ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μετα-ευρεστικές Μέθοδοι Αποικίας Μυρμηγκιών και Σμήνους Σωματιδίων: Εφαρμογή στο Πρόβλημα της Εύρεσης Βαρύτερης Κλίκας σε Γράφο.

Τιμόθεος Π. Μισίκος

Επιβλέπων: **Παναγιώτης Σταματόπουλος**, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ
Μάρτιος 2017

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μετα-ευρεστικές Μεθόδοι Αποικίας Μυρμηγκιών και Σμήνους Σωματιδίων: Εφαρμογή στο Πρόβλημα της Εύρεσης Βαρύτερης Κλίκας σε γράφο.

Τιμόθεος Π. Μισίκος

Α.Μ.: 1115200900258

Επιβλέπων: Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Στην παρούσα έρευνα γίνεται σύγκριση δύο μετα-ευρεστικών αλγορίθμων βελτιστοποίησης της συλλογικής νοημοσύνης, της Αποικίας μυρμηγκιών (ACO), που προτάθηκε από τον Marco Dorigo και της νέας εκδοχής του Δυαδικού Σμήνους σωματιδίων (NPSO), που εκδόθηκε από τους Khanesar, Teshnehlab και Shoorehdeli. Το πρόβλημα στο οποίο γίνεται η σύγκριση είναι αυτό της βαρύτερης κλίκας γράφου με βάρη στους κόμβους (MVWCP) το οποίο είναι NP-Hard. Σε αυτό το έγγραφο γίνεται ανάλυση των εκδοχών των προαναφερθέντων αλγορίθμων, οι οποίοι δοκιμάστηκαν σε ένα εύρος γράφων από εκατό μέχρι χίλιους κόμβους συμπεριλαμβανομένων κάποιων από των DIMACS benchmark instances.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη, Θεωρία γράφων
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Βαρύτερη κλίκα γράφου, PSO, ACO, Μετα-ευρεστικές μέθοδοι αναζήτησης, αλγόριθμοι εμπνευσμένοι από την φύση.

ABSTRACT

In this research, two meta-heuristic optimisation algorithms are compared. The Ant Colony Optimisation (ACO) introduced by Marco Dorigo and a newer version of the Binary Particle Swarm Optimization (NPSO), suggested by Khanesar, Teshnehlab, and Shoorehdeli (NPSO). The comparison is applied to the problem of the Maximum Vertex Weight Clique Problem (MVWCP), which is an NP-Hard problem. In this paper, there is an in-depth analysis of the above-mentioned algorithms, which were tested on graphs of one hundred to one thousand vertices, including some graphs of the well-known DIMACS benchmark instances.

SUBJECT AREAS: Artificial Intelligence, Graph Theory
KEYWORDS: Maximum Weight Clique, PSO, ACO, Meta-heuristic search methods, nature-inspired algorithms

Αφιερώνεται στην γυναίκα μου
Χάρις Μουγκράκη για την μεγάλη
της συμπαράσταση και βοήθεια
τα ακαδημαϊκά έτη 2015 – 2017

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα επικ. καθ. Παναγιώτη Σταματόπουλο για την υπομονή του, την γρήγορη ανταπόκρισή του στις ανάγκες της ερευνητικής αυτής εργασίας, τους γονείς μου που έθεσαν σωστά θεμέλια για την πορεία της ζωής μου, την γυναίκα μου για την υποστήριξή της σε κάθε δυσκολία, και τον Θεό μου για το μυαλό, την υγεία και όλα τα παραπάνω.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η εργασία αυτή έχει ως κύριο στόχο της την παρουσίαση δύο μετα-ευρεστικών αλγορίθμων, της Αποικίας Μυρμηγκιών και του Σμήνους Σωματιδίων, για την επίλυση του προβλήματος της βαρύτερης κλίκας με βάρη στους κόμβους. Οι παραπάνω αλγόριθμοι υλοποιήθηκαν σε Python, και έγινε μελέτη της συμπεριφοράς και των αποτελεσμάτων τους. Η παρούσα έρευνα διενεργήθηκε στα πλαίσια εκπόνησης πτυχιακής εργασίας στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών της Σχολής Θετικών Επιστημών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών, υπό την επίβλεψη του καθηγητή Σταματόπουλου Παναγιώτη, τον οποίο ευχαριστώ για την άμεση ανταπόκριση του στα ερωτήματα μου για την πρόοδο του έργου.

1. ΕΙΣΑΓΩΓΗ

1.1. Περιγραφή του προβλήματος

Η αναζήτηση της βαρύτερης κλίκας γράφου (Maximum Weight Clique) αποτελεί ένα κλασσικό πρόβλημα της θεωρίας γράφων. Η λύση στο πρόβλημα αυτό είναι ένα σύνολο κόμβων που συνδέονται όλοι μεταξύ τους και έχουν το μέγιστο δυνατό βάρος, όπου το βάρος μετριέται είτε σε βάρος πλευράς είτε σε βάρος κόμβου. Στην συγκεκριμένη ερευνητική εργασία τα βάρη είναι στους κόμβους (Maximum Vertex Weight Clique Problem), στο εξής MVWCP.

Το πρόβλημα αυτό αποτελεί γενίκευση του προβλήματος της μεγαλύτερης κλίκας (Maximum Clique Problem – MCP). Έχει αποδειχθεί ότι η εκδοχή του προβλήματος ως πρόβλημα απόφασης (Clique Decision Problem ή K-Clique Problem) είναι NP-Complete. Συνεπώς, το πρόβλημα αναζήτησης βαρύτερης κλίκας γράφου έχει τουλάχιστον NP-Complete επίπεδο δυσκολίας [1].

Εάν υποθέσουμε ότι έχουμε άπειρη υπολογιστική ισχύ και μνήμη, το πρόβλημα θα μπορούσε να επιλυθεί με τον εξής απλό τρόπο:

1. Βρες όλες τις τοπικά μέγιστες κλίκες του γράφου.
2. Υπολόγισε τα βάρη τους.
3. Επέστρεψε την βαρύτερη κλίκα.

Βεβαίως το παραπάνω δεν αποτελεί καλή λύση και στην πραγματικότητα αυτός ο αλγόριθμος δεν είναι κατάλληλος ούτε για προβλήματα 100 κόμβων, λόγω της εκθετικής πολυπλοκότητας του βήματος 1.

