 5, and the flow of the program will continue to the false branch of the if statement, which is the node number 4 (max = num2).

We should note here that the Start node is always painted gray when there is a node in the set of nodes contained in the path that the algorithm is intending to cover in that test-case that has as a parent the node Start. In a similar fashion, the node End is painted gray (covered) when there is at least one node in the set of nodes of the path that is covered and has the node End as a child.

Scenario B : The generation was based on the Control Flow graph.
Similarly, when the execution is finished, the user should hear a "beep" sound indicating that the process is over. This is a signal for the activation (enabling) of the results button for the GA of the control flow graph.
When ready, the user can open the test-case results window for the test-case generation based on the Control Flow graph, as shown below :



When clicking this button, the results button, designed in a similar way, will open as we can see below :



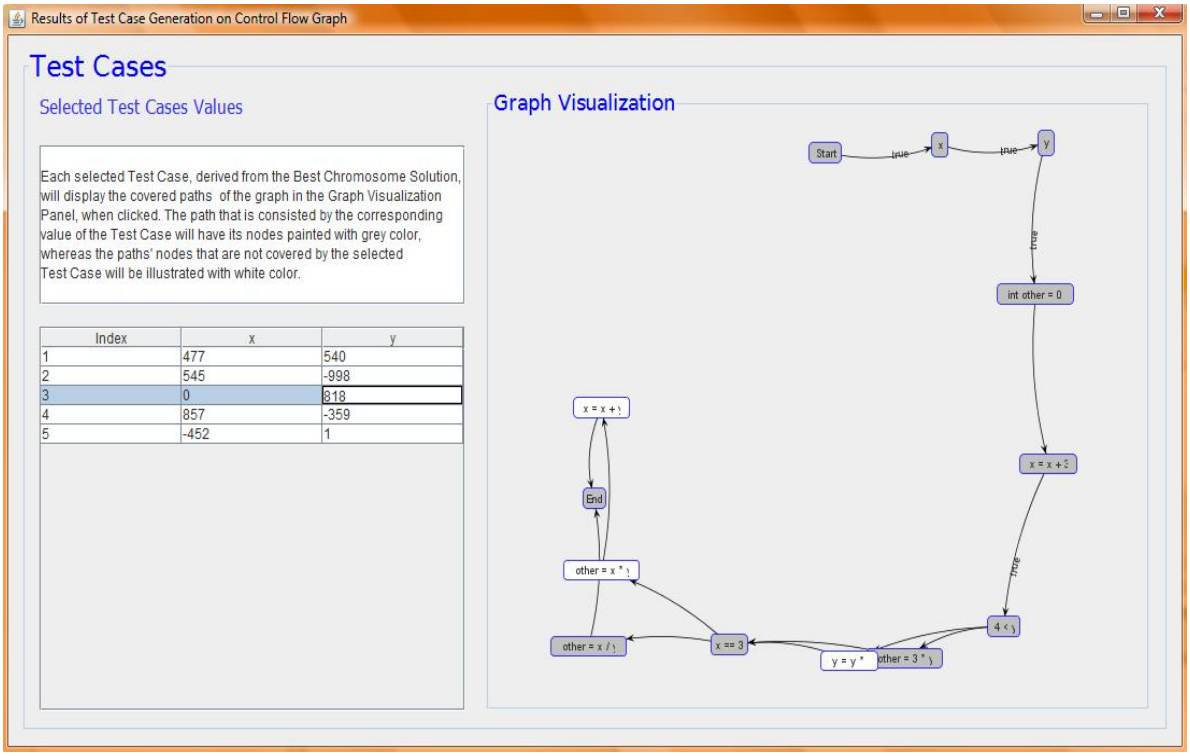
We can see the selected test cases, and the numbers assigned to the parameters. Suppose the user would like to chose the third test case, which assigns the values :

- x= 0
- y= 818

We realize that using this test-case, the flow will arrive from Start to x=x+3, making the value of x=3.
The next node visited is the if statement 4<y. Taking into account that y= 818, the comparison is 4< 818 and therefore the result is TRUE. This means that the flow will continue with the true branch, which is other = 3*y. The next node visited is the if statement x==3, which happens to be TRUE because the new value of x is 3. Therefore, the flow continues with the true branch of the statement, which is other = x/y, and then we reach the node End.

Note that there is the assumption that the node Start is always covered since every execution will start from this node. In the same fashion, the node End is also always covered, and painted grey, since we know that one way or another the flow of the program will reach the End.

In the following scheme we can see the pathe covered in the graph, when we click on the test-case mentioned before :



