ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

Τμήμα Ψηφιακών Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

με τίτλο:

ΣΥΣΤΗΜΑ ΒΕΛΤΙΣΤΗΣ ΑΝΑΠΤΥΞΗΣ ΥΠΗΡΕΣΙΩΝ (SERVICE DEPLOYMENT) ΣΕ ΠΟΛΥΠΛΟΚΑ ΔΥΝΑΜΙΚΑ ΣΥΣΤΗΜΑΤΑ

Κυριακόπουλος Χρήστος

Αριθμός Μητρώου: ΜΕ07058

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Ευχαριστίες

Κατ' αρχάς θα ήθελα να ευχαριστήσω την επιβλέπουσά μου επίκουρη καθηγήτρια Βέρα Σταυρουλάκη που μου εμπιστεύτηκε το θέμα της παρούσας διπλωματικής εργασίας.

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου για την στήριξη που μου παρείχαν καθ' όλη την διάρκεια των ακαδημαϊκών μου σπουδών καθώς και της εκπόνησης της διπλωματικής μου εργασίας.

Τέλος θα ήθελα να ευχαριστήσω τους φίλους μου Χρήστο Χριστοδουλόπουλο, Κάτια Σαρσεμπάεβα και Χρυσοβαλάντη Κεφαλίδη για την ηθική συμπαράσταση και εμψύχωση που μου παρείχαν κατά την διαδικασία εκπόνησης της διπλωματικής μου εργασίας.

Κεφάλαιο 1: Εισαγωγή

1.1 Αντικείμενο και στόχος της διπλωματικής εργασίας

Τα σύγχρονα κατανεμημένα συστήματα έχουν αναδειχτεί σε μία δυνατή πλατφόρμα που παρέχει την υψηλή υπολογιστική ισχύ που χρειάζεται ένα ευρύ φάσμα εφαρμογών. Σε αυτά μπορεί να αναπτυχθεί ένα μεγάλο πλήθος υπηρεσιών, οι οποίες να συνθέτουν την εκάστοτε εφαρμογή που εκτελείται κατανεμημένα στο δίκτυο.

Η απόδοση μιας εφαρμογής που εκτελείται παράλληλα σε διάφορους σταθμούς εργασίας του κατανεμημένου συστήματος εξαρτάται σε μεγάλο βαθμό τόσο από τα χαρακτηριστικά της ίδιας (κόστος εκτέλεσης, κόστος επικοινωνίας των υπηρεσιών της κτλ), όσο και από χαρακτηριστικά της πλατφόρμας (υπολογιστική ισχύς των επεξεργαστών, εύρος ζώνης των καναλιών επικοινωνίας, μέγεθος μνήμης κτλ). Επίσης το κατανεμημένο δίκτυο μπορεί να αποτελείται από ετερογενείς συσκευές (hardware) ή να παρέχει ένα δυναμικό περιβάλλον για τις υπηρεσίες που εκτελούνται σε αυτό, δηλαδή να αλλάζει συνεχώς η δομή του με απρόβλεπτο τρόπο, έχοντας κόμβους που συνδέονται ή αποσυνδέονται από αυτό ευκαιριακά (ad-hoc network).

Μία σημαντική πρόκληση για την αποτελεσματική αξιοποίηση των κατανεμημένων συστημάτων είναι η ανάθεση των υπηρεσιών της εκάστοτε εφαρμογής στους επεξεργαστές του συστήματος έτσι ώστε να επιτυγχάνεται κάποιος στόχος, όπως εξισορρόπηση του φόρτου εργασίας, ελαχιστοποίηση της επικοινωνίας μεταξύ των επεξεργαστών, είτε κάποιος συνδυασμός των δύο για πιο πολύπλοκες εφαρμογές[1]. Επίσης σε συγκεκριμένες εφαρμογές μπορεί να υπερτερούν και άλλα κριτήρια όπως η ποιότητα της παρεχόμενης υπηρεσίας (Quality of Service), καθώς η κατανομή των υπηρεσιών σε συσχετισμό με πιθανή μετακίνηση των επεξεργαστών μέσα στο δίκτυο ή ακόμα και ή αποχώρηση τους από αυτό καλεί για εφαρμογή αναδιαρθρώσιμων τεχνικών στο σύστημα (reconfigurable network) που θα λαμβάνουν υπ’ όψιν τους την δυνατότητα

αναδιάρθρωσης του κατανεμημένου συστήματος ώστε να εξυπηρετούνται οι απαιτήσεις ποιότητας της παρεχόμενης υπηρεσίας [2].

Στόχος της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η ανάπτυξη ενός συστήματος ανάπτυξης υπηρεσιών μίας εφαρμογής που εκτελείται σε κατανεμημένο υπολογιστικό σύστημα, με στόχο την μείωση του συνολικού χρόνου επεξεργασίας της εφαρμογής με παράλληλη ελαχιστοποίηση του κόστους επικοινωνίας μεταξύ των επεξεργαστών, έτσι ώστε να αυξηθεί η αποδοτικότητα του συστήματος.

Παρόλα αυτά υπάρχει σύγκρουση ανάμεσα στα δύο αυτά κριτήρια, καθώς η μείωση του συνολικού χρόνου επεξεργασίας επιτυγχάνεται με την μέγιστη δυνατή κατανομή των υπηρεσιών σε διαφορετικούς επεξεργαστές, ενώ η ελαχιστοποίηση του κόστους επικοινωνίας μεταξύ των επεξεργαστών επιτυγχάνεται με την ελάχιστη δυνατή κατανομή των υπηρεσιών σε επεξεργαστές, δηλαδή με την ανάθεση όλων των υπηρεσιών στον ίδιο επεξεργαστή. Για το λόγο αυτό το σύστημα που θα αναπτυχθεί θα πρέπει χρήση κάποιου αλγόριθμου βελτιστοποίησης, ο οποίος με χρήση μιας κατάλληλης συνάρτησης κόστους θα εξισορροπεί τα δύο αντίθετα αυτά κριτήρια προτείνοντας μια βέλτιστη συμβιβαστική λύση.

1.2 Πεδίο εφαρμογής της διπλωματικής εργασίας

Τα κατανεμημένα συστήματα είναι διαδεδομένα από την δεκαετία του ’80 καθώς παρέχουν ευκολία στην πρόσβαση απομακρυσμένων πόρων, είναι ευέλικτα, αποδοτικά, αξιόπιστα αρθρωτά και (σχετικά) οικονομικά [3]. Τις τελευταίες δεκαετίες έχουν γνωρίσει ραγδαία εξέλιξη και πλέον τα σύγχρονα κατανεμημένα συστήματα δεν είναι μόνο στατικά, αλλά μπορούν να αποτελούνται από κινητές συσκευές (κινητά τηλέφωνα, φορητοί υπολογιστές, υπολογιστές παλάμης κτλ) που συνδέονται ευκαιριακά (ad-hoc) για την εκτέλεση κάποιας εργασίας. Για το λόγο αυτό τα σύγχρονα κατανεμημένα συστήματα έχουν την ανάγκη να είναι αναδιαρθρώσιμα και προσαρμοστικά. Αυτό απαιτεί την αποσύνθεση του συστήματος σε συστατικά με κατάλληλους μηχανισμούς αυτόνομης

σύνθεσης και δυναμικής προσαρμογής που λαμβάνουν υπ' όψιν την εξέλιξη της επεξεργασίας των διαφόρων εργασιών που εκτελούνται, καθώς και τις διακυμάνσεις του περιβάλλοντος [4].

Η διαδικασία αναδιάρθρωσης του δικτύου μπορεί να εφαρμοστεί σε διάφορα συστατικά του, όπως τις υπηρεσίες και τις εφαρμογές που εκτελούνται, τις αρχιτεκτονικές τους συστήματος, τις υποδομές καθώς και τους μηχανισμούς διαχείρισης του συστήματος. Ένα σημαντικό πρόβλημα που αφορά την αναδιάρθρωση του συστήματος είναι η επίτευξη καθολικής συνοχής της νέας διαμόρφωσης που πραγματοποιήθηκε. Ένα άλλο πρόβλημα είναι η εγγύηση ότι οι τρέχουσες δραστηριότητες του συστήματος θα συνεχίσουν να εκτελούνται σωστά κατά την διάρκεια της διαδικασίας αναδιάρθρωσης [4].

Η δυναμική προσαρμογή του συστήματος είναι ιδιαίτερα κατάλληλη όταν χρειάζονται γρήγορες και συχνές αντιδράσεις στις αλλαγές που συντελούνται στο σύστημα. Οι προσαρμογές αυτές θα πρέπει να πραγματοποιούνται χωρίς μα υποβαθμίζεται η διαθεσιμότητα των υπηρεσιών. Για να παρέχεται τέτοια "ελαφριά" προσαρμοστικότητα, το σύστημα θα πρέπει εκ των πρότερων να καθορίζει πως θα διατηρεί την συνοχή του και πώς να πραγματοποιεί την σύνδεση των συστατικών του και των διαύλων επικοινωνίας [4].

Μέσα στα πλαίσια των αναδιαρθρώσιμων (reconfigurable) και δυναμικών προσαρμοζόμενων (dynamic, adaptive) κατανεμημένων συστημάτων θα πρέπει να λαμβάνονται υπ' όψιν από το σύστημα που θα αναπτυχθεί όλες οι αλλαγές που συντελούνται στο υπόβαθρο του συστήματος. Για τους σκοπούς της παρούσας εργασίας, το σύστημα που θα αναπτυχθεί θα δέχεται ως είσοδο τις όποιες αλλαγές επιτελούνται στο κατανεμημένο σύστημα (θεωρητικά παρεχόμενες ως πληροφορίες από το ίδιο το σύστημα) και θα εκτελεί κάθε φορά που αυτές συντελούνται τον αλγόριθμο βελτιστοποίησης που θα καθορίζει ποια είναι η βέλτιστη ανάπτυξη των υπηρεσιών με τα νέα δεδομένα που διαμορφώθηκαν στο σύστημα.

Το σύστημα που θα αναπτυχθεί θα εστιάζει αυστηρά στην βελτιστοποίηση του κόστους επεξεργασίας των υπηρεσιών που αποτελούν την εφαρμογή καθώς και του κόστους επικοινωνίας μεταξύ των επεξεργαστών του κατανεμημένου συστήματος. Δεν θα ληφθούν υπ' όψιν παράγοντες που αφορούν την φύση της εκάστοτε εφαρμογής, όπως απαιτήσεις για ποιότητα παρεχόμενης υπηρεσίας, (για παράδειγμα στην υποθετική περίπτωση μίας εφαρμογής τηλεδιάσκεψης με βίντεο). Επίσης δεν θα ληφθούν υπ' όψιν παράγοντες που αφορούν το εκάστοτε υλικό του κατανεμημένου συστήματος. Θα γίνει η υπόθεση ότι το σύστημα είναι ένα ετερογενές κατανεμημένο σύστημα, αλλά δεν θα γίνει ειδίκευση αν είναι ένα δίκτυο φορητών υπολογιστών, ένα δίκτυο αισθητήρων, ή ένα καιροσκοπικό κινητό δίκτυο (mobile ad-hoc network - MANET) κτλ. Ο λόγος που γίνεται αυτό είναι ότι η υλική υποδομή του δικτύου δεν επηρεάζει το εξεταζόμενο πρόβλημα ούτε την λύση που θα προταθεί, καθώς αυτή θα είναι μια γενική λύση που μπορεί να εφαρμοστεί σε οποιοδήποτε κατανεμημένο σύστημα το οποίο θα παρέχει τις απαιτούμενες πληροφορίες για την υλοποίηση της λύσης.

1.3 Δομή της διπλωματικής εργασίας

Για την μελέτη του προβλήματος που εξετάζει η παρούσα διπλωματική εργασία θα ακολουθηθεί η εξής διαδικασία:

Στο κεφάλαιο 2 θα εξετασθούν όλες οι διαθέσιμες τεχνολογίες για την επίλυση του προβλήματος, δηλαδή οι διάφοροι αλγόριθμοι βελτιστοποίησης που υπάρχουν στην επιστημονική βιβλιογραφία και θα δοθεί έμφαση στους πιο σύγχρονους και αποτελεσματικούς από αυτούς.

Στο κεφάλαιο 3 θα γίνει επισκόπηση επιστημονικών εργασιών που υπάρχουν στην επιστημονική βιβλιογραφία και χρησιμοποιούν αλγορίθμους βελτιστοποίησης για να λύσουν είτε το ίδιο είτε παρεμφερή με το εξεταζόμενο θέματα.

Στο κεφάλαιο 4 θα γίνει η μοντελοποίηση του προτεινόμενου συστήματος για την λύση του προβλήματος, ενώ παράλληλα θα εξετασθεί

το πώς διαφοροποιείται το συγκεκριμένο σύστημα από τα συστήματα της επιστημονικής βιβλιογραφίας που εξετάστηκαν στο κεφάλαιο 3.

Στο κεφάλαιο 5 θα γίνει παρουσίαση του συστήματος που αναπτύχθηκε για την λύση του προβλήματος, με παρουσίαση του σκεπτικού λειτουργίας και των μεθόδων που το αποτελούν, καθώς και του γραφικού περιβάλλοντος χρήστη.

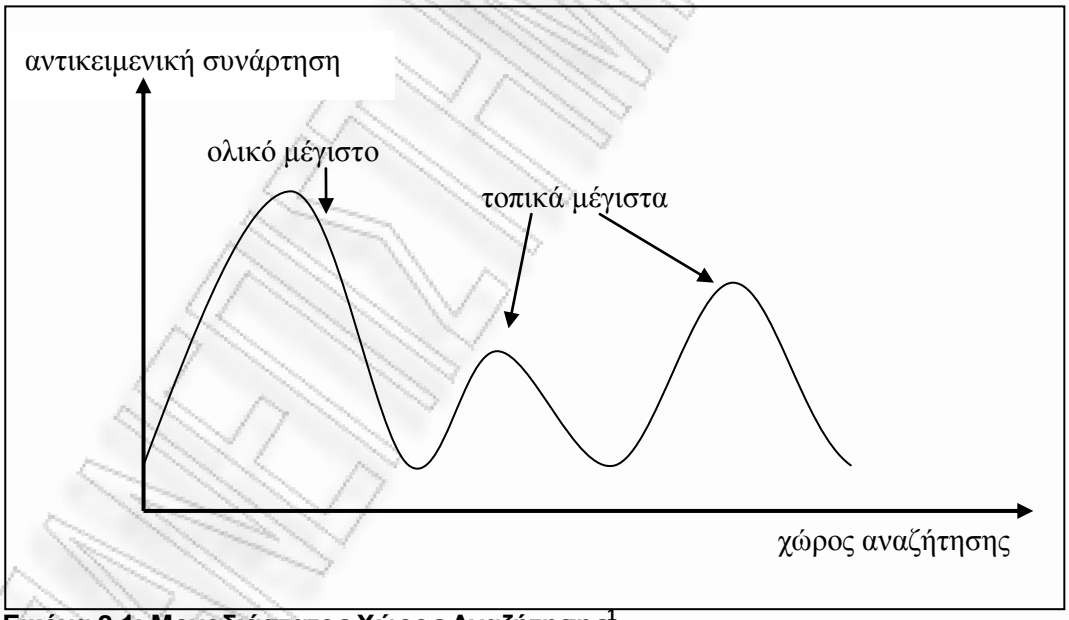
Στο κεφάλαιο 6 θα γίνει εκτίμηση της αποδοτικότητας του συστήματος που αναπτύχθηκε με χρήση στατιστικών αποτελεσμάτων για διάφορα σετ δοκιμών που εφαρμόστηκαν σε τρεις ελέγχους με διαφορετικό αριθμό υπηρεσιών και επεξεργαστών/κόμβων δικτύου.

Στο κεφάλαιο 8 θα γίνει μία ανακεφαλαίωση των ζητημάτων που ασχολείται η παρούσα εργασία και θα προταθούν κατευθύνσεις για περαιτέρω μελέτη πάνω στο εξεταζόμενο θέμα.

Κεφάλαιο 2: Επισκόπηση διαθέσιμων αλγόριθμων βελτιστοποίησης

Τα προβλήματα βελτιστοποίησης επιλύονται με τη χρήση **αλγορίθμων ολικής αναζήτησης** (global search algorithms) ή αλλιώς **αλγόριθμοι βελτιστοποίησης** (optimization algorithms). Οι αλγόριθμοι αυτοί είναι ικανοί να βρίσκουν λύσεις μέσα σε μεγάλους και συνεχείς χώρους καταστάσεων με την χρήση ποικίλων μεθοδολογιών που θα εξεταστούν στη συνέχεια.

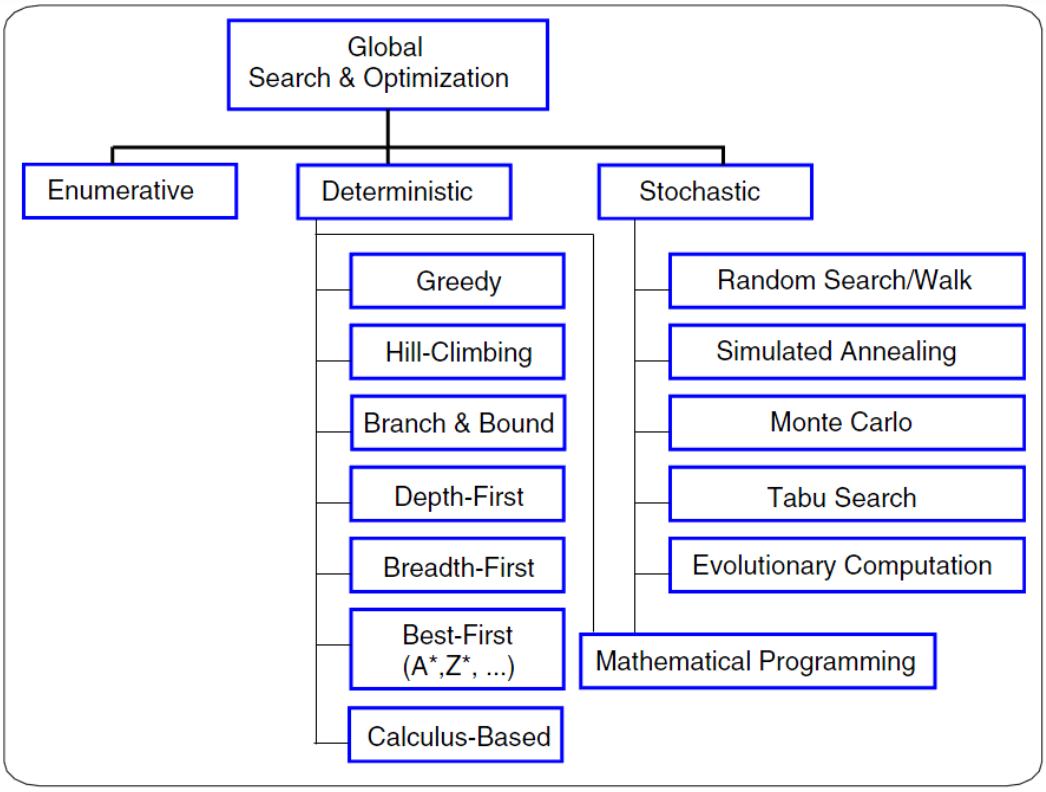
Ως **χώρος αναζήτησης** (search space) θα οριστεί το σύνολο των πιθανών τιμών της αντικειμενικής συνάρτησης (ή συνάρτηση κόστους όπως αλλιώς καλείται). Ένα παράδειγμα χώρου αναζήτησης φαίνεται στην εικόνα 2.1:



Εικόνα 2.1: Μονοδιάστατος Χώρος Αναζήτησης[1]

Οι αλγόριθμοι ολικής αναζήτησης ή βελτιστοποίησης χωρίζονται σε τρεις κατηγορίες [5]: απαριθμητικοί (enumerative), ντετερμινιστικοί (deterministic) και στοχαστικοί, όπως φαίνεται στην εικόνα 2.2.

[1] Σε πραγματικά προβλήματα βελτιστοποίησης ο χώρος αναζήτησης έχει αρκετά περισσότερες διαστάσεις. Η απλοποίηση γίνεται χωρίς βλάβη της γενικότητας.



Εικόνα 2.2: Κατηγοριοποίηση αλγορίθμων αναζήτησης και βελτιστοποίησης [5]

2.1 Απαριθμητικοί αλγόριθμοι

Οι απαριθμητικοί αλγόριθμοι παρόλο που είναι ντετερμινιστικοί διαχωρίζονται από αυτούς, καθώς δεν χρησιμοποιούν ευρετικές τεχνικές (heuristics). Αποτελούν την πιο απλή στρατηγική εξεύρεσης λύσης αφού υπολογίζουν και αξιολογούν όλες τις πιθανές λύσεις του χώρου αναζήτησης. Μπορεί να γίνει εύκολα αντιληπτό ότι τέτοιοι αλγόριθμοι δεν μπορούν να εφαρμοστούν σε πολύπλοκα προβλήματα με μεγάλους χώρους αναζήτησης καθώς αδυνατούν να δώσουν αποτέλεσμα μέσα σε αποδεκτό χρόνο [5].

2.2 Ντετερμινιστικοί αλγόριθμοι

Οι ντετερμινιστικοί αλγόριθμοι λειτουργούν αποκτώντας γνώση του πεδίου καταστάσεων και χρησιμοποιώντας την για να καθορίσουν την

βέλτιστη λύση [5]. Εφαρμόζονται κυρίως σε προβλήματα που περιέχουν γράφους και δενδρικές δομές.

Οι **Greedy** αλγόριθμοι λειτουργούν βρίσκοντας τοπικά βέλτιστες λύσεις σε κάθε βήμα, θεωρώντας ότι οι λύσεις αυτές είναι πάντα κομμάτι της συνολικής βέλτιστης λύσης [6]. Σε περιπτώσεις λοιπόν που δεν ισχύει αυτό οι αλγόριθμοι αυτοί αποτυγχάνουν.

Ο **Hill-Climbing** (αναρριχητής λόφων) αλγόριθμος αποτελείται από ένα βρόχο (loop) ο οποίος για κάθε τρέχουσα κατάσταση εξετάζει τις γειτονικές της και προχωρά προς την καλύτερη. Αν καμία γειτονική κατάσταση δεν έχει μεγαλύτερη τιμή από την τρέχουσα τότε ο αλγόριθμος σταματάει [7]. Αν ο αλγόριθμος βρεθεί σε μια κατάσταση που αποτελεί τοπικό μέγιστο τότε θα «κολλήσει», αφού δεν μπορεί να διακρίνει καταστάσεις πέρα από τις δύο γειτονικές του.

Οι τεχνικές **Branch and bound** (Διακλάδωσης και οριοθέτησης) υπολογίζουν όρια σε διάφορα υποσύνολα λύσεων σε δενδρικές δομές και έπειτα αξιολογούν τα όρια αυτά απορρίπτοντας ολόκληρα υποσύνολα λύσεων και επιλέγοντας μόνο το πιο ελπιδοφόρο [5] [8].

Ο αλγόριθμος αναζήτησης **depth-first** εφαρμόζεται σε δενδρικές δομές και γράφους και δουλεύει επιλέγοντας ένα κόμβο και ψάχνοντας διαδοχικά όλες τις βέλτιστες υπο-λύσεις του κόμβου μέχρι να τις εξετάσει όλες πριν αρχίσει να εξετάζει υπο-λύσεις άλλων κόμβων που δεν ήταν βέλτιστοι [5].

Ο αλγόριθμός αναζήτησης **Breadth-first** εφαρμόζεται και αυτός σε δενδρικές δομές και δουλεύει επιλέγοντας έναν κόμβο και ψάχνοντας πρώτα λύσεις σε όλους τους κόμβους που είναι γειτονικοί με αυτόν. [5].

Οι αλγόριθμοι τύπου **Best-first** χρησιμοποιούν ευριστικές πληροφορίες για να αποδώσουν αριθμητικές τιμές στους κόμβους ενός γράφου. Έπειτα εξετάζουν πρώτα τον πιο ελπιδοφόρο κόμβο [5]. Οι A*, Z* και άλλες παραλλαγές πέρα από τον πιο ελπιδοφόρο κόμβο λαμβάνουν υπ’ όψιν τους και το συνολικό κόστος για να φτάσουν σε αυτόν τον κόμβο [5].

Οι **calculus-based** (βασισμένες σε λογισμό) τεχνικές αναζήτησης βασίζονται στο να υπάρχει συνέχεια σε ένα πεδίο μεταβλητών ώστε να εξάγουν την καλύτερη λύση [5].

2.3 Στοχαστικοί αλγόριθμοι

Οι αλγόριθμοι για απαριθμητική ή ντετερμινιστική αναζήτηση, αδυνατούν να δώσουν λύσεις σε αρκετά πολύπλοκα επιστημονικά προβλήματα με ανώμαλους χώρους αναζήτησης. Γι αυτό το λόγο έχουν αναπτυχθεί αλγόριθμοι στοχαστικής αναζήτησης και βελτιστοποίησης οι οποίοι μπορούν να διαχειριστούν τέτοια πολύπλοκα συστήματα. Τέτοιοι αλγόριθμοι είναι ο **Simulated Annealing**, ο **Monte Carlo**, ο **Tabu search**, και **Evolutionary Computation.** Οι στοχαστικές μέθοδοι δουλεύουν αναθέτοντας τιμές σε μία συνάρτηση κόστους και αξιολογώντας τις πιθανές ή μερικές λύσεις που προκύπτουν. Αν και οι περισσότερες μέθοδοι βρίσκουν τελικά μία βέλτιστη λύση, δεν μπορούν να εγγυηθούν ότι αυτή είναι η καθολικά βέλτιστη λύση του πεδίου καταστάσεων [5].

Η **τυχαία αναζήτηση** (random search) είναι η απλούστερη στοχαστική στρατηγική αναζήτησης. Αξιολογεί ένα σύνολο τυχαίων λύσεων στο πεδίο καταστάσεων και επιλέγει την καλύτερη από αυτές [9]. Υπάρχουν διάφορες παραλλαγές του βασικού αλγορίθμου όπως ο αρκετά συναφής **random walk** ο οποίος αντί να ψάχνει σε κάθε βήμα μια τυχαία λύση χρησιμοποιεί την λύση κάθε βήματος ως αφετηρία για να επιλέξει την επόμενη τυχαία λύση [5].

Ο αλγόριθμος **simulated annealing** (προσομοιωμένη ανόπτηση) δουλεύει προσομοιώνοντας την φυσική διαδικασία της ανόπτησης, κατά την οποία ένα υλικό θερμαίνεται αρκετά και έπειτα αφήνεται σταδιακά ψυχθεί με σκοπό να αλλάξουν οι ιδιότητες του υλικού (π.χ. κατασκευή γυαλιού από πυρίτη). Ο αλγόριθμος σε κάθε βήμα αντικαθιστά την τρέχουσα λύση με μία τυχαία κοντινή λύση η οποία επιλέγεται βάση πιθανότητας και εξαρτάται από μία παράμετρο Τα, η οποία αρχικά έχει μία μεγάλη τιμή επιτρέποντας την αναζήτηση σε μακρινές λύσεις (σε

σχέση με την αρχική), ενώ σταδιακά μειώνεται (“ψύχεται”), περιορίζοντας έτσι τις αναζητήσεις σε πιο κοντινά διαστήματα ως προς την τρέχουσα λύση. Ο αλγόριθμος σταματάει τις αναζητήσεις όταν η τιμής της μεταβλητής Τ μηδενιστεί και επιστρέφει την “βέλτιστη” λύση που έχει εντοπίσει [10].

Οι αλγόριθμοι τύπου **Monte Carlo** δουλεύουν στηριζόμενοι σε στατιστικές μεθόδους όπου κάθε τυχαία λύση είναι εντελώς ανεξάρτητη από την προηγούμενη λύση και το αποτέλεσμά της. Η καλύτερη λύση σε κάθε βήμα αποφασίζεται βάση μεταβλητών απόφασης που στηρίζονται σε στατιστικά σφάλματα [5].

Ο αλγόριθμος **Tabu search** (αναζήτηση ταμπού) είναι ένας αλγόριθμος που λειτουργεί αξιολογώντας γειτονικές λύσεις (ως προς την τρέχουσα) στο χώρο αναζήτησης και χρησιμοποιώντας μνήμη όλων των προηγούμενων λύσεων καθώς και το πώς αυτές επιτεύχθηκαν. Χρησιμοποιεί ως κριτήρια αξιολόγησης την ποιότητα της λύσης, την συχνότητα που εμφανίζεται στον χώρο αναζήτησης, το πόσο πρόσφατη είναι καθώς και τι επιρροή έχει στην η εκάστοτε λύση στην βελτίωση της ποιότητας (δηλαδή κατά πόσο μεταβάλλεται η βέλτιστη λύση). Κάθε φορά που υπάρχει βελτίωση της λύσης αυτή μαρκάρεται ως “ταμπού” ώστε ο αλγόριθμος να μην την ξαναεπισκεφτεί κατά την διαδικασία αναζήτησης [11].

Η κατηγορία **mathematical programming** (μαθηματικός προγραμματισμός) αφορά μία οικογένεια αλγορίθμων που έχει αναπτυχθεί από την κοινότητα του κλάδου της “Επιχειρησιακής έρευνας” (Operation research). Η αλγόριθμοι αυτοί εστιάζουν γύρω από τους περιορισμούς του προβλήματος [5]. Ο **γραμμικός προγραμματισμός** (linear programming) χρησιμοποιείται σε προβλήματα που η συνάρτηση κόστους καθώς και όλοι οι περιορισμοί έχουν γραμμικές σχέσεις μεταξύ τους. Ο **μη-γραμμικός προγραμματισμός** (non-linear programming) χρησιμοποιείται σε προβλήματα όπου δεν έχουν γραμμικές σχέσεις μεταξύ τους και χρησιμοποιεί πολύπλοκες συναρτήσεις περιορισμών (constraint functions) για να εξάγει αποτελέσματα. Τέλος ο **στοχαστικός προγραμματισμός**

(stochastic programming) χρησιμοποιείται όταν οι συναρτήσεις κόστους καθώς και παράμετροι που λαμβάνουν τυχαίες τιμές υπόκεινται σε στατιστικές διακυμάνσεις οι οποίες είναι μέρος του ορισμού του προβλήματος [5].

Ο όρος **evolutionary computation** (εξελικτικός προγραμματισμός) περιγράφει μία οικογένεια μεθόδων που προσομοιώνουν υπολογιστικά την εξελικτική διαδικασία. Βασίζονται σε πληθυσμούς μεμονωμένων λύσεων, οι οποίοι εξελίσσονται στον χρόνο βάση της δαρβινικής θεωρίας περί επιβίωσης του καταλληλότερου [5]. Οι **evolutionary algorithms** (εξελικτικοί αλγόριθμοι) είναι η κυριότερη υποκατηγορία μεθόδων εξελικτικού προγραμματισμού και χρησιμοποιούν τις διαδικασίες της αναπαραγωγής, της επιλογής, της τυχαίας διακύμανσης και του ανταγωνισμού για την διαχείριση ενός πληθυσμού λύσεων οι οποίες αξιολογούνται από μία συνάρτηση καταλληλότητας [5].