Πολλά σύγχρονα προβλήματα μπορούν να αναπαρασταθούν και να επιλυθούν σαν προβλήματα αναζήτησης βαρύτερης κλίκας γράφου. Παραδείγματος χάριν, στον τομέα της τεχνητής όρασης (computer vision), μπορεί να χρησιμοποιηθεί σαν αναζήτηση ταιριάσματος εικόνων (image matching) [2].

1.2. Μέθοδοι και αλγόριθμοι που έχουν χρησιμοποιηθεί

Σε αύτη την υποενότητα γίνεται μία αναφορά των κύριων μεθόδων που έχουν χρησιμοποιηθεί για την επίλυση του MVWCP. Στις περισσότερες περιπτώσεις έχουν χρησιμοποιηθεί είτε συνδυαστικά μεταξύ τους είτε με άλλες τεχνικές, ευρεστικές ή μετά-ευρεστικές μεθόδους.

1.2.1. Αναζήτηση με οπισθοδρόμηση (Backtracking)

Μία από τις πλέον χρησιμοποιούμενες μεθόδους σε προβλήματα αναζήτησης είναι η αναζήτηση με οπισθοδρόμηση. Ο αλγόριθμος αυτός σταδιακά δημιουργεί υποψήφιες λύσεις, τις οποίες ελέγχει σε κάθε βήμα επέκτασης τους. Όταν κάποια υποψήφια λύση δεν είναι έγκυρη ο αλγόριθμος δεν συνεχίζει να την αναπτύσσει αλλά επιστρέφει (backtracks) στην τελευταία διακλάδωση με επόμενους κλάδους που δεν έχουν δοκιμαστεί ακόμα, και η αναζήτηση συνεχίζεται από εκεί με όμοιο τρόπο μέχρι να βρεθεί κάποια λύση.

1.2.2. Μέθοδος διακλάδωσης και αποκοπής (Branch and bound)

Αυτή η μέθοδος χρησιμοποιείται από τους περισσότερους αλγορίθμους που εγγυώνται βέλτιστη λύση στο συγκεκριμένο πρόβλημα. Σημαντική είναι η προσφορά του Παναγιώτη Παρδαλού [3] που εισήγαγε αυτήν την μέθοδο στο πρόβλημα της μέγιστης κλίκας (MCP). Κατά την μέθοδο αυτή, σταδιακά «χτίζεται» ένα δέντρο αναζήτησης διαφορετικών λύσεων. Στον αλγόριθμο αυτό ορίζεται και μία ευρεστική συνάρτηση για τον υπολογισμό μεγαλύτερου (ή μικρότερου σε άλλα προβλήματα) ορίου κάθε κλάδου του δέντρου. Ο αλγόριθμος σε κάθε βήμα επεκτείνει τον πιο πολλά υποσχόμενο κλάδο, όταν βρεθεί κάποια «λύση», ο αλγόριθμος δεν σταματά αλλά συνεχίζει την επέκταση των κλάδων που «υπόσχονται» καλύτερη λύση, ενώ ταυτόχρονα αποκόπτει τους κλάδους που δεν είναι πλέον ανταγωνιστικοί σύμφωνα με την ευρεστική συνάρτηση ή την ήδη υπάρχουσα λύση. Με αυτόν τον τρόπο γίνεται σημαντική εξοικονόμηση πόρων και υπάρχει εγγύηση της βέλτιστης λύσης.

1.2.3. Χρωματισμός κόμβων γράφου (Vertex Colouring)

Έστω ότι έχουμε έναν γράφο και θέλουμε να χρωματίσουμε τους κόμβους του με όσο το δυνατόν λιγότερα χρώματα, όμως κανένας κόμβος δεν μπορεί να έχει ίδιο χρώμα με κάποιο γειτονικό του. Ο ελάχιστος αριθμός χρωμάτων που μπορεί να χρησιμοποιηθεί σε ένα γράφο ονομάζεται χρωματικός αριθμός γράφου και ισχύει ότι δεν υπάρχει καμία κλίκα με περισσότερους κόμβους από αυτόν τον αριθμό. Το πρόβλημα αυτό είναι επίσης NP-hard, παρόλα αυτά έχει χρησιμοποιηθεί συνδυαστικά με άλλες μεθόδους ώστε να δώσει το μέγιστο αριθμό κόμβων κλίκας γράφου ή υπογράφων του γράφου.

1.2.4. Αναζήτηση με απαγορεύσεις (Tabu Search)

Αυτή η μέθοδος αναζήτησης υπάγεται στην κατηγορία των μεθόδων τοπικής αναζήτησης, δηλαδή δοθέντος μίας αρχικής «λύσης», η μέθοδος αυτή αναζήτα την βέλτιστη ή έστω μία καλύτερη. Το ιδιαίτερο χαρακτηριστικό της αναζήτησης με απαγορεύσεις είναι ότι μέσω των απαγορεύσεων εμποδίζει την επαναλαμβανόμενη αναζήτηση των «περιοχών» που προσπέλασε – αναζήτησε πρόσφατα. Ο αλγόριθμος διατηρεί βραχυπρόθεσμα τις πιο πρόσφατες αλλαγές και απαγορεύει την ακύρωση τους στο μέλλον. Υπάρχουν επίσης εκδοχές όπου υπάρχει εκτός από την βραχυπρόθεσμη μνήμη, υπάρχει μία ενδιάμεση ώστε να εντατικοποιήσουν (intensification) την αναζήτηση προς μία πιο πολλά υποσχόμενη περιοχή του χώρου αναζήτησης ή μία μακροπρόθεσμη ώστε να ενθαρρύνουν την διαφοροποίηση (diversification) των λύσεων με αναζήτηση σε όλο το εύρος του χώρου αναζήτησης.

1.3. Εκδοχές και λύσεις στο παρελθόν

Αρκετές προσπάθειες έχουν γίνει για να βρεθεί μία «καλή» κλίκα μέσα σε ένα εύλογο χρονικό διάστημα. Αυτές οι προσπάθειες μπορούν εύκολα να κατηγοριοποιηθούν σε απόλυτους – ακριβείς αλγορίθμους και αλγορίθμους που χρησιμοποιούν ευρεστικές μεθόδους.

1.3.1. Απόλυτοι – Ακριβείς αλγόριθμοι

Αυτοί οι αλγόριθμοι εγγυώνται βέλτιστη λύση, αλλά σε πολύ μεγάλους ή πολύπλοκους γράφους είναι εύκολο να αποτύχουν λόγω της εκθετικής πολυπλοκότητας τους.