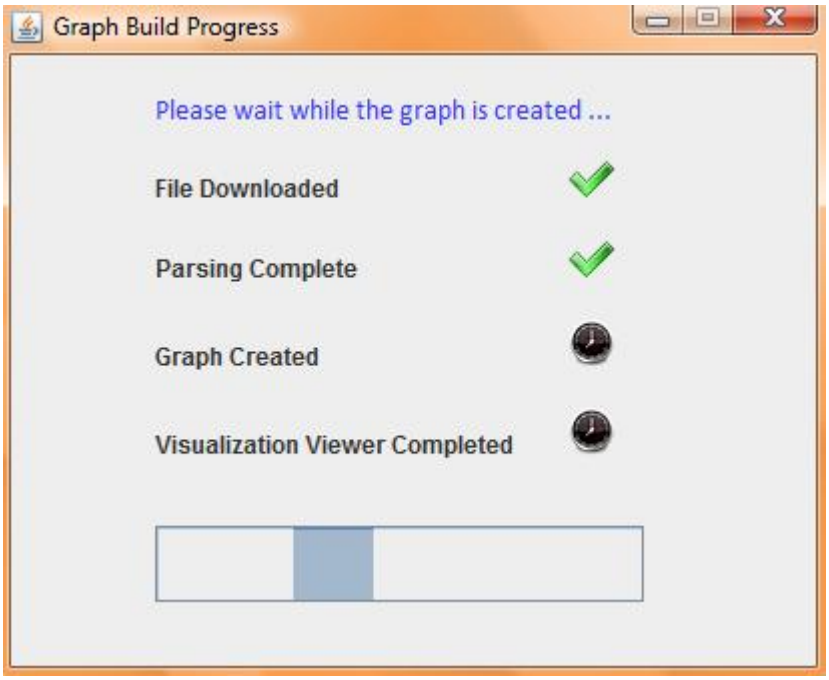
Multithreading in the Tool

Multithreading computers have hardware support to efficiently execute multiple threads. These are distinguished from multiprocessing systems (such as multi-core systems) in that the threads have to share the resources of single core: the computing units, the CPU caches and the translation lookaside buffer (TLB). Where multiprocessing systems include multiple complete processing units, multithreading aims to increase utilization of a single core by leveraging thread-level as well as instruction-level parallelism. As the two techniques are complementary, they are sometimes combined in systems with multiple multithreading CPUs and in CPUs with multiple multithreading cores.

Taking into account the benefits from implementing multithreading, especially on multi-core systems, it has been carefully used in the application, when this seemed to be necessary. By this i mean tasks that consume a big amount of time to complete, tasks that can be executed in a parallel fashion, and tasks that need to offer the user information about their status, and therefore, need to run independently in separate threads.

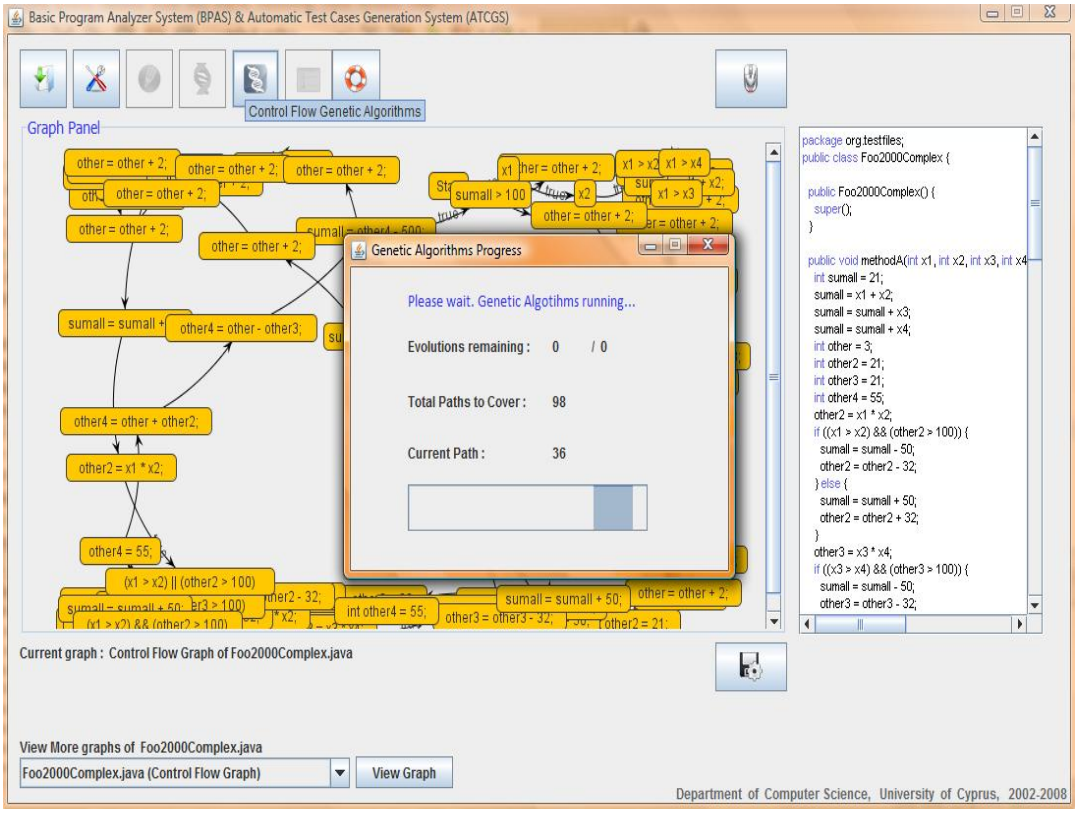
More specifically, these Tasks include :

- The graph creation process, every graph type can be created independently, and since many types of graph are allowed to be created, their progress status must be shown to the user. This design is not only following good Human-Computer Interaction principles, but is also necessary in some cases, like when the source code consists of many lines of code and needs time to complete the creation of the graph (and might give the user the impression that the application is not responding, when it is actually working).