Οι εξελικτικοί αλγόριθμοι είναι σχετικά οι πιο πρόσφατοι στο χώρο των αλγόριθμων βελτιστοποίησης και επίσης είναι οι πιο “δυνατοί” στο να βρίσκουν ολικά βέλτιστες λύσεις σε μία πληθώρα από διαφορετικά πολύπλοκα επιστημονικά προβλήματα, χωρίς να “παγιδεύονται” σε τοπικά βέλτιστες λύσεις [5][7][12][13]. Για τον λόγο αυτό οι κυριότεροι εξελικτικοί α αλγόριθμοι θα μελετηθούν πιο διεξοδικά στις επόμενες υπο-ενότητες, καθώς είναι οι πλέον κατάλληλοι για την λύση του προβλήματος που εξετάζει η παρούσα εργασία.

2.3.1 Έρευνα με Στοχαστική Διασπορά

Ο αλγόριθμος **Stochastic Diffusion Search - SDS** (έρευνα με στοχαστική διασπορά) προτάθηκε από τον J.M. Bishop το 1989 ως αλγόριθμος βελτιστοποίησης με εύρεση ομοιοτήτων (pattern matching), βασισμένος σε πληθυσμό [14] και αργότερα γενικεύτηκε ως αλγόριθμος βελτιστοποίησης οποιασδήποτε συνάρτησης.

Για να εντοπίσει τη βέλτιστη λύση της συνάρτησης, ο αλγόριθμος SDS χρησιμοποιεί ένα σμήνος από πράκτορες, καθένας από τους οποίους

διατηρεί μία υπόθεση (x_i) για τη βέλτιστη λύση. Έπειτα ο αλγόριθμος εκτελεί επαναληπτικά διαδικασίες ελέγχου των υποθέσεων και διασποράς (μετάδοσης) των αποτελεσμάτων στους υπόλοιπους πράκτορες, μέχρι το σμήνος να συγκλίνει στην βέλτιστη λύση.

Ο αλγόριθμος έχει ως εξής [15]:

```
INITIALISE (agents);
REPEAT
  TEST (agents);
  DIFFUSE (agents);
UNTIL CONVERGENCE (agents);
```

- INITIALISE (agents): Αρχικοποίηση. Σε αυτή τη φάση του αλγορίθμου δίνεται μία τυχαία αρχική υπόθεση σε κάθε πράκτορα προς έλεγχο.
- TEST (agents): Έλεγχος. Κάθε πράκτορας χρησιμοποιεί μία συνάρτηση για να ελέγξει την υπόθεσή του έναντι σε κάποια κριτήρια και αποφασίζει αν η υπόθεσή του τα ικανοποιεί. Η συνάρτηση ελέγχου επιστρέφει την τιμή “Αληθές” (True) ή “Ψευδές” (False) ανάλογα με το αν ικανοποιούνται τα κριτήρια ή όχι. Σε περίπτωση που η συνάρτηση επιστρέψει “Αληθές” ο πράκτορας γίνεται “ενεργό” μέλος του σμήνους, αλλιώς γίνεται “παθητικό”.
- DIFFUSE (agents): Διασπορά. Σε αυτή τη φάση όλοι οι “παθητικοί” πράκτορες του σμήνους επικοινωνούν τυχαία με ένα άλλο πράκτορα του σμήνους. Αν ο άλλος πράκτορας είναι “ενεργός” τότε μεταδίδει στον “παθητικό” πράκτορα την υπόθεσή του. Σε περίπτωση που ο κόμβος που επιλέχθηκε τυχαία είναι παθητικός, τότε ο κόμβος που τον επέλεξε, επιλέγει τυχαία μία νέα υπόθεση από τον χώρο αναζήτησης για να την εξετάσει στον επόμενο κύκλο.
- CONVERGENCE (agents): Σύγκλιση. Χρησιμοποιούνται δύο κριτήρια για να διαπιστωθεί αν υπάρχει σύγκλιση. Το πρώτο λέγεται “Ισχυρό Κριτήριο Τερματισμού” (Strong Halting Criterion), το οποίο ελέγχει

αν μία ομάδα πρακτόρων έχει συμπληρώσει έναν ορισμένο αριθμό μελών και έχει σταθερό στοχαστικά αριθμό μελών για κάποιες επαναλήψεις του αλγορίθμου. Το δεύτερο ονομάζεται “Ασθενές Κριτήριο Τερματισμού” (Weak Halting Criterion) και ελέγχει τη σταθερότητα και το ελάχιστο μέγεθος των συνολικών “ενεργών” πρακτόρων, καθώς η συνολική δραστηριότητα σχετίζεται με την καλύτερη τιμή μέχρι στιγμής [15].

2.3.2 Γενετικοί Αλγόριθμοι

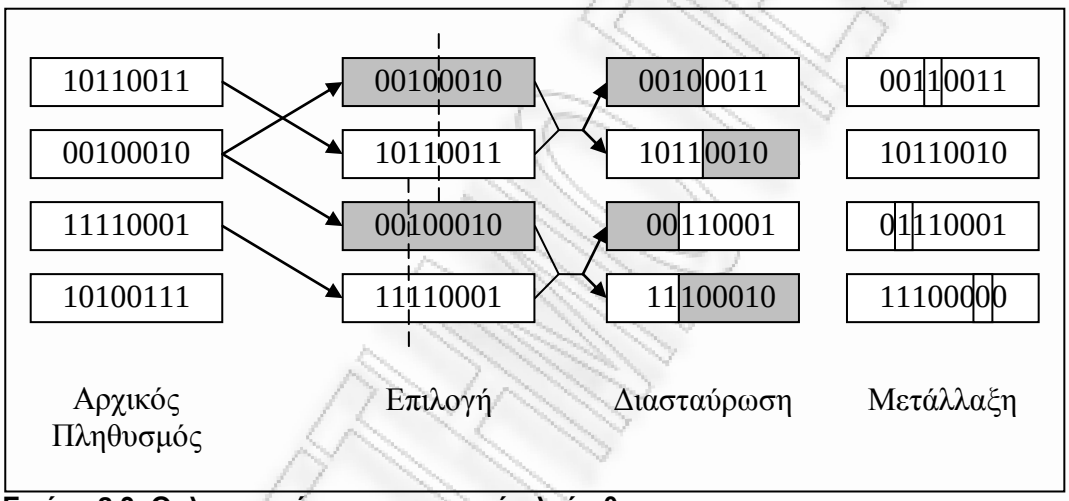
Οι **γενετικοί αλγόριθμοι** (Genetic Algorithms – GA’s) εφευρέθηκαν από τον John Holland τη δεκαετία του 1960 με σκοπό την μελέτη των εξελικτικών μοντέλων της φύσης και την εισαγωγή τους στους ηλεκτρονικούς υπολογιστές [16]. Στο χρονικό διάστημα μέχρι σήμερα έχουν αναπτυχθεί αρκετοί αλγόριθμοι βασισμένοι ή όχι στην αρχική ιδέα οι οποίοι καλούνται γενετικοί. Για να είναι κάποιοι αλγόριθμοι γενετικοί πρέπει να έχουν τα εξής χαρακτηριστικά: α) πληθυσμούς από "χρωμοσώματα" ή αλλιώς υποψήφιες λύσεις, β) επιλογή ανάλογα με την καταλληλότητα, γ) διασταύρωση για την δημιουργία απογόνων του πληθυσμού και δ) τυχαία μετάλλαξη των απογόνων [16].

- Πληθυσμός από “χρωμοσώματα” (Chromospheres population): Είναι ο αρχικός πληθυσμός των πρακτόρων, οι οποίοι φέρουν λύσεις που αντιστοιχούν στον χώρο αναζήτησης. Κάθε νέος πληθυσμός που δημιουργείται από τον αλγόριθμο ονομάζεται γενεά (generation).
- Επιλογή (Selection): Η επιλογή είναι μία λειτουργία η οποία επιλέγει τα “χρωμοσώματα” που θα αναπαραχθούν από το σύνολο του πληθυσμού. Η πιθανότητα επιλογής είναι σχετίζεται άμεσα με την καταλληλότητα (fitness) του “χρωμοσώματος”.
- Διασταύρωση (Crossover): Η διασταύρωση είναι μία λειτουργία που επιλέγει ένα μέρος της λύσης του καθενός από τα δύο “χρωμοσώματα” προς διασταύρωση και ανταλλάσσει τις υπο-λύσεις

μεταξύ των “χρωμοσωμάτων” αυτών προκειμένου να παράγει τους απογόνους των αρχικών “χρωμοσωμάτων”.

• Μετάλλαξη (Mutation): Η λειτουργία αυτή παραποιεί τυχαία μέρος της υπο-λύσης ενός απογόνου. Η πιθανότητα μετάλλαξης είναι πολύ μικρή (της τάξης του 1%).

Η εικόνα 2.3 δείχνει τις βασικές λειτουργίες του γενετικού αλγόριθμου εφαρμοσμένες στο παράδειγμα ενός πληθυσμού “χρωμοσωμάτων” αποτελούμενο από bits:



Εικόνα 2.3: Οι λειτουργίες του γενετικού αλγόριθμου

Ο αλγόριθμος έχει ως εξής [16]:

1. Εκκίνηση με τυχαίο πληθυσμό “χρωμοσωμάτων”.
2. Υπολογισμός της καταλληλότητας (fitness) του κάθε “χρωμοσώματος” του πληθυσμού.
3. Επανάληψη των ακόλουθων βημάτων μέχρι να παραχθεί ο ν-οστός απόγονος:

   α. Επιλογή ενός ζευγαριού γονέων από τον πληθυσμό με πιθανότητα επιλογής να αυξάνει σε σχέση με την καταλληλότητα του “χρωμοσώματος”. Ένα “χρωμόσωμα” μπορεί να επιλεγεί περισσότερες από μία φορές για γονέας.

   β. Με πιθανότητα p_c (πιθανότητα διασταύρωσης), γίνεται διασταύρωση του ζεύγους σε ένα τυχαίο σημείο της λύσης (επιλεγμένο με κανονική

κατανομή) για το σχηματισμό 2 απογόνων. Αν δεν πραγματοποιηθούν διασταυρώσεις, τότε σχηματίζονται οι απόγονοι ως ακριβή αντίγραφα των γονέων.

γ. Μετάλλαξη των απογόνων σε μέρος της υπο-λύσης τους με πιθανότητα p_m (πιθανότητα μετάλλαξης) και τοποθέτηση των απογόνων στον νέο πληθυσμό. Αν το ν είναι μονός αριθμός τότε ένα μέλος του καινούριου πληθυσμού απομακρύνεται τυχαία.

4. Αντικατάσταση του παλιού με τον νέο πληθυσμό
5. Επιστροφή στο βήμα 2.

Ο αλγόριθμος εκτελείται συνήθως για 50 ως 500 επαναλήψεις και η τελική λύση εξάγεται από τα "χρωμοσώματα" που στο τέλος της εκτέλεσης παρουσιάζουν τα καλύτερα χαρακτηριστικά λύσης. Ο αριθμός των επαναλήψεων και οι πιθανότητες p_c και p_m καθορίζουν την αποδοτικότητα του αλγορίθμου [16].

2.3.3 Βελτιστοποίηση με Αποικία Μυρμηγκιών

Η **βελτιστοποίηση με Αποικία Μυρμηγκιών** (Ant Colony Optimization - ACO) είναι ένας αλγόριθμος που προτάθηκε από τον Marco Doringo το 1992 [17], ως μέθοδος επίλυσης υπολογιστικών προβλημάτων με πιθανότητες, που έγκειται στην εύρεση βέλτιστων διαδρομών σε γράφους. Ο αλγόριθμος είναι εμπνευσμένος από τον τρόπο που μετακινούνται τα μυρμήγκια βρίσκοντας διαδρομές από την αποικία τους προς την τροφή. Οι πράκτορες αποτελούν εικονικά μυρμήγκια και είναι στην πραγματικότητα στοχαστικές διαδικασίες που οικοδομούν μία λύση. Η λύση αυτή δημιουργείται προσθέτοντας συστατικά της λύσης σε ημιτελείς λύσεις που έχουν ήδη οικοδομηθεί. Αυτό το κάνουν λαμβάνοντας υπ' όψιν τους ευριστικές (heuretic) πληροφορίες για το στιγμιότυπο του προβλήματος που λύνεται, αν είναι διαθέσιμες, καθώς και τεχνητά "ίχνη φερομόνης" (feromone trails), τα οποία αλλάζουν δυναμικά για να

αναπαραστήσουν την εμπειρία που αποκτούν οι πράκτορες στην αναζήτηση [18].

Ο αλγόριθμος έχει ως εξής [18]:

```
Αρχικοποίηση_παραμέτρων
Πρόγραμμα_δραστηριοτήτων
    Ενέργειες_Μυρμηγκιών
    Εξάτμιση_Ιχνών_Φερομόνης
    Ενέργειες_δαίμονα (προαιρετικά)
Τέλος_προγράμματος_δραστηριοτήτων
```

- Αρχικοποίηση_παραμέτρων: Σε αυτή τη φάση αρχικοποιούνται οι πράκτορες-μυρμήγκια, με αρχικές υποθέσεις, οι οποίες αποτελούν τα αρχικά μέρη της λύσης πάνω στα οποία θα οικοδομήσουν τα νέα συστατικά λύσης.
- Πρόγραμμα δραστηριοτήτων: Το πρόγραμμα δραστηριοτήτων αφορά είναι όλες οι ενέργειες που εκτελούνται επαναληπτικά από τον αλγόριθμο. Οι ενέργειες μέσα στο Πρόγραμμα_δραστηριοτήτων μπορούν να εκτελεστούν παράλληλά ή σειριακά και με διαφορετική σειρά, ανάλογα με την υλοποίηση του αλγόριθμου από τον εκάστοτε προγραμματιστή. Ο αριθμός των επαναλήψεων για το Πρόγραμμα_δραστηριοτήτων είναι καθορισμένος από την αρχή.
- Ενέργειες_Μυρμηγκιών: Κατά τη φάση αυτή οι πράκτορες-μυρμήγκια κινούνται σε μία γειτονική μερική λύση του προβλήματος, την οποία προσθέτουν στη λύση τους, κατασκευάζοντας έτσι ένα μονοπάτι στον γράφο αποφάσεων. Έπειτα οι πράκτορες-μυρμήγκια χρησιμοποιούν την συνάρτηση ελέγχου που διαθέτουν για να εκτιμήσουν την καταλληλότητα της λύσης που έχουν οικοδομήσει, συνεκτιμώντας τα ίχνη φερομόνης που τυχόν την σημαδεύουν. Αν η δομημένη λύση τους ικανοποιεί τα κριτήρια τότε αφήνουν επιπρόσθετα ίχνη φερομόνης -δηλαδή σημαδεύουν- τα συστατικά της λύσης ή τη διαδρομή που οδηγεί στη συγκεκριμένη κατασκευασμένη λύση.

• Εξάτμιση_Ιχνών_Φερομόνης: Σε αυτή την φάση ο αλγόριθμος μειώνει την τιμή των ιχνών φερομόνης. Αυτό γίνεται ώστε να μην γίνεται γρήγορη σύγκλιση των πρακτόρων-μυρμηγκιών σε μία περιοχή του χώρου αναζήτησης, αλλά να εξερευνώνται νέες περιοχές.

• Ενέργειες_δαίμονα: Αυτή η φάση είναι προαιρετική και αφορά τον κεντρικό έλεγχο των πρακτόρων. Μπορεί μία διεργασία-δαίμονας να συλλέγει τις πληροφορίες των πρακτόρων-μυρμηγκιών και να κάνει κεντρική εκτίμηση της καταλληλότητας (fitness) ορισμένων λύσεων ή περιοχών του χώρου αναζήτησης και να επεμβαίνει ενισχύοντας παραδείγματος χάριν τα ίχνη φερομόνης που αφορούν την δημιουργία της καλύτερης λύσης.

Αφού ολοκληρωθεί ο προκαθορισμένος αριθμός επαναλήψεων του αλγορίθμου τότε εξάγονται τα τελικά αποτελέσματα από το σημείο σύγκλισης των πρακτόρων-μυρμηγκιών.

2.3.4 Βελτιστοποίηση με Σμήνος Σωματιδίων

Ο αλγόριθμος **Βελτιστοποίησης με Σμήνος Σωματιδίων** (Particle Swarm Optimization - PSO) αναπτύχθηκε από τον J. Kennedy αρχικά σαν μέθοδος προσομοίωσης κοινωνικής συμπεριφοράς, ενώ το 1995 παρουσιάστηκε ως μέθοδος επίλυσης προβλημάτων βελτιστοποίησης [19]. Ο αλγόριθμος βασίζεται στον τρόπο που αλληλεπιδρούν τα μέλη ενός σμήνους πουλιών ή μίας αγέλης ζώων, όταν κινούνται ομαδικά, ψάχνοντας προς εύρεση τροφής ή αποφεύγοντας κυνηγούς. Σε αυτές οι ομάδες τα άτομα κινούνται προσπαθώντας να διατηρήσουν μία βέλτιστη απόσταση σε σχέση του γείτονές τους, ώστε να μην απομακρυνθούν από την ομάδα. Αυτό το επιτυγχάνουν ανταλλάσοντας πληροφορίες σχετικά με την κίνησή τους οι οποίες διαδίδονται σε όλη την ομάδα [19]. Ο αλγόριθμος χρησιμοποιεί σωματίδια (ή πράκτορες) τα οποία κινούνται με μία ταχύτητα στον χώρο αναζήτησης και τα οποία ανταλλάσσουν πληροφορίες για τις μετρήσεις τους με τους γείτονές τους, ώστε να

επαναπροσδιορίσουν την ταχύτητά τους στον χώρο. Η εύρεση της βέλτιστης λύσης επιτυγχάνεται όταν έπειτα από επαναλήψεις αυτής της διαδικασίας το σμήνος συγκλίνει σε αυτήν που θεωρεί βέλτιστη λύση.

Πιο αναλυτικά, ο αλγόριθμος περιγράφεται ως εξής [20]:

Υπάρχει ένα σμήνος από σωματίδια που ψάχνουν να βρουν την καλύτερη τιμή ερευνώντας σε υποομάδες (clusters) τον χώρο αναζήτησης. Κάθε *i* σωματίδιο έχει ένα διάνυσμα συντεταγμένων $(\vec{x}_i)$ στον χώρο, καθώς και ένα διάνυσμα ταχύτητας $(\vec{u}_i)$. Τα διανύσματα θέσης και ταχύτητας αναλύονται σε συνιστώσες $\vec{x}_i = (\vec{x}_{i1}, \vec{x}_{i2}, ..., \vec{x}_{iD})$ και $\vec{u}_i = (\vec{u}_{i1}, \vec{u}_{i2,}, ..., \vec{u}_{iD})$ αντίστοιχα, όπου *1,2,…,D* οι διαστάσεις (dimensions) του χώρου αναζήτησης.

Ακόμα κάθε *i* σωματίδιο ξέρει την τιμή της συνάρτησης προς βελτιστοποίηση στο σημείο που βρίσκεται και ακόμα έχει μνήμη της καλύτερης τιμής που έχει βρει ποτέ (pBest) και των συντεταγμένων της $(\vec{p}_i)$, καθώς και της καλύτερης τιμής (gBest) που έχει βρει ποτέ κάποιο άλλο σωματίδιο από την υποομάδα του και των συντεταγμένων της $(\vec{p}_g)$, δηλαδή την καλύτερη τιμή και συντεταγμένες που έχει επιτύχει η υποομάδα.

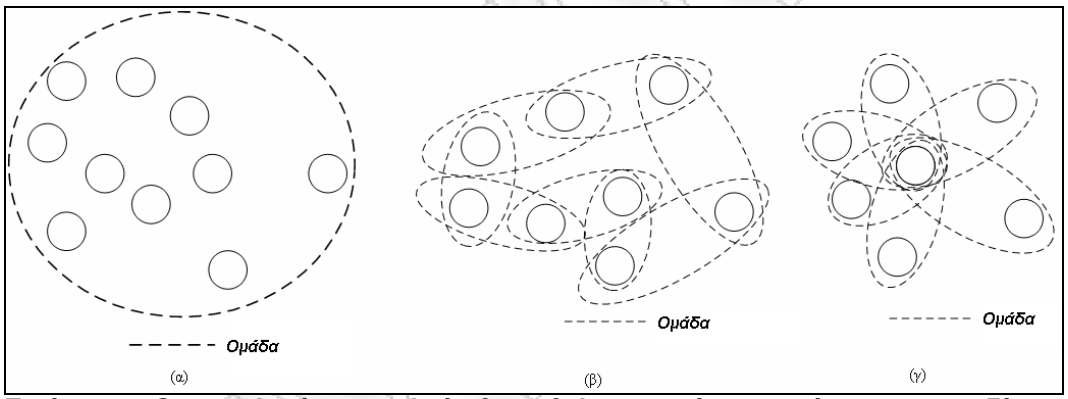
Η υποομάδα (cluster) ενός σωματιδίου έχει αποφασιστεί κατά την αρχικοποίηση του αλγορίθμου από τον προγραμματιστή. Κάθε σωματίδιο μπορεί να είναι μέλος σε πάνω από μία υποομάδα και ο σκοπός είναι όλες οι υποομάδες να έχουν κοινά μέλη, ώστε να υπάρχει συνεργασία μεταξύ τους και να σχηματίζουν μία καθολική ομάδα, που είναι ολόκληρο το σμήνος. Οι τύποι ομάδων που έχουν μελετηθεί διεξοδικά στη βιβλιογραφία είναι οι εξής [21]:

- Καθολική (Global): Σε αυτή τη τοπολογία όλα τα σωματίδια του σμήνους ανήκουν στην ίδια ομάδα. Το μέγιστο της ομάδας είναι και το ολικό μέγιστο και έτσι η ενημέρωση του συνόλου των σωματιδίων γίνεται σε ένα βήμα. Η καθολική τοπολογία φαίνεται στην εικόνα 2.4.α.

- Κύκλος (Circle): Σε αυτή τη τοπολογία κάθε σωματίδιο συνδέεται με N γειτονικά του σωματίδια (συνήθως N=2 ή 3). Για N=2 η τοπολογία

αυτή ονομάζεται και δακτύλιος (ring). Η τοπολογία Κύκλου έχει ως χαρακτηριστικό την αρκετά αργή (για μικρό N) διάδοση της πληροφορίας μεταξύ των υποομάδων του σμήνους, με αποτέλεσμα το σύνολο των σωματιδίων να ενημερώνεται για ένα νέο βέλτιστο σε βήματα ίσα με το μισό του αριθμού των σωματιδίων. Η τοπολογία κύκλου για N=2 (δακτύλιος) φαίνεται στην εικόνα 2.4.β.

- Αστέρας ή Τροχός (Star or Wheel): Σε αυτή τη τοπολογία υπάρχει ένα κεντρικό σωματίδιο και όλα τα άλλα σχηματίζουν ανά ένα ομάδες με αυτό. Επειδή όλες οι πληροφορίες μεταδίδονται μέσω του κεντρικού σωματιδίου, η ενημέρωση του συνόλου των σωματιδίων γίνεται σε δύο βήματα. Η τοπολογία αστέρα φαίνεται στην εικόνα 2.4.γ.



Εικόνα 2.4: Οι τοπολογίες του αλγόριθμου βελτιστοποίησης σμήνους σωματιδίων

Ο αλγόριθμος έχει ως εξής [20]:

Επανάληψη_μέχρι_να_ικανοποιηθεί_το_κριτήριο_τερματισμού{
  Για *i* = 1 μέχρι *αριθμός_σωματιδίων_σμήνους*{
    Αν $G(\vec{x}_i) > G(\vec{p}_i)$ τότε{
      Για *d* = 1 μέχρι *αριθμός_διαστάσεων*
        $(p_{id}) = (x_{id})$
      }
    }
    $g = i$
    Για *j* =1 μέχρι *λίστα_γειτόνων*

Αν $G(\vec{p}_j) > G(\vec{p}_g)$ τότε{

$g = j$

}

}

Για *d* = 1 μέχρι *αριθμός_διαστάσεων*{

$\upsilon_{id}(t) = \upsilon_{id}(t-1) + \varphi_1 \quad p_{id} - x_{id}(t-1) \quad + \varphi_2 \quad p_{gd} - x_{id}(t-1)$

$x_{id}(t) = x_{id}(t-1) + \upsilon_{id}(t)$

}

}

}

Όπου:

$G()$ εκτιμά την καταλληλότητα της λύσης

g είναι ο δείκτης του μέγιστου της υποομάδας

t είναι τρέχον βήμα του αλγορίθμου

φ_1, φ_2 είναι δύο τυχαίοι αριθμοί, οι οποίοι προσδίδουν κάποια τυχαιότητα στο αποτέλεσμα της νέας ταχύτητας.

Οι τύποι που χρησιμοποιεί ο αλγόριθμος όπως παρουσιάστηκε παραπάνω, έχουν το μειονέκτημα ότι δεν εγγυούνται την σύγκλιση των σωματιδίων σε μία λύση, καθώς υπάρχει πιθανότητα η ταχύτητά τους να μην μηδενιστεί ποτέ [22]. Γι' αυτό τον λόγο έχει προταθεί μία παραλλαγή του κλασσικού αλγορίθμου, η οποία ονομάζεται PSO Type 1" [22] και η οποία εισάγει έναν παράγοντα χ ο οποίος περιορίζει την ταχύτητα.

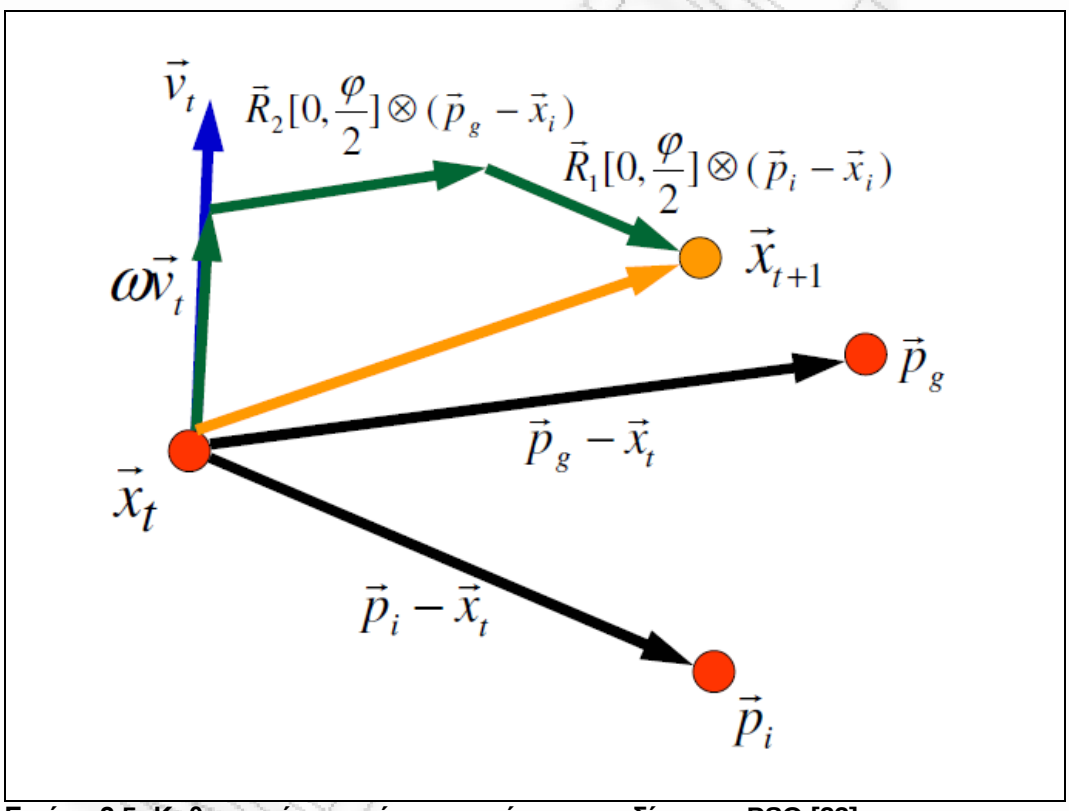
Έτσι ο τύπος που δίνει την νέα ταχύτητα του κάθε πράκτορα αλλάζει από: $\upsilon_{id}(t) = \upsilon_{id}(t-1) + \varphi_1 \quad p_{id} - x_{id}(t-1) \quad + \varphi_2 \quad p_{gd} - x_{id}(t-1)$

σε: $\upsilon_{id}(t) = \chi\left[\upsilon_{id}(t-1) + \varphi_1 \quad p_{id} - x_{id}(t-1) \quad + \varphi_2 \quad p_{gd} - x_{id}(t-1)\right]$

όπου: $\chi = \begin{cases} \dfrac{2\kappa}{\varphi - 2 + \sqrt{\varphi^2 - 4\varphi}} \text{ , για } \varphi > 4 \\ \text{αλλιώς } \sqrt{\kappa} \end{cases}$

Το φ είναι η μέγιστη τιμή τυχαιότητας που μπορούν να λάβουν οι παράγοντες φ_1,φ_2, δηλαδή $\varphi_1,\varphi_2 \in (0,\varphi]$. Ο κ είναι ένας παράγοντας που ο οποίος επηρεάζει το πόσο γρήγορα συγκλίνουν μεταξύ τους τα σωματίδια του αλγορίθμου και παίρνει τιμές στο διάστημα (0,1]. Όσο μεγαλύτερη είναι η τιμή του, τόσο περισσότερο καθυστερεί η σύγκλιση των σωματιδίων σε μία λύση [22].

Η εικόνα 2.5 δείχνει σχηματικά τον τρόπο με τον οποίο καθορίζεται το διάνυσμα ταχύτητας ενός σωματιδίου σε κάθε βήμα του αλγορίθμου.



Εικόνα 2.5: Καθορισμός της κίνησης ενός σωματιδίου του PSO [23]

Όπως φαίνεται στην εικόνα το σωματίδιο βρίσκεται στην θέση $\vec{x}_t$ στο τρέχον βήμα του αλγορίθμου, $\vec{v}_t$ είναι το διάνυσμα της ταχύτητας που υπολογίζεται για το επόμενο βήμα, $\omega\vec{v}_t$ είναι το διάνυσμα που προκύπτει έπειτα από περιορισμό της ταχύτητας βάση του παράγοντα χ (αναπαρίσταται στο σχήμα ως ω ώστε να μην δημιουργείται σύγχυση σε σχέση με το διάνυσμα θέσης). Στο διάγραμμα απεικονίζονται επίσης τα

διανύσματα θέσης $\vec{p}_i, \vec{g}_i$ και τα διανύσματά απόστασής τους από το $\vec{x}_t$. Τα υπόλοιπα δύο πράσινα διανύσματα δείχνουν την επίδραση των διανυσμάτων απόστασης των $\vec{p}_i, \vec{g}_i$ στην τελική διαμόρφωση του διανύσματος θέσεως (λαμβανομένης υπ' όψιν της τυχαιότητας $\vec{R}_1$ και $\vec{R}_2$ αντίστοιχα). Το άθροισμα των πράσινων διανυσμάτων δίνει το τελικό πορτοκαλί διάνυσμα θέσης του σωματιδίου $\vec{x}_{t+1}$ για το επόμενο βήμα του αλγορίθμου.

Κεφάλαιο 3: Επισκόπηση αλγοριθμικών λύσεων για την ανάπτυξη υπηρεσιών

Στο προηγούμενο κεφάλαιο παρουσιάστηκαν οι διάφορες κατηγορίες των αλγορίθμων βελτιστοποίησης που μπορούν να χρησιμοποιηθούν για να επιλύσουν το πρόβλημα της βέλτιστης ανάπτυξης υπηρεσιών σε πολύπλοκα δυναμικά συστήματα. Στο κεφάλαιο αυτό παρουσιάζονται τρεις βιβλιογραφικές πηγές που κάνουν χρήση των κυριότερων εξελικτικών αλγορίθμων (evolutionary algorithms) σε θέματα που αφορούν την ανάπτυξη υπηρεσιών σε πολύπλοκα δυναμικά συστήματα.