Μέχρι σήμερα έχουν εκδοθεί διάφοροι τέτοιοι αλγόριθμοι όπως ο αλγόριθμος του Östergård [4] ο οποίος συνδυάζει τον αλγόριθμο διακλάδωσης και αποκοπής για «κλάδεμα» των κακών λύσεων και τον αλγόριθμο χρωματισμού κόμβων γράφου (vertex colouring). Επίσης το 2004, ο Kumlander [5] εισήγαγε έναν αλγόριθμο με χρήση της αναζήτησης με οπισθοδρόμηση σε δέντρο (backtrack tree search) και του χρωματισμού κόμβων γράφου.

Οι Tomita και Seki [6] επίσης πρότειναν έναν αλγόριθμο διακλάδωσης και αποκοπής, ο οποίος χρησιμοποιεί μία ευρεστική συνάρτηση ταξινόμησης κόμβων για διαχωρισμό των ανεξάρτητων συνόλων. Αυτός ο αλγόριθμος βελτιώθηκε ακόμα περισσότερο με τον δυναμικό υπολογισμό του βαθμού των κόμβων, ο οποίος οδήγησε στον αλγόριθμο MaxCliqueDyn του Konc του Janezic το 2007 [7].

Οι πλέον πρόσφατες εκδοχές είναι των Wu and Hao [8] οι οποίοι παρουσίασαν νέες τεχνικές στον αλγόριθμο διακλάδωσης και αποκοπής επίσης με χρήση χρωματισμού γράφου και ταξινόμησης, και των Fang, Li and Xu [9] οι οποίοι παρουσίασαν έναν αλγόριθμο που χρησιμοποιεί την λογική της μέγιστης ικανοποίησης (Maximum Satisfiability Reasoning) σαν περιοριστική μέθοδο. Αυτοί οι αλγόριθμοι επιστρέφουν βέλτιστη λύση.

1.3.2. Ευρεστικοί αλγόριθμοι

Σε αυτήν την κατηγορία υπάγονται οι αλγόριθμοι που αναζητούν λύση με κάποιο προσεγγιστικό τρόπο – με ευρεστικούς κανόνες. Τέτοιοι κανόνες μπορεί να χρησιμοποιούνται και στους αλγορίθμους της προηγούμενης κατηγορίας, όμως σε αυτήν την περίπτωση οι αλγόριθμοι δεν εγγυώνται βέλτιστες λύσεις άλλα δουλεύουν και φέρνουν αποτελέσματα μέσα σε ένα εύλογο χρονικό διάστημα.

Οι Bomze, Pelillo και Stix [10] υλοποίησαν και δοκίμασαν έναν αλγόριθμο με χρήση παράλληλης, κατανεμημένης ευρεστικής συνάρτησης με δυναμικούς κανόνες. Επίσης ο Wayne Pullan επέκτεινε τον αλγόριθμο Phased Local Search που πρότεινε για τον MCP ώστε να δουλεύει για το MWVCP το 2008 [11].

Τέλος πολλοί αλγόριθμοι τοπικής αναζήτησης έχουν προταθεί όπως ο MN/NT του Wu, Hao και Glover [12], ο αλγόριθμος τοπικής αναζήτησης με ισχυρό έλεγχο διαμόρφωσης - σχηματισμού για αποφυγή κυκλικής αναζήτησης (LSCC), και μία ακόμα πιο βελτιωμένη εκδοχή του τελευταίου αλγορίθμου με χρήση μίας ακόμα ευρεστικής συνάρτησης που λέγεται Best from Multiple Selection (LSCC +BMS) [13].

Στα πλαίσια αυτής της εργασίας χρησιμοποιήθηκαν δύο μετά-ευρεστικοί αλγόριθμοι, που μιμούνται συμπεριφορές που βρίσκουμε στην φύση. Στις επόμενες ενότητες θα γίνει παρουσίαση των δύο αλγορίθμων, και στο τέλος υπάρχει η σύγκριση των αποτελεσμάτων σε γράφους.

2. ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΜΕ ΣΜΗΝΟΣ ΣΩΜΑΤΙΔΙΩΝ

2.1. Γενική περιγραφή του αλγορίθμου βελτιστοποίησης με σμήνος σωματιδίων

Η βελτιστοποίηση με χρήση σμήνους σωματιδίων, στο εξής PSO, μιμείται την κίνηση πουλιών, ψαριών ή μελισσών και κατηγοριοποιείται στις μετά-ευρεστικές μεθόδους βελτιστοποίησης. Εν προκειμένω, ένα προκαθορισμένο πλήθος ψηφιακών πρακτόρων, τα «σωματίδια», κινούνται στο χώρο ψευδοτυχαία με σκοπό να βρουν την καλύτερη θέση σύμφωνα με κάποια συνάρτηση αξιολόγησης (fitness function). Η κίνηση στο χώρο επηρεάζεται από την ήδη υπάρχουσα πορεία κάθε σωματιδίου, λόγω της αδράνειας μάζας, τη βέλτιστη θέση που έχει βρει το ίδιο και την βέλτιστη θέση που έχει βρεθεί από όλα τα σωματίδια [14], δηλαδή από το σμήνος. Με αυτόν τον τρόπο υπάρχει η ιδέα ότι σε κάποια «καλή» λύση είναι πολύ πιθανό να υπάρχουν «κοντινές» καλύτερες λύσεις. Το τέλος του αλγορίθμου είναι, αργά ή γρήγορα, να γίνει σύγκλιση στην καλύτερη λύση που έχει βρεθεί, η οποία μπορεί να είναι ένα τοπικό ελάχιστο ή μέγιστο ανάλογα με την περίπτωση. Στην δική μας περίπτωση η καλύτερη λύση είναι η βαρύτερη λύση.

Η αρχική έκδοση του αλγορίθμου προτάθηκε από τους Kennedy και Eberhart το 1995 [15] και αποτελεί μέθοδο επίλυσης προβλημάτων σε διαστάσεις συνεχόμενων τιμών. Οι προαναφερθέντες πρότειναν μία διακριτή – δυαδική εκδοχή του αλγορίθμου για επίλυση προβλημάτων δυαδικών χώρων αναζήτησης [16], στο εξής BPSO. Η βασική διαφορά μεταξύ των PSO και BPSO είναι ότι στην δεύτερη μέθοδο βελτιστοποίησης η επιτάχυνση των σωματιδίων ορίζεται με χρήση πιθανοτήτων για το εάν η τιμή κάποιας διάστασης θα είναι 0 ή 1, συνεπώς γίνεται χρήση κάποιας συνάρτησης $f(x): R \rightarrow [0,1]$ για κανονικοποίηση. Τέλος, η μάζα των σωματιδίων ορίζεται σε κάθε γύρο με τυχαίο τρόπο, μεταξύ 0 και 1. Έχει προταθεί βέβαια και η ιδέα να μην υπάρχει καθόλου η παράμετρος της μάζας [17] αλλά δεν έχει γίνει κάποια βαθύτερη έρευνα πάνω σε αυτό.