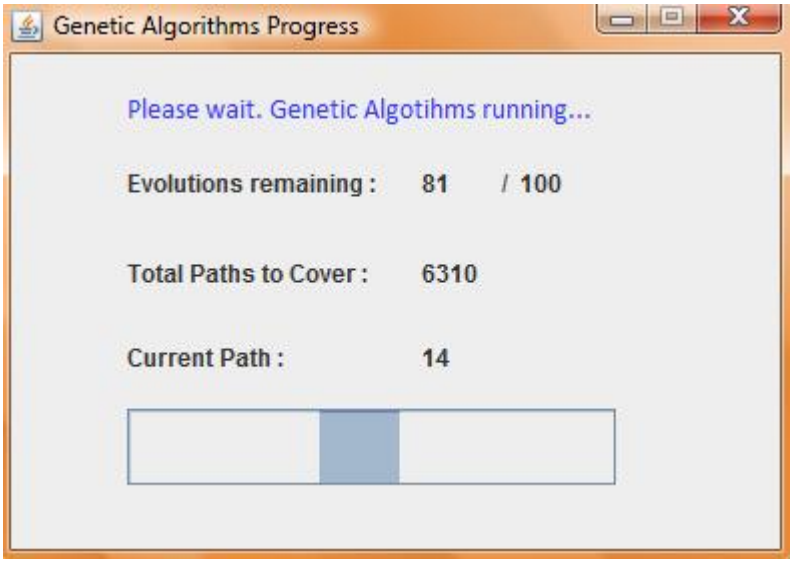


- The test-case generation with GA based on the Control Flow graph. This task, depending on the source code under test, may need time to complete. For this reason, information is available to the user through the progess info window, showing the current path testing, the total number of paths to be covered and the evolutions taking place. Moreover, apart from this useful information running multithreading can offer, using a separate thread for this task no longer makes the application "freeze", allowing the user to continue

his navigation through the tool and work on other tasks.



- The test-case generation with GA based on the Data Flow graph. This task, depending on the source code under test, may need time to complete. For the same reasons, the genetic algorithms are running on an individual thread, in order that the user can see all the information he needs about his task that is running and continue working with the tool, without wasting time, waiting for the completion of a potentially time-consuming task.

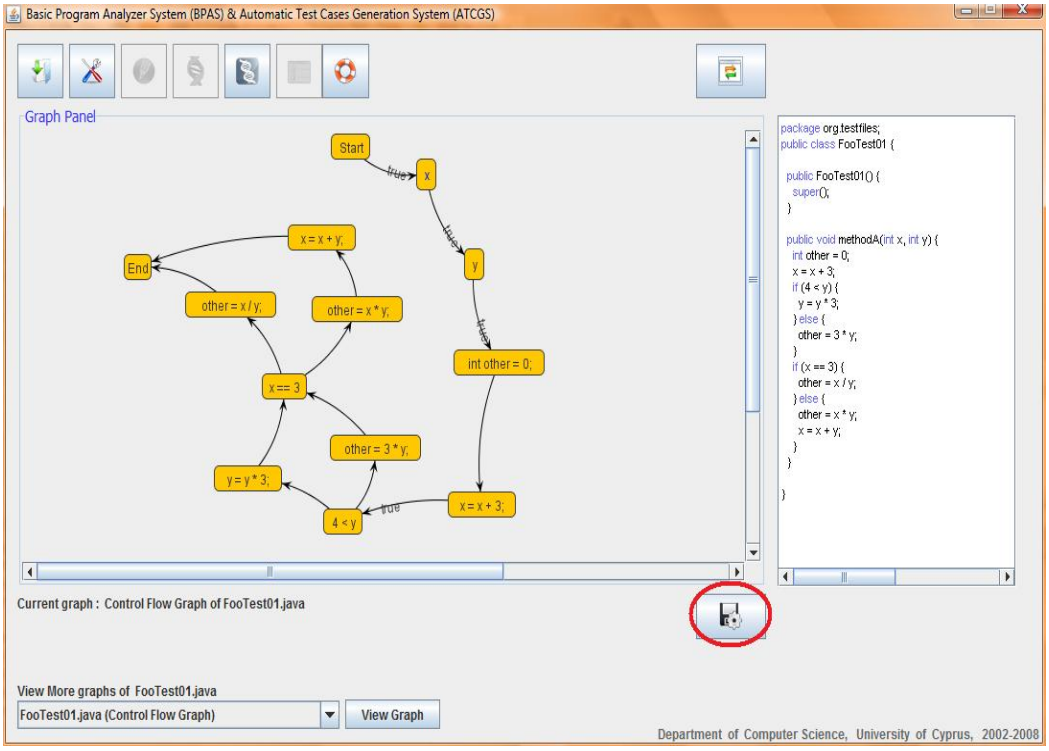


Can i save the graph as an image?