3.1 Οργάνωση κατανεμημένων υπηρεσιών σε δίκτυα αισθητήρων με χρήση του αλγορίθμου "Ant colony optimization"

Ο F. Rammig κ.α. μελετάνε στο [24] ένα τρόπο κατανομής των υπηρεσιών του λειτουργικού συστήματος σε ένα ασύρματο δίκτυο αισθητήρων, με χρήση του αλγορίθμου βελτιστοποίησης **Ant colony optimization** (βελτιστοποίηση με αποικία μυρμηγκιών).

Η λύση που προτείνουν στο πρόβλημα που εξετάζουν αποτελείται δύο στάδια. Στο πρώτο στάδιο οι κόμβοι του δικτύου οργανώνονται σε ομάδες (clusters) με χρήση κατάλληλου αλγορίθμου κατάτμησης (clustering), έτσι ώστε να επιτευχθεί η ελαχιστοποίηση των πόρων που χρησιμοποιεί η κάθε ομάδα (cluster), ενώ ταυτόχρονα η κάθε ομάδα (cluster) να αποτελεί όσο γίνεται μεγαλύτερο ανεξάρτητο κομμάτι του δικτύου.

Στο δεύτερο στάδιο γίνεται χρήση του αλγόριθμου **ant colony optimization** με σκοπό την ελαχιστοποίηση του κόστους επικοινωνίας μεταξύ των υπηρεσιών του λειτουργικού συστήματος και των εργασιών των διαφόρων εφαρμογών. Για κάθε σύνδεσμο (link) υπάρχει μία

συνάρτηση βάρους που αναπαριστά την ποιότητά του. Οι υπηρεσίες του λειτουργικού συστήματος αναπαριστούνται ως πηγές τροφής (food resources), ενώ η κλήσεις των υπηρεσιών αποτελούν τα μυρμήγκια (ants) και οι σύνδεσμοι τα μονοπάτια (paths) του αλγορίθμου. Καθώς οι κλήσεις για υπηρεσίες δρομολογούνται στο δίκτυο, αφήνουν ίχνη φερομόνης στους κόμβους τους οποίους επισκέπτονται και ο αλγόριθμος εξάγει συμπεράσματα με τον τρόπο που περιγράφτηκε στο προηγούμενο κεφάλαιο. Ο αλγόριθμος χρησιμοποιεί τον εξής περιορισμό για την λειτουργία του: οι υπηρεσίες του λειτουργικού συστήματος και οι εργασίες που ανατίθενται σε κάθε κόμβο δεν γίνεται να είναι περισσότερες από τους διαθέσιμους πόρους του κόμβου.

Στα αποτελέσματα της μελέτης επικυρώνεται η βιωσιμότητα της προτεινόμενης λύσης, αλλά σημειώνεται ότι παρατηρούνται προβλήματα που προκύπτουν από την “λαίμαργη” (greedy) φύση του αλγορίθμου όταν διαφορετικοί αιτούντες ζητούν την ίδια υπηρεσία χρησιμοποιώντας διαφορετικά μονοπάτια (paths) δρομολόγησης.

3.2 Δόμηση και ανάπτυξη υπηρεσιών με τη χρήση του γενετικού αλγόριθμου

Ο Y. Vanrompay κ.α. μελετάνε στο [25] τη χρήση του **γενετικού αλγορίθμου** (genetic algorithm) για την σύνθεση υπηρεσιών από συστατικά και την ανάπτυξή τους σε συστήματα κινητών συσκευών (mobile networks).

Στο σύστημα που προτείνουν και μελετάνε κάθε υπηρεσία απαρτίζεται από ένα σύνολο διαφορετικών συστατικών. Υπάρχει μεταβλητότητα στα συστατικά που αποτελούν τις υπηρεσίες και το σύστημα κατά τη διάρκεια της λειτουργίας του επιλέγει ποια παραλλαγή υπηρεσίας είναι κάθε φορά η καταλληλότερη για χρήση.

Ο γενετικός αλγόριθμος χρησιμοποιείται δύο φορές, αρχικά για την σύνθεση της εκάστοτε υπηρεσίας που θα εκτελείται σε ένα κόμβο από τα διάφορα διαθέσιμα συστατικά, και έπειτα για την ανάπτυξη της κάθε

υπηρεσίας σε κόμβο του δικτύου, λαμβάνοντας υπ' όψιν τα κόστη επικοινωνίας των κόμβων.

Στο πρώτο στάδιο η κάθε λύση κωδικοποιείται ως ένα χρωμόσωμα που αποτελείται από ένα πίνακα ακέραιων με τόσα στοιχεία, όσα και ο αριθμός των διαφορετικών συστατικών που συνθέτουν την υπηρεσία. Κάθε στοιχείο του πίνακα αποθηκεύει και μία εναλλακτική σύνθεση συστατικών. Έπειτα χρησιμοποιείται ο γενετικός αλγόριθμος, όπως περιγράφεται στο προηγούμενο κεφάλαιο, για να εξάγει την βέλτιστη σύνθεση της κάθε υπηρεσίας για κάθε κόμβο βασιζόμενος στην εξής συνάρτηση καταλληλότητας: $F = Qos - D$, όπου:

$$Qos = \frac{w1 \cdot \Delta\iota\alpha\theta\varepsilon\sigma\iota\mu\acute{o}\tau\eta\tau\alpha + w2 \cdot \text{Α}\xi\iota o\pi\iota\sigma\tau\acute{\iota}\alpha}{w3 \cdot \text{Κ}\acute{o}\sigma\tau o\varsigma + w4 \cdot \text{Χ}\rho\acute{o}\nu o\varsigma\_\alpha\pi\acute{o}\kappa\rho\iota\sigma\eta\varsigma}$$

είναι η συνάρτηση μέτρησης της ποιότητας των υπηρεσιών. Τα *w1, w2, w3, w4* είναι βάρη που ρυθμίζουν το πόσο μεγάλο ρόλο παίζει κάθε μία από της παραμέτρους της υπηρεσίας (Διαθεσιμότητα, Αξιοπιστία, Κόστος, Χρόνος Απόκρισης). Η συνάρτηση *D* χρησιμοποιείται για να "τιμωρήσει" (penalize) τις λύσεις που δεν ικανοποιούν τους περιορισμούς του προβλήματος.

Αφού εξαχθούν τα αποτελέσματα του γενετικού αλγορίθμου και υπολογιστεί η βέλτιστη καταλληλότητα κάθε υπηρεσίας για κάθε κόμβο, χρησιμοποιείται εκ νέου ο γενετικός αλγόριθμος για να καθοριστεί ποια υπηρεσία θα αναπτυχθεί σε ποιον κόμβο, λαμβάνοντας υπ όψιν τόσο την καταλληλότητά που έχει υπολογιστεί, όσο και τα κόστη επικοινωνίας μεταξύ των κόμβων από την ανάθεση.

Στα αποτελέσματα της μελέτης επικυρώνεται η βιωσιμότητα της λύσης η οποία βρίσκει μία σχεδόν βέλτιστη λύση, έχει καλή επεκτασιμότητα και λαμβάνει υπ' όψιν της τους περιορισμένους πόρους του κινητού συστήματος (κατά το στάδιο σύνθεσης της υπηρεσίας). Τέλος ο αλγόριθμος έχει την δυνατότητα να διακόψει την εκτέλεση του οποτεδήποτε είναι επιθυμητό, παρέχοντας μία λύση, αν και η λύση αυτή θα είναι κατώτερης ποιότητας από ότι αν ο αλγόριθμος είχε ολοκληρώσει την λειτουργία του.

3.3 Ανάθεση εργασιών με τον αλγόριθμο βελτιστοποίησης με σμήνος σωματιδίων

Ο A. Salman κ.α. [26] προτείνουν την χρήση της βελτιστοποίησης με σμήνος σωματιδίων (particle swarm optimization) για την ανάθεση εργασιών (tasks) σε κατανεμημένα συστήματα.

Το σύστημα το οποίο εξετάζουν κάνει στατική ανάθεση των εργασιών στους κόμβους του συστήματος, δηλαδή η κατανομή των υπηρεσιών δεν αλλάζει κατά την διάρκεια ζωής των εργασιών. Επίσης το σύστημα είναι ομογενές, δηλαδή όλοι οι επεξεργαστές του συστήματος θεωρείται ότι έχουν την ίδια επεξεργαστική ισχύ και η κάθε εργασία (task) χρειάζεται τον ίδιο χρόνο να ολοκληρωθεί σε οποιοδήποτε επεξεργαστή του συστήματος.

Το σύστημα κάνει χρήση του αλγορίθμου βελτιστοποίησης με σμήνος σωματιδίων (particle swarm optimization), που περιγράφτηκε στο προηγούμενο κεφάλαιο, με στόχο την μείωση του συνολικού χρόνου επεξεργασίας του πιο χρονοβόρου επεξεργαστή. Κάθε εργασία που είναι προς ανάθεση αποτελεί μία διάσταση στον χώρο καταστάσεων και κάθε σωματίδιο (particle) του αλγορίθμου αποτελείται από ένα διάνυσμα που περιέχει την θέση της κάθε εργασίας στον αντίστοιχο χώρο. Ως κριτήριο τερματισμού του αλγόριθμου χρησιμοποιείται ο αριθμός των γενεών των σωματιδίων, ο οποίος τίθεται διπλάσιος του αριθμού των εργασιών.

Οι συγγραφείς αντιπαραθέτουν στατιστικά το σύστημα που ανέπτυξαν, με δική τους υλοποίηση αντίστοιχου συστήματος που βασίζεται στον γενετικό αλγόριθμο και εξάγουν το συμπέρασμα ότι με χρήση του αλγόριθμου βελτιστοποίησης με σμήνος σωματιδίων υπάρχει βελτίωση της συνάρτησης κόστους κατά 18.1%, ενώ ο αλγόριθμος τρέχει κατά 1.62 φορές γρηγορότερα από τον αντίστοιχο γενετικό.

Κεφάλαιο 4: Μοντελοποίηση λύσης

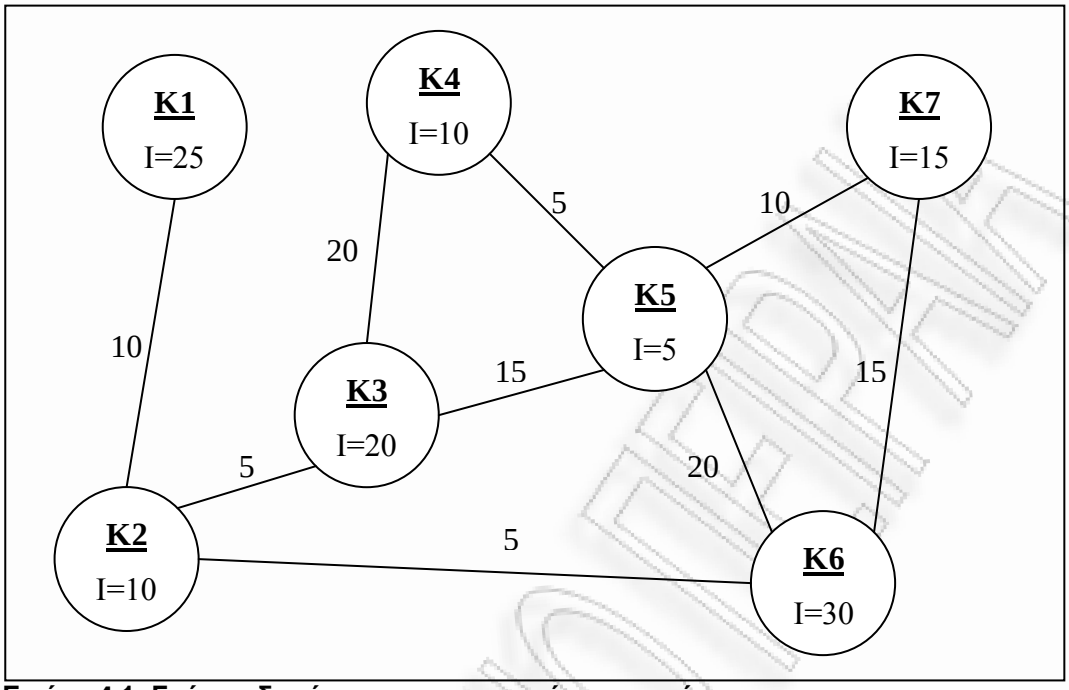
Στο παρόν κεφάλαιο περιγράφεται διεξοδικά η λύση που προτείνεται για την επίλυση του προβλήματος της δημιουργίας συστήματος βέλτιστης ανάπτυξης υπηρεσιών σε πολύπλοκα δυναμικά συστήματα.

4.1 Δεδομένα προβλήματος

Για την λύση του προβλήματος που εξετάζει η παρούσα εργασία είναι απαραίτητα κάποια δεδομένα τα οποία θεωρείται ότι παρέχονται από το κατανεμημένο σύστημα.

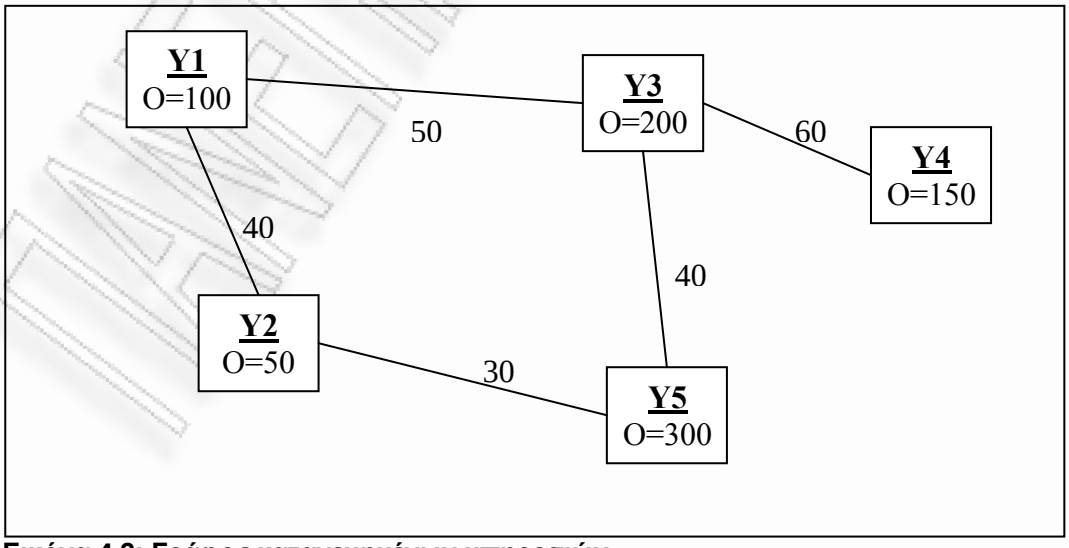
Αυτά είναι:

- Ο γράφος με τους κόμβους του κατανεμημένου συστήματος και τις συνδέσεις τους.
- Το κόστος επικοινωνίας μεταξύ των κόμβων του δικτύου.
- Η υπολογιστική ισχύς των επεξεργαστών των κόμβων, μετρούμενη σε KBps.
- Ο γράφος με τις υπηρεσίες που πρόκειται να αναπτυχθούν και τις αλληλεξαρτήσεις τους.
- Ο όγκος επεξεργασίας της κάθε υπηρεσίας που αναπτύσσεται, μετρούμενος σε Kbytes.



Εικόνα 4.1: Γράφος δικτύου του κατανεμημένου συστήματος

Στην εικόνα 4.1 δίνεται το παράδειγμα ενός δικτύου με εφτά κόμβους (K1 ως K7), την επεξεργαστική ισχύ τους (I), καθώς και το κόστος επικοινωνίας μεταξύ τους. Στην εικόνα 4.2 δίνεται το παράδειγμα μερικών κατανεμημένων υπηρεσιών (Y1 ως Y5), των αλληλεξαρτήσεών τους, του όγκου επεξεργασίας (O) που παράγουν και του κόστους επικοινωνίας μεταξύ τους.



Εικόνα 4.2: Γράφος κατανεμημένων υπηρεσιών

4.2 Επιλογή αλγορίθμου βελτιστοποίησης

Η βάση του συστήματος είναι ο αλγόριθμος βελτιστοποίησης που θα χρησιμοποιηθεί για την λύση του εξεταζόμενου προβλήματος. Στο κεφάλαιο 2 έγινε μία παρουσίαση των αλγόριθμων βελτιστοποίησης που υπάρχουν στην βιβλιογραφία, ενώ δόθηκε έμφαση στους εξελικτικούς αλγόριθμους (evolutionary algorithms), οι οποίοι σύμφωνα με την βιβλιογραφία [5], [7], [12], [13] είναι οι πλέον κατάλληλοι για την επίλυση πολύπλοκων επιστημονικών προβλημάτων. Στο κεφάλαιο 3 παρουσιάστηκαν εργασίες που εντοπίστηκαν στην επιστημονική βιβλιογραφία και κάνουν χρήση εξελικτικών αλγορίθμων για να αντιμετωπίσουν προβλήματα παρεμφερή με το πρόβλημα που εξετάζει η παρούσα εργασία. Ακολουθεί το σκεπτικό επιλογής του καταλληλότερου αλγορίθμου βελτιστοποίησης με βάση τις πληροφορίες που παρουσιάστηκαν στα δύο αυτά κεφάλαια, ενώ γίνεται εμφανές από την βιβλιογραφία, ότι ο αλγόριθμος αυτός θα πρέπει να επιλεγεί από την κατηγορία των εξελικτικών αλγορίθμων.

Ο αλγόριθμος **έρευνας με στοχαστική διασπορά** (Stochastic Diffusion Search) μπορεί να εφαρμοστεί για να λύσει το παρόν πρόβλημα, αλλά το γεγονός ότι ο κάθε πράκτορας του αλγορίθμου ψάχνει εντελώς τυχαία για τη λύση και δεν συσχετίζει την προηγούμενη με την επόμενη προσπάθεια, καθιστά τον αλγόριθμο αναποτελεσματικό σε σχέση με τους υπόλοιπους αλγορίθμους της κατηγορίας που αξιοποιούν περισσότερες διαθέσιμες πληροφορίες για την επίλυση του προβλήματος.

Ο αλγόριθμος **βελτιστοποίησης με αποικία μυρμηγκιών** (Ant Colony Optimization) είναι βιώσιμη λύση για το πρόβλημα που εξετάζεται, όπως καταδεικνύεται και στην σχετική εργασία [24] που περιγράφηκε στο προηγούμενο κεφάλαιο, αλλά όπως φαίνεται στα αποτελέσματα της συγκεκριμένης εργασίας παρουσιάζει προβλήματα όταν προκύπτουν συγκεκριμένες καταστάσεις κατά την εκτέλεση του αλγορίθμου. Γι’ αυτό το λόγο απορρίπτεται και αυτός ως πιθανή λύση.

Ο **γενετικός αλγόριθμος** (genetic algorithm) καθώς και ο αλγόριθμος **βελτιστοποίησης με σμήνος σωματιδίων** (particle swarm optimization) είναι αμφότεροι ικανοί να διαχειριστούν το εξεταζόμενο πρόβλημα, όπως φαίνεται και από την σχετική βιβλιογραφία που εξετάστηκε [25], [26]. Όπως υποστηρίζεται όμως στο [26], ο αλγόριθμος βελτιστοποίησης με σμήνος σωματιδίων είναι αποτελεσματικότερος σε σχέση με τον γενετικό αλγόριθμο. Αυτό επιβεβαιώνεται και στη βιβλιογραφία καθώς υπάρχουν μελέτες [27], [28], [29] που υποστηρίζουν ότι το μαθηματικό μοντέλο του αλγορίθμου βελτιστοποίησης με σμήνος σωματιδίων είναι εξ' ίσου αποδοτικό και πολλές φορές αποδοτικότερο από άλλους σχετικούς αλγορίθμους. Συγκεκριμένα στο [29], η Rania Hassan κ.α. κάνουν μία ενδελεχή στατιστική εξέταση των δύο αλγορίθμων με χρήση οκτώ δύσκολων προβλημάτων βελτιστοποίησης, έξι προερχόμενων από τον τομέα των μαθηματικών και δύο "πραγματικά" επιστημονικά προβλήματα. Καταλήγουν στο συμπέρασμα ότι και οι δύο αλγόριθμοι είναι αρκετά ικανοί στην εύρεση της βέλτιστης λύσης με επίπεδο εμπιστοσύνης 99%, αλλά ο αλγόριθμος βελτιστοποίησης με σμήνος σωματιδίων παρουσιάζει καλύτερα αποτελέσματα σε αρκετά από τα τεστ, καθώς και είναι γρηγορότερος στο σύνολο αυτών. Τα αποτελέσματα αυτά συνάγουν με εκείνα της αντίστοιχης σχετικής εργασίας που παρουσιάστηκε στο προηγούμενο κεφάλαιο [26].

Για τους παραπάνω λόγους θα επιλεχθεί ο αλγόριθμος βελτιστοποίησης με σμήνος σωματιδίων (Particle Swarm Optimization), ο οποίος θα αποκαλείται εφεξής **PSO**, ως ο πλέον κατάλληλος αλγόριθμος για την λύση του προβλήματος που εξετάζει η παρούσα εργασία. Συγκεκριμένα θα χρησιμοποιηθεί ο αλγόριθμος PSO Type 1" με καθολική τοπολογία, έτσι ώστε να διασφαλίζεται ότι θα υπάρχει σύγκλιση των σωματιδίων, ενώ παράλληλα σε κάθε βήμα του αλγορίθμου θα ενημερώνονται όλα τα σωματίδια που χρησιμοποιούνται για την νέα γνώση που αποκτήθηκε.

4.2 Συνάρτηση κόστους

Για να επιτευχθεί ο στόχος του εξεταζόμενου προβλήματος πρέπει να ελαχιστοποιηθεί το κόστος που παράγουν στο σύστημα οι υπηρεσίες που αναπτύσσονται. Για τους σκοπούς της παρούσας εργασίας το κόστος που παράγουν οι υπηρεσίες αποτελείται από δύο επί μέρους κόστη τα οποία θα πρέπει να ελαχιστοποιηθούν, το κόστος επικοινωνίας (communication cost) μεταξύ των κατανεμημένων υπηρεσιών και το υπολογιστικό κόστος (processing cost) που προσθέτουν οι υπηρεσίες στον κόμβο του δικτύου που αναπτύσσονται.

Στη σχετική με το πρόβλημα βιβλιογραφία, που εξετάστηκε στο προηγούμενο κεφάλαιο, ακολουθούνται διάφορες προσεγγίσεις στο θέμα του κόστους που παράγουν οι υπηρεσίες.

Στην εργασία που εξετάζει την βέλτιστη ανάπτυξη υπηρεσιών λειτουργικού συστήματος σε ασύρματα δίκτυα αισθητήρων [24] το κόστος επεξεργασίας ελαχιστοποιείται με την χρήση αλγορίθμου κατάτμησης (clustering), ο οποίος έχει στόχο την ελαχιστοποίηση της χρήσης πόρων του συστήματος. Αφού ελαχιστοποιηθεί το κόστος επικοινωνίας, τότε ελαχιστοποιείται το κόστος επικοινωνίας των υπηρεσιών με χρήση του αλγορίθμου βελτιστοποίησης με αποικία μυρμηγκιών.

Στην εργασία που κάνει ανάπτυξη υπηρεσιών με χρήση του γενετικού αλγόριθμου [25], πρώτα συντίθενται οι υπηρεσίες από διάφορα επί μέρους συστατικά, έτσι ώστε να βελτιστοποιηθεί η ποιότητα που παρέχουν. Μέσα στο κόστος ποιότητας εμπεριέχεται και το κόστος επεξεργασίας της υπηρεσίας, ενώ τίθεται ο περιορισμός ότι μόνο μία σύνθετη υπηρεσία μπορεί να αναπτυχθεί σε κάθε κόμβο. Αφού βελτιστοποιηθεί η σύνθεση της κάθε υπηρεσίας με χρήση του γενετικού αλγορίθμου, έπειτα χρησιμοποιείται εκ νέου ο γενετικός αλγόριθμος για να ελαχιστοποιηθεί το κόστος επικοινωνίας κατά την ανάπτυξη των υπηρεσιών.

Τέλος, στην εργασία που αναθέτει εργασίες σε κατανεμημένο σύστημα με χρήση της βελτιστοποίησης με σμήνος σωματιδίων [26], το

εξεταζόμενο σύστημα είναι ομογενές και ο κάθε επεξεργαστής μπορεί να έχει μόνο μία εργασία οπότε το κόστος επεξεργασίας είναι ίδιο για κάθε επεξεργαστή του συστήματος και άρα δεν λαμβάνεται υπ ‘όψιν.

Οι δύο πρώτες εργασίες [24], [25] βελτιστοποιούν τα δύο διαφορετικά κόστη κάνοντας χρήση του λεγόμενου “pareto optimum”. Η τεχνική αυτή είναι απαραίτητη όταν υπάρχουν προς βελτιστοποίηση διαφορετικά μεγέθη στο ίδιο πρόβλημα [5] [30]. Η τεχνική αυτή λειτουργεί βελτιστοποιώντας τη λύση πρώτα ως προς το ένα κόστος και έπειτα χρησιμοποιεί τα αποτελέσματα της βελτιστοποίησης αυτής για να κάνει βελτιστοποίηση της λύσης ως προς ένα άλλο κόστος. Η δεύτερη βελτιστοποίηση μπορεί μόνο να βελτιώσει την υπάρχουσα λύση που έχει προκύψει και δεν μπορεί να οδηγήσει σε χειρότερη λύση από αυτή που επιτεύχθηκε με την πρώτη βελτιστοποίηση. Μπορούν να υπάρξουν και περεταίρω βελτιστοποιήσεις, ως προς άλλα κόστη, έπειτα από την δεύτερη. Η μέθοδος αυτή δεν μπορεί να εγγυηθεί την εύρεση μία βέλτιστης (ή σχεδόν βέλτιστης) λύσης, καθώς η σειρά με την οποία επιλέγονται τα κόστη που θα βελτιστοποιηθούν επηρεάζει άμεσα το αποτέλεσμα της μεθόδου [5] [30]. Έτσι η λύση που επιτυγχάνεται είναι μία εν δυνάμει “βέλτιστη” λύση μεταξύ πολλών εναλλακτικών.

Στην λύση που προτείνεται με την παρούσα εργασία, δεν θα ακολουθηθεί η τεχνική του “pareto optimum”, αλλά τα κόστη που εντοπίζονται (επικοινωνίας και της επεξεργασίας), θα αναχθούν στην ίδια μονάδα μέτρησης, ώστε να ομογενοποιηθούν και να είναι διαχειρίσιμα ως ένα ενιαίο αθροιστικό κόστος. Η μονάδα αυτή θα είναι ο χρόνος. Επίσης θα υπάρχει η δυνατότητα ανάθεσης περισσότερων από μία υπηρεσιών σε ένα επεξεργαστή, εάν αυτό κρίνεται σκόπιμο από τον αλγόριθμο.

Με βάση τα παραπάνω η συνάρτηση κόστους του προβλήματος διαμορφώνεται ως εξής:

$$F = \max(NC_{n,i}) + \left(\sum_{\forall LC_{i,j}} LC_{i,j} \cdot I_{LC} \right) / 2$$

όπου NCn,i είναι το κόστος επεξεργασίας που προκύπτει από την ανάθεση της υπηρεσίας *i* στον κόμβο *n*, για *i* = (0,…,*k*) και $LC_{i,j}$ είναι το κόστος επικοινωνίας μεταξύ δύο υπηρεσιών *i, j*. Η συνάρτηση I_{LC} είναι μία συνάρτηση δείκτη, που παίρνει τιμές:

$$I_{LC} = \begin{cases} 0, & (i,n) = (j,n) \\ 1, & (i,n) \neq (j,n) \end{cases}$$

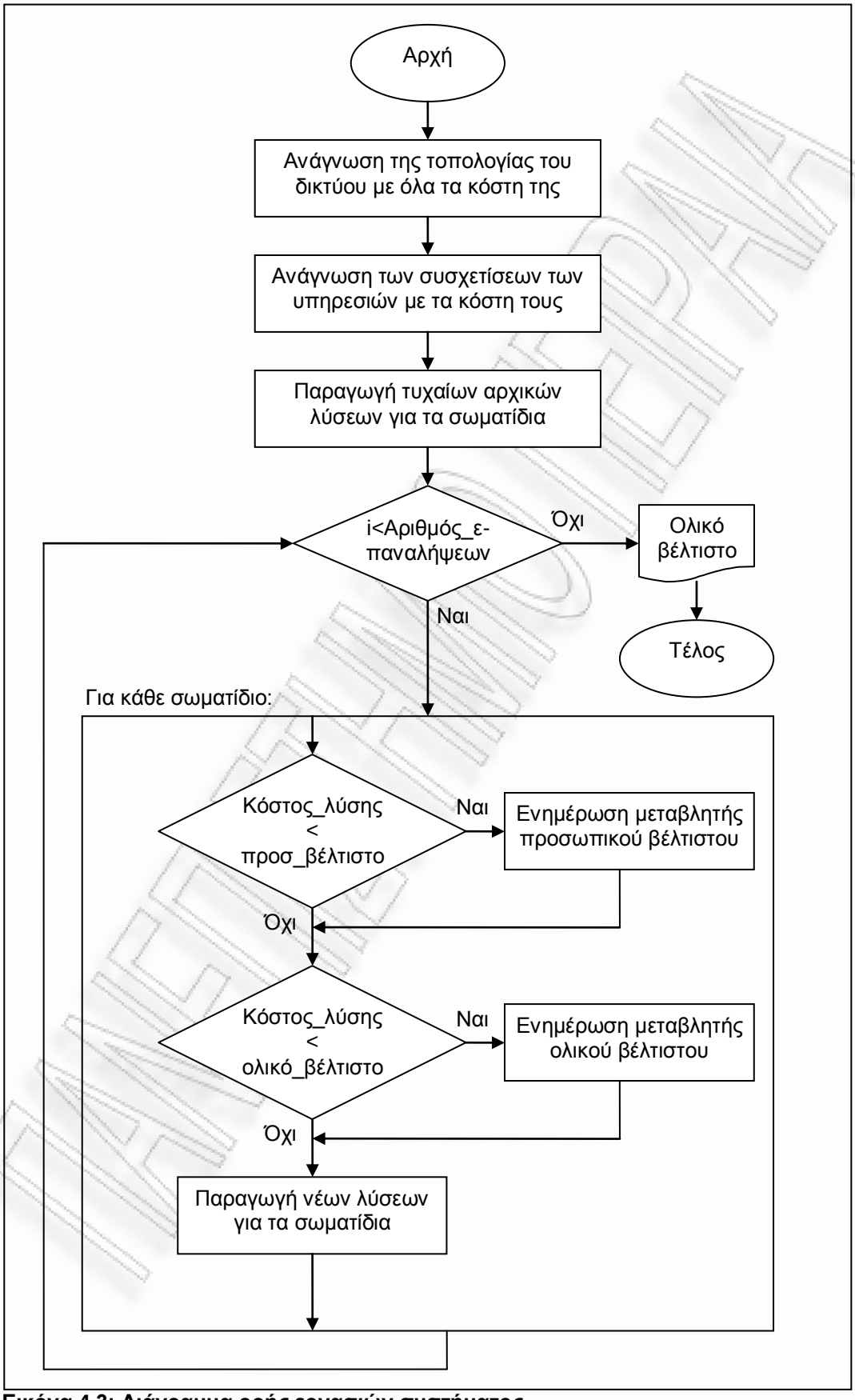
δηλαδή παίρνει την τιμή 0, όταν οι υπηρεσίες *i, j* που επικοινωνούν έχουν ανατεθεί στον ίδιο κόμβο *n*, ενώ παίρνει την τιμή 1 όταν οι υπηρεσίες αυτές έχουν ανατεθεί σε διαφορετικούς κόμβους.