Στην δική μας περίπτωση χρησιμοποιήθηκε μία ακόμα πιο πρόσφατη εκδοχή του BPSO η οποία έχει καλύτερα αποτελέσματα από το κλασσικό BPSO [18]. Στην συγκεκριμένη περίπτωση η επιτάχυνση ορίζεται από δύο επιταχύνσεις – τάσεις, αυτήν που τείνει στο 0 και αυτήν που τείνει στο 1. Όσον αφορά την μάζα στην συγκεκριμένη περίπτωση είναι σταθερή όπως και στο PSO. Αυτός είναι ο αλγόριθμος που περιγράφεται παρακάτω, στο εξής NPSO.

2.2. Μορφή του αλγορίθμου για το πρόβλημα της βαρύτερης κλίκας γράφου

Έστω το στοιχείο i και η διάσταση j, ορίζουμε ως $x_{ij}(t)$ την θέση που βρίσκεται το στοιχείο i στην διάσταση j στον γύρο t. Επίσης ορίζουμε την επιτάχυνση $V_{ij}^0, V_{ij}^1 \in \mathbb{R}$ ως την επιτάχυνση του στοιχείου i στην διάσταση j προς το 0 και το 1 αντίστοιχα. Η θέση κάθε στοιχείου είναι της μορφής $(x_{i0}, x_{i1}, x_{ij}, \dots, x_{in})$ όπου n ισούται με τον αριθμό των κόμβων του γράφου, και τα $x_{ij} = 0$ ή $x_{ij} = 1 \; \forall \; i, j$.

Ισχύουν οι παρακάτω κανόνες διαχείρισης της επιτάχυνσης και θέσεως κάθε στοιχείου:

Κανόνας 1:

$$V_{ij}^1(t) = w\, V_{ij}^1(t-1) + d_{ij,1}^1 + \; d_{ij,2}^1$$
$$V_{ij}^0(t) = w\, V_{ij}^0(t-1) + d_{ij,1}^0 + \; d_{ij,2}^0$$

Σχετικά με τα $d_{ij,1}^0,\; d_{ij,2}^0,\; d_{ij,1}^1,\; d_{ij,2}^1$ ισχύει ότι:

$$Εάν\; P_{ibest}^j = 1\; τότε\; d_{ij,1}^1 = c_1 r_1\; και\; d_{ij,1}^0 = -c_1 r_1$$

$$Εάν\; P_{ibest}^j = 0\; τότε\; d_{ij,1}^0 = c_1 r_1\; και\; d_{ij,1}^1 = -c_1 r_1$$

$$Εάν\; P_{gbest}^j = 1\; τότε\; d_{ij,2}^1 = c_2 r_2\; και\; d_{ij,2}^0 = -c_2 r_2$$

$$Εάν\; P_{gbest}^j = 0\; τότε\; d_{ij,2}^0 = c_2 r_2\; και\; d_{ij,2}^1 = -c_2 r_2$$

Όπου $P_{ibest}^j\; και\; P_{gbest}^j$ είναι η τιμή της διάστασης της καλύτερης θέσης του στοιχείου i και του σμήνους αντίστοιχα. Τα $d_{ij}^0\; d_{ij}^1$ είναι προσωρινές μεταβλητές , τα $r_1, r_2 \in [0,1]$ τυχαίες μεταβλητές που αλλάζουν σε κάθε γύρο και c_1, c_2 μεταβλητές που ορίζονται από το χρήστη στην αρχή του προγράμματος.

Κανόνας 2:

Για την τελική επιτάχυνση ανά γύρο ισχύει:

$$V_{ij}^C = \begin{cases} V_{ij}^1, & if\; x_{ij} = 0 \\ V_{ij}^0, & if\; x_{ij} = 1 \end{cases}$$

Κανόνας 3:

Η νέα θέση του στοιχείου ανά διάσταση ορίζεται ως εξής:

$$x_{ij}(t+1) = \begin{cases} \overline{x_{ij}}\ (t)\,, & if\ \ r_{ij} < V'_{ij} \\ x_{ij}(t)\,, & if\ \ r_{ij} > V'_{ij} \end{cases}$$

Όπου $\mathrm{V'_{ij}} = \mathrm{sigmoid}(\mathrm{V_{ij}^{C}}) = \frac{1}{1+\mathrm{e}^{-\mathrm{V_{ij}^{C}}}}$. Χρησιμοποιείται η σιγμοειδής συνάρτηση ώστε να γίνει κανονικοποίηση των τιμών στο $[0,1]$. Επίσης το $\overline{\mathrm{x_{ij}(t)}}$ σημαίνει μετατροπής της τιμής 0 σε 1 και της τιμής 1 σε μηδέν, αν ισχύει η συνθήκη, ενώ το $\mathrm{x_{ij}(t)}$ σημαίνει διατήρηση της τιμής στον επόμενο γύρο.

Με τους παραπάνω κανόνες ο αλγόριθμος έχει την παρακάτω μορφή:

Παράμετροι : αριθμός στοιχείων, γράφος, γύροι, βάρος – μάζα, βάρος εμπειρίας στοιχείου, βάρος εμπειρίας σμήνους, c_1, c_2,

1. Θέσε κάθε στοιχείο του σμήνους σε έναν από τους κόμβους τυχαία.
2. Μέχρι την λήξη (τέλος γύρων ή σύγκλιση στοιχείων) κάνε:
    a. Δημιούργησε τυχαία μεταβλητή $r_{ij} \in [0,1]$.
    b. Για κάθε στοιχείο ι:
        i. Αξιολόγησε την θέση x_i κάθε στοιχείου.
        ii. Θέσε το $x_i\ \sigma\alpha\nu\ P_{ibest}$ αν η νέα θέση είναι βαρύτερη απ' την προηγούμενη βέλτιστη του στοιχείου.
        iii. Θέσε το $x_i\ \sigma\alpha\nu\ Pgbest$ αν η νέα θέση έχει το μεγαλύτερο βάρος.
        iv. Ανανέωσε τα $V_i^0\ \kappa\alpha\iota\ V_i^1$ σύμφωνα με τον κανόνα (1).
        v. Υπολόγισε το V^C κατά τον κανόνα (2).
        vi. Μετακίνησε το στοιχείο στην νέα θέση κατά τον κανόνα (3).
3. Εκτύπωσε αποτελέσματα.