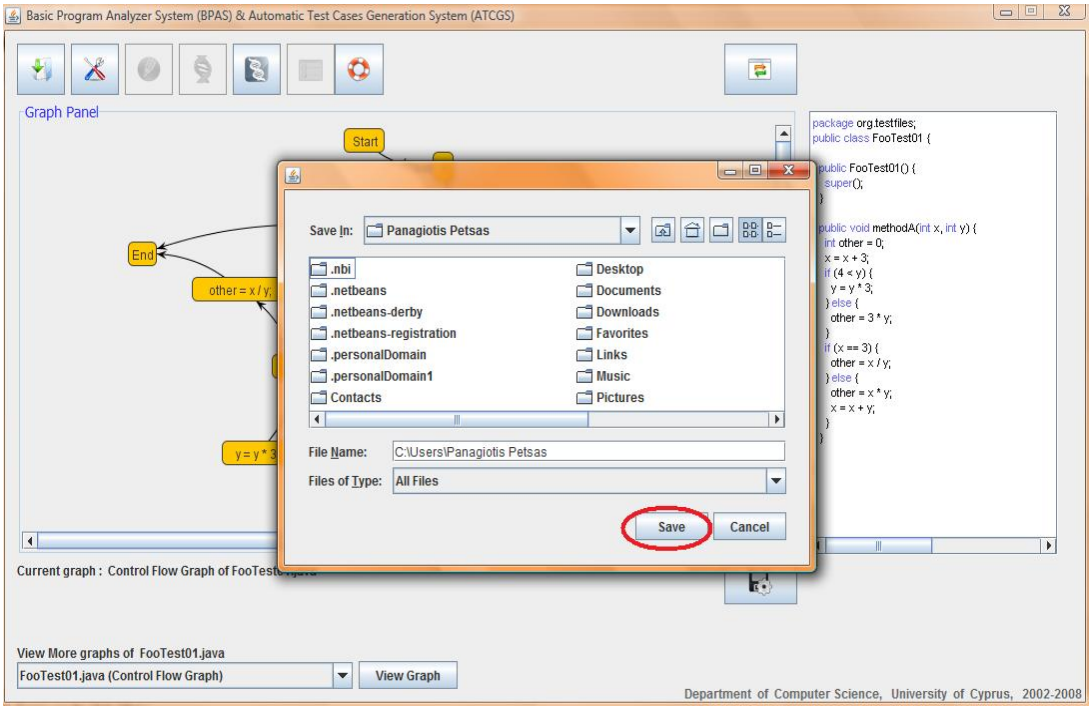
The answer is yes. A new feature, that was added in one of the latest versions of the tool allows the user to save the graph that he has created in an image format in a directory on his hard disk that he can chose.

This can be done by following the simple steps shown below :

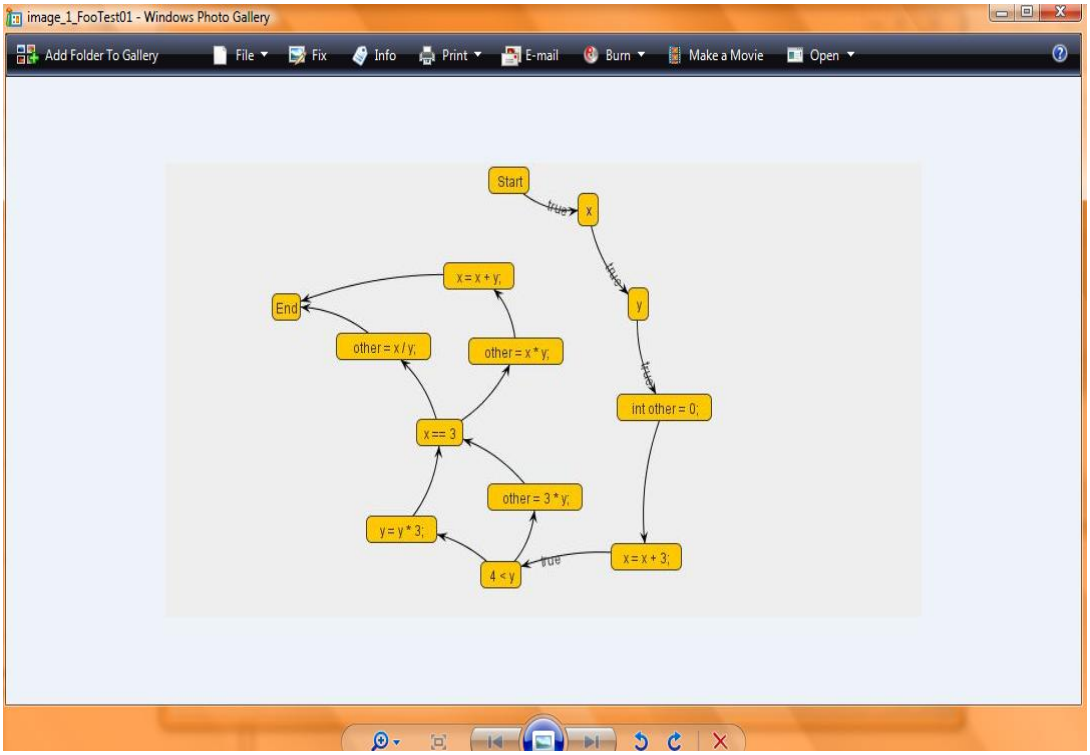
- Once the graph is created and added inside the graph panel, click on the button show below :



- Notice that a browser window will open, asking for the location where the .jpg image will be saved.
- All that needs to be done is to chose the folder that the image will be saved.
- There is not need to specify the name of the .jpg file. This will be done automatically by the tool, giving the name to the image in this format : image<image_number>_<Source Code file name>.JPG
- When the desired folder has been selected , click on the button "Save" shown below :



- And it's done! You can find your graph in the directory selected.