Η διαίρεση του αθροίσματος του κόστους επικοινωνίας δια 2 γίνεται ώστε να μην υπολογίζονται εις διπλούν τα κόστη επικοινωνίας μεταξύ των υπηρεσιών καθώς το κόστος (*i, j*) είναι ίδιο με το κόστος (*j, i*).

Σκοπός λοιπόν του αλγόριθμου PSO είναι να ελαχιστοποιήσει την συνάρτηση *F*, που αντιπροσωπεύει τον συνολικό χρόνο επεξεργασίας και επικοινωνίας στο κατανεμημένο σύστημα.

4.4 Διάγραμμα ροής εργασιών

Στην ενότητα αυτή θα παρουσιαστεί το διάγραμμα ροής εργασιών του συστήματος που πρόκειται να αναπτυχθεί για να διαχειρίζεται την ανάπτυξη των υπηρεσιών σε κατανεμημένα συστήματα.. Το σύστημα αυτό θα ενεργοποιείται κάθε φορά που ανιχνεύεται αλλαγή στην τοπολογία του δικτύου, με την πρόσθεση ή κατάργηση κόμβου του δικτύου, ή την μεταβολή της ποιότητας των συνδέσμων επικοινωνίας του δικτύου πέρα από κάποιο κατώφλι. Η εικόνα 4.3 δείχνει διάγραμμα ροής εργασιών του προτεινόμενου συστήματος:



Εικόνα 4.3: Διάγραμμα ροής εργασιών συστήματος

Κεφάλαιο 5: Υλοποίηση λύσης

Στο παρόν κεφάλαιο γίνεται ανάλυση της υλοποίησης της λύσης που προτάθηκε στο προηγούμενο κεφάλαιο. Η λύση αποτελεί μία προσομοίωση του συστήματος σε προγραμματιστική γλώσσα Java, που σκοπό έχει να καταδείξει τη βιωσιμότητα της προτεινόμενης λύσης.

Το πρόγραμμα εστιάζει στην υλοποίηση του συστήματος ανάπτυξης των υπηρεσιών στους κόμβους του κατανεμημένου συστήματος και δέχεται ως δεδομένα από αρχείο, όλες τις πληροφορίες που αφορούν το δίκτυο και τις υπηρεσίες που πρόκειται να αναπτυχθούν, θεωρώντας ότι αυτές παρέχονται από το ίδιο το σύστημα. Ο χρήστης μπορεί να αλλάξει τα δεδομένα αυτά με πρόσβαση απευθείας στα αρχεία που τα περιέχουν, όπως θα αναλυθεί διεξοδικότερα στη συνέχεια του κεφαλαίου. Επίσης από την οθόνη του προγράμματος ο χρήστης έχει τη δυνατότητα να αλλάζει όλες τις παραμέτρους που αφορούν την λειτουργία του αλγορίθμου PSO.

Στην ενότητα 5.1 θα γίνει μία επισκόπηση των μεθόδων που αποτελούν το πρόγραμμα προσομοίωσης καθώς και του τρόπου λειτουργίας τους, ενώ στην ενότητα 5.2 θα παρουσιαστεί η οθόνη λειτουργίας του προγράμματος καθώς και μερικές ενδεικτικές οθόνες αποτελεσμάτων. Τέλος στην εικόνα 5.3 θα παρουσιαστεί η δομή των πληροφοριών στα αρχεία δεδομένων που χρησιμοποιεί ως είσοδο το πρόγραμμα.

5.1 Επισκόπηση των μεθόδων του προγράμματος προσομοίωσης

Στην παρούσα ενότητα θα γίνει μια περιληπτική περιγραφή του τρόπου λειτουργίας των μεθόδων του προγράμματος με παρουσίαση των δεδομένων που χρειάζονται ως είσοδο, της λειτουργίας που επιτελούν και των αποτελεσμάτων που επιστρέφουν, με σκοπό να γίνει κατανοητός ο τρόπος λειτουργίας των επί μέρους συστατικών που απαρτίζουν το

πρόγραμμα προσομοίωσης. Η παρουσίαση θα γίνει με την σειρά που απαντώνται οι μέθοδοι μέσα στον κώδικα κατά την εκτέλεση του προγράμματος. Ο πλήρης κώδικας του προγράμματος προσομοίωσης βρίσκεται στο παράρτημα 1.

- **gui():** Η πρώτη μέθοδος που καλεί το πρόγραμμα κατά την έναρξή του, έχει σκοπό την δημιουργία του γραφικού περιβάλλοντος του προγράμματος προσομοίωσης. Δημιουργεί το παράθυρο της εφαρμογής, τα πεδία εισόδου δεδομένων τα οποία αρχικοποιούνται με ενδεικτικές τιμές τις οποίες μπορεί να αλλάξει ο χρήστης, δημιουργεί την περιοχή σχεδίασης όπου θα σχεδιαστεί η λύση που παράγει το πρόγραμμα και τέλος δημιουργεί το κουμπί έναρξης του αλγορίθμου βελτιστοποίησης με τίτλο “Execute”.
- **actionPerformed():** Η μέθοδος αυτή καλείται κάθε φορά που ο χρήστης κάνει χρήση του κουμπιού “Execute”. Η μέθοδος αυτή αρχικά ελέγχει αν οι τιμές που έχει συμπληρώσει ο χρήστης στο γραφικό περιβάλλον της εφαρμογής είναι αριθμητικές και αν εμπίπτουν στους περιορισμούς που υπάρχουν (όπως όχι αρνητικές τιμές). Αν οι τιμές που εισήγαγε ο χρήστης δεν είναι αποδεκτές παρουσιάζεται μήνυμα σφάλματος στο χρήστη. Σε διαφορετική περίπτωση η μέθοδος καλεί διαδοχικά τις μεθόδους **readNetworkFile()**, **runDijkstra()**, **readComponentFile()**, **pso()** και **graphPanel.draw()** οι οποίες θα παρουσιαστούν στην συνέχεια.
- **readNetworkFile():** Η μέθοδος αυτή εντοπίζει στον κατάλογο που είναι αποθηκευμένο το πρόγραμμα προσομοίωσης το αρχείο με όνομα “network.xsl” και διαβάζει από αυτό των αριθμό των κόμβων του δικτύου του συστήματος, την επεξεργαστική ισχύ τους, τους συνδέσμους μεταξύ τους και το κόστος επικοινωνίας μέσω αυτών των συνδέσμων.
- **runDijkstra():** Η μέθοδος αυτή χρησιμοποιείται επαναληπτικά ώστε να αποκτηθούν από το πρόγραμμα τα ελάχιστα κόστη επικοινωνίας από κάθε κόμβο του δικτύου προς κάθε άλλο κόμβο. Δέχεται κάθε φορά ως όρισμα έναν κόμβο του δικτύου και επιστρέφει το ελάχιστο

κόστος επικοινωνίας με όλους τους υπόλοιπους κόμβους, ακόμα και όταν δεν υπάρχει απ' ευθείας σύνδεση. Ουσιαστικά η μέθοδος είναι μια υλοποίηση του γνωστού αλγόριθμου δρομολόγησης του Dijkstra [31], για δίκτυα υπολογιστών.

- **readComponentFile():** Η μέθοδος αυτή εντοπίζει στον κατάλογο που είναι αποθηκευμένο το πρόγραμμα προσομοίωσης το αρχείο με όνομα "components.xsl" και διαβάζει από αυτό των αριθμό των υπηρεσιών που πρόκειται να αναπτυχθούν, τον όγκο επεξεργασίας της κάθε μίας καθώς και τις αλληλεξαρτήσεις τους, στις οποίες έχει τεθεί προγραμματιστικός περιορισμός και δεν μπορούν να υπερβαίνουν τις δέκα.
- **pso():** Η συγκεκριμένη μέθοδος εσωκλείει όλες τις λειτουργίες που αφορούν την εκτέλεση του αλγορίθμου PSO. Αρχικά δημιουργεί τόσα αντικείμενα psoAgent(), όσα και η τιμή που έχει δώσει ο χρήστης. Τα αντικείμενα αυτά αποτελούν τα σωματίδια (ή πράκτορες) του αλγορίθμου και στην δομή τους αποθηκεύουν ζωτικές πληροφορίες για την εκτέλεση του αλγορίθμου όπως το διάνυσμα της τρέχουσας λύσης που υπολογίζει το καθένα σε κάθε βήμα του αλγορίθμου, το διάνυσμα της βέλτιστης ατομικής λύσης που έχει υπολογίσει το καθένα καθώς και το κόστους που παράγουν αυτές οι λύσεις (τρέχουσα και καλύτερη προσωπική). Αφού δημιουργηθούν τα σωματίδια, αρχικοποιούνται με ένα τυχαίο διάνυσμα θέσης μέσα στο δίκτυο (που ισοδυναμεί με μια τυχαία ανάθεση των υπηρεσιών στους κόμβους του δικτύου), μία τυχαία αρχική ταχύτητα για κάθε σωματίδιο, που δείχνει την τάση του να αλλάξει την θέση της κάθε ανάθεσης, καθώς και ένα πολύ μεγάλο αριθμό ως "καλύτερη" προσωπική λύση.

Έπειτα αρχίζει η επαναληπτική λειτουργία του αλγορίθμου PSO, όπου σε κάθε βήμα του αλγορίθμου και για κάθε σωματίδιο γίνονται τα ακόλουθα: α) Υπολογισμός του μέγιστου χρόνου επεξεργασίας για τρέχον διάνυσμα θέσεων.

β) Υπολογισμός του συνολικού κόστους επικοινωνίας για το τρέχον διάνυσμα θέσεων.

γ) Άθροισμα των δύο χρόνων που υπολογίστηκαν.

δ) Ενημέρωση της καλύτερης προσωπικής λύσης του κάθε πράκτορα σε περίπτωση που η τρέχουσα λύση είναι καλύτερη από αυτή που έχει αποθηκευμένη.

ε) Ενημέρωση της ολικά καλύτερης λύσης του αλγορίθμου σε περίπτωση που η τρέχουσα λύση είναι καλύτερη από οποιαδήποτε άλλη λύση έχει βρεθεί ως τώρα.

στ) Ορισμός νέων ταχυτήτων για τα σωματίδια του αλγορίθμου.

ζ) Ορισμός νέων θέσεων που προκύπτουν από τις νέες ταχύτητες.

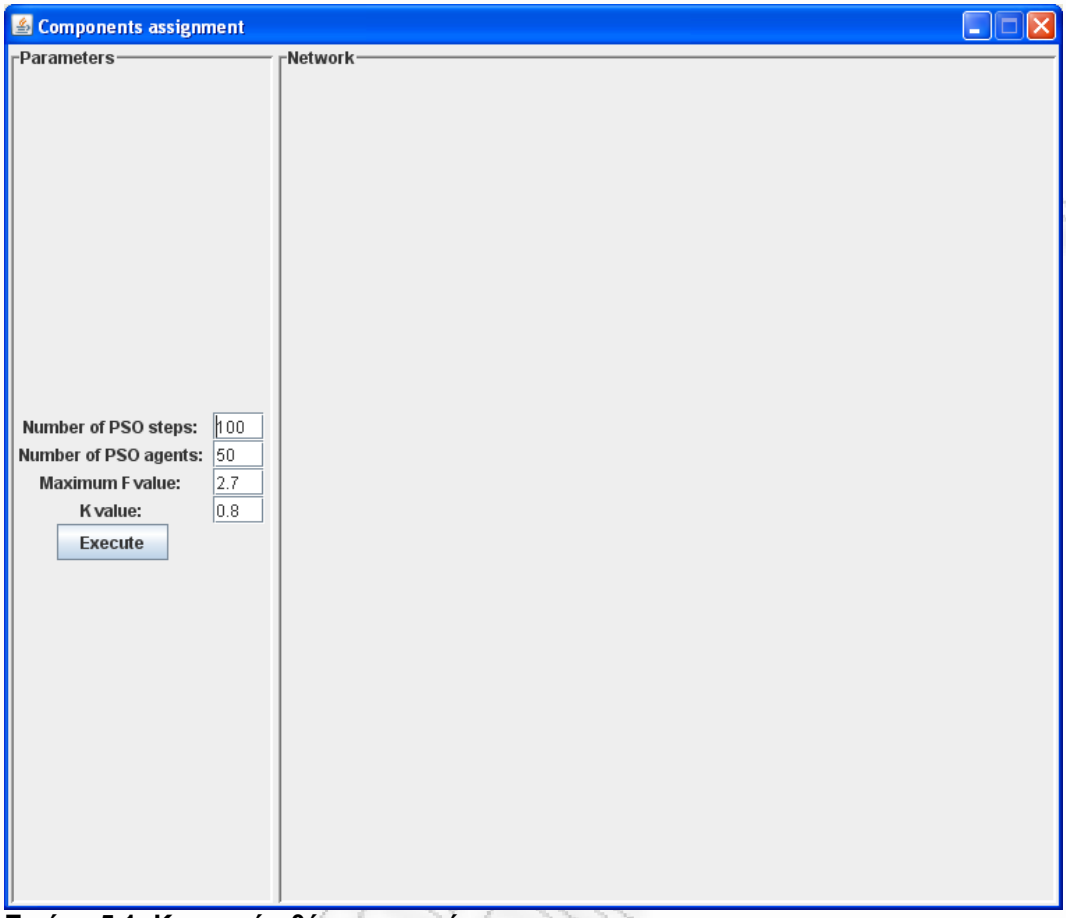
graphPanel.draw(): Η τελευταία μέθοδος που καλεί το πρόγραμμα, έχει σκοπό την σχεδίαση της λύσης του αλγορίθμου στο παράθυρο της εφαρμογής. Χρησιμοποιεί τα διαθέσιμα δεδομένα που αφορούν την δομή του δικτύου και το διάνυσμα της βέλτιστης λύσης που υπολογίζει ο αλγόριθμος PSO και σχεδιάζει στην οθόνη τον γράφο του δικτύου με τα κόστη επικοινωνίας, καθώς και τον αριθμό της κάθε υπηρεσίας δίπλα στον κόμβο τον οποίο έχει ανατεθεί.

5.2 Γραφικό περιβάλλον προγράμματος προσομοίωσης

Στην ενότητα αυτή θα παρουσιαστούν οι οθόνες χρήστη του προγράμματος που αναπτύχθηκε.

Στην εικόνα 5.1 παρουσιάζεται η κεντρική οθόνη του προγράμματος πριν την χρησιμοποίηση του. Στο αριστερό μέρος ο χρήστης μπορεί να καθορίσει τις τιμές των διαφόρων παραμέτρων του προγράμματος:

- **Number of PSO steps:** Ο αριθμός των βημάτων του PSO
- **Number of PSO agents:** Ο αριθμός των σωματιδίων του PSO
- **Maximum F value:** Η μεγίστη τιμή που θα μπορεί να πάρει η παράμετρος τυχαιότητας F
- **K value:** Η τιμή της παραμέτρου σύγκλισης K, η οποία θα πρέπει να κυμαίνεται στο διάστημα (0,1].

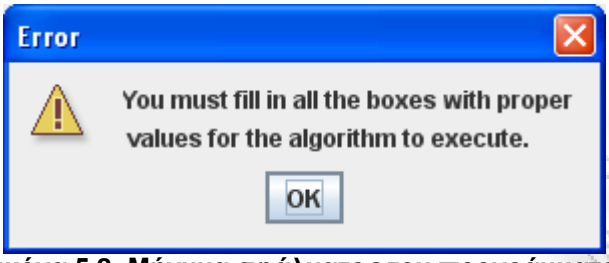


Εικόνα 5.1: Κεντρική οθόνη προγράμματος

Κάτω από τα κουτάκια με τις τιμές των παραμέτρων βρίσκεται το κουμπί “Execute”, με την χρήση του οποίου εκτελείται ο αλγόριθμος βελτιστοποίησης PSO και επιστρέφονται στην οθόνη τα αποτελέσματα.

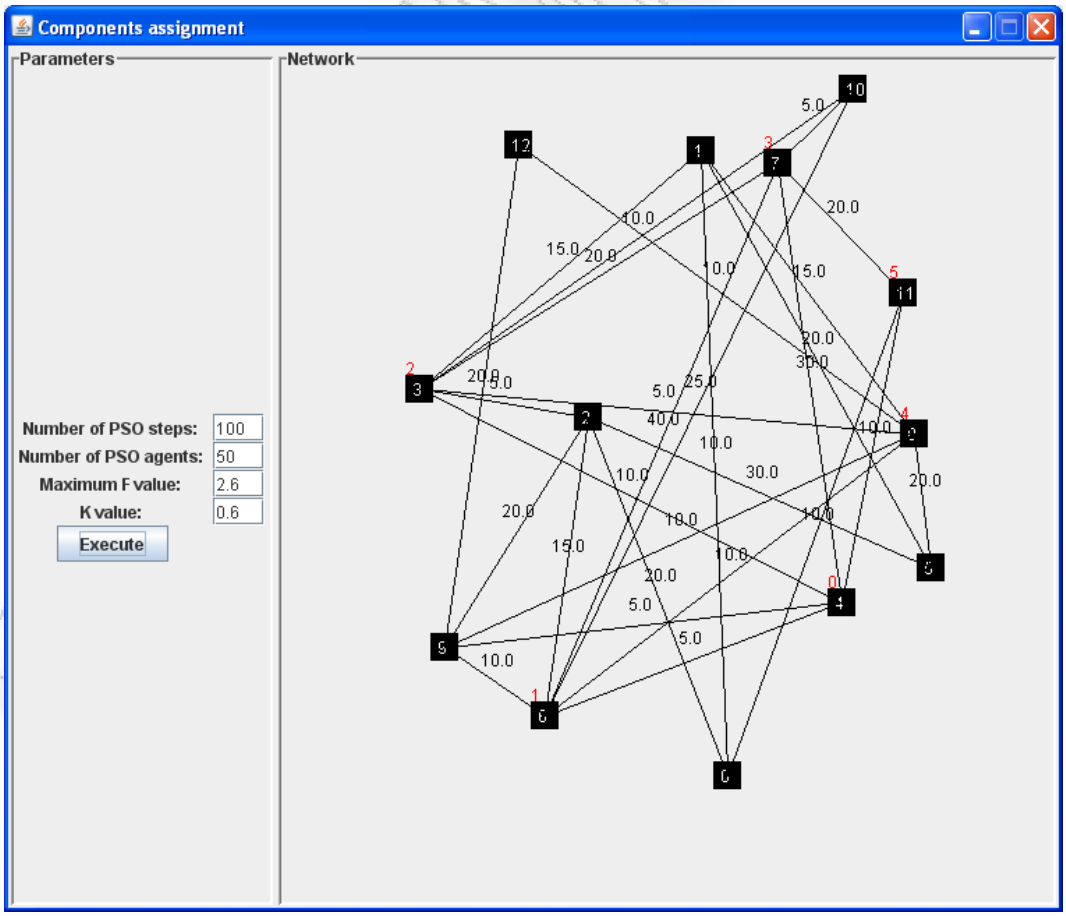
Στην δεξιά πλευρά της οθόνης χρήστη βρίσκεται η περιοχή όπου παρουσιάζονται τα αποτελέσματα της προσομοίωσης.

Σε περίπτωση που ο χρήστης δεν εισάγει αποδεκτά για το πρόγραμμα δεδομένα, όπως αρνητικές τιμές για τις μεταβλητές, είτε τιμές εκτός του διαστήματος (0,1] για την παράμετρο K, τότε στην οθόνη εμφανίζεται μήνυμα σφάλματος που ειδοποιεί τον χρήστη να εισάγει αποδεκτές τιμές. Η εικόνα 5.2 δείχνει ένα παράδειγμα με μήνυμα σφάλματος που προκλήθηκε από την εισαγωγή τιμής μεγαλύτερης της μονάδας για την παράμετρο K.



Εικόνα 5.2: Μήνυμα σφάλματος του προγράμματος

Η εικόνα 5.3 δείχνει την λύση που προέκυψε έπειτα από το πάτημα του κουμπιού “Execute” για ένα δίκτυο δεκατριών κόμβων και έξι υπηρεσιών. Το πρόγραμμα σχεδίασε σε τυχαία θέση στην οθόνη τους δεκατρείς κόμβους του δικτύου (0-12), τις μεταξύ τους συνδέσεις και το κόστος τους, και επίσης πρόσθεσε με κόκκινο χρώμα τον αριθμό της κάθε υπηρεσίας (0-5) πάνω από τον κόμβο στον οποίο ανατέθηκε, σύμφωνα με την βέλτιστη λύση που καθόρισε ο αλγόριθμος PSO.

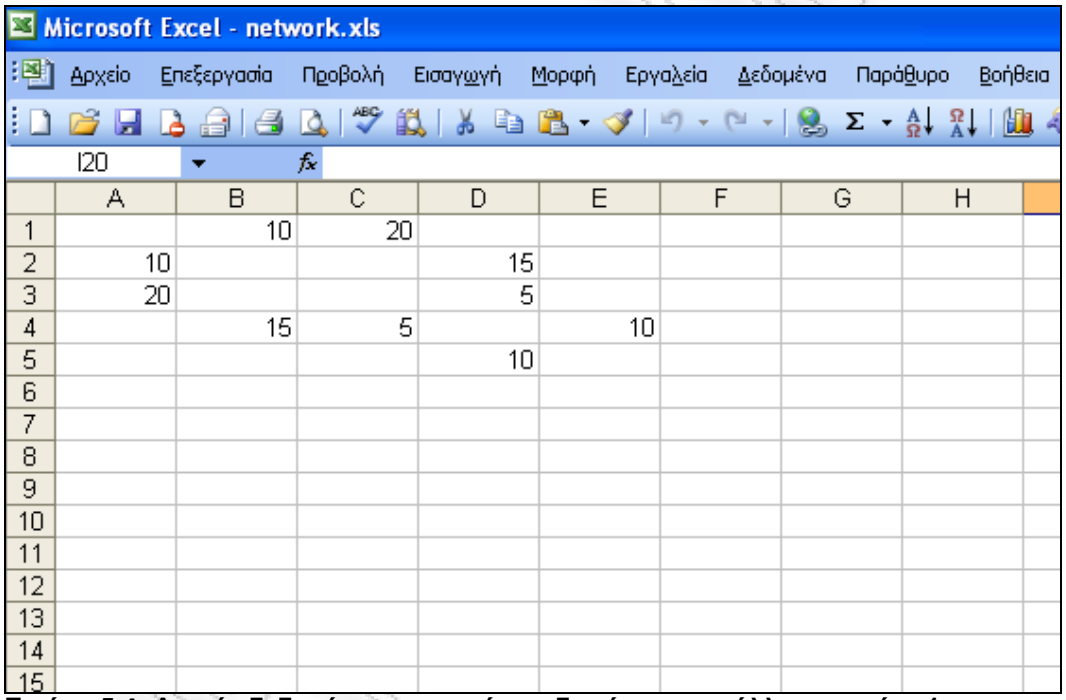


Εικόνα 5.3: Κεντρική οθόνη προγράμματος μετά την εκτέλεση του αλγορίθμου PSO

5.3 Δομή αρχείων δεδομένων

Ο χρήστης του προγράμματος έχει την δυνατότητα να αλλάξει όλες τις παραμέτρους του γράφου του δικτύου, καθώς και του γράφου υπηρεσιών με άμεση πρόσβαση στα αρχεία excel στα οποία είναι αποθηκευμένες οι δομές των δύο γράφων. Στη συνέχεια θα περιγραφεί ο τρόπος με τον οποίο είναι δομημένα τα αρχεία των δύο γράφων.

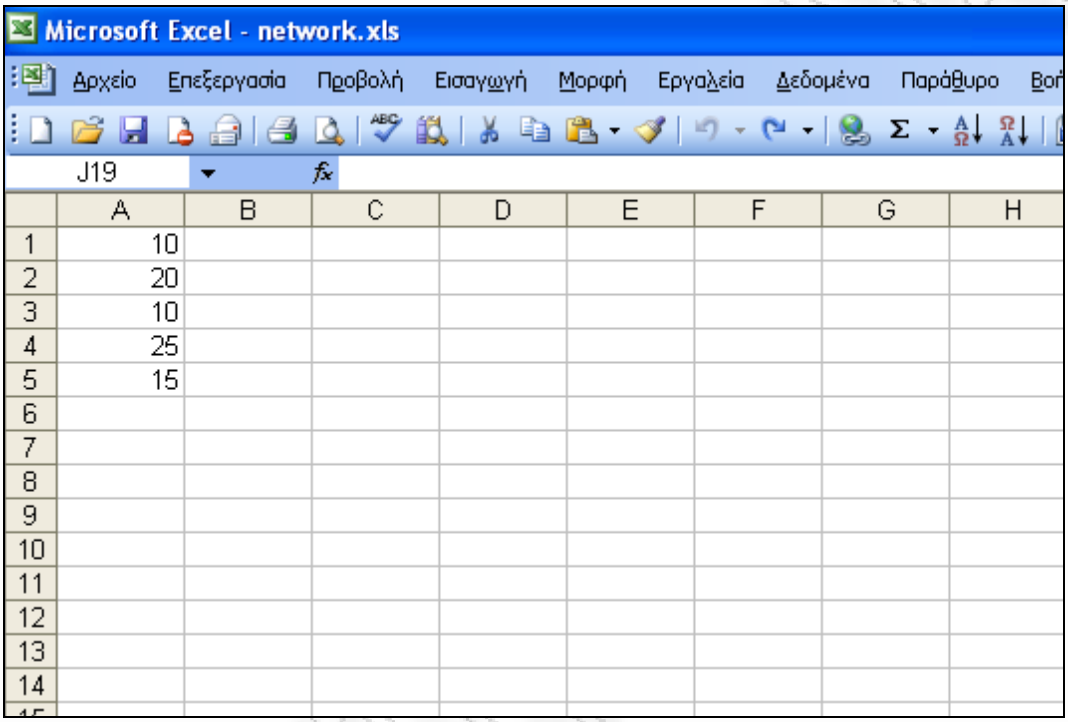
Στην εικόνα 5.4 φαίνεται το αρχείο δεδομένων του γράφου του δικτύου με όνομα “network.xls”, ανοιγμένο στο φύλο εργασίας 1 (Sheet1).



Εικόνα 5.4: Αρχείο δεδομένων του γράφου δικτύου στο φύλλο εργασίας 1

Κάθε γραμμή του αρχείου καθώς και κάθε στήλη αναπαριστά και έναν κόμβο του δικτύου. Οι τιμές στα κελιά αναπαριστούν το κόστος επικοινωνίας μεταξύ των συγκεκριμένων δύο κόμβων, ενώ τα άδεια κελιά υποδηλώνουν ότι δεν υπάρχει απευθείας δίαυλος επικοινωνίας μεταξύ των κόμβων. Όπως γίνεται ορατό ο πίνακας είναι συμμετρικός ως προς την διαγώνιό του, καθώς οι τιμές επαναλαμβάνονται για τα ζευγάρια επικοινωνούντων κόμβων. Αυτό συμβαίνει ώστε να είναι πιο εύκολη η διαχείριση της πληροφορίας προγραμματιστικά και ο χρήστης θα πρέπει

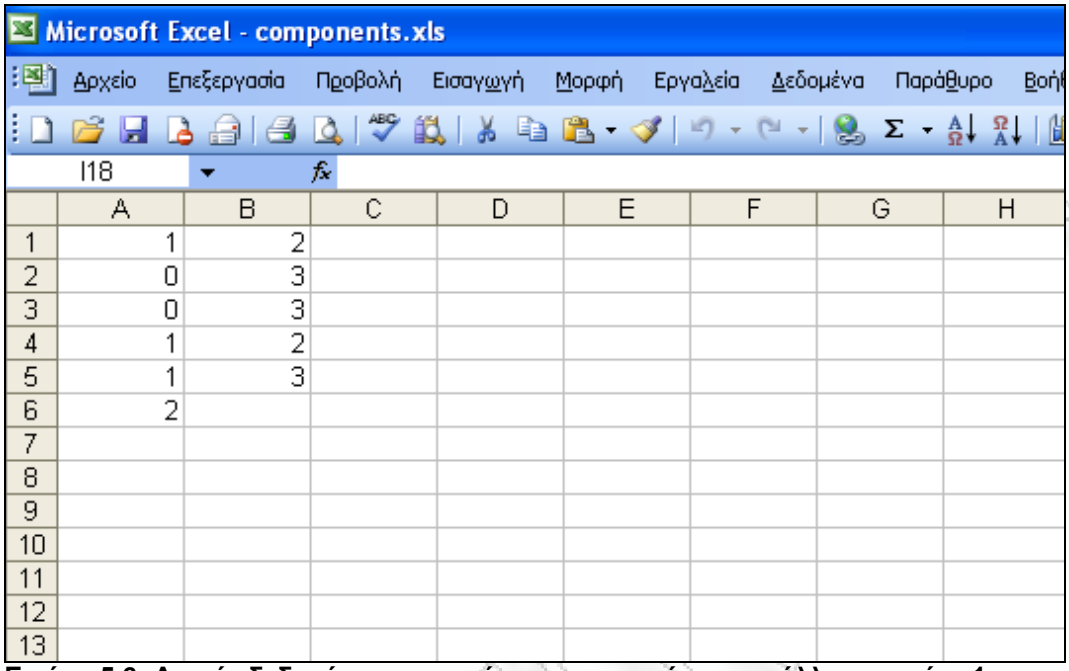
να είναι προσεκτικός κατά την αλλαγή των δεδομένων έτσι ώστε ο πίνακας να είναι πάντοτε συμμετρικός ως προς την διαγώνιό του, διαφορετικά ο αλγόριθμος PSO δεν θα παράξει σωστά αποτελέσματα.