Σχήμα 1: Ο Αλγόριθμος του NPSO

2.3. Σύγκριση αποτελεσμάτων των διάφορων εκδοχών που υλοποιήθηκαν

Ο παραπάνω αλγόριθμος στην αρχική του δομή είχε πολύ κακά αποτελέσματα ακόμα και σε γράφους διαστάσεων 10 κόμβων. Εξαιτίας αυτού, υλοποιήθηκαν κάποιες ευρεστικές μέθοδοι για να γίνουν τα σωματίδια «εξυπνότερα» όσον αφορά την επιλογή των θέσεων. Δοκιμάστηκαν, επίσης, διαφορετικές εκδοχές για την ανταπόκριση του προγράμματος σε περίπτωση σύγκλισης.

Συνεπώς έχουμε, σε περίπτωση σύγκλισης:

- Καμία ενέργεια: το πρόγραμμα τρέχει μέχρι τέλους παρόλο που δεν πρόκειται να εμφανιστούν νέες λύσεις ή κίνηση.
- Τερματισμός: στην συγκεκριμένη περίπτωση το πρόγραμμα τελειώνει πριν ολοκληρώσει όλους τους γύρους.
- Επανατοποθέτηση: σε περίπτωση σύγκλισης θέσε τα στοιχεία σε νέες τυχαίες θέσεις διατηρώντας όμως τις τοπικά και ολικά βέλτιστες τιμές.

Σχετικά με τις ευρεστικές συναρτήσεις έχουμε τις παρακάτω επιλογές:

- Καμία ενέργεια: το πρόγραμμα τρέχει ακριβώς όπως έχει αναφερθεί παραπάνω.
- Προτροπή γειτόνων: Η ευρεστική συνάρτηση προσθέτει στο V_{ij}^{C} μία ορισμένη από το χρήστη σταθερά, για να αυξήσει τις πιθανότητες επιλογής κόμβων που οδηγούν σε μεγαλύτερες κλίκες (μόνο αν το V_{ij}^{C} είναι το V_{ij}^{1}). Η σταθερά αυτή επηρεάζει την τιμή μόνο για τον συγκεκριμένο γύρο, δηλαδή δεν προστίθεται στο V_{ij}^{1} για να επηρεάσει επόμενους γύρους. Αυτό υπολογίζεται με τον ίδιο τρόπο όπως οι γείτονες της κλίκας στην ενότητα 2.4.2. και στο παράδειγμα της ACO, της ενότητας 3.3.
- Αναρρίχηση: σε αυτήν την περίπτωση κάθε στοιχείο από την θέση που είναι υποχρεούται να «ανέβει» σε μία τοπικά μέγιστη κλίκα, με σταδιακή προσθήκη τυχαίων έγκυρων κόμβων. Αυτό γίνεται με την τυχαία επιλογή επόμενων κόμβων από τους γείτονες της κλίκας.

Από τις παραπάνω διαφορετικές εκδοχές, κάνοντας δοκιμές σε μικρούς γράφους ο συνδυασμός της επανατοποθέτησης και της αναρρίχησης έδωσε πολύ καλύτερα αποτελέσματα, ακόμα και βέλτιστα σε γράφους κάτω των 50 κόμβων. Στην επόμενη ενότητα θα γίνει σύγκριση αυτής της εκδοχής με τη βελτιστοποίηση της αποικίας μυρμηγκιών.

Στο παρακάτω διάγραμμα φαίνονται τα αποτελέσματα από τις παραπάνω διαφορετικές εκδοχές. Για τους παρακάτω συνδυασμούς παραμέτρων το πρόγραμμα έτρεξε 10 φορές σε κάθε ένα από τους 3 γράφους. Για κάθε διαφορετικό συνδυασμό και γράφο φαίνονται η βέλτιστη λύση, ο μέσος όρος των λύσεων και η χείριστη λύση αντίστοιχα σε κάθε μία από τις οριζόντιες γραμμές του «Ε».