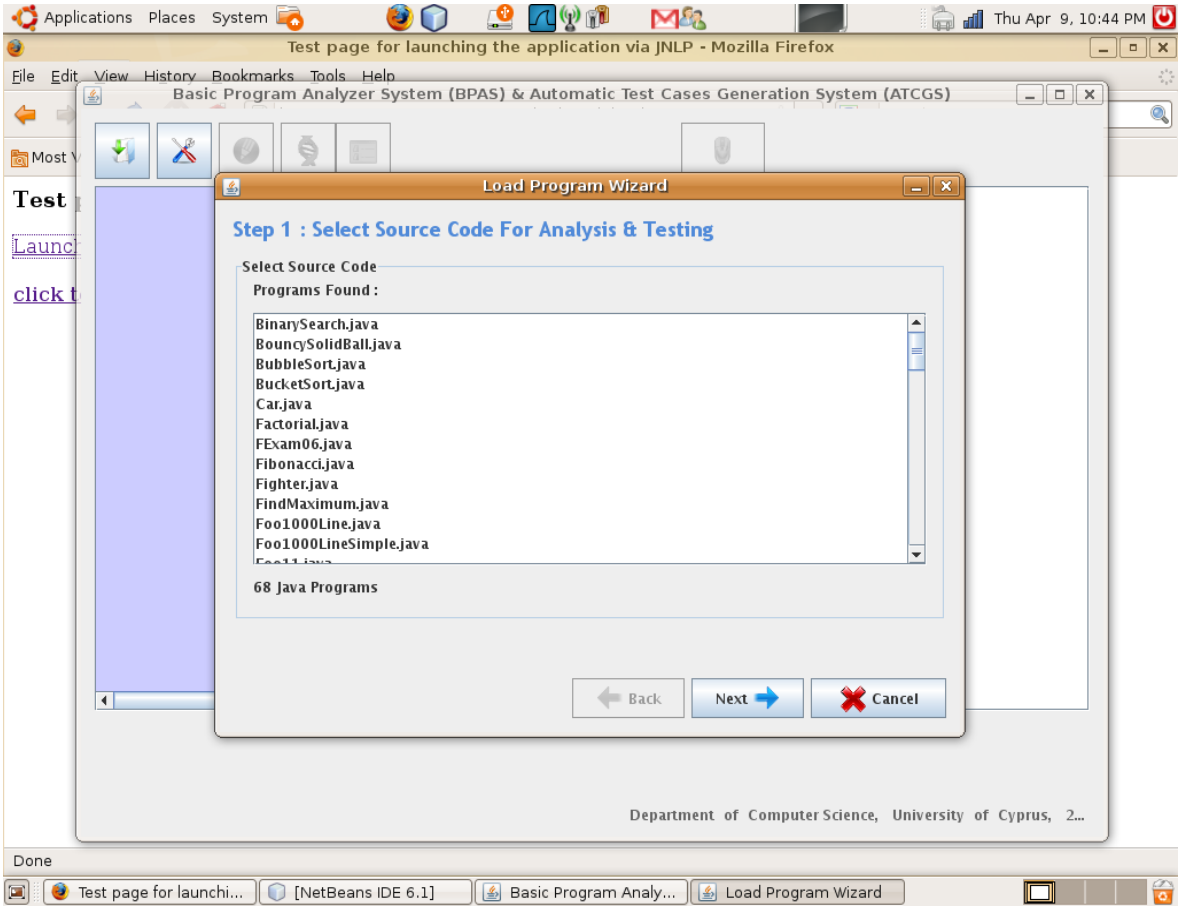
Can i use the tool in other Operating System ?