Εικόνα 5.5: Αρχείο δεδομένων του γράφου δικτύου στο φύλλο εργασίας 2

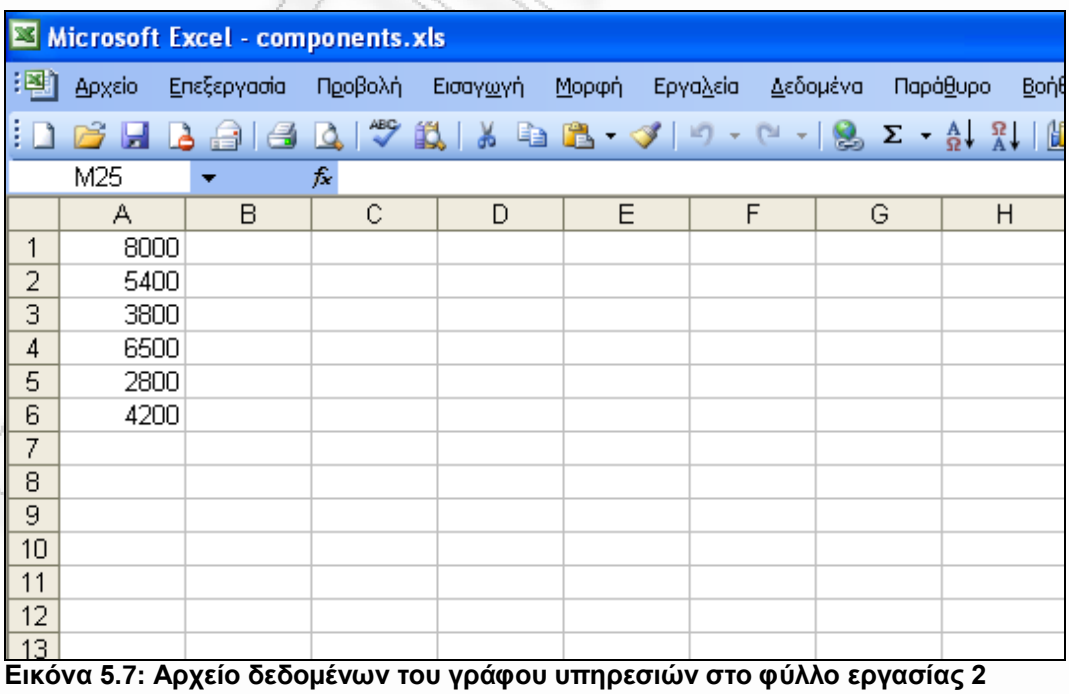
Η εικόνα 5.5 δείχνει το φύλο εργασίας 2 (Sheet2) του αρχείου δεδομένων “network.xls”. Σε αυτό το φύλο εργασίας είναι αποθηκευμένη η επεξεργαστική ισχύς του καθενός από τους πέντε κόμβους που απαρτίζουν το δίκτυο.

Στην εικόνα 5.6 φαίνεται το αρχείο δεδομένων του γράφου των υπηρεσιών με όνομα “components.xls”, ανοιγμένο στο φύλο εργασίας 1 (Sheet1). Η δομή αυτού του αρχείου είναι διαφορετική από το προηγούμενο. Στο αρχείο αυτό κάθε γραμμή αποτελεί μία υπηρεσία ενώ κάθε στήλη δείχνει με ποια άλλη υπηρεσία αλληλεπιδρά η συγκεκριμένη υπηρεσία (υπενθυμίζεται ότι ο μέγιστος αριθμός αλληλεπιδράσεων έχει τεθεί σε δέκα). Επειδή προγραμματιστικά η αρίθμηση ξεκινάει από το μηδέν, εξυπηρετεί προγραμματιστικά η κάθε καταχώρηση σε κελί είναι η τιμή της υπηρεσίας μείον ένα, οπότε πρέπει να δοθεί η δέουσα προσοχή κατά την επεξεργασία των συγκεκριμένων δεδομένων από τον χρήστη.



Εικόνα 5.6: Αρχείο δεδομένων του γράφου υπηρεσιών στο φύλλο εργασίας 1

Η εικόνα 5.7 δείχνει το φύλο εργασίας 2 (Sheet2) του αρχείου υπηρεσιών “components.xls”. Κάθε καταχώρηση σε κελί απεικονίζει το υπολογιστικό φόρτο που επιφέρει η αντίστοιχη υπηρεσία.



Εικόνα 5.7: Αρχείο δεδομένων του γράφου υπηρεσιών στο φύλλο εργασίας 2

Κεφάλαιο 6: Εκτίμηση απόδοσης λύσης

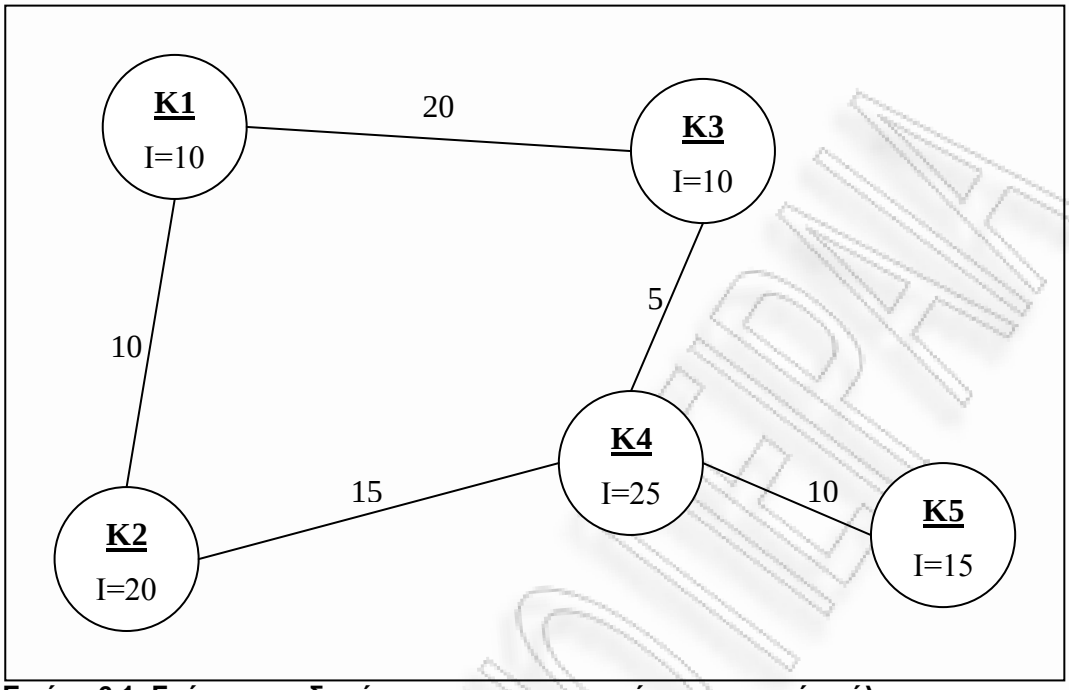
Στο παρόν κεφάλαιο θα γίνει εκτίμηση της αποδοτικότητας της λύσης που σχεδιάστηκε και αναπτύχθηκε για την επίλυση του προβλήματος της ανάπτυξης υπηρεσιών σε δυναμικά συστήματα.

Η λύση που αναπτύχθηκε θα ελεγχθεί στατιστικά ως προς την κάθε παράμετρο που επηρεάζει τον αλγόριθμο βελτιστοποίησης σε ένα μέγεθος δείγματος διακοσίων εκτελέσεων του αλγορίθμου για κάθε σετ παραμέτρων που θα δοκιμαστεί. Σκοπός είναι να μετρηθεί πως επιδρά η κάθε παράμετρος στην αποδοτικότητα του αλγορίθμου, καθώς και να καθοριστεί πια είναι η απόδοση του αλγορίθμου για το βέλτιστο σετ παραμέτρων.

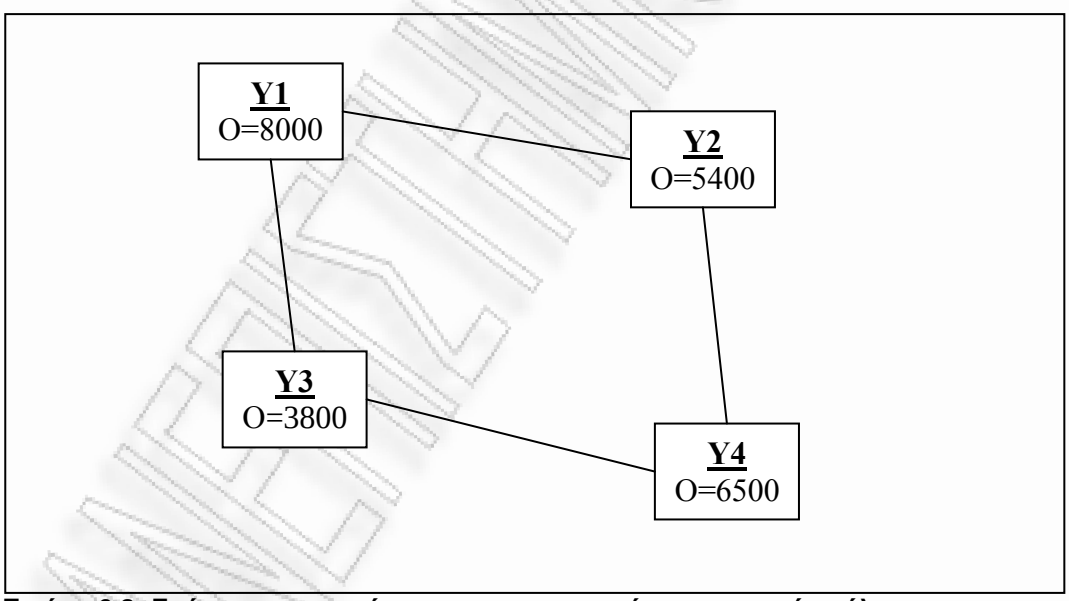
Ο έλεγχος θα γίνει πρώτα σε ένα απλό δίκτυο πέντε κόμβων, όπου θα αναπτυχθούν τέσσερις υπηρεσίες ώστε να ελεγχθεί αρχικά η βιωσιμότητα της λύσης. Έπειτα θα γίνει έλεγχος σε ένα πιο πολύπλοκο δίκτυο όπου θα αναπτυχθούν πάλι τέσσερις υπηρεσίες αλλά σε δεκατρείς κόμβους, ώστε να μελετηθεί η συμπεριφορά του αλγορίθμου σε πιο πολύπλοκα δίκτυα. Τέλος θα γίνει ένας τρίτος έλεγχος, όπου θα αναπτυχθούν έξι υπηρεσίες -με περισσότερες μεταξύ τους εξαρτήσεις- σε δίκτυο δεκατριών κόμβων, ώστε να μελετηθεί η συμπεριφορά του PSO σε ακόμα πιο πολύπλοκες καταστάσεις.

6.1 Έλεγχος αποδοτικότητας σε δίκτυο πέντε κόμβων και τεσσάρων υπηρεσιών

Ο συγκεκριμένος έλεγχος αποδοτικότητας θα εφαρμοστεί στο γράφο του δικτύου με τα χαρακτηριστικά που φαίνονται στην εικόνα 6.1 και για τον γράφο των υπηρεσιών με τα χαρακτηριστικά που φαίνονται στην εικόνα 6.2.



Εικόνα 6.1: Γράφος του δικτύου που χρησιμοποιείται στον πρώτο έλεγχο



Εικόνα 6.2: Γράφος υπηρεσιών που χρησιμοποιείται στον πρώτο έλεγχο

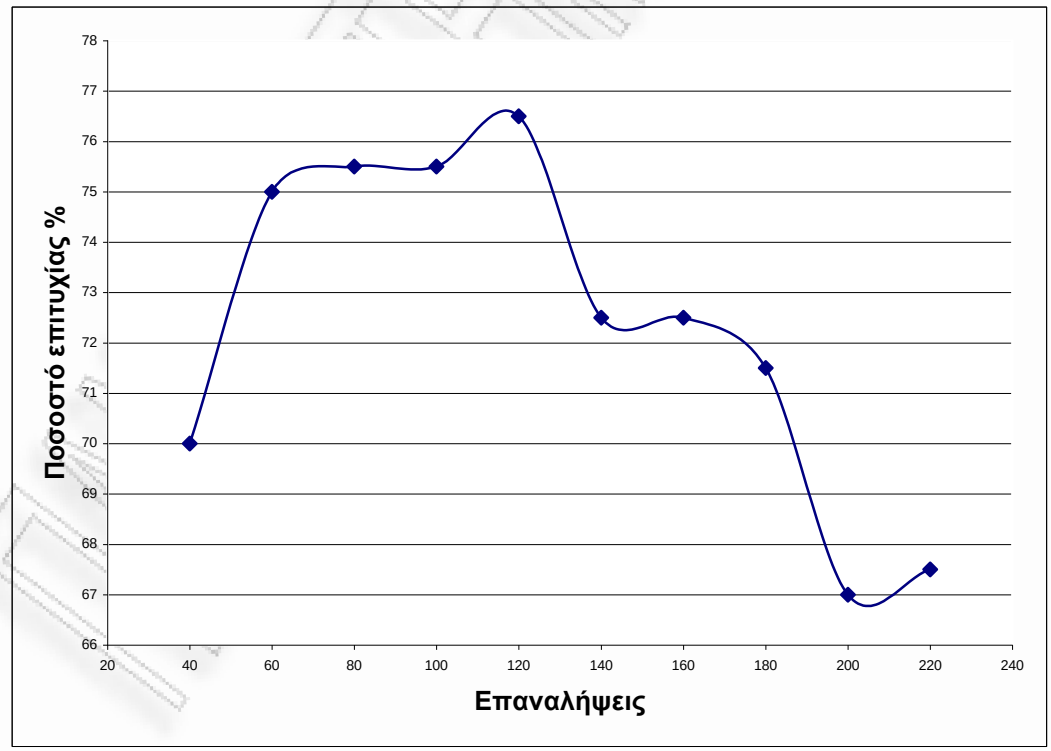
Για να μετρηθεί η αποδοτικότητα του αλγορίθμου θα μετρηθεί πόσες φορές εντοπίστηκε η βέλτιστη λύση (optimum solution), η οποία είναι γνωστή εκ των πρότερων, καθώς και πόσες φορές εντοπίστηκε η αμέσως επόμενη καλύτερη λύση ή σχεδόν βέλτιστη (near-optimum solution), στο σύνολο διακοσίων εκτελέσεων του αλγορίθμου με κάθε σετ δεδομένων και θα εκφραστεί ως ποσοστό επί τις εκατό.

Στο πρώτο σετ δοκιμών θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.1 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος του αριθμού επαναλήψεων του αλγορίθμου από 40 ως 220 με βήμα 20.

Αριθμός βημάτων PSO	Μεταβάλλεται
Αριθμός σωματιδίων PSO	50
Μέγιστη τιμή τυχαιότητας F	2,7
Τιμή παραμέτρου K	0,8

Πίνακας 6.1: Τιμές μεταβλητών του πρώτου σετ δοκιμών (πρώτος ελεγχος)

Στην εικόνα 6.3 φαίνονται τα στατιστικά αποτελέσματα του πρώτου σετ δοκιμών. Όπως παρατηρείται η απόδοση του αλγορίθμου μεγιστοποιείται στις 120 επαναλήψεις, ενώ η απόδοση μειώνεται απότομα όσο αυξάνονται οι επαναλήψεις.



Εικόνα 6.3: Ποσοστό επιτυχίας του PSO κατά την μεταβολή των επαναλήψεων του αλγορίθμου (πρώτος έλεγχος)

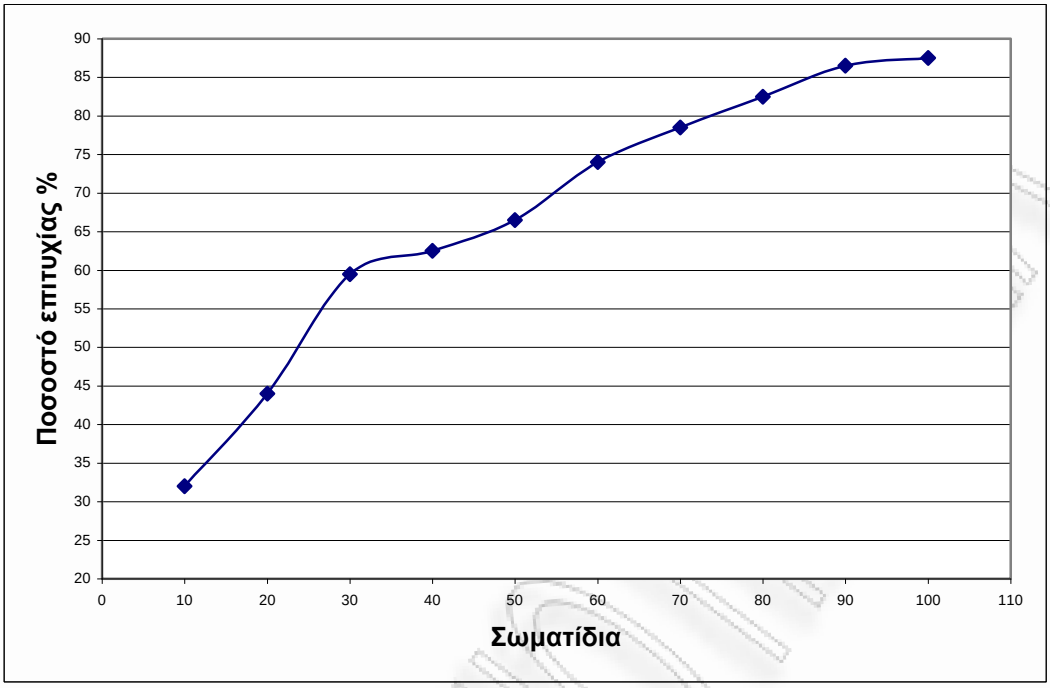
Η μείωση της απόδοσης του αλγορίθμου συμβαίνει επειδή με την αύξηση των επαναλήψεων επιτρέπεται στα σωματίδια να εξερευνούν για περισσότερο χρόνο το πεδίο τιμών αυτόνομα, αργώντας περισσότερο να συγκλίνουν μεταξύ τους, λόγο και της αργής σύγκλισης k=0,9 που εφαρμόζεται στο σύνολο των επαναλήψεων. Αυτό έχει ως αποτέλεσμα τα σωματίδια να χάνουν την δυναμική τους (ταχύτητα) πριν προλάβουν να συγκλίνουν προς την βέλτιστη λύση, και να εγκλωβίζονται συχνότερα σε τοπικά μέγιστα, τα οποία έπειτα δεν μπορούν να ξεπεράσουν παρά τον μεγαλύτερο αριθμό επαναλήψεων του αλγορίθμου, αφού δεν έχουν πλέον αρκετή δυναμική.

Στο δεύτερο σετ δοκιμών θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.2 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος του αριθμού των σωματιδίων του αλγορίθμου από 40 ως 220 με βήμα 20.

Αριθμός βημάτων PSO	100
Αριθμός σωματιδίων PSO	Μεταβάλλεται
Μέγιστη τιμή τυχαιότητας F	2,7
Τιμή παραμέτρου K	0,8

Πίνακας 6.2: Τιμές μεταβλητών του δεύτερου σετ δοκιμών (πρώτος έλεγχος)

Στην εικόνα 6.4 φαίνονται τα στατιστικά αποτελέσματα του δεύτερου σετ δοκιμών. Όπως παρατηρείται η απόδοση του αλγορίθμου αυξάνεται συνεχώς όσο αυξάνονται και τα σωματίδια του αλγορίθμου που αλληλεπιδρούν μεταξύ τους.



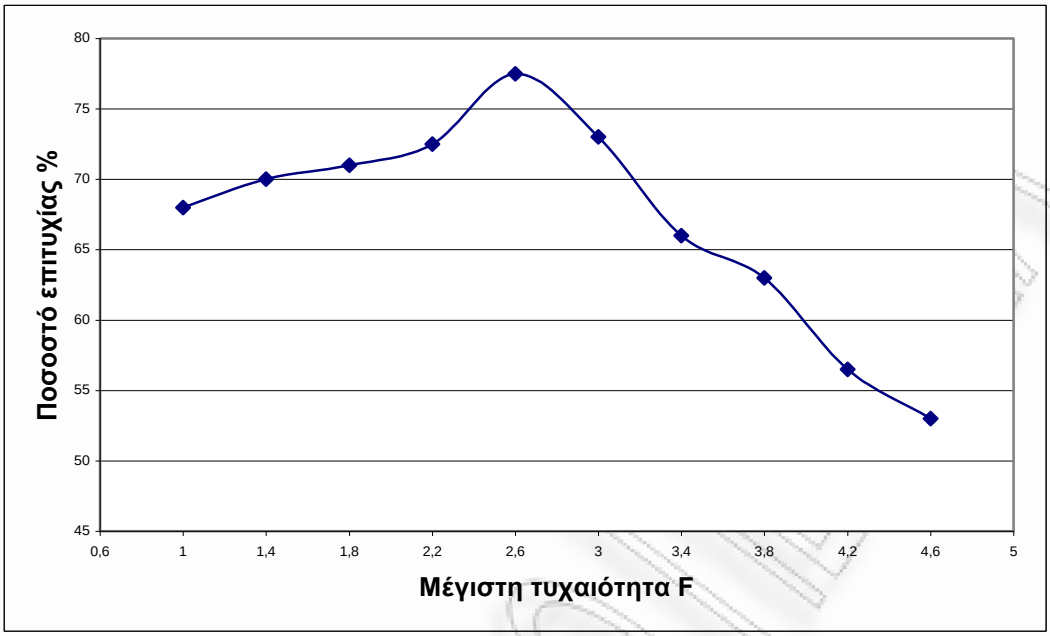
Εικόνα 6.4: Ποσοστό επιτυχίας του PSO κατά την μεταβολή του αριθμού των σωματιδίων του (πρώτος έλεγχος)

Στο τρίτο σετ δοκιμών θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.3 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος της μέγιστης τυχαιότητας που μπορεί να εισαχθεί στην κίνηση των σωματιδίων του αλγορίθμου από 1 ως 4,6 με βήμα 0,4.

Αριθμός βημάτων PSO	100
Αριθμός σωματιδίων PSO	50
Μέγιστη τιμή τυχαιότητας F	Μεταβάλλεται
Τιμή παραμέτρου K	0,8

Πίνακας 6.3: Τιμές μεταβλητών του τρίτου σετ δοκιμών (πρώτος έλεγχος)

Στην εικόνα 6.5 φαίνονται τα στατιστικά αποτελέσματα του τρίτου σετ δοκιμών. Όπως παρατηρείται η απόδοση του αλγορίθμου μεγιστοποιείται για την τιμή μέγιστης τυχαιότητας 2.6, ενώ φθίνει ραγδαία για τιμές μεγαλύτερες του 3.



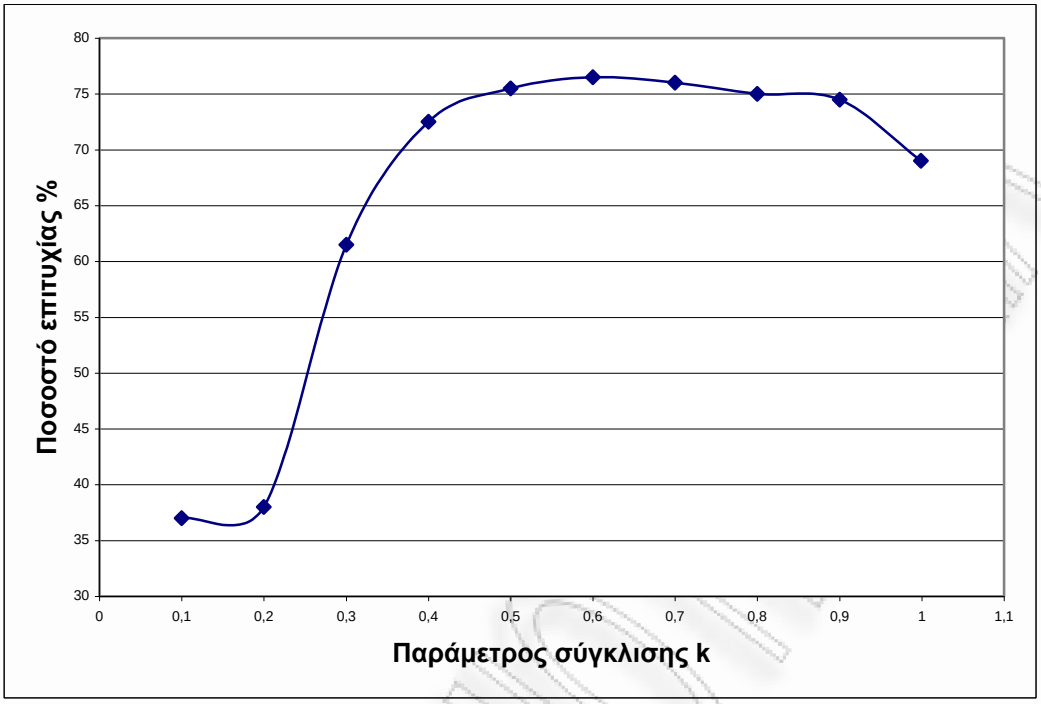
Εικόνα 6.5: Ποσοστό επιτυχίας του PSO κατά την μεταβολή της τυχαιότητας F (πρώτος έλεγχος)

Στο τέταρτο σετ δοκιμών θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.4 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος k που αφορά το πόσο γρήγορα συγκλίνουν μεταξύ τους τα σωματίδια του αλγορίθμου PSO, από 0.1 ως 0.999 με βήμα 0.1.

Αριθμός βημάτων PSO	100
Αριθμός σωματιδίων PSO	50
Μέγιστη τιμή τυχαιότητας F	2,7
Τιμή παραμέτρου K	Μεταβάλλεται

Πίνακας 6.4: Τιμές μεταβλητών του τέταρτου σετ δοκιμών (πρώτος έλεγχος)

Στην εικόνα 6.6 φαίνονται τα στατιστικά αποτελέσματα του τέταρτου σετ δοκιμών. Γίνεται εμφανές ότι από ένα σημείο και έπειτα υπάρχει πολύ μικρή διαφοροποίηση της απόδοσης του αλγορίθμου, οποία μεγιστοποιείται για την τιμή k ίση με 0.6.



Εικόνα 6.6: Ποσοστό επιτυχίας του PSO κατά την μεταβολή της παραμέτρου k (πρώτος έλεγχος)

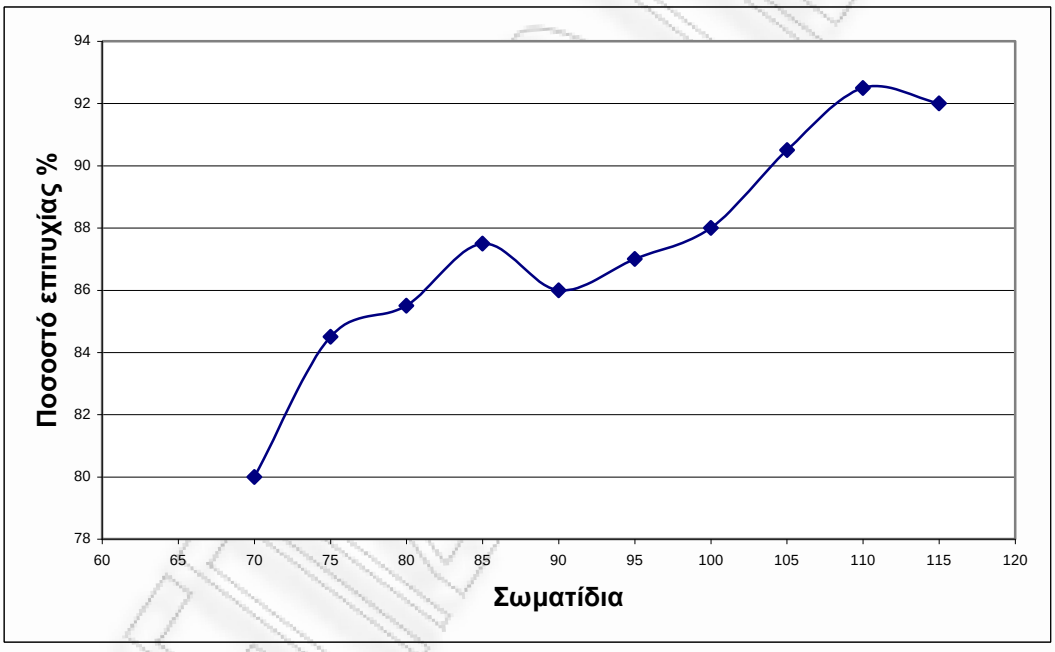
Τα παραπάνω σετ δοκιμών δείχνουν την παραμετροποίηση που χρειάζεται ο αλγόριθμος για να αποδώσει καλύτερα στο συγκεκριμένο πρόβλημα. Όπως γίνεται κατανοητό σημαντικότερη παράμετρος για την απόδοση του αλγορίθμου είναι ο αριθμός των σωματιδίων, ενώ ο αριθμός των επαναλήψεων του αλγορίθμου από ένα σημείο και έπειτα δεν παίζει τόσο σημαντικό ρόλο στην επιτυχία του αλγορίθμου, αλλά αντίθετα επιδρά αρνητικά σε αυτή.

Λαμβάνοντας υπ' όψιν τα παραπάνω αποτελέσματα θα διεξαχθεί ένα πέμπτο σετ δοκιμών όπου οι παράμετροι θα λάβουν τις βέλτιστες τιμές που βρέθηκαν στα προηγούμενα τεστ, ενώ θα μεταβάλλεται και πάλι ο αριθμός των σωματιδίων του αλγορίθμου (αφού είναι η σημαντικότερη παράμετρος επιτυχίας του αλγορίθμου), στο διάστημα 70 ως 110 με βήμα 5. Οι τιμές των βέλτιστων παραμέτρων που θα χρησιμοποιηθούν φαίνονται στον πίνακα 6.5.

Αριθμός βημάτων PSO	120
Αριθμός σωματιδίων PSO	Μεταβάλλεται
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	0,6

Πίνακας 6.5: Τιμές μεταβλητών του τέταρτου σετ δοκιμών (πρώτος έλεγχος)

Στην εικόνα 6.7 φαίνονται τα στατιστικά αποτελέσματα του πέμπτου σετ δοκιμών.



Εικόνα 6.7: Ποσοστό επιτυχίας του PSO κατά την μεταβολή του αριθμού των σωματιδίων, με βέλτιστες τιμές για τις υπόλοιπες παραμέτρους (πρώτος έλεγχος)

Όπως γίνεται εμφανές από την εικόνα 6.7 η απόδοση του αλγορίθμου αυξάνεται συνεχώς, με μικρές αποκλίσεις που οφείλονται σε στατιστικό σφάλμα και φτάνει σε ποσοστά ως και 92.5% για το συγκεκριμένο πρόβλημα. Περισσότερα σωματίδια θα οδηγούσαν πιθανότατα σε μεγαλύτερα ποσοστά επιτυχίας του αλγορίθμου. Μια πιο αναλυτική εξέταση των δεδομένων που χρησιμοποιήθηκαν για το διάγραμμα της εικόνας 6.7 και τα οποία φαίνονται στον πίνακα 6.6, φανερώνει ότι ο αλγόριθμος συνεχώς βελτιώνεται ως προς το ποσοστό

της βέλτιστης λύσης που υπολογίζει. Το ποσοστό εντοπισμού της βέλτιστης λύσης φτάνει το 71.5% για 115 σωματίδια, ενώ η αμέσως επόμενη καλύτερη τιμή της συνάρτησης κόστους εντοπίζεται το 20,5% των περιπτώσεων (υπενθυμίζεται ότι τα αποτελέσματα βγαίνουν στο σύνολο 200 εκτελέσεων του PSO με το κάθε σετ δεδομένων).