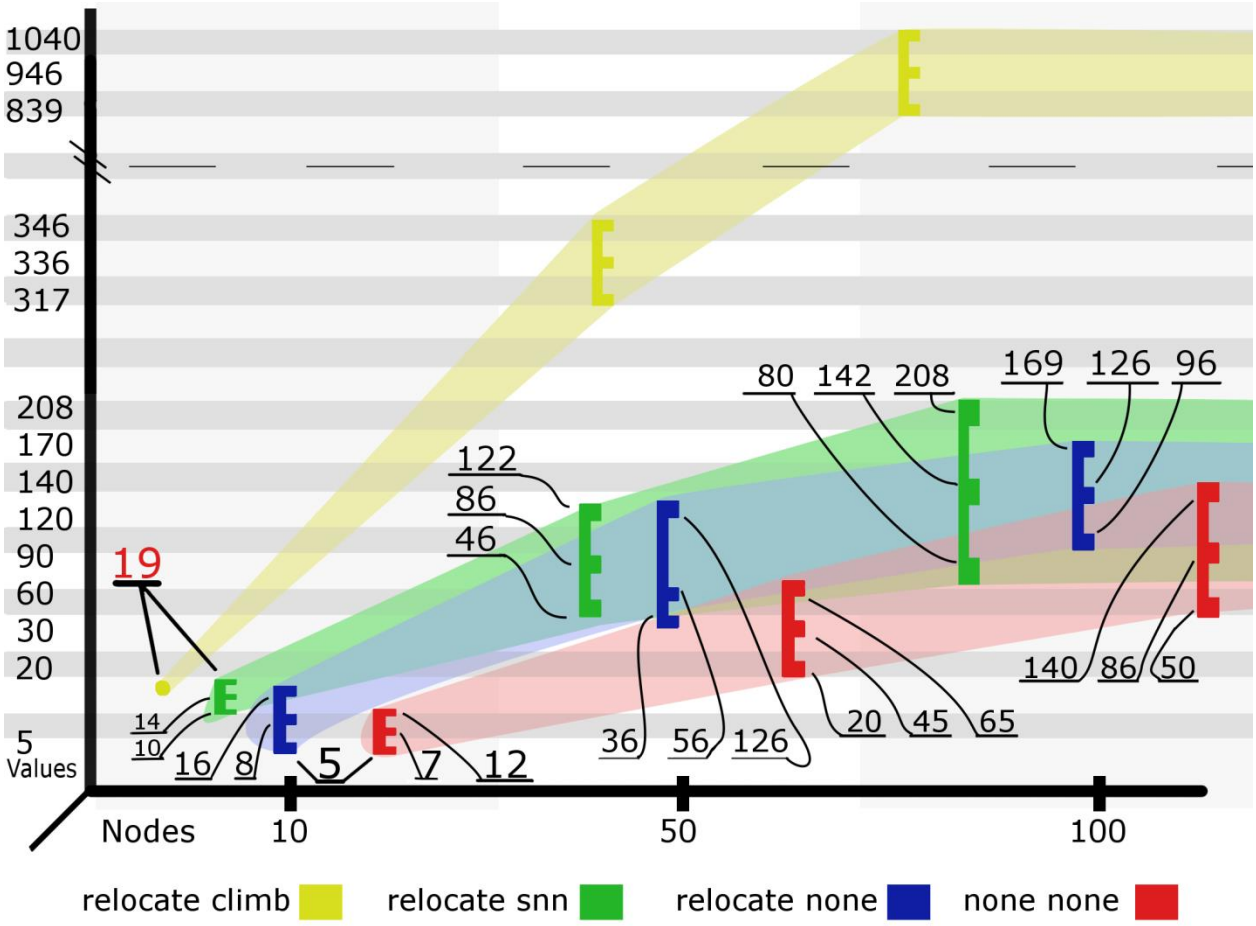


Figure 1: Σύγκριση διαφορετικών εκδοχών του NPSO στο MVWCP

Εδώ πρέπει να αναφερθεί ότι στο γράφο αυτό δεν υπάρχουν συνδυασμοί για την επιλογή του «τερματισμού» σε περίπτωση σύγκλισης, διότι οι τιμές των αποτελεσμάτων δεν θα είναι καλύτερες συγκριτικά με αυτήν της «επανατοποθέτησης». Αυτό συμβαίνει επειδή μόλις γίνει σύγκλιση, ο «τερματισμός» τερματίζει με τις τιμές της θέσης σύγκλισης, ενώ στην «επανατοποθέτηση» τα στοιχεία αποφεύγουν την ολική σύγκλιση με χρήση τυχαίας τοποθέτησης και διατηρείται η καλύτερη λύση.

Τέλος, είναι σημαντικό να παρατηρηθεί ότι η τιμή 19 και 1040 στους αντίστοιχους γράφους είναι βέλτιστες λύσεις και βρέθηκαν από τον συνδυασμό που χρησιμοποιούμε παρακάτω στην σύγκριση του NPSO και του ACO.

Στην ενότητα 2.5. θα γίνουν φανεροί οι λόγοι που υπάρχουν τα παραπάνω αποτελέσματα.

2.4. Δομές του προγράμματος του NPSO

Σε αυτήν την ενότητα γίνεται αναφορά των δομών του προγράμματος που υλοποιήθηκε η NPSO. Η γλώσσα υλοποίησης και των δύο προγραμμάτων, των δομών και οι δοκιμές έγιναν σε Python 2.8 σε συνδυασμό με την βιβλιοθήκη NumPy, που είναι μία βιβλιοθήκη για επιστημονικούς υπολογισμούς σε Python.

2.4.1. Δομή των αρχείων των γράφων

Η αναπαράσταση του εκάστοτε γράφου στο αρχείο έχει την εξής δομή:

{όνομα κόμβου} {βάρος κλίκας} [γειτονικοί κόμβοι]

Κάθε γραμμή είναι και ένας κόμβος, παραδείγματος χάριν για ένα γράφο 100 κόμβων, το αρχείο θα έχει 100 γραμμές. Εδώ είναι σημαντικό να αναφερθεί τι γίνεται με τους γειτονικούς κόμβους του κόμβου. Θα ήταν σπατάλη μνήμης να γραφτούν 2 φορές για κάθε ζευγάρι, γι αυτό το λόγο ακολουθήθηκε ο εξής κανόνας:

Έστω οι κόμβοι X και Y ενώνονται και το X προηγείται στο αρχείο από το Y. Τότε το Y θα είναι στην λίστα γειτόνων του κόμβου X, αλλά το X δεν θα είναι στην λίστα του Y.

2.4.2. Δομή και λειτουργίες της κλάσης του γράφου

Για την διαχείριση των δεδομένων των γράφων και κάποιων βασικών λειτουργιών υλοποιήθηκε η κλάση του γράφου (**Graph**).

Ευθύνη της κλάσης του γράφου είναι:

1. Η φόρτωση του γράφου στο πρόγραμμα από αρχείο της προαναφερθείσας μορφής.
2. Η επιστροφή των άμεσων γειτόνων ενός κόμβου (απόσταση ίση με 1).
3. Η επιστροφή όλων των κόμβων που μεγαλώνουν κάποια δοθείσα κλίκα.

Η κλάση του γράφου έχει τα εξής χαρακτηριστικά:

1. Ένα λεξικό (dictionary) – πίνακα κατακερματισμού (hash table) για την αποθήκευση των κόμβων, των βαρών και της λίστας των γειτόνων του εκάστοτε κόμβου. Η λίστα γειτόνων κάθε κόμβου είναι ίδια με αυτή στο αρχείο σύμφωνα με τον κανόνα της ενότητας 2.4.1.

2. Μια λίστα όλων των κόμβων με διατήρηση της σειράς εμφάνισης τους στο αρχείο.

3. Το πλήθος των ακμών.

4. Το πλήθος των κόμβων.

Βασικές λειτουργίες – συναρτήσεις της κλάσης του γράφου είναι:

1. **getNeighbours(** node **)**: Δοθέντος ενός κόμβου, αυτή η συνάρτηση επιστρέφει μία λίστα από γειτονικούς κόμβους.

2. **weightOfClique (** list_of_nodes **)**: Αυτή είναι η συνάρτηση αξιολόγησης (fitness function), όπου δεδομένης μίας λίστας κόμβων η συνάρτηση αυτή ελέγχει αν οι παραπάνω κόμβοι είναι κλίκα, επιστρέφοντας το βάρος της ή μηδέν αν δεν είναι κλίκα.