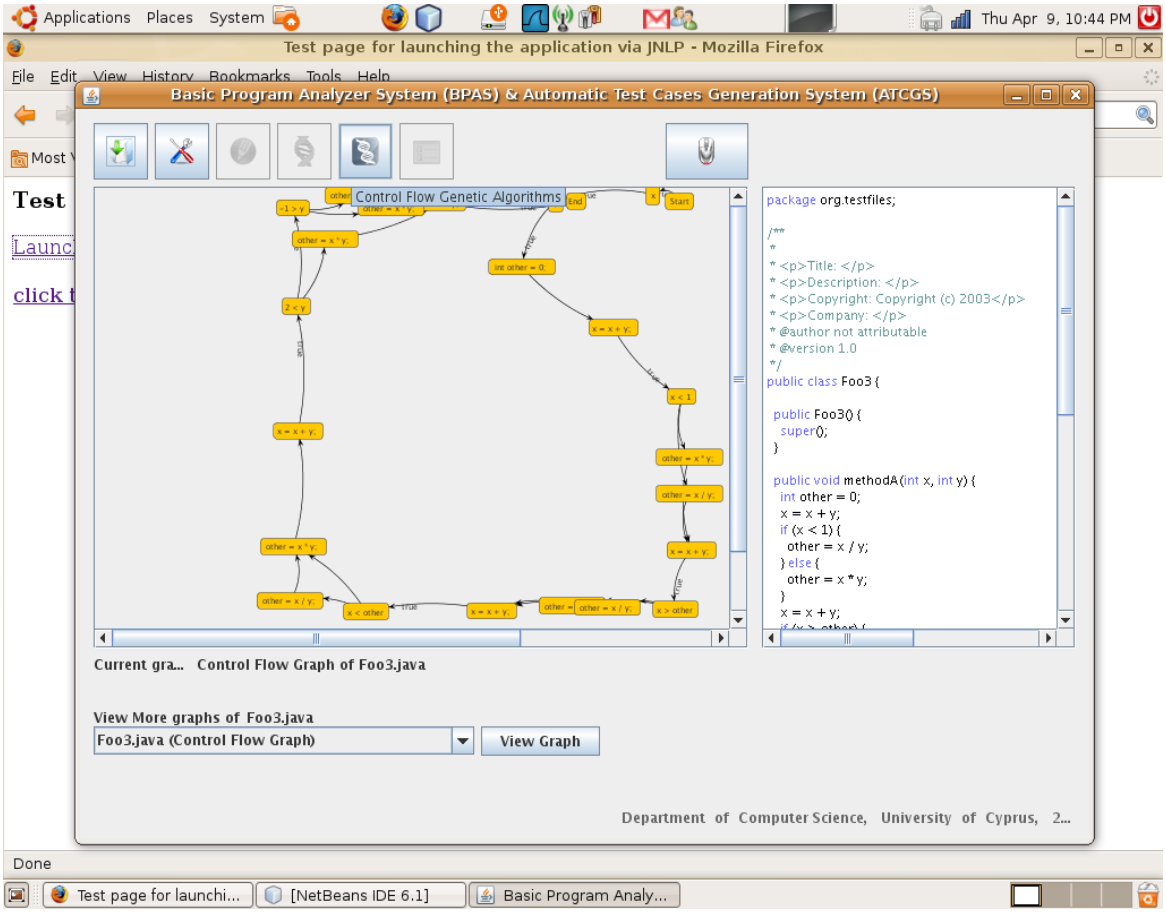
Taking advantage of the Java Platform, and its ability to run on any machine, regardless its operating system, this tool can run on any operating system that has java 5 or better. The developing of the tool took place in various Operating Systems, such as Microsoft Windows XP, Microsoft Windows Vista, Linux, Ubuntu (Feisty Fawn) 7.04, Ubuntu (Intrepid Ibex) 8.10.

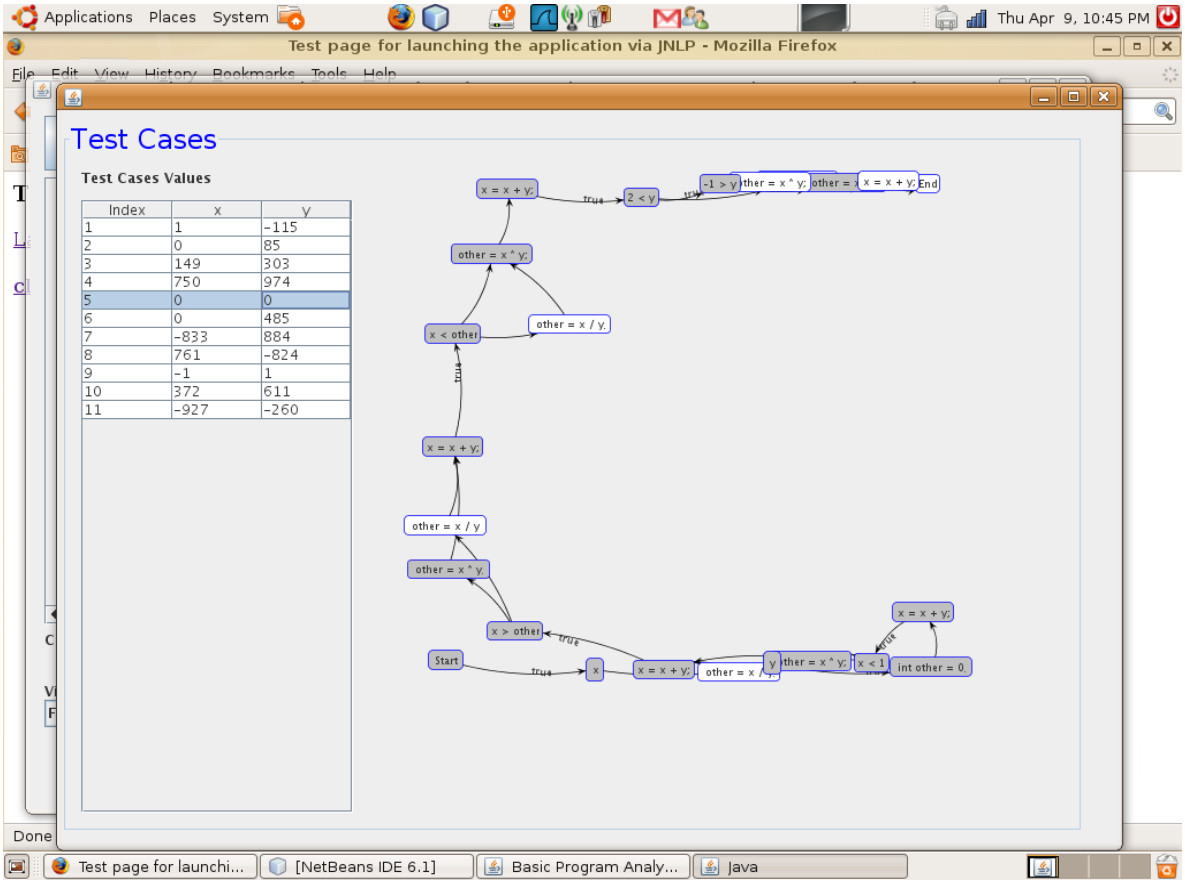
This ensured that the tool runs with the same performance, stability, predictability and user-friendly interface designed in the Operating Systems mentioned above.