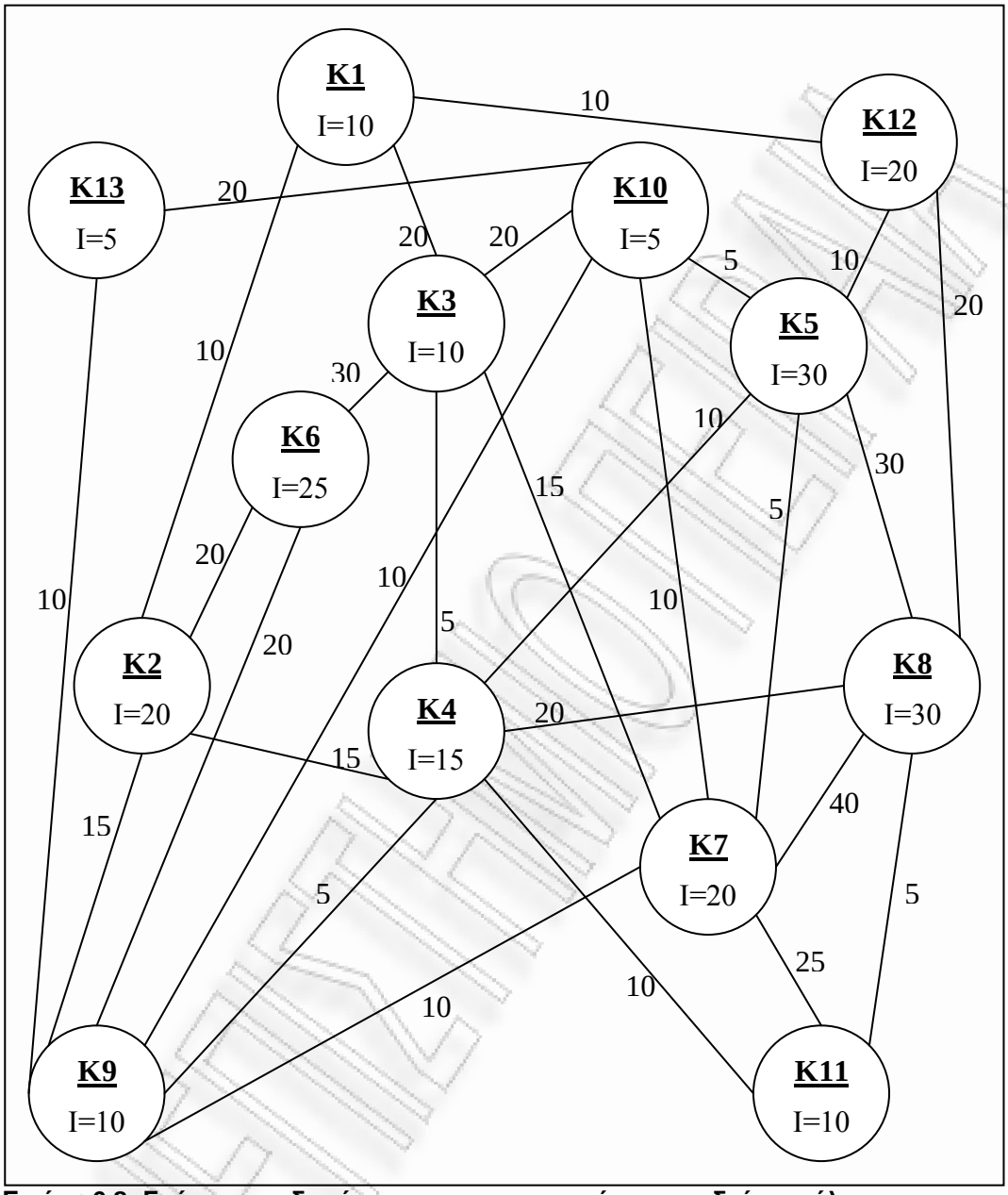
Σωματίδια	Βέλτιστη τιμή	Σχεδόν βέλτιστη τιμή	Συνολικό ποσοστό %
70	113	47	80
75	118	51	84,5
80	118	53	85,5
85	120	55	87,5
90	126	46	86
95	135	39	87
100	133	43	88
105	143	38	90,5
110	138	47	92,5
115	143	41	92

Πίνακας 6.6: Δεδομένα διαγράμματος εικόνας 6.7

Όπως φαίνεται από τα αποτελέσματα ο αλγόριθμος PSO μπορεί να έχει μια πολύ καλή απόδοση στο συγκεκριμένο απλό δίκτυο και να εντοπίζει εύκολα καλές λύσεις για την καλύτερη ανάπτυξη υπηρεσιών σε αυτό, μέσα από το εύρος $6^4 = 1296$ λύσεων.

6.2 Έλεγχος αποδοτικότητας σε δίκτυο δεκατριών κόμβων και τεσσάρων υπηρεσιών

Ο συγκεκριμένος έλεγχος αποδοτικότητας θα εφαρμοστεί στο γράφο του δικτύου με τα χαρακτηριστικά που φαίνονται στην εικόνα 6.8 και για τον γράφο των υπηρεσιών με τα χαρακτηριστικά που φαίνονται στην εικόνα 6.2 (ίδιος με τον πρώτο έλεγχο).



Εικόνα 6.8: Γράφος του δικτύου που χρησιμοποιείται στον δεύτερο έλεγχο

Όπως και με τον προηγούμενο έλεγχο, θα μετρηθεί και εδώ η αποδοτικότητα του αλγορίθμου ως ποσοστό επί τις εκατό, ανάλογα με το πόσες φορές εντοπίστηκε η βέλτιστη λύση (optimum solution, καθώς και πόσες φορές εντοπίστηκε η αμέσως επόμενη καλύτερη λύση ή σχεδόν βέλτιστη (near-optimum solution), στο σύνολο διακοσίων εκτελέσεων του αλγορίθμου με κάθε σετ δεδομένων.

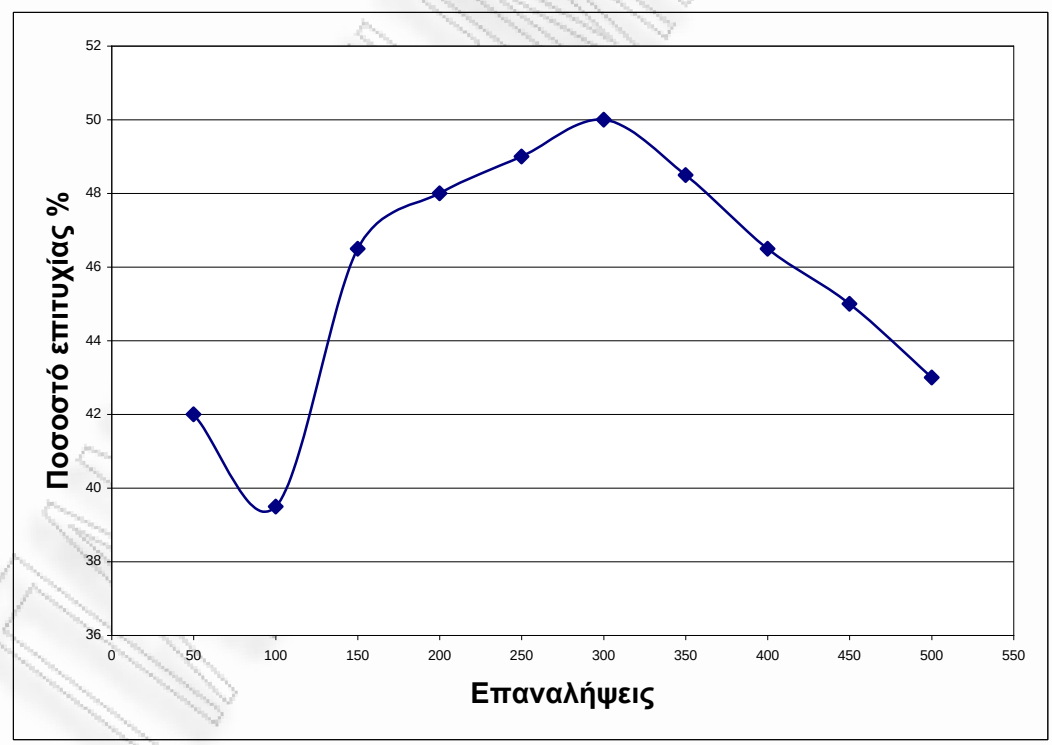
Στο πρώτο σετ δοκιμών για το συγκεκριμένο δίκτυο θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.7 με τις αντίστοιχες

τιμές, ενώ θα μεταβάλλεται η παράμετρος του αριθμού επαναλήψεων του αλγορίθμου από 50 ως 500 με βήμα 50.

Αριθμός βημάτων PSO	Μεταβάλλεται
Αριθμός σωματιδίων PSO	100
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	0,6

Πίνακας 6.7: Τιμές μεταβλητών του πρώτου σετ δοκιμών (δεύτερος έλεγχος)

Στην εικόνα 6.9 φαίνονται τα στατιστικά αποτελέσματα του πρώτου σετ δοκιμών για το δεύτερο δίκτυο. Όπως παρατηρείται η απόδοση του αλγόριθμου μεγιστοποιείται στις 300 επαναλήψεις, ενώ η απόδοση μειώνεται απότομα όσο αυξάνονται οι επαναλήψεις, για τον λόγο που περιγράφτηκε στον αντίστοιχο πρώτο έλεγχο (Εικόνα 6.3).



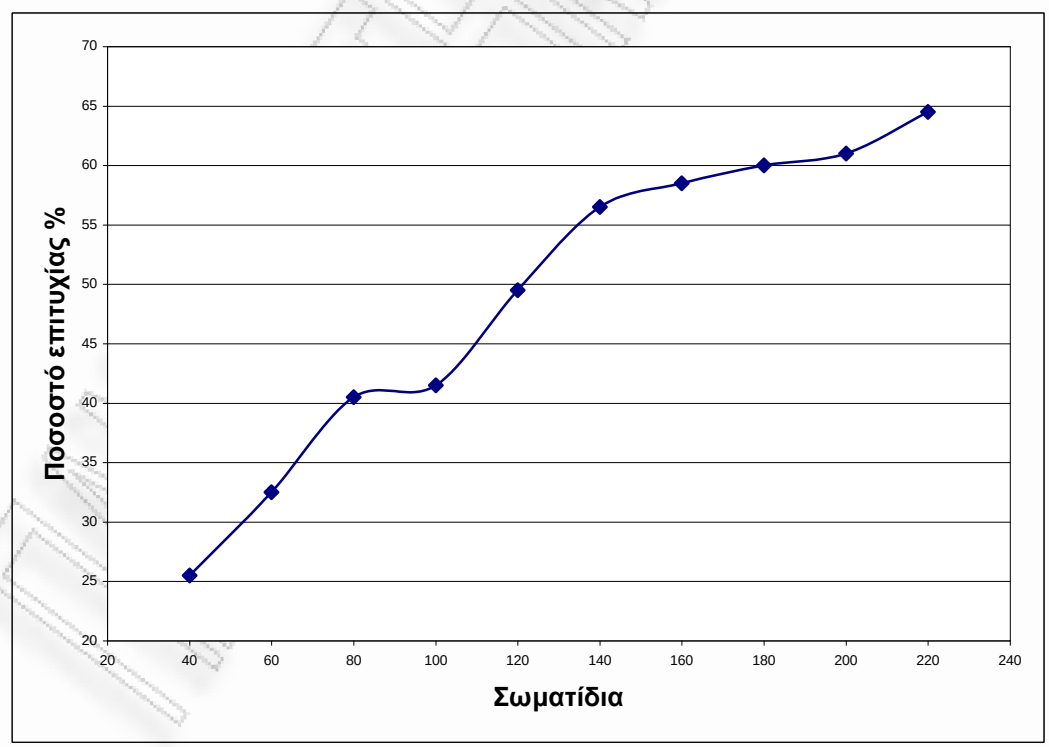
Εικόνα 6.9: Ποσοστό επιτυχίας του PSO κατά την μεταβολή των επαναλήψεων του αλγορίθμου (δεύτερος έλεγχος)

Στο δεύτερο σετ δοκιμών για το δίκτυο της εικόνας 6.8 θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.8 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος του αριθμού των σωματιδίων του αλγορίθμου από 50 ως 500 με βήμα 50.

Αριθμός βημάτων PSO	300
Αριθμός σωματιδίων PSO	Μεταβάλλεται
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	0,6

Πίνακας 6.8: Τιμές μεταβλητών του δεύτερου σετ δοκιμών (δεύτερος έλεγχος)

Στην εικόνα 6.10 φαίνονται τα στατιστικά αποτελέσματα του δεύτερου σετ δοκιμών για το εξεταζόμενο δίκτυο. Όπως παρατηρείται η απόδοση του αλγορίθμου αυξάνεται συνεχώς με την αύξηση του αριθμού των σωματιδίων.



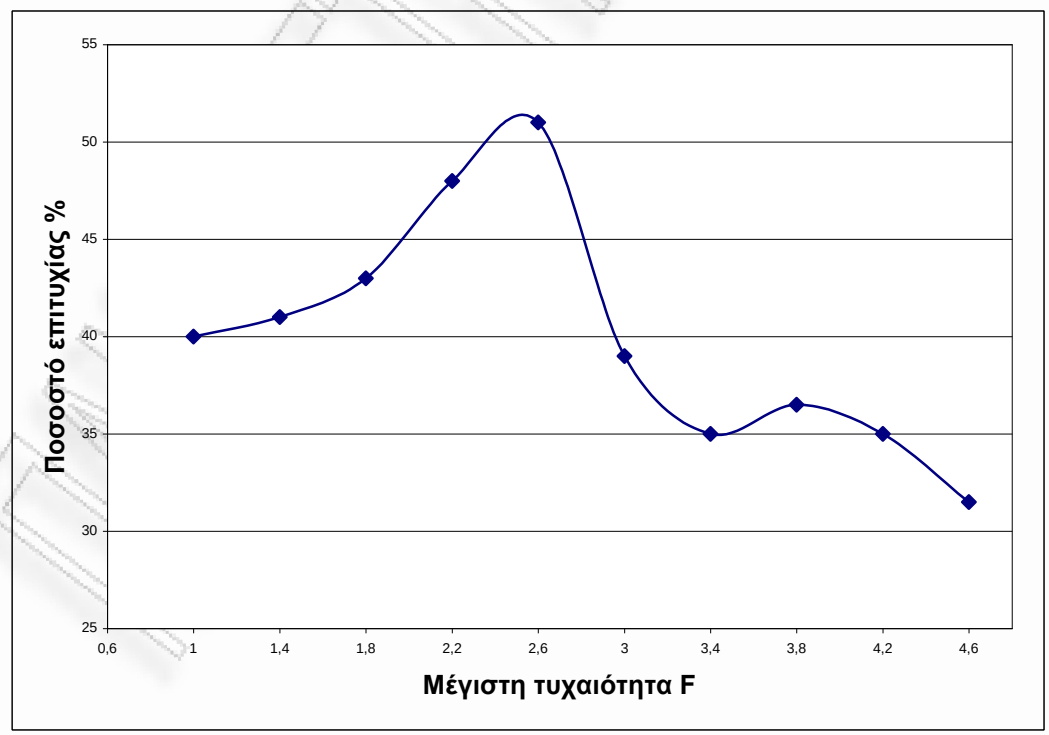
Εικόνα 6.10: Ποσοστό επιτυχίας του PSO κατά την μεταβολή του αριθμού των σωματιδίων του (δεύτερος έλεγχος)

Στο τρίτο σετ δοκιμών για το δίκτυο της εικόνας 6.8 θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.9 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος της μέγιστης τυχαιότητας που μπορεί να εισαχθεί στην κίνηση των σωματιδίων του αλγορίθμου από 1 ως 4,6 με βήμα 0,4.

Αριθμός βημάτων PSO	300
Αριθμός σωματιδίων PSO	100
Μέγιστη τιμή τυχαιότητας F	Μεταβάλλεται
Τιμή παραμέτρου K	0,6

Πίνακας 6.9: Τιμές μεταβλητών του τρίτου σετ δοκιμών (δεύτερος έλεγχος)

Στην εικόνα 6.11 φαίνονται τα στατιστικά αποτελέσματα του τρίτου σετ δοκιμών για τον δεύτερο έλεγχο. Όπως παρατηρείται η απόδοση του αλγορίθμου μεγιστοποιείται για την τιμή μέγιστης τυχαιότητας 2.6, ακριβώς όπως και στον πρώτο έλεγχο που έγινε.



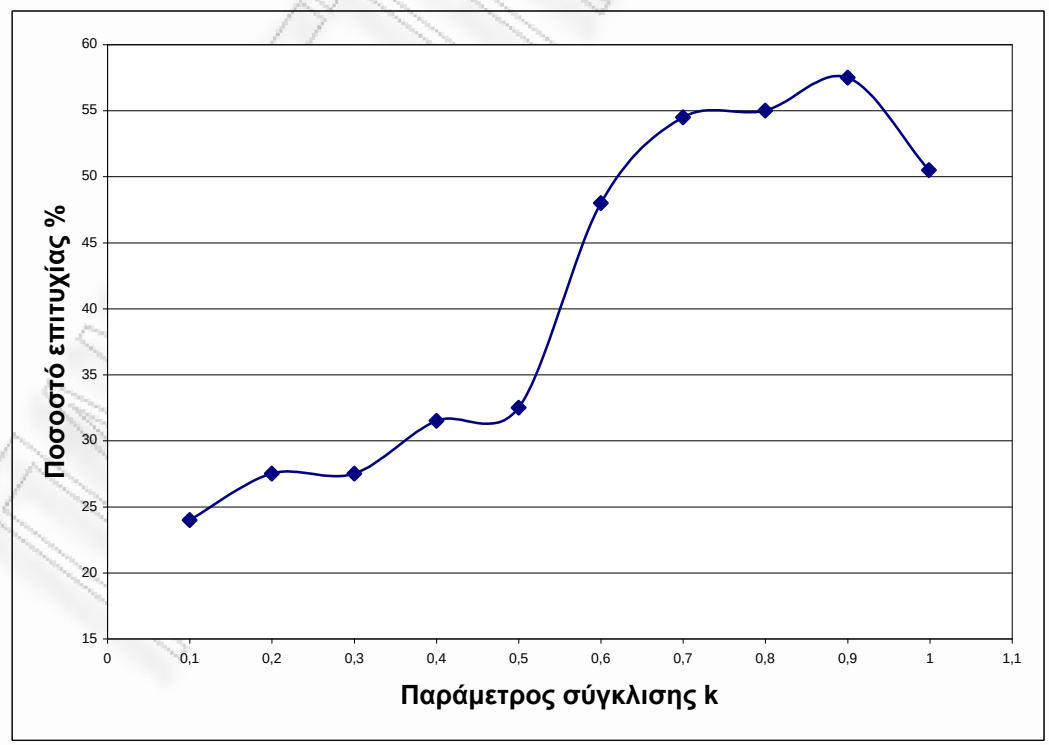
Εικόνα 6.11: Ποσοστό επιτυχίας του PSO κατά την μεταβολή της τυχαιότητας F (δεύτερος έλεγχος)

Στο τέταρτο σετ δοκιμών για τον δεύτερο έλεγχο θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.10 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος k που αφορά το πόσο γρήγορα συγκλίνουν μεταξύ τους τα σωματίδια του αλγορίθμου PSO, από 0.1 ως 0.999 με βήμα 0.1.

Αριθμός βημάτων PSO	300
Αριθμός σωματιδίων PSO	100
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	Μεταβάλλεται

Πίνακας 6.10: Τιμές μεταβλητών του τέταρτου σετ δοκιμών (δεύτερος έλεγχος)

Στην εικόνα 6.12 φαίνονται τα στατιστικά αποτελέσματα του τέταρτου σετ δοκιμών. Όπως παρατηρείται, η απόδοση μεγιστοποιείται για την τιμή 0.9, γεγονός που καταδεικνύει ότι τα σωματίδια χρειάζονται πιο αργή σύγκλιση ώστε να προλάβουν να εξερευνήσουν περισσότερο τον πολύπλοκο γράφο του δικτύου.



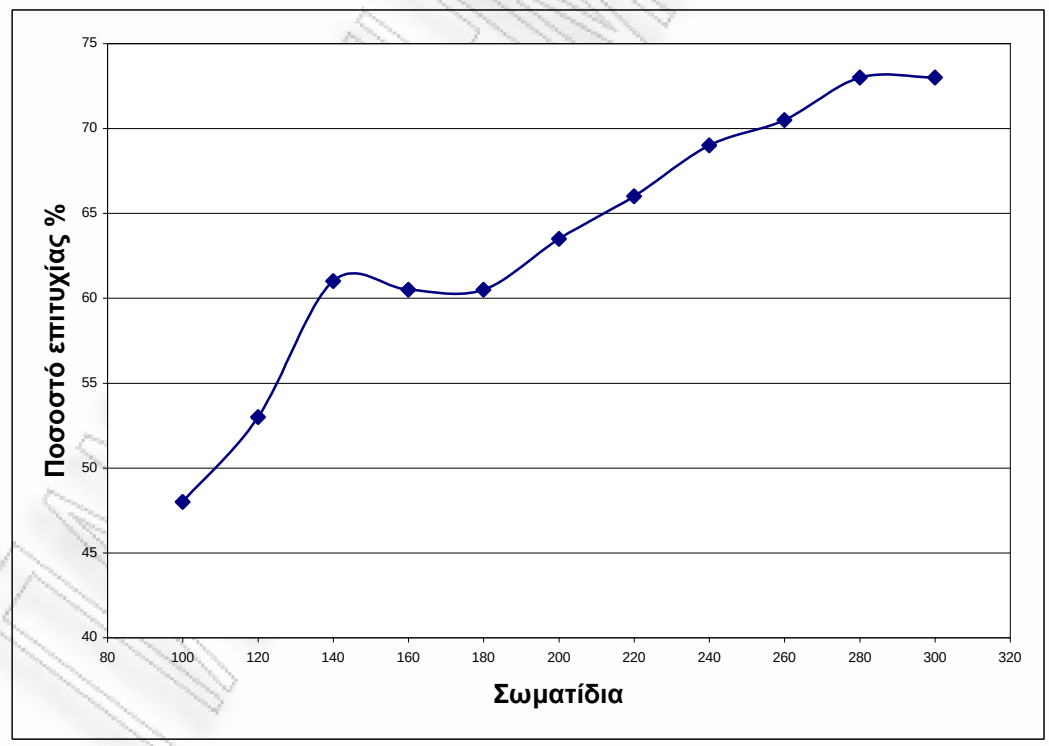
Εικόνα 6.12: Ποσοστό επιτυχίας του PSO κατά την μεταβολή της παραμέτρου k (δεύτερος έλεγχος)

Λαμβάνοντας υπ' όψιν τα παραπάνω αποτελέσματα θα διεξαχθεί ένα πέμπτο σετ δοκιμών όπου οι παράμετροι θα λάβουν τις βέλτιστες τιμές που βρέθηκαν στα προηγούμενα τεστ, ενώ θα μεταβάλλεται και πάλι ο αριθμός των σωματιδίων του αλγορίθμου (αφού είναι η σημαντικότερη παράμετρος επιτυχίας του αλγορίθμου), στο διάστημα 100 ως 300 με βήμα 20. Οι τιμές των βέλτιστων παραμέτρων φαίνεται στον πίνακα 6.11.

Αριθμός βημάτων PSO	300
Αριθμός σωματιδίων PSO	Μεταβάλλεται
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	0,9

Πίνακας 6.11: Τιμές μεταβλητών του τέταρτου σετ δοκιμών (δεύτερος έλεγχος)

Στην εικόνα 6.13 φαίνονται τα στατιστικά αποτελέσματα του πέμπτου σετ δοκιμών.



Εικόνα 6.13: Ποσοστό επιτυχίας του PSO κατά την μεταβολή του αριθμού των σωματιδίων, με βέλτιστες τιμές για τις υπόλοιπες παραμέτρους (δεύτερος έλεγχος)

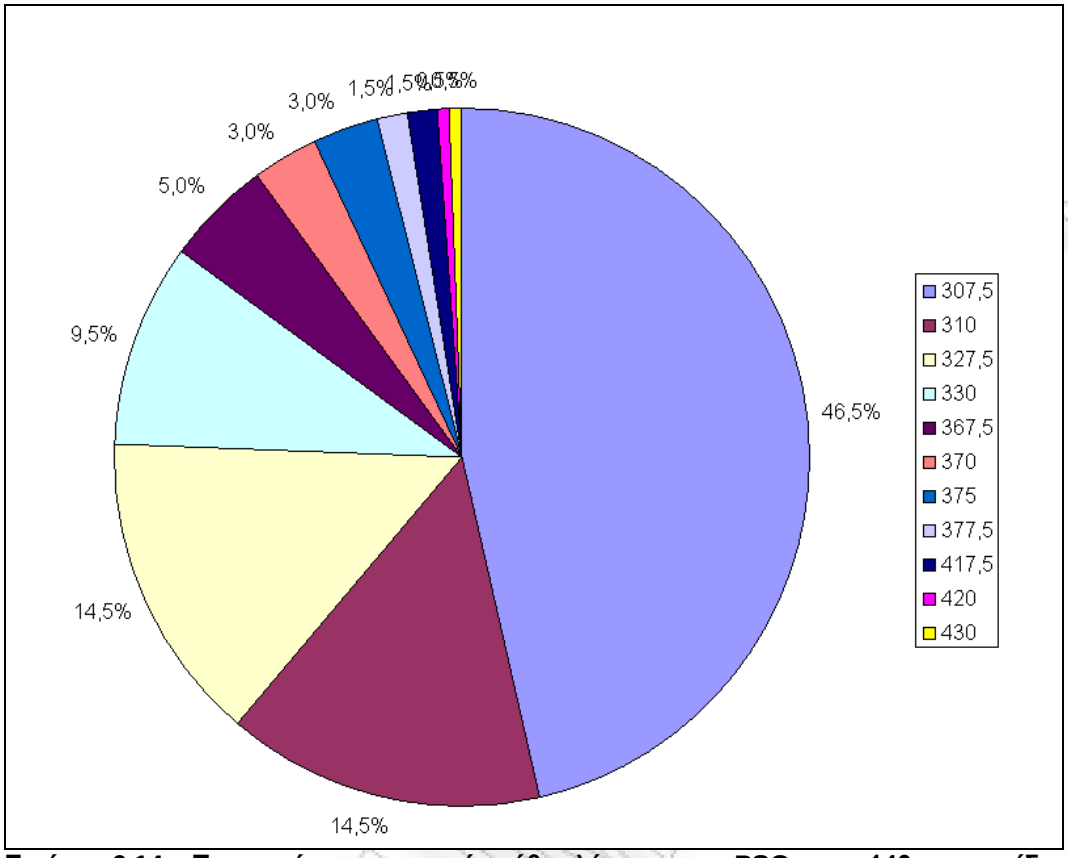
Όπως γίνεται εμφανές από το σχεδιάγραμμα η απόδοση του PSO αυξάνεται σταδιακά όσο αυξάνονται τα σωματίδια του αλγορίθμου, αλλά με πολύ μικρό ρυθμό παρόλη την μεγάλη αύξηση του αριθμού των σωματιδίων. Στον πίνακα 6.12 παραθέτονται τα δεδομένα που χρησιμοποιήθηκαν για το διάγραμμα της εικόνας 6.13.

Σωματίδια	Βέλτιστη τιμή	Σχεδόν βέλτιστη τιμή	Συνολικό ποσοστό %
100	73	23	47,5
120	85	21	53
140	93	29	61
160	102	19	60,5
180	101	20	60,5
200	101	26	63,5
220	101	31	66
240	111	27	69
260	118	23	70,5
280	116	30	73
300	122	24	73

Πίνακας 6.12: Δεδομένα διαγράμματος εικόνας 6.13

Η εξέταση των δεδομένων φανερώνει ότι ο αλγόριθμος βρίσκει κατά μέσο όρο την βέλτιστη λύση περίπου στο 50% των περιπτώσεων, ενώ εντοπίζει την δεύτερη καλύτερη λύση στο υπόλοιπο 10%-15% των περιπτώσεων, χωρίς να παρουσιάζει ουσιαστική βελτίωση σε σχέση με την μεγάλη αύξηση των σωματιδίων.

Κατόπιν τούτου θα επιχειρηθεί και μία διαφορετική αποτίμηση του αλγορίθμου. Χρησιμοποιώντας τα στοιχεία της τρίτης γραμμής του πίνακα 6.12, δηλαδή για 140 σωματίδια του αλγορίθμου, θα απεικονισθούν όλες οι λύσεις που εντόπισε ως “βέλτιστες” ο αλγόριθμος στις διακόσιες επαναλήψεις του. Τα αποτελέσματα της απεικόνισης φαίνονται στην εικόνα 6.14.



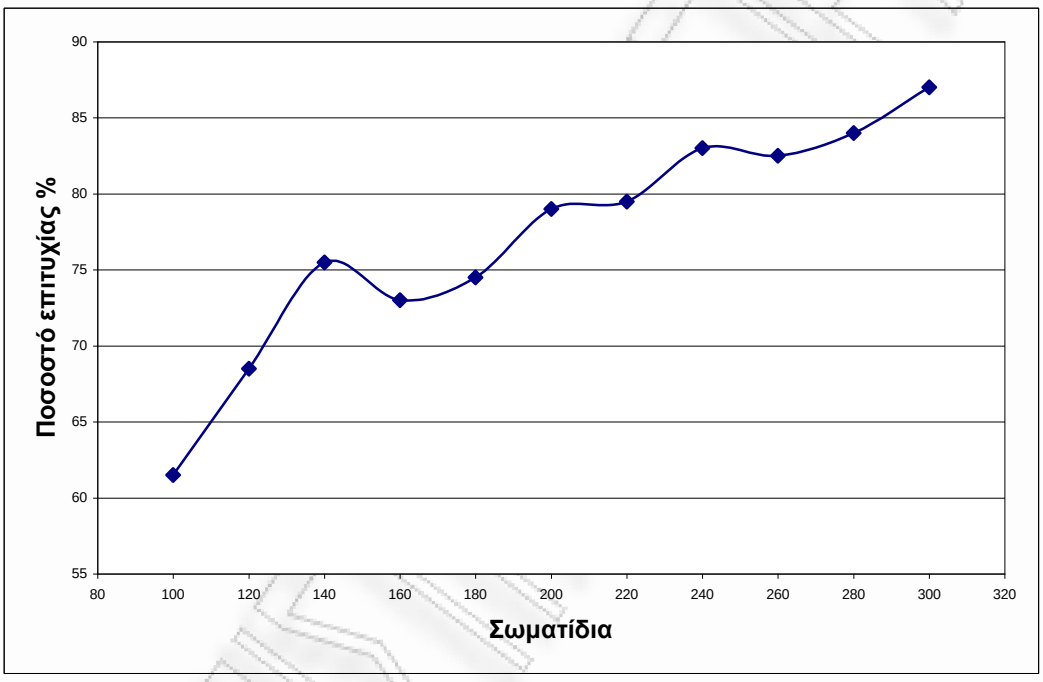
Εικόνα 6.14: Ποσοστό εντοπισμού κάθε λύσης του PSO για 140 σωματίδια (δεύτερος έλεγχος)

Όπως φαίνεται από την εικόνα 6.14, ο αλγόριθμος εντοπίζει έντεκα διαφορετικές λύσεις στο σύνολο των διακοσίων επαναλήψεων, ως "βέλτιστες", από το σύνολο 13^4 = 28.561 λύσεων. Η πραγματική βέλτιστη λύση επιτυγχάνεται με ποσοστό 64,5%, ενώ η σχεδόν βέλτιστη δεύτερη λύση επιτυγχάνεται σε ποσοστό 14,5%. Επίσης ένα σημαντικό ποσοστό 14,5% καταλαμβάνει και η τρίτη καλύτερη λύση, η οποία είναι πολύ κοντά στην δεύτερη καλύτερη αριθμητικά.