3. **findAvailableNodesForClique (** list_of_clique_neighbours , new_node**)**: Στην περίπτωση του σμήνους σωματιδίων, καθώς και της αποικίας μυρμηγκιών, οι κλίκες αναπτύσσονται, εν πολλοίς, σταδιακά ξεκινώντας με ένα κόμβο και τους άμεσους γείτονες του, στη συνέχεια προστίθεται ένας κόμβος από το σύνολο των γειτόνων και η γείτονες της κλίκας πλέον είναι η τομή των συνόλων των γειτόνων των υπαρχόντων κόμβων της κλίκας. Έτσι λοιπόν, η παραπάνω συνάρτηση δεδομένων των γειτόνων μιας κλίκας και ενός από αυτούς τους κόμβους επιστρέφει τους γείτονες της νέας κλίκας που δημιουργείται από την παλιά μαζί με τον νέο κόμβο new_node (βλ. παράδειγμα στην 3.3).

4. **findNodesForClique (** list_of_nodes **)**: Δοθείσης μίας κλίκας, η συνάρτηση αυτή επιστρέφει μία λίστα με κόμβους που μπορούν να επεκτείνουν την κλίκα. Αν στην κλίκα προστεθεί ένας από αυτούς τους κόμβους τότε το αποτέλεσμα θα είναι κλίκα. Επιπλέον, αν η κλίκα είναι μέγιστη, τότε επιστρέφεται κενή λίστα. Η συνάρτηση αυτή εσωτερικά χρησιμοποιεί την findAvailableNodesForClique.

2.4.3. Το σμήνος

Η κύρια κλάση του προγράμματος είναι το Σμήνος (Swarm) και το σωματίδιο (Particle). Το σμήνος είναι η κλάση που ελέγχει όλο το πρόγραμμα. Αυτή διαχειρίζεται τα σωματίδια, και κρατάει την βέλτιστη λύση και τιμή. Οι λειτουργίες αυτής της κλάσης είναι η δημιουργία των σωματιδίων και η επανατοποθέτηση τους (βλ. ενότητα 2.3), η διατήρηση και η ενημέρωση των σωματιδίων για την βέλτιστη λύση, και τέλος μέσω αυτής της κλάσης γίνεται εκκίνηση υπολογισμού επιταχύνσεων και νέων θέσεων από τα στοιχεία.

2.4.4. Το σωματίδιο

Η κλάση του στοιχείου (Particle) – ο ψηφιακός πράκτορας του συγκεκριμένου αλγορίθμου – είναι αυτή που διατηρεί την θέση του εκάστοτε στοιχείου, τις δύο επιταχύνσεις, και τους πιθανούς επόμενους γείτονες στην περίπτωση της προτροπής γειτόνων ή αναρρίχησης (βλ. ενότητα 2.3). Επίσης μέσω αυτής της κλάσης γίνεται δημιουργία τυχαίων θέσεων για την εκκίνηση των σωματιδίων, υπολογισμός των επιταχύνσεων V_{ij}^{0}, V_{ij}^{1} και V_{ij}^{C}και τέλος, υπολογίζει την επόμενη θέση κάθε στοιχείου.

2.4.5. Επιπλέον συναρτήσεις

Τέλος, υλοποιήθηκαν και κάποιες βοηθητικές συναρτήσεις για την μετατροπή των θέσεων από την αναπαράσταση του δυαδικού υπέρ – κύβου του NPSO (πχ. (0,1,0,0,1,0)) στην μορφή του γράφου (πχ. [‘B’, ‘E’]).

2.5. Ένα παράδειγμα του NPSO

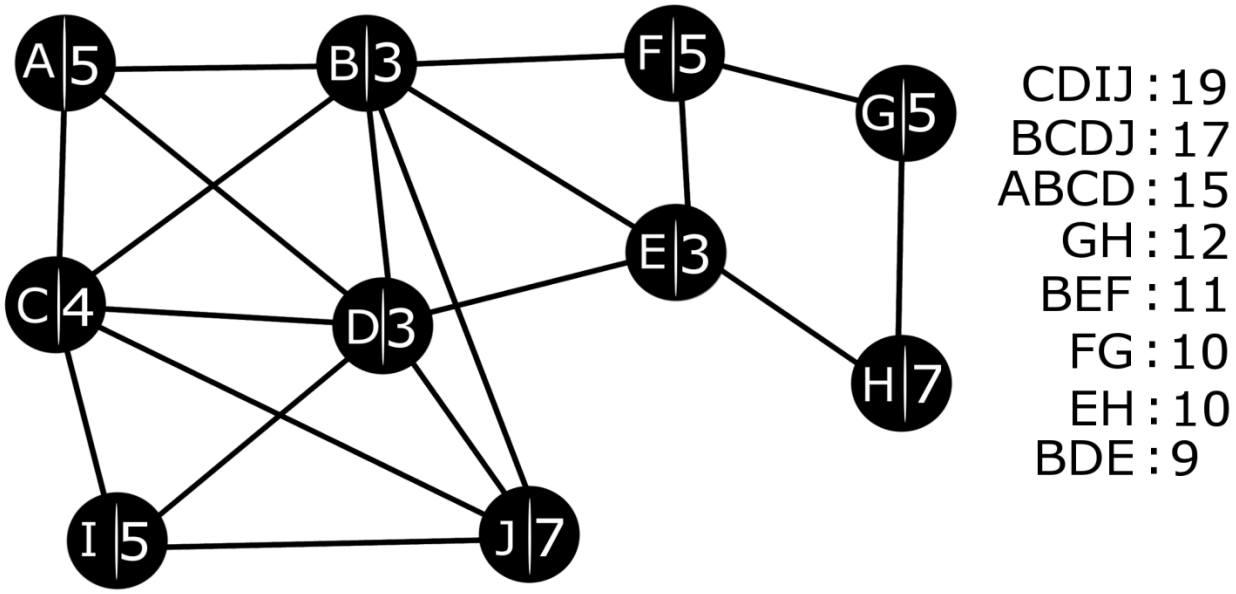


Figure 2: Γράφος παραδειγμάτων και οι κλίκες του

Έστω ο παραπάνω γράφος, στα δεξιά του οποίου είναι όλες η τοπικά μέγιστες κλίκες. Εφόσον ο γράφος έχει 10 κόμβους, ο χώρος κίνησης των σωματιδίων είναι ένας δέκα–διάστατος υπέρ-κύβος. Για λόγους απλοποίησης και διευκόλυνσης δεν θα χρησιμοποιηθεί η αναπαράσταση του προγράμματος με τις τιμές 0 και 1 για το αν ο κόμβος ανήκει ή όχι στην κλίκα αλλά με τα ονόματα των κόμβων. Έτσι αντί να έχουμε μία θέση – λύση της μορφής (0, 1, 0, 0, 1, 1, 0, 0, 0, 0), αυτό θα αντιστοιχεί για το παράδειγμα μας στο (B, E, F).

Έστω ότι στο πρόγραμμα έχουμε 2 στοιχεία (P_1, P_2) και έχουν τεθεί οι παράμετροι ως εξής:

$$Μάζα = \mathbf{0.8}, \quad Εμπειρία\ στοιχείου = \mathbf{2}, \quad Εμπειρία\ σμήνους = \mathbf{2}, \quad γύροι = \mathbf{100}$$

Έστω οι τυχαίες θέσεις $X_{P_1} = (G),\ X_{P_2} = (I)$ γίνεται η αξιολόγηση και η τιμές είναι 5 και στις δύο περιπτώσεις. Το I γίνεται η βέλτιστη λύση της κλίκας. Για τον επόμενο γύρο με $c_1 = 0.394, c_2 = 0.935$ με βάση τον κανόνα 1 και 2 η φερορμόνη θα γίνει ως εξής:

$V_{P_1}^0$: $[2.839,\ 3.337,\ 3.023,\ 3.130, 3.303, 3.227, 1.355, 2.866, 3.269, -0.951]$
$V_{P_2}^1$: $[-2.338, -2.539, -2.068, -2.270, -2.623, -2.546, -0.466, -2.302, -2.637, 1.651]$

$V_{P_1}^C$: $[-2.338, -2.539, -2.068, -2.270, -2.623, -2.546, 1.355, -2.302, -2.637, 1.651]$

$V_{P_2}^0$: $[2.989, 3.083, 2.945, 3.449, 3.246, 2.839, 3.291, 2.677, 2.813, -2.238]$

$V_{P_1}^1$: $[-2.025, -2.283, -2.343, -1.955, -2.510, -2.306, -2.016, -2.238, -2.241, 2.811]$

$V_{P_2}^C$: [-2.025, -2.283, -2.343, -1.955, -2.510, -2.306, -2.016, -2.238, -2.241, -2.238]

Οι θέσεις των P_1 και P_2 με τυχαία μεταβλητή γύρου ίση με 0,095 και 0,781 αντίστοιχα, και σύμφωνα με τον κανόνα 3 έχουμε $\mathrm{X}_{\mathrm{P}_1} = (\mathrm{C}, \mathrm{J}),\ \mathrm{X}_{\mathrm{P}_2} = (\mathrm{I})$, τώρα οι λύσεις αυτές έχουν βάρος 11 και 5 και η θέση (C,J) γίνεται η βέλτιστη θέση του σμήνους.

Οι γύροι συνεχίζονται και τα στοιχεία δεν αλλάζουν θέσεις, μετά από τρείς γύρους με την αλλαγή φερορμόνης θα γίνει εν τέλει σύγκλιση στο (C, J).

Είναι φανερό ότι αυτά τα αποτελέσματα είναι πολύ κακά για ένα γράφο 10 κόμβων, καθώς το αποτέλεσμα που βρέθηκε δεν είναι καν μια τοπικά μέγιστη κλίκα. Στο ίδιο πρόβλημα με χρήση της προτροπής γειτόνων το αποτέλεσμα θα ήταν μία τοπικά μέγιστη κλίκα, ενώ με επανατοποθέτηση και αναρρίχηση η πιθανότητα να μην βρούμε την βέλτιστη λύση θα ήταν πολύ μικρή.

3. ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΜΕ ΑΠΟΙΚΙΑ ΜΥΡΜΗΓΚΙΩΝ

3.1. Γενική περιγραφή του αλγορίθμου του Ant Colony Optimization

Ο αλγόριθμος που χρησιμοποιείται ανήκει στην οικογένεια των αλγορίθμων της αποικίας μυρμηγκιών, οι οποίοι με τη σειρά τους είναι μέρος της ευρύτερης κατηγορίας της συλλογικής νοημοσύνης ή νοημοσύνης σμήνους (Swarm Intelligence) και μετα-ευρεστικών μεθόδων βελτιστοποίησης (metaheuristic optimization).

Η πρώτη έκδοση του μυρμηγκιού, το σύστημα του μυρμηγκιού (Ant System), προτάθηκε αρχικά από τον Marco Dorigo το 1992 [19] σαν ένας τρόπος αναζήτησης βέλτιστου μονοπατιού σε γράφο, και εφαρμόστηκε αρχικά στο πρόβλημα του πλανόδιου πωλητή (Traveling Salesman Problem).

Η γενική ιδέα έχει ως εξής: τα μυρμήγκια κινούνται τυχαία στο χώρο αναζητώντας τροφή. Όταν κάποιο μυρμήγκι βρει τροφή, επιστρέφει στην φωλιά εκκρίνοντας μία χημική ουσία σε όλη την διαδρομή, την φερορμόνη. Όταν τα υπόλοιπα μυρμήγκια την ανιχνεύσουν ακολουθούν το μονοπάτι που τους οδηγεί στην τροφή. Εάν η τροφή υπάρχει ακόμα τότε εκκρίνουν και αυτά φερορμόνη στο μονοπάτι, ενισχύοντας το. Η φερορμόνη όμως εξατμίζεται λόγω του αέρα, οπότε τα μακρύτερα και τα λιγότερο χρησιμοποιούμενα μονοπάτια χάνουν την φερορμόνη που είχαν και τα μυρμήγκια τα προτιμούν λιγότερο. Με αυτόν τον τρόπο τα μυρμήγκια βρίσκουν σχεδόν βέλτιστες διαδρομές μεταξύ φωλιάς και τροφής. Ομοίως τα ψηφιακά μυρμήγκια διατρέχουν τον δοθέντα γράφο και εκκρίνουν φερορμόνη στις ακμές του γράφου. Στο τέλος κάθε γύρου η φερορμόνη ελαττώνεται, ώστε τα πιο μικρά σε μήκος και πολυσύχναστα μονοπάτια, να έχουν περισσότερη φερορμόνη και να είναι πιο «ελκυστικά» στα ψηφιακά μυρμήγκια. Κατ' αυτόν τον τρόπο δημιουργείται μία ευρεστική συνάρτηση με κατεύθυνση τα καλύτερα