Having already seen the tool being executed on Windows Vista, we can see below some examples, as instances of execution of the tool in Ubuntu :


Applications Places System
Thu Apr 9, 10:44 PM
Test page for launching the application via JNLP - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.cs.ucy.ac.cy/~cs04pp2/dist/launch.html
Google
Most Visited
Getting Started
Prolog Tutorial -- 2.7
Latest Headlines
Test page for launching the application via JNLP
Launch the application
click to see API
Java™ starting...
http://www.cs.ucy.ac.cy/~cs04pp2/dist/launch.jnlp
Test page for launchi...
[NetBeans IDE 6.1]


Applications Places System
Thu Apr 9, 10:44 PM
Test page for launching the application via JNLP - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Basic Program Analyzer System (BPAS) & Automatic Test Cases Generation System (ATCGS)
Load Program Wizard
Step 1 : Select Source Code For Analysis & Testing
Select Source Code
Programs Found :
BinarySearch.java
BouncySolidBall.java
BubbleSort.java
BucketSort.java
Car.java
Factorial.java
FExam06.java
Fibonacci.java
Fighter.java
FindMaximum.java
Foo1000Line.java
Foo1000LineSimple.java
68 Java Programs
Back
Next
Cancel
Department of Computer Science, University of Cyprus, 2...
Done
Test page for launchi...
[NetBeans IDE 6.1]
Basic Program Analy...
Load Program Wizard


Applications Places System
Thu Apr 9, 10:44 PM
Test page for launching the application via JNLP - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Basic Program Analyzer System (BPAS) & Automatic Test Cases Generation System (ATCGS)
Control Flow Genetic Algorithms
package org.testfiles;
/**
*
* Title:
* Description:
* Copyright: Copyright (c) 2003
* Company:
* @author not attributable
* @version 1.0
*/
public class Foo3 {
public Foo3() {
super();
}
public void methodA(int x, int y) {
int other = 0;
x = x + y;
if (x < 1) {
other = x / y;
} else {
other = x * y;
}
x = x + y;
Current gra... Control Flow Graph of Foo3.java
View More graphs of Foo3.java
Foo3.java (Control Flow Graph)
View Graph
Department of Computer Science, University of Cyprus, 2...
Done
Test page for launchi...
[NetBeans IDE 6.1]
Basic Program Analy...


Applications
Places
System
Thu Apr 9, 10:45 PM
Test page for launching the application via JNLP - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Test Cases
Test Cases Values
Index x y
1 1 -115
2 0 85
3 149 303
4 750 974
5 0 0
6 0 485
7 -833 884
8 761 -824
9 -1 1
10 372 611
11 -927 -260
Start
x = x + y;
2 < y
-1 > y
other = x ^ y;
other = x / y;
x < other
x > other
other = x / y
other = x ^ y;
x < 1
int other = 0;
End
true
Done
Test page for launchi...
[NetBeans IDE 6.1]
Basic Program Analy...
Java

TroubleShooting

Issue: When opening the application with Web Start, the downloading stalls.

Cause: Due to the size of the application and the necessary libraries, around 21MB, it may be slow, needing around 3-4 minutes. It might stall because of low download speed too.

Solution: You can either wait for it to finish downloading, or press the button "Cancel" and try to donwload it again by clicking on the link "Launch Application". The downloading will continue from the point it was interrupted.