Όπως έχει αναφερθεί και στο κεφάλαιο επισκόπησης της βιβλιογραφίας κανένας εξελικτικός αλγόριθμος δεν εγγυάται ότι βρίσκει την βέλτιστη λύση, αλλά ότι βρίσκουν σχεδόν-βέλτιστες λύσεις (near optimum-solutions), σε σύντομο χρόνο. Ως εκ τούτου και λόγο της διαφαινόμενης πολυπλοκότητας του προβλήματος και των πολλών διαφορετικών λύσεων με τις οποίες τερματίζει ο αλγόριθμος, θα επιχειρηθεί η χαλάρωση των

κριτηρίων επιτυχίας του και θα παρουσιαστούν τα ποσοστά επιτυχίας του βάση των τριών επικρατέστερων λύσεων που εξάγονται από τον PSO.

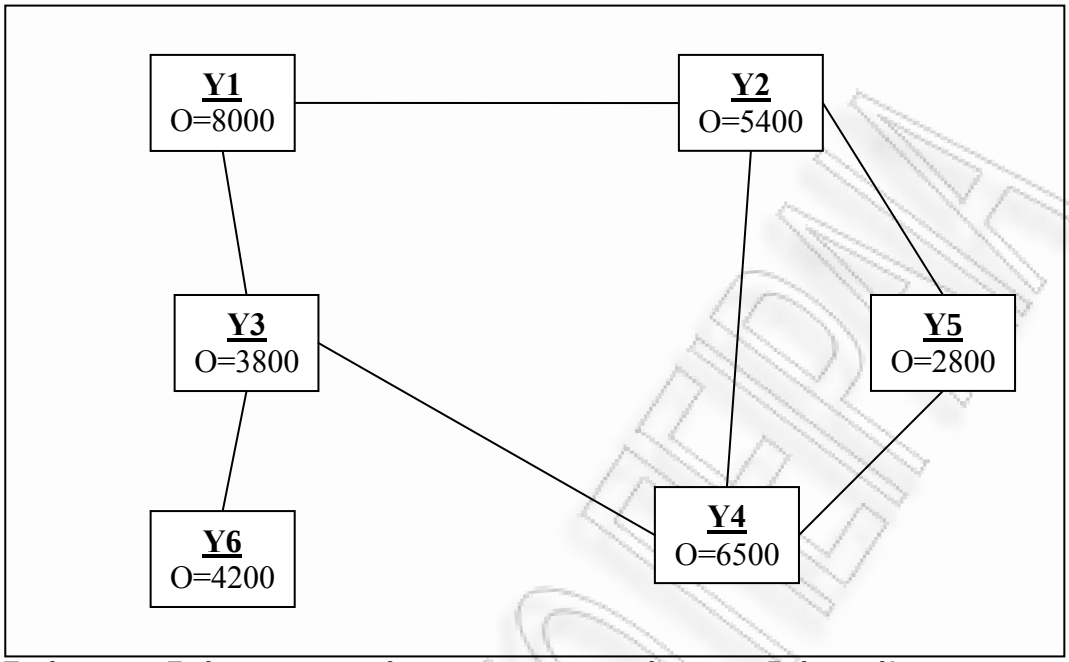
Για τον λόγο αυτό θα προστεθεί στα αποτελέσματα του πέμπτου σετ δοκιμών που και η συνεισφορά της τρίτης καλύτερης λύσης που υπολογίστηκε κατά τις στατιστικές δοκιμές. Το νέο αποτέλεσμα φαίνεται στην εικόνα 6.15, όπου το ποσοστό επιτυχίας είναι αρκετά βελτιωμένο.



Εικόνα 6.15: Ποσοστό επιτυχίας του PSO κατά την μεταβολή του αριθμού των σωματιδίων, για τις τρεις καλύτερες τιμές κόστους (δεύτερος έλεγχος)

6.3 Έλεγχος αποδοτικότητας σε δίκτυο δεκατριών κόμβων και έξι υπηρεσιών

Ο συγκεκριμένος έλεγχος αποδοτικότητας θα εφαρμοστεί στο γράφο του δικτύου με τα χαρακτηριστικά που φαίνονται στην εικόνα 6.8 (ίδιος με τον δεύτερο έλεγχο) και για τον γράφο των υπηρεσιών με τα χαρακτηριστικά που φαίνονται στην εικόνα 6.14.

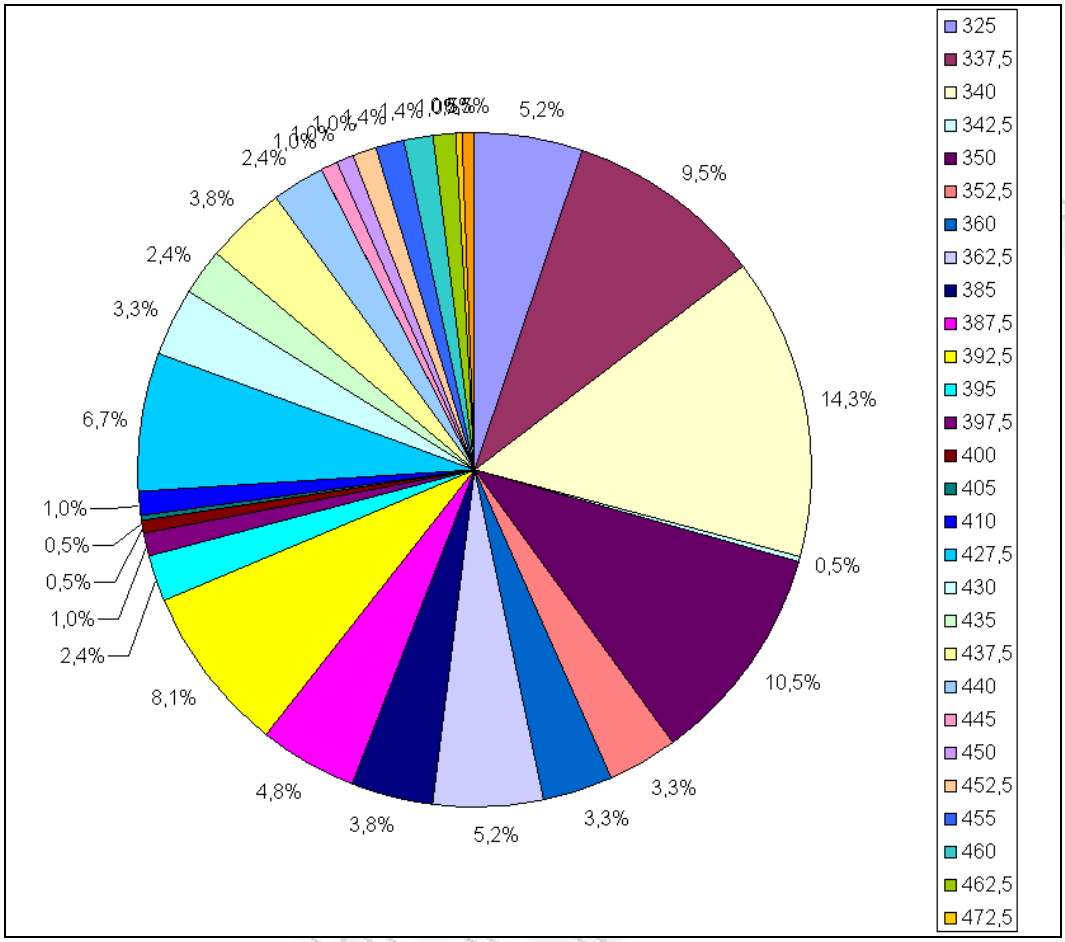


Εικόνα 6.16: Γράφος υπηρεσιών που χρησιμοποιείται στον δεύτερο έλεγχο

Όπως έγινε φανερό από τον προηγούμενο έλεγχο, όσο πιο περίπλοκο γίνεται το πρόβλημα , τόσο πρέπει να προσαρμόζονται και τα κριτήρια επιτυχίας του αλγορίθμου. Για το λόγο αυτό θα γίνει και σε αυτόν τον έλεγχο πρώτα μια αναπαράσταση όλων των λύσεων που εντοπίζει ως "βέλτιστες" ο αλγόριθμος για να διαπιστωθεί το πλήθος τους και η ποιότητά τους. Στην εικόνα 6.17 φαίνονται τα αποτελέσματα μίας δοκιμαστικής απεικόνισης που έγινε για τις τιμές των μεταβλητών που φαίνονται στον πίνακα 6.13.

Αριθμός βημάτων PSO	300
Αριθμός σωματιδίων PSO	200
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	0,9

Πίνακας 6.13: Τιμές μεταβλητών δοκιμαστικού σετ (τρίτος έλεγχος)



Εικόνα 6.17: Ποσοστό εντοπισμού κάθε λύσης του PSO για 200 σωματίδια (τρίτος έλεγχος)

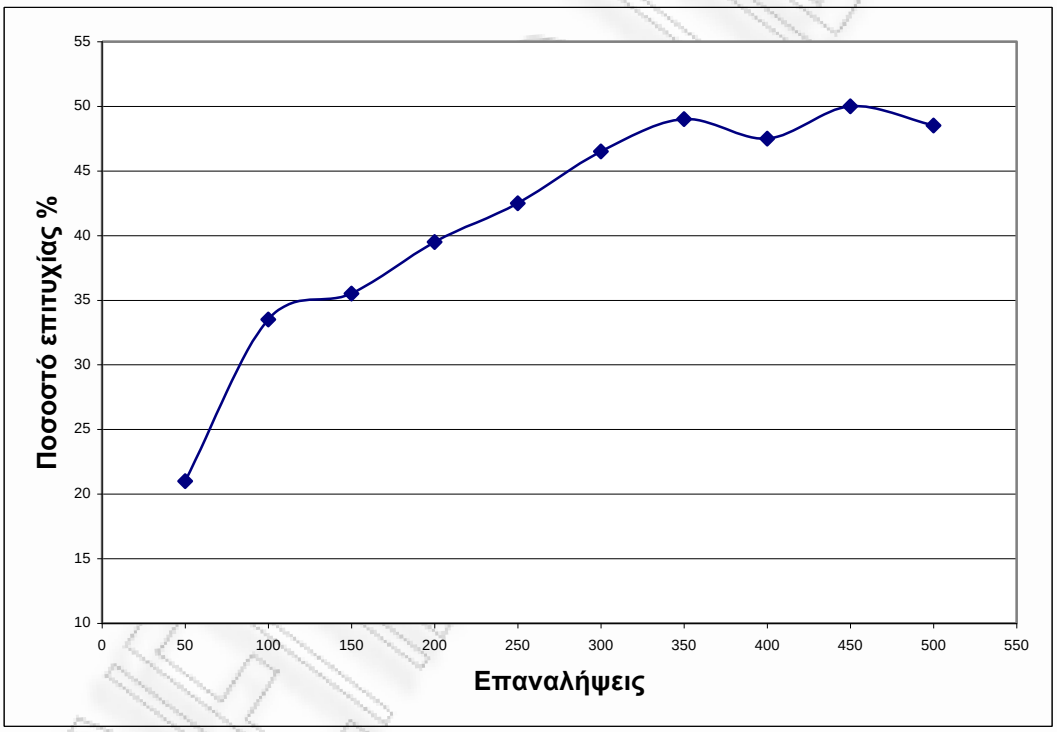
Όπως φαίνεται από την εικόνα 6.17, ο αλγόριθμος εντοπίζει εικοσιεννέα διαφορετικές λύσεις στο σύνολο των διακοσίων επαναλήψεων ως “βέλτιστες”, από ένα πολύ μεγαλύτερο χώρο πιθανών λύσεων. Η πραγματική βέλτιστη λύση επιτυγχάνεται με ποσοστό μόλις 5,2%, ενώ υπάρχει εγκλωβισμός του αλγορίθμου σε υποδεέστερες λύσεις με μη αμελητέα ποσοστά. Για τις δοκιμές που θα ακολουθήσουν θα ληφθεί υπ’ όψιν στο ποσοστό επιτυχίας του αλγορίθμου η συνεισφορά των λύσεων που έχουν κόστος μικρότερο ή ίσο με 352,5.

Στο πρώτο σετ δοκιμών για τον τρίτο έλεγχο θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.14 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος του αριθμού επαναλήψεων του αλγορίθμου από 50 ως 500 με βήμα 50.

Αριθμός βημάτων PSO	Μεταβάλλεται
Αριθμός σωματιδίων PSO	200
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	0,9

Πίνακας 6.14: Τιμές μεταβλητών του πρώτου σετ δοκιμών (τρίτος έλεγχος)

Στην εικόνα 6.18 φαίνονται τα στατιστικά αποτελέσματα του πρώτου σετ δοκιμών για το τρίτο δίκτυο. Όπως παρατηρείται η απόδοση του αλγορίθμου μεγιστοποιείται στις 450 επαναλήψεις.



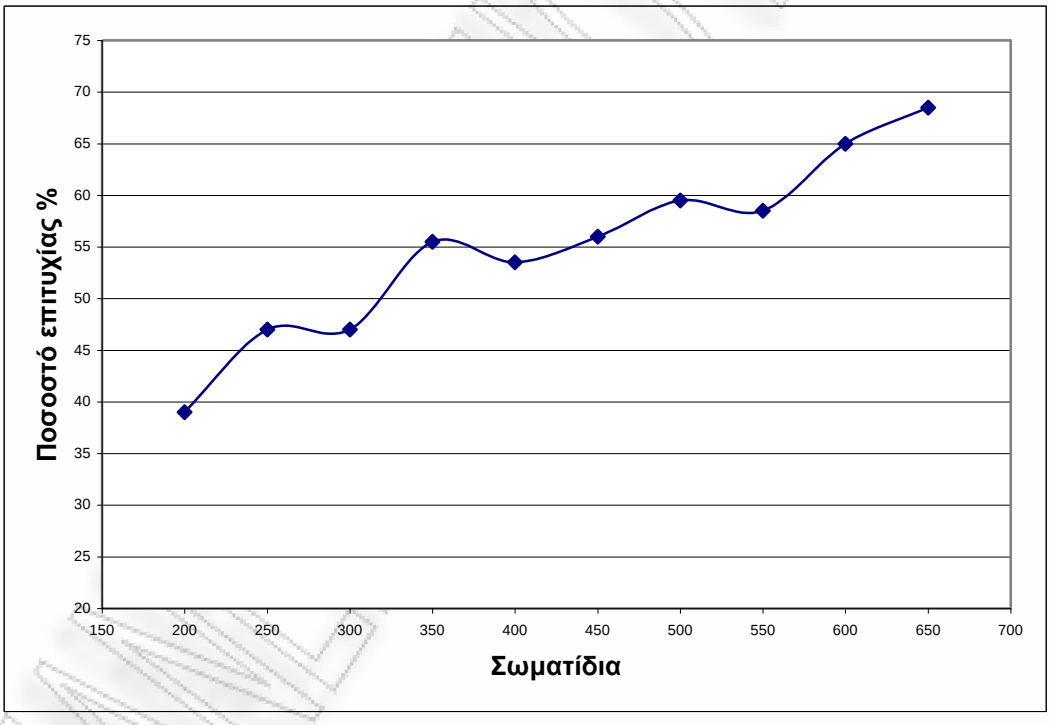
Εικόνα 6.18: Ποσοστό επιτυχίας του PSO κατά την μεταβολή του αριθμού των επαναλήψεων (τρίτος έλεγχος)

Στο δεύτερο σετ δοκιμών για τον τρίτο έλεγχο θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.15 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος του αριθμού των σωματιδίων του αλγορίθμου από 50 ως 500 με βήμα 50.

Αριθμός βημάτων PSO	450
Αριθμός σωματιδίων PSO	Μεταβάλλεται
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	0,9

Πίνακας 6.15: Τιμές μεταβλητών του δεύτερου σετ δοκιμών (τρίτος έλεγχος)

Στην εικόνα 6.19 φαίνονται τα στατιστικά αποτελέσματα του δεύτερου σετ δοκιμών για τον τρίτο έλεγχο. Όπως παρατηρείται η απόδοση του αλγορίθμου αυξάνεται πολύ ελαφρώς με την αύξηση του αριθμού των σωματιδίων, παρόλο που η αύξηση γίνεται κατά μεγάλα διαστήματα τιμών.



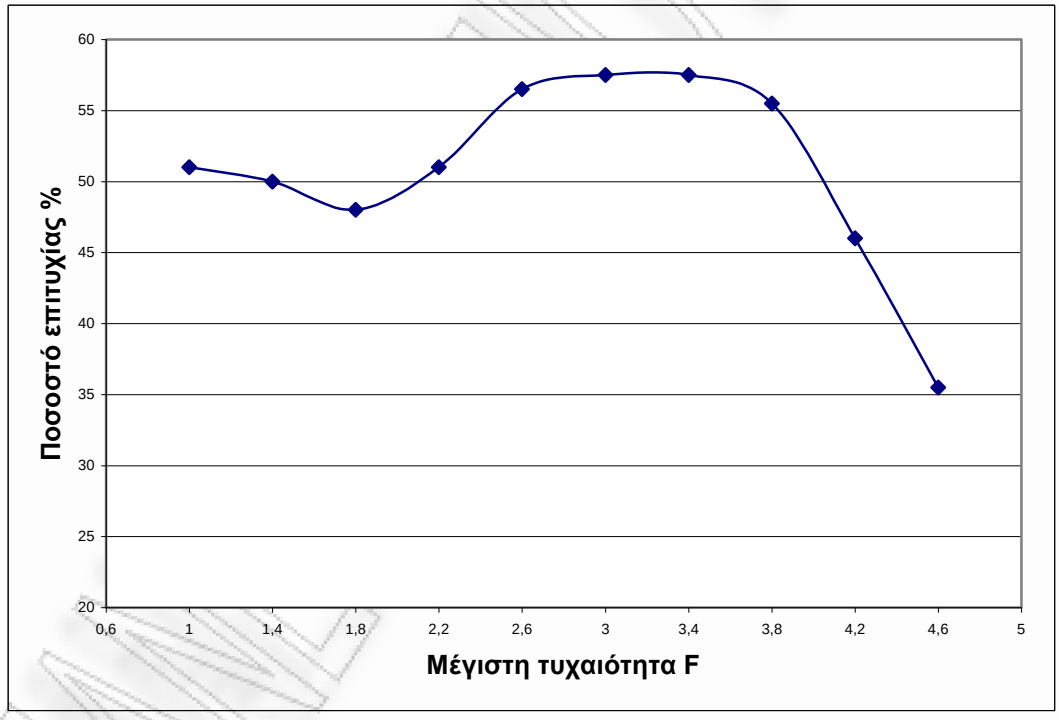
Εικόνα 6.19: Ποσοστό επιτυχίας του PSO κατά την μεταβολή του αριθμού των σωματιδίων (τρίτος έλεγχος)

Στο τρίτο σετ δοκιμών θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.16 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος της μέγιστης τυχαιότητας που μπορεί να εισαχθεί στην κίνηση των σωματιδίων του αλγορίθμου από 1 ως 4,6 με βήμα 0,4.

Αριθμός βημάτων PSO	450
Αριθμός σωματιδίων PSO	400
Μέγιστη τιμή τυχαιότητας F	Μεταβάλλεται
Τιμή παραμέτρου K	0,9

Πίνακας 6.16: Τιμές μεταβλητών του τρίτου σετ δοκιμών (τρίτος έλεγχος)

Στην εικόνα 6.20 φαίνονται τα στατιστικά αποτελέσματα του τρίτου σετ δοκιμών για τον τρίτο έλεγχο. Όπως παρατηρείται η απόδοση του αλγορίθμου μεγιστοποιείται για την τιμές μέγιστης τυχαιότητας στο διάστημα 3,0 - 3,4 με τις γειτονικές τιμές να δίνουν παραπλήσια ποσοστά επιτυχίας.



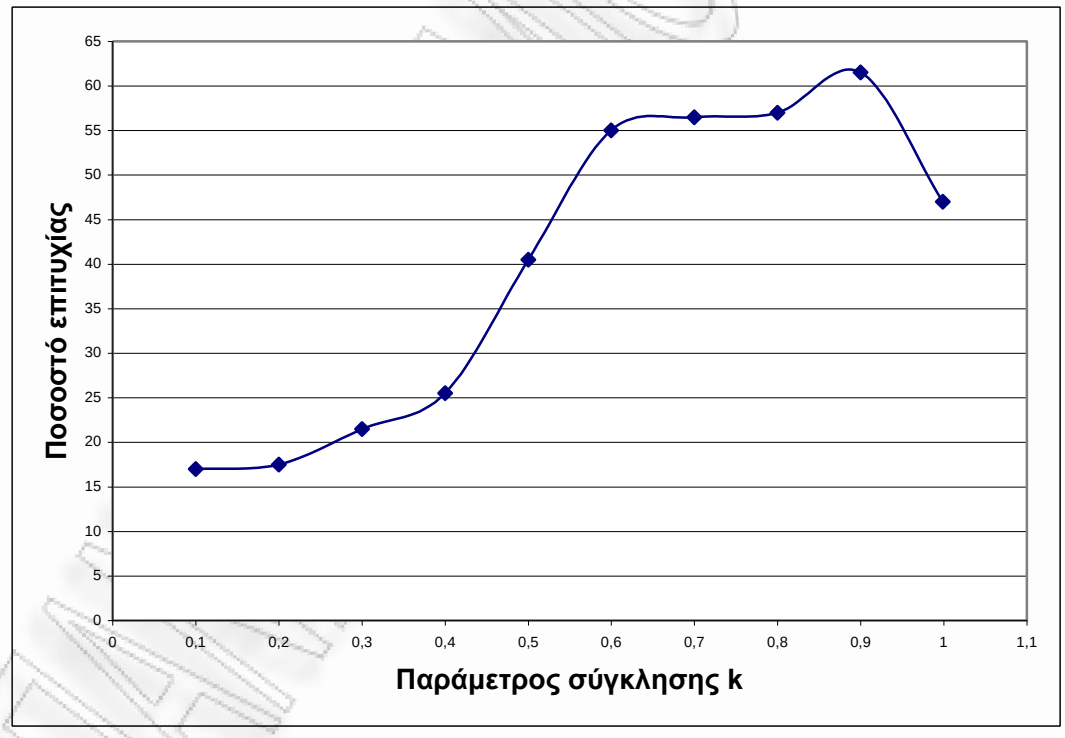
Εικόνα 6.20: Ποσοστό επιτυχίας του PSO κατά την μεταβολή της τυχαιότητας F (τρίτος έλεγχος)

Στο τέταρτο σετ δοκιμών για τον τρίτο έλεγχο θα μείνουν σταθερές οι παράμετροι που φαίνονται στον πίνακα 6.17 με τις αντίστοιχες τιμές, ενώ θα μεταβάλλεται η παράμετρος σύγκλισης k, από 0.1 ως 0.999 με βήμα 0.1.

Αριθμός βημάτων PSO	450
Αριθμός σωματιδίων PSO	400
Μέγιστη τιμή τυχαιότητας F	2,6
Τιμή παραμέτρου K	Μεταβάλλεται

Πίνακας 6.17: Τιμές μεταβλητών του τέταρτου σετ δοκιμών (τρίτος έλεγχος)

Στην εικόνα 6.21 φαίνονται τα στατιστικά αποτελέσματα του τέταρτου σετ δοκιμών. Όπως παρατηρείται, η απόδοση μεγιστοποιείται για την τιμή 0.9, γεγονός που καταδεικνύει ότι τα σωματίδια χρειάζονται αργή σύγκλιση ώστε να προλάβουν να εξερευνήσουν τον πολύπλοκο γράφο του δικτύου, ακριβώς όπως και στον δεύτερο έλεγχο.



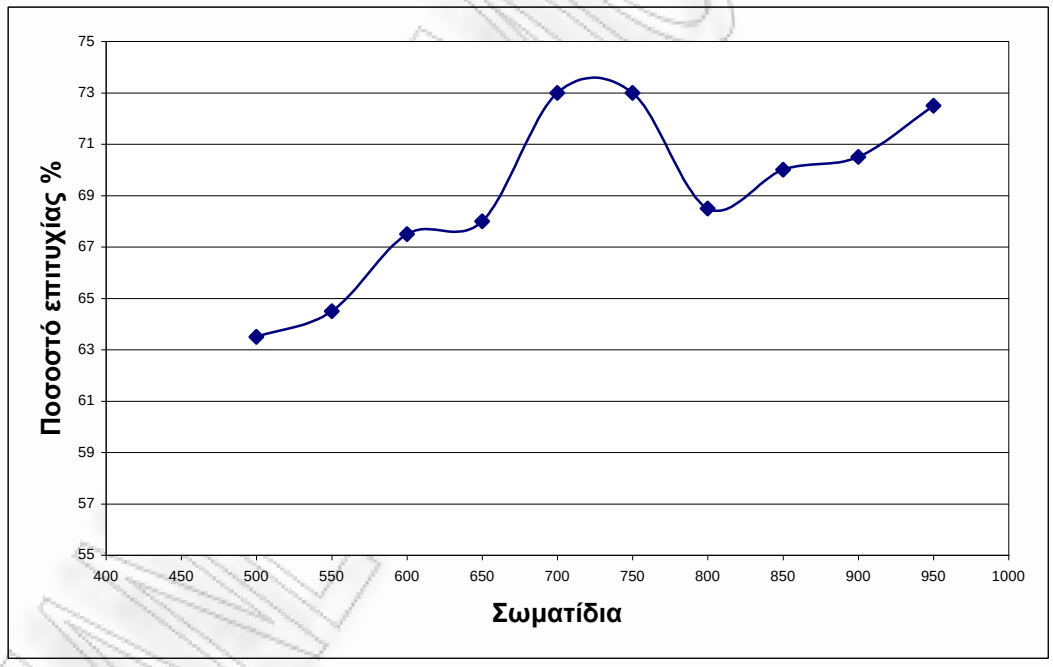
Εικόνα 6.21: Ποσοστό επιτυχίας του PSO κατά την μεταβολή της παραμέτρου σύγκλισης k (τρίτος έλεγχος)

Παρόλα αυτά η τιμή της παραμέτρου k=0.999 που χρησιμοποιείται δεν παράγει τόσο καλά αποτελέσματα αφού σχεδόν απαλείφει τον παράγοντα σύγκλισης με αποτέλεσμα να μην συγκλίνουν τα σωματίδια μεταξύ τους.

Όπως και με τους προηγούμενους δύο ελέγχους, θα διεξαχθεί και εδώ ένας πέμπτος έλεγχος με βέλτιστες παραμέτρους (πίνακας 6.18), παρόλο που όπως φαίνεται ήδη από τα υπάρχοντα αποτελέσματα ο αλγόριθμος δεν έχει μεγάλη απόδοση. Για το λόγο αυτό θα επιχειρηθεί μια σημαντική αύξηση του αριθμού των σωματιδίων του αλγορίθμου τα οποία θα μεταβάλλονται από 500 ως 950 με βήμα 50.

Αριθμός βημάτων PSO	450
Αριθμός σωματιδίων PSO	Μεταβάλλεται
Μέγιστη τιμή τυχαιότητας F	3,2
Τιμή παραμέτρου K	0,9

Πίνακας 6.18: Τιμές μεταβλητών του πέμπτου σετ δοκιμών (τρίτος έλεγχος)



Εικόνα 6.22: Ποσοστό επιτυχίας του PSO κατά την μεταβολή των σωματιδίων, με βέλτιστες τιμές για τις υπόλοιπες παραμέτρους (τρίτος έλεγχος)

Τα αποτελέσματα που αναπαρίστανται στην εικόνα 6.22 δείχνουν ότι ο αλγόριθμος έχει ένα ποσοστό επιτυχίας γύρω στο 70% για το κατώφλι επιτυχίας που τέθηκε αυθαίρετα (οι έξι μικρότερες τιμές που

υπολογίστηκαν στην συνάρτηση κόστους). Υπάρχει ελάχιστη βελτίωση στα ποσοστά επιτυχίας του PSO παρά τον μεγάλο αριθμό των σωματιδίων και μάλιστα υπάρχουν στατιστικές διακυμάνσεις που καταδεικνύουν ότι δεν υπάρχει κανένα ουσιαστικό κέρδος από την μεγάλη αύξηση του αριθμού των σωματιδίων. Οι διακυμάνσεις αυτές είναι της τάξης του 4,5% και οφείλονται στην τυχαιότητα που ενυπάρχει στον αλγόριθμο από την παράμετρο F. Μικρές διακυμάνσεις της τάξης του 2% στην απόδοση, σε σχέση με τον αριθμό των σωματιδίων, παρατηρούνται και στους προηγούμενους ελέγχους, αλλά στον παρόντα έλεγχο υπάρχει μεγαλύτερη απόκλιση επειδή το εξεταζόμενο σύστημα είναι αρκετά πιο πολύπλοκο, με πολύ περισσότερες πιθανές λύσεις οι οποίες εγκλωβίζουν τα σωματίδια του αλγόριθμου σε τοπικά μέγιστα.

Το σύνολο των πιθανών λύσεων για τον συγκεκριμένο έλεγχο ανέρχεται στις 13^6 = 4.826.809, γεγονός που αποδεικνύει τον βαθμό πολυπλοκότητας του εξεταζόμενου προβλήματος. Οπότε παρόλο που εκ πρώτης όψεως τα αποτελέσματα του αλγορίθμου δεν είναι τόσο ενθαρρυντικά σε απόλυτα νούμερα, δηλαδή στον εντοπισμό της πραγματικής βέλτιστης λύσης ή ακόμα και των κοντινών σε αυτή λύσεων, ο αλγόριθμος αποδίδει σχετικά καλά σε αναλογία με τον πολύ μεγάλο χώρο καταστάσεων που πρέπει να εξερευνήσει. Αυτό δικαιολογείται από το γεγονός ότι σε ελάχιστο χρονικό διάστημα προτείνει μία λύση που σύμφωνα με τα αποτελέσματα του ελέγχου που απεικονίζονται στην εικόνα 6.17 -δηλαδή 29 διαφορετικές λύσεις προτεινόμενες ως “βέλτιστες”- ο αλγόριθμος εντοπίζει λύσεις με πολύ μικρή απόκλιση από την βέλτιστη σε σχέση με τον συνολικό αριθμό πιθανών λύσεων που αναφέρθηκε (13^6).

Κεφάλαιο 7: Ανακεφαλαίωση - Συμπεράσματα

7.1 Ανακεφαλαίωση

Η παρούσα διπλωματική εργασία ασχολείται με το πρόβλημα της βέλτιστης ανάπτυξης υπηρεσιών σε πολύπλοκα δυναμικά συστήματα. Το πρόβλημα αυτό προέκυψε καθώς με την εξέλιξη των κατανεμημένων συστημάτων αυτά μετατράπηκαν από στατικά δίκτυα σταθερών υπολογιστών σε πολύπλοκα δυναμικά συστήματα δυνητικά κινητών συσκευών οι οποίες μπορούν να συνδέονται μεταξύ τους ευκαιριακά (ad-hoc) για να επιτελέσουν διάφορες λειτουργίες.

Στα ανωτέρω πλαίσια των αναδιαρθρώσιμων και δυναμικών προσαρμόσιμων συστημάτων εξετάζεται το πρόβλημα της βέλτιστης ανάπτυξης υπηρεσιών, ενώ αυτό ειδικεύεται στην βέλτιστη εξισορρόπηση μεταξύ του κόστους επεξεργασίας των υπηρεσιών στους επεξεργαστές του κατανεμημένου συστήματος και του κόστους επικοινωνίας μεταξύ των επεξεργαστών, ανεξάρτητα από την υλική υποδομή του δικτύου και την φύση των υπηρεσιών που πρόκειται να αναπτυχθούν.