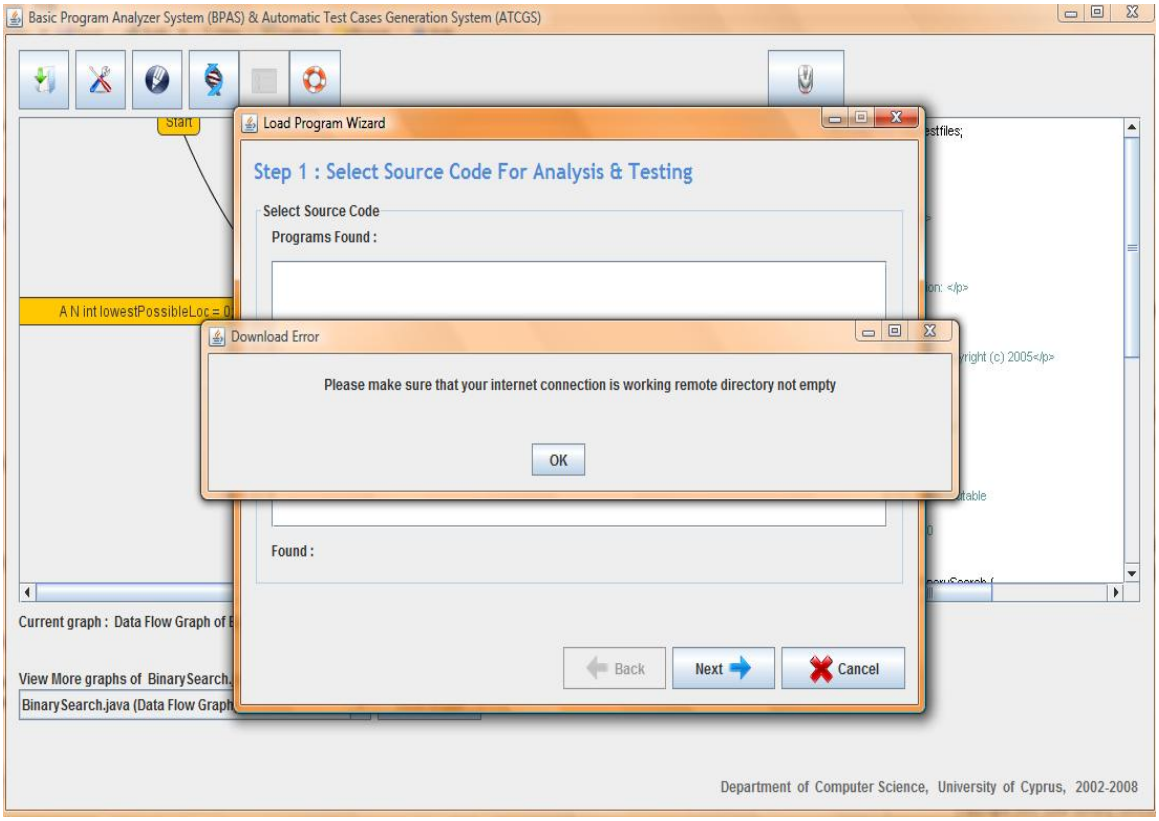
Issue: When creating the graph, the dialog box of the graph's contstruction's progress seems to take a long time at the "Graph Created" Step.

Cause: Either the source code you have selected has high complexity or many lines of code, or the specific type of graph is not supported for this source code due to unsupportedo perations. Another possibble cause is that the user has written his own source code and renamed it after the name of an existing program in the list of testfiles_index. This code might contain syntax errors or unsupported operations.

Solution: Incompatible source codes are usually being removed, taking into account the type of graph selected or the user is being warned about the graphs that can create with a program and the graphs that cannot create. If the case is the user has written his own program and used it as an input, try avoiding this and using instead the sample programs offered

by the tool or make sure it is syntactically correct and free of unsupported operations like Arrays and Recursion.

Issue: When opening the Load wizard to create a new graph the following screen with the error dialog window appears :



Cause: You didn't have internet connection (or problems with the http

protocol) the moment you wanted to open the Load Wizard and the attempt to download the available files for testing and load them inside the program list was not succesful.

Solution: Make sure your http is working fine and that your connection to the internet is not facing any problems. To check this you can simply try to see if the application can find what it needs to start at this step of the wizard, you can try to open the page with the test-files :

http://www.cs.ucy.ac.cy/~cs04pp2/testfiles_index.txt

If this does not work, another solution is to go to the settings menu and select and click on the Offline Mode Allowed checkbox, and then on the button "Apply". An important requirement is that the testfiles_index.txt already exists on your computer and it is not empty. Moreover, the file you want to use for testing must also exist locally, in the directory of the application (this directory that the .jnlp file and CodeTester.jar is downloaded).

Issue: I am sure that the connection is working fine, i tried to open the link mentioned in the previous issue and it opens, loading in the browser a list of .java programs, but the graph is not still created.

Cause: This should probably mean that, even though the location of the index of testfiles exists and is not empty, the folder containing the files has been removed or you do not have access.

Solution: This problem should never occur. But if it does, try to contact me at
cs04pp2@cs.ucy.ac.cy, or

panagiotis.petsas@gmail.com

Issue: I have chosen the Spring Layout from the Algorithms tab in the Settings Menu and applied it on a graph. The graph appears with this Layout corectly inside the graph panel. After a while the graph has disappeared from the graph panel.

Cause: Remember that the Spring Layout algorithm is an animated layout that forces the graph to move continuously. It is possible that after some time it moves outside the boundaries of the visible graph panel.

Solution: Zoom out, as much as you need to see the graph, and then zoom in the area that the graph appears.

Issue: After creating a data flow graph i tried to start the test-case generation. When the GA start running, i can see that it takes a long time to change the current path and makes the whole procedure need a lot of time.

Cause: The source code you have selected probably is very complex and/or consists of many lines of code (LOC), therefore it is necessary to take some time to cover the required paths and result in a high percentage of coverage.

Solution: In the Genetic Settings Frame you can select in the "Population & Evolutions" tab to decrease the number of evolutions. In this way, for every path focused the number maximum number of evolutions will be smaller. Therefore, with this action, you will save some time but this might probably result in a lower coverage percentage.

Issue: When trying to save the graph in a .jpeg format using the graph-shot tool, some parts of the graph do not appear in the image

Cause: This operation is designed to capture the part of the graph that is visible inside the graph panel.

Solution: Try to move the graph so that it is totally visibly inside the graph panel and repeat the operation