Εργαλείο για τη βελτιστοποίηση του κόστους ανάπτυξης των υπηρεσιών αποτελούν οι διάφοροι αλγόριθμοι βελτιστοποίησης. Για το λόγο αυτό γίνεται μία παρουσίαση των διαφόρων αλγορίθμων εξερεύνησης και βελτιστοποίησης, ενώ δίνεται ιδιαίτερη έμφαση στους πιο σύγχρονους και ισχυρούς (βάση βιβλιογραφίας) αλγορίθμους που είναι ικανότεροι να επιλύουν πολύπλοκα επιστημονικά προβλήματα όπως το θέμα της εργασίας.

Έπειτα γίνεται μία ανασκόπηση της σχετικής επιστημονικής βιβλιογραφίας, με εργασίες που χρησιμοποιούν τους κυριότερους από τους εξετασθέντες αλγορίθμους βελτιστοποίησης για να προσεγγίσουν το πρόβλημα που εξετάζει η εργασία.

Με βάση όλες τις πληροφορίες που συγκεντρώθηκαν από την εξέταση των βιβλιογραφικών πηγών, παρουσιάζεται η προτεινόμενη μοντελοποίηση της λύσης για το εξεταζόμενο πρόβλημα, ενώ παράλληλα

εξετάζονται οι διαφοροποιήσεις της παρούσας λύσης σε σχέση με τις αντίστοιχες της ερευνητικής βιβλιογραφίας.

Έπειτα γίνεται παρουσίαση της υλοποίησης που αναπτύχθηκε για την προτεινόμενη λύση, δηλαδή ένα εργαλείο προσομοίωσης σε προγραμματιστικό περιβάλλον java που δέχεται ως είσοδο τα χαρακτηριστικά των υπηρεσιών προς ανάπτυξη, τα χαρακτηριστικά του δικτύου (κάθε φορά που υπάρχει αλλαγή σε αυτό) καθώς και τιμές για της παραμέτρους του αλγορίθμου βελτιστοποίησης και παράγει ως έξοδο το σχέδιο του γράφου του δικτύου όπου φαίνονται ανεπτυγμένες οι υπηρεσίες στους διάφορους επεξεργαστές/κόμβους του συστήματος σύμφωνα με την λύση που υπολόγισε ο αλγόριθμος βελτιστοποίησης.

Τέλος πραγματοποιούνται στατιστικοί έλεγχοι με χρήση του εργαλείου προσομοίωσης του αλγορίθμου ώστε να εκτιμηθεί η απόδοση της προτεινόμενης λύσης. Συγκεκριμένα πραγματοποιούνται τρεις έλεγχοι κλιμακούμενου βαθμού δυσκολίας και με εξέταση της επιρροής όλων των παραμέτρων του αλγορίθμου στο παραγόμενο αποτέλεσμα.

7.2 Συμπεράσματα

Τα αποτελέσματα των στατιστικών ελέγχων που διεξήχθησαν δείχνουν ότι ο αλγόριθμος που προτάθηκε ως λύση συμπεριφέρεται άψογα σε πολύ μικρά δίκτυα που αναπτύσσονται λίγες υπηρεσίες, εντοπίζοντας την βέλτιστη λύση σε πολύ μεγάλο ποσοστό και για μικρές τιμές των παραμέτρων του. Η απόδοση του αλγορίθμου πέφτει σε μεγαλύτερα και πιο πολύπλοκα δίκτυα όπου απαιτούνται μεγαλύτερες τιμές των παραμέτρων του για την επίτευξη σημαντικών ποσοστών επιτυχίας. Τέλος, η αύξηση των υπηρεσιών προς ανάπτυξη επηρεάζει δραματικά την ικανότητα του αλγορίθμου να εντοπίζει την βέλτιστη λύση (ή λύσεις πολύ κοντά σε αυτή), καθώς η πολυπλοκότητα του προβλήματος αυξάνεται εκθετικά όσο προστίθενται επιπλέον υπηρεσίες προς ανάπτυξη. Αν και η επιτυχία του αλγορίθμου να βρίσκει τη πραγματική βέλτιστη λύση ή λύσεις πολύ κοντά σε αυτή περιορίζεται σε ποσοστά της τάξης του 50%

ακόμα και για πολύ μεγάλες τιμές των παραμέτρων του, μπορεί να θεωρηθεί ότι ο αλγόριθμος είναι αποδοτικός, υπό το πρίσμα του τεραστίου χώρου καταστάσεων (λύσεων) που έχει να εξετάσει. Δηλαδή, οι λύσεις που παράγει ο αλγόριθμος παρόλο που αρκετές φορές αποκλίνουν από την πραγματική βέλτιστη λύση, δεν αποκλίνουν ωστόσο σε μεγάλο ποσοστό αναλογικά με τον τεράστιο χώρο καταστάσεων και αποτελούν ένα πολύ μικρό ποσοστό “ελίτ” λύσεων.

7.3 Περαιτέρω μελέτη

Για της ανάγκες της παρούσας εργασίας το πολύπλευρο πρόβλημα της βέλτιστης ανάπτυξης υπηρεσιών σε πολύπλοκα δυναμικά συστήματα έπρεπε να περιοριστεί, ώστε να καταστεί δυνατή η μελέτη του προβλήματος.

Για την προτεινόμενη λύση επιλέχθηκε από την βιβλιογραφία συγκεκριμένος αλγόριθμος βελτιστοποίησης και εφαρμόστηκε πάνω στο συγκεκριμένο πρόβλημα. Θα μπορούσε να γίνει περαιτέρω συγκριτική μελέτη της απόδοσης του επιλεχθέντος αλγορίθμου με άλλους αλγορίθμους πάνω στο συγκεκριμένο πρόβλημα και την μοντελοποίηση του.

Μία πολύ ενδιαφέρουσα μελέτη θα ήταν επίσης η χρήση του αλγορίθμου με διαφορετικό κριτήριο βελτιστοποίησης (διαφορετική συνάρτηση κόστους) που θα εστιάζει σε διαφορετικά κριτήρια όπως η ποιότητα παρεχόμενης υπηρεσίας (quality of service), ή διαφορετικό βάρος μεταξύ των παραγόντων προς βελτιστοποίηση (δηλαδή το κόστος επεξεργασίας και επικοινωνίας).

Τέλος περαιτέρω μελέτη θα μπορούσε να διεξαχθεί πάνω στα στατιστικά αποτελέσματα που παράγει η συγκεκριμένη υλοποίηση του αλγορίθμου, ώστε να καθοριστεί με πιο αντικειμενικά κριτήρια ένα κατώφλι λύσεων που θα θεωρούνται αποδεκτές ως “σχεδόν-βέλτιστες” για το ποσοστό επιτυχίας, αναλογικά με το μέγεθος του δικτύου του συστήματος και του αριθμού υπηρεσιών προς ανάπτυξη.

Βιβλιογραφία

[1] Salman A. et al, ‘‘Particle swarm optimization for task assignment problem”, Microprocessors and Microsystems 26 (2002) 363–371, Elsevier Science B.V.

[2] Csorba M. et al, “Adaptable Model-based Component Deployment Guided by Artificial Ants”, Autonomics '08 Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems, Belgium ©2008

[3] Efe K., “Heuristic Models of Task Assignment Scheduling in Distributed Systems”, IEEE Computer Magazine, Vol. 15, no. 6, pp. 50-56, June 1982

[4] Aksit M. et al, “Dynamic, Adaptive and Reconfigurable Systems Overview and Prospective Vision”, Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW’03), IEEE 2003

[5] Coello C. et al, “Evolutionary Algorithms for Solving Multi-Objective Problems”, Second Edition, Springer

[6] Black P.E., "Greedy algorithm", Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 2 February 2005

[7] Russel S. et al, “Τεχνητή Νοημοσύνη Μια Σύγχρονη Προσέγγιση”, Εκδόσεις, Κλειδάριθμος, 2004

[8] Land A. H. and Doig A. G., "An automatic method of solving discrete programming problems". Econometrica 28 (3): pp. 497-520, 1960

[9] Rastrigin L.A., "The convergence of the random search method in the extremal control of a many parameter system". Automation and Remote Control 24 (10): 1337–1342, 1963

[10] Kirkpatrick, S.; C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing", 05-1983

[11] Glover, F. and M. Laguna, “Tabu Search”, Kluwer Academic Publishers, Boston, 1997

[12] Hinchey M. et al, “Swarms and Swarm Intelligence”, IEEE Computer Magazine, April 2007

[13] A. Reimman, “Evolutionary Algorithms and Optimization”, Phd Dissertation in Physics, Humboldt University, Berlin

[14] Bishop J. M., “Stochastic Searching Network”, Proc. 1st IEE Conf. ANNs, London, 1989

[15] Bishop J.M., Torr P., “The Stochastic Search Network”, in Linggard R., Myers D.J., Nightingale C. (eds.), “Neural Networks for Images, Speech and Natural Language”, New York, Chapman & Hall, 1992

[16] Mitchell, M., “An Introduction to Genetic Algorithms”, MIT Press, 1999

[17] Dorigo M., “Optimization, Learning and Natural Algorithms”, PhD thesis, Politecnico di Milano, Italy, 1992

[18] Dorigo M., Stützle T., "The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances", Handbook of Metaheuristics, 2002

[19] Kennedy, J., Eberhart, R., “Particle Swarm Optimization”, Proceedings IEEE International Conference on Neural Networks, IV: pp. 1942-1948, IEEE Service Center, 1995

[20] Kennedy, J., Eberhart, R., “Swarm Intelligence”, Morgan Kaufmann Publishers, 2001

[21] Kennedy, J., “Small worlds and mega-minds: effects of neighbourhood topology on particle swarm performance”, Proceedings of the Congress on Evolutionary Computation, 1999, Volume 3, Page(s): -1938, 1999

[22] Clerk, M., Kennedy, J., “The Particle Swarm – Explosion, Stability, and Convergence in a Multi Dimensional Complex Space”, IEEE Transactions on Evolutionary Computation, Vol. 6, No.1, 2002

[23] Li X., “Particle Swarm Optimization, an introduction and its recent developments”, Tutorial prepared for the 6th international conference “Simulated evolution and learning”, Hefei, China, 2006

[24] Rammig F. et al, “Biologically Inspired Methods for Organizing Distributed Services on Sensor Networks”, Dagstuhl Seminar Proceedings 08141, University of Paderborn, Paderborn,Germany

[25] Vanrompay Y., “Genetic Algorithm-Based Optimization of Service Composition and Deployment”, 3rd ACM International Workshop on Services Integration in Pervasive Environments, Sorrento, Italy, July 2008

[26] Salman A. et al, “Particle swarm optimization for task assignment problem”, Microprocessors and Microsystems 26 (2002) 363–371, Elsevier Science B.V

[27] Eberhart R.C., Kennedy J., “A New Optimizer Using Particle Swarm Theory”, Proceedings of the Sixth Symposium on Micro Machine and Human Science, pp. 39–43, 1995

[28] Kennedy, J., Eberhart, R., “Swarm Intelligence”, Morgan Kaufmann Publishers, 2001

[29] Hassan R. et al, “A Comparison Of Particle Swarm Optimization And The Genetic Algorithm”, Proceedings of the 1st AIAA Multidisciplinary Design Optimization Specialist Conference, AIAA-2005-2135, Austin, Texas, April 2005

[30] Coello C. et al., “Handling multiple objectives with particle swarm optimization”, IEEE Transactions on evolutionary computation, vol. 8, no. 3, June 2004

[31] Dijkstra E., "A note on two problems in connexion with graphs", Numerische Mathematik 1: 269–271, 1959

Παράρτημα 1. Κώδικας προσομοίωσης

Π 1.1 - Αρχείο Deployment.java

```
import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.poifs.filesystem.POIFSFileSystem;
import org.apache.poi.hssf.usermodel.*;

public class Deployment {
        private static JFrame mFrame;
        private JPanel paramPanel;
        private DrawPanel graphPanel;
        private JButton Run;
        private JLabel
psoStepsLabel,psoAgentsLabel,psoFvalueLabel,psoKvalueLabel;
        private JTextField
psoStepsField,psoAgentsField,psoFvalueField,psoKvalueField;
        int psoSteps=50,psoAgentsNum=15;
```

```
        double psoFmaxValue=2.6,psoKvalue=0.9;
        public int[] globalBestSolution;
        public Vertex[] networkGraph;
        public int[][]  components;
        public double[] nodeProssCapacity, componentProssCost,
processTime,gBestArray=new double[200];
        public int[] assignedComponents;
        public double[][] networkArray, communicationCosts;

        public Deployment(){
                gui();
        };
        public static void main(String[] args) {
                new Deployment();
        }
        public void gui(){
                // Στήσιμο γραφικού περιβάλλοντος χρήστη
                mFrame=new JFrame("Components assignment");
                paramPanel=new JPanel();

        paramPanel.setBorder(BorderFactory.createTitledBorder(BorderFa
ctory.createLoweredBevelBorder(),"Parameters"));
                // Στήσιμο του πάνελ παραμέτρων
                GridBagLayout gBL=new GridBagLayout();
                paramPanel.setLayout(gBL);
                GridBagConstraints cs=new GridBagConstraints();
                //Προσθήκη συστατικών στο πλέγμα
                psoStepsLabel=new JLabel("Number of PSO steps: ");
                paramPanel.add(psoStepsLabel);
                cs.gridx=0;
                cs.gridy=0;
                cs.insets.left=2;
```

```
cs.insets.right=2;
gBL.setConstraints(psoStepsLabel,cs);

psoStepsField=new JTextField(3);
psoStepsField.setText("100");
paramPanel.add(psoStepsField);
cs.gridx=1;
cs.gridy=0;
gBL.setConstraints(psoStepsField,cs);

psoAgentsLabel=new JLabel("Number of PSO agents: ");
paramPanel.add(psoAgentsLabel);
cs.gridx=0;
cs.gridy=1;
gBL.setConstraints(psoAgentsLabel,cs);

psoAgentsField=new JTextField(3);
psoAgentsField.setText("50");
paramPanel.add(psoAgentsField);
cs.gridx=1;
cs.gridy=1;
gBL.setConstraints(psoAgentsField,cs);

psoFvalueLabel=new JLabel("Maximum F value: ");
paramPanel.add(psoFvalueLabel);
cs.gridx=0;
cs.gridy=2;
gBL.setConstraints(psoFvalueLabel,cs);

psoFvalueField=new JTextField(3);
psoFvalueField.setText("2.6");
paramPanel.add(psoFvalueField);
```

```
                cs.gridx=1;
                cs.gridy=2;
                gBL.setConstraints(psoFvalueField,cs);

                psoKvalueLabel=new JLabel("K value: ");
                paramPanel.add(psoKvalueLabel);
                cs.gridx=0;
                cs.gridy=3;
                gBL.setConstraints(psoKvalueLabel,cs);

                psoKvalueField=new JTextField(3);
                psoKvalueField.setText("0.6");
                paramPanel.add(psoKvalueField);
                cs.gridx=1;
                cs.gridy=3;
                gBL.setConstraints(psoKvalueField,cs);

                Run=new JButton("Execute");
                RunButtonListener dbl = new RunButtonListener();
                Run.addActionListener(dbl);
                paramPanel.add(Run);
                cs.gridx=0;
                cs.gridy=4;
                gBL.setConstraints(Run,cs);

                //Στήσιμο του πάνελ γραφικών
                graphPanel=new DrawPanel();
        graphPanel.setBorder(BorderFactory.createTitledBorder(BorderFac
tory.createLoweredBevelBorder(),"Network"));

                mFrame.getContentPane().setLayout(new BorderLayout());
                mFrame.setSize(760,650);
```

```
                mFrame.setLocation(200, 50);
                mFrame.getContentPane().add(paramPanel,
BorderLayout.WEST);
                mFrame.getContentPane().add(graphPanel);
                mFrame.setVisible(true);
                mFrame.setResizable(false);
        mFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        //ActionListener για το κουμπί Execute
        private class RunButtonListener implements ActionListener{
                public void actionPerformed (ActionEvent fileEvnt){
                        boolean problem = false;
                        //Έλεγχος αν οι τιμές των πεδίων που εισάγει ο
χρήστης είναι αριθμοί
                        try{
        psoSteps=Integer.parseInt(psoStepsField.getText().toString());
        psoAgentsNum=Integer.parseInt(psoAgentsField.getText().toString(
));
        psoFmaxValue=Double.parseDouble(psoFvalueField.getText().toSt
ring());
        psoKvalue=Double.parseDouble(psoKvalueField.getText().toString(
));
                        }
                        catch(NumberFormatException e){
                                /* Η μεταβλητή problem γίνεται αληθής για να
δηλώσει
                                 * ότι υπάρχει εξαίρεση και να γίνει μετά στον
κώδικα
                                 * κατάλληλη διαχείρηση.
                                 */
                                problem=true;
```

```
                    }
                    //Έλεγχος αν οι τιμές που εισήγαγε ο χρήστης είναι
αποδεκτές
                    if(psoSteps<1)
                            problem=true;
                    else if(psoAgentsNum<1)
                            problem=true;
                    else if(psoFmaxValue<=0)
                            problem=true;
                    else if(psoKvalue<=0 || psoKvalue>=1)
                            problem=true;

                    if (problem) JOptionPane.showMessageDialog(null,
"You must fill in all the boxes with proper \n values for the algorithm to
execute.", "Error", JOptionPane.WARNING_MESSAGE);
                    else{
                            readNetworkFile();
                            //Πίνακας που αποθηκεύει τις τιμές κόστους
των αποστάσεων όλων των κόμβωνgiven by Dijkstra's algorithm
                            //Every row represents the node's name and
the values in the columns the given distance costs
                            communicationCosts = new
double[networkArray.length][networkArray.length];
                            for(int node=0; node<networkArray.length;
node++){
                                    double[] distance = runDijkstra(node);
                                    for (int x=0; x<networkArray.length;
x++){
        communicationCosts[node][x]=distance[x];
                                    }
                            }
                            readComponentFile();
```

```
                    pso();
                graphPanel.draw(networkArray,globalBestSolution);
                }
            }
        }
        public void readNetworkFile(){
            int numOfRows=0;
            try{
                FileInputStream networkXSL = new
FileInputStream("network.xls");
                HSSFWorkbook wb = new
HSSFWorkbook(networkXSL);
                HSSFSheet sheet = wb.getSheetAt(0); //Πρώτο φύλλο
εργασίας
                HSSFSheet sheet2 = wb.getSheet("Sheet2");
//Δεύτερο φύλλο εργασίας
                HSSFRow row,row2;
                HSSFCell cell,cell2;
                numOfRows = 1+sheet.getLastRowNum(); //Για να
μετρήσει και το μηδενικό
                networkArray = new
double[numOfRows][numOfRows];
                nodeProssCapacity = new double[numOfRows];
                for(int i=0;i<numOfRows;i++){
                    row = sheet.getRow(i);
                    row2 = sheet2.getRow(i);
                    cell2 = row2.getCell((short) 0);

        nodeProssCapacity[i]=cell2.getNumericCellValue();
                    for (int j = 0; j < numOfRows; j++){
                        cell = row.getCell((short) j);
                        if (cell != null)
```

```
                                        networkArray[i][j] =
cell.getNumericCellValue();
                                }
                        }
                        networkXSL.close();
                }
                catch(Exception x){
                        System.err.println("File not found " + x.getMessage());
                }
        }
        public void readComponentFile(){
                int numOfRows=0;
                try{
                        FileInputStream componentsXSL = new
FileInputStream("components.xls");
                        HSSFWorkbook wb = new
HSSFWorkbook(componentsXSL);
                        HSSFSheet sheet = wb.getSheetAt(0); //Πρώτο φύλλο
εργασίας
                        HSSFSheet sheet2 = wb.getSheet("Sheet2");
//Δεύτερο φύλλο εργασίας
                        HSSFRow row,row2;
                        HSSFCell cell,cell2;
                        numOfRows = 1+sheet.getLastRowNum(); //Για να
μετρήσει και το μηδενικό
                        components = new int[numOfRows][10]; //Ένα
component μπορεί να επικοινωνεί το πολύ με άλλα 10
                        componentProssCost = new double
[components.length];
                        for(int i=0;i<numOfRows;i++){
                                row = sheet.getRow(i);
                                cell = row.getCell((short) 0);
```

```
                        row2 = sheet2.getRow(i);
                        cell2 = row2.getCell((short) 0);

        componentProssCost[i]=cell2.getNumericCellValue();
                        for (int j = 0; j < numOfRows; j++){
                                cell = row.getCell((short) j);
                                if (cell != null)
                                        components[i][j] = (int)
cell.getNumericCellValue();
                        }
                }
                componentsXSL.close();
        }
        catch(Exception x){
                System.err.println("File not found" + x.getMessage());
        }
    }
    public double[] runDijkstra(int node){
        int size = networkArray.length;
        double[] distances= new double[size];
        networkGraph = new Vertex[size];
        for(int i=0;i<size;i++){
            int counter=0;
            //Όνομα κορυφής
            networkGraph[i]= new Vertex(Integer.toString(i));
            //Εύρεση των ακμών της κάθε κορυφής
            for (int j = 0; j < size; j++){
                if (networkArray[i][j] != 0)
                    counter++;
            }
            networkGraph[i].adjacencies = new Edge[counter];
        }
```

```
                for(int i=0;i<size;i++){
                        int counter2=0;
                        for (int j = 0; j < size; j++){
                                if (networkArray[i][j] != 0){
                                        networkGraph[i].adjacencies[counter2]=
new Edge(networkGraph[j], networkArray[i][j]);
                                        counter2++;
                                }
                        }
                }
                Dijkstra.computePaths(networkGraph[node]);
                //for (Vertex ve : networkGraph)
                for(int i=0;i<size;i++){
                        //System.out.println("Distance to " + networkGraph[i] +
": " + networkGraph[i].minDistance);
                        //List<Vertex> path =
Dijkstra.getShortestPathTo(networkGraph[i]);
                        //System.out.println("Path: " + path);
                        distances[i] = networkGraph[i].minDistance;
                }
                return distances;
        }
        public double pso(){
                double globalBestCost=9999999;
                globalBestSolution= new int[components.length];
                double xi;

                //Αρχικοποίηση του σμήνους με λύσεις
                psoAgent[] agents = new psoAgent[psoAgentsNum];
                int numOfComponents= components.length;
                int numOfNetworkNodes= networkArray.length;
```

```
                for(int i=0;i<psoAgentsNum;i++){
                        //Το assignedComponents αποθηκεύει την τιμή του
κόμβου που έχει ανατεθεί το κάθε compnent
                        assignedComponents = new int[numOfComponents];
                        for(int j=0;j<numOfComponents;j++){
                                int randomNetNode = (int)
(numOfNetworkNodes * Math.random());
                                assignedComponents[j]=randomNetNode;
                                agents[i] = new psoAgent(i, psoAgentsNum,
assignedComponents, networkArray.length);
                        }
                }
                //Αλγόριθμος
                for(int stpCnt=0;stpCnt<psoSteps;stpCnt++){
                        //System.out.println("PSO Step: "+stpCnt);
                        //Για κάθε σωματίδιο
                        for(int agNum=0;agNum<psoAgentsNum;agNum++){
                                //System.out.println("Agent: " + agNum);
                                int[] agentAssignment =
agents[agNum].getSolution();
                                //Υπολογισμός χρόνου επεξεργασίας για κάθε
κόμβο
                                processTime = new double
[networkArray.length];
                                //Για κάθε component
                                for(int i=0;i<agentAssignment.length;i++){
                                        int node = agentAssignment[i];
                                        processTime[node] =
processTime[node] + (componentProssCost[i]/nodeProssCapacity[node]);
                                }
                                //Υπολογισμός μέγιστου χρόνου επεξεργασίας
                                double maxProcTime=0;
```

```
                    for(int i=0;i<processTime.length;i++){
                        if(processTime[i]>maxProcTime){
                            maxProcTime = processTime[i];
                        }
                    }
                    //Υπολογισμός κόστους επικοινωνίας
                    double totalLinkCost = 0;
                    for(int i=0;i<numOfComponents;i++){
                        int nodeA = agentAssignment[i];
                        for(int j=0;j<10;j++){ //Υπάρχουν το πολύ
10 συνδέσεις ανά component
                            if(components[i][j]!=0){
                                int temp = components[i][j];
                                int nodeB =
agentAssignment[temp];
                                totalLinkCost =
totalLinkCost+=(communicationCosts[nodeA][nodeB]);
                            }
                        }
                    }
                    totalLinkCost/=2;
                    double solutionTotalCost = maxProcTime +
totalLinkCost;
                    //Ενημέρωση της τιμής του προσωπικού
βέλτιστου

        if(solutionTotalCost<=agents[agNum].getPerBestCost()){
        agents[agNum].setPerBest(solutionTotalCost,agentAssignment);
                    }
                    //Ενημέρωση της τιμής του ολικού βέλτιστου
                    if(solutionTotalCost<=globalBestCost){
                        globalBestCost=solutionTotalCost;
```

```
                                        for(int
x=0;x<agentAssignment.length;x++){
        globalBestSolution[x]=agentAssignment[x];
                                        }
                                }
                        }
                        for(int agNum=0;agNum<psoAgentsNum;agNum++){
                                //Ρύθμιση της ταχύτητας του σωματιδίου
                                double f1=psoFmaxValue*Math.random();
                                double f2=psoFmaxValue*Math.random();
                                double f=f1+f2;
                                if(f>4){
                                        xi=(2*psoKvalue)/(f-
2+Math.sqrt(Math.pow(f,2)-4*f));
                                }
                                else{
                                        xi=Math.sqrt(psoKvalue);
                                }

        agents[agNum].setNewPosition(f1,f2,xi,globalBestSolution);
                        }
                }
                System.out.println("Best solution: ");
                for(int i=0;i<components.length;i++){
                        System.out.print(globalBestSolution[i]);
                }
                System.out.println("Total Cost: " + globalBestCost);
                return globalBestCost;
        }
}
```

Π 1.2 - Αρχείο Dijkstra.java

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.PriorityQueue;

class Vertex implements Comparable<Vertex>{
    public final String name;
    public Edge[] adjacencies;
    public double minDistance = Double.POSITIVE_INFINITY;
    public Vertex previous;
    public Vertex(String argName) { name = argName; }
    public String toString() { return name; }
    public int compareTo(Vertex other)
    {
        return Double.compare(minDistance, other.minDistance);
    }
}
class Edge{
    public final Vertex target;
    public final double weight;
    public Edge(Vertex argTarget, double argWeight){
         target = argTarget; weight = argWeight; }
}

public class Dijkstra {
        public static void computePaths(Vertex source){
            source.minDistance = 0.;
            PriorityQueue<Vertex> vertexQueue = new
PriorityQueue<Vertex>();
            vertexQueue.add(source);
```

```
            while (!vertexQueue.isEmpty()){
                Vertex u = vertexQueue.poll();

                // Επισκέψου κάθε ακμή που ξεκινά από το u
                for (Edge e : u.adjacencies){
                    Vertex ver = e.target;
                    double weight = e.weight;
                    double distanceThroughU = u.minDistance + weight;
                    if (distanceThroughU < ver.minDistance){
                            vertexQueue.remove(ver);
                        ver.minDistance = distanceThroughU ;
                        ver.previous = u;
                        vertexQueue.add(ver);
                    }

                }
            }
        }
        public static List<Vertex> getShortestPathTo(Vertex target){
            List<Vertex> path = new ArrayList<Vertex>();
            for (Vertex vertex = target; vertex != null; vertex =
vertex.previous)
                path.add(vertex);

            Collections.reverse(path);
            return path;
        }
}
```

Π 1.3 - Αρχείο psoAgent.java

```
public class psoAgent {
	private int agNum, swarmSize, netSize;
	private int[] solution,pBestSolution;
	private double pBest;
	private int agVel[];
	public psoAgent(int number,int swSize, int[] assignment, int networkSize){
		agNum=number;
		swarmSize=swSize;
		solution=assignment;
		pBest=9999999;
		netSize=networkSize;
		agVel = new int [assignment.length];
		pBestSolution=new int[assignment.length];
		//Αρχικοποίηση της ταχύτητας με μία τιμή στο διάστημα [1,5]
		for(int i=0;i<solution.length;i++){
			agVel[i] =(int) (1+Math.random()*4);
		}
	}
	public int[] getSolution(){
		return solution;
	}
	public double getPerBestCost(){
		return pBest;
	}
	public int[] getPerBestSolution(){
		return pBestSolution;
	}
	public void setPerBest(double value, int[] agentAssignment){
		pBest=value;
```

```
                pBestSolution=agentAssignment;
        }
        public void setNewPosition(double f1,double f2,double xi,int[]
gBestPos){
                for(int i=0;i<solution.length;i++){
                        //Υπολογισμός ταχύτητας σωματιδίου
                        agVel[i]=(int)
Math.round(xi*(agVel[i]+f1*(pBestSolution[i]-solution[i])+f2*(gBestPos[i]-
solution[i])));
                        //Υπολογισμός θέσης σωματιδίου
                        solution[i] = solution[i] + agVel[i];
                        if(solution[i]<0){
                                solution[i]=0;
                        }
                        else if(solution[i]>netSize-1){
                                solution[i]=netSize-1; //Μέτρηση από το μηδέν
                        }
                }

        }
}
```

Π 1.4 - Αρχείο DrawPanel.java

```
import java.awt.*;

import javax.swing.*;

public class DrawPanel extends JPanel{
	private int nodesNum;
	private double[][] links;
	public int[] bestSolution;
	String dispSolution;
	public void paint(Graphics g){
		super.paint(g);
		int[][] layout = new int[nodesNum][2];
		for(int i=0;i<nodesNum;i++){
			layout[i][0]=(int) (20 + Math.random()*500);
			layout[i][1]=(int) (20 + Math.random()*500);
			g.setColor(Color.BLACK);
			//Σχεδίαση του κόμβου
			g.fillRect(layout[i][0],layout[i][1],20,20);
			g.setColor(Color.WHITE);
			//Σχεδίαση του ενόματος κόμβου

	g.drawString(Integer.toString(i),(layout[i][0]+5),(layout[i][1]+15));
			}
		g.setColor(Color.BLACK);
		//StringBuffer sb=new StringBuffer(40);
		//sb.append("Selected nodes are:");
		for(int i=0;i<nodesNum;i++){
			for(int j=0;j<nodesNum;j++){
				if(links[i][j]!=0){
					//Σχεδίαση συνδέσμου κόμβου
```

```
                                        g.drawLine(layout[i][0]+10,
layout[i][1]+10, layout[j][0]+10, layout[j][1]+10);
                                        //Σχεδίαση κόστους συνδέσμου

        g.drawString(Double.toString(links[i][j]),((layout[j][0]+layout[i][0])/2),(
(layout[j][1]+layout[i][1])/2));
                                        //Add node to solution string
                                        //sb.append(" "+bestSolution[i]);
                                    }
                                }
                            }
                            if(bestSolution!=null){
                                g.setColor(Color.RED);
                                for(int z=0;z<bestSolution.length;z++){
                                    g.drawString(Integer.toString(z),
layout[bestSolution[z]][0], layout[bestSolution[z]][1]);
                                }
                                //dispSolution=sb.toString();
                                //System.out.println(dispSolution);
                                //g.drawString(dispSolution, 500, 10);
                            }
                        }

                        public void draw(double[][] nodes, int[] solution){
                            nodesNum=nodes.length;
                            links=nodes;
                            bestSolution=solution;
                            repaint();
                        }
}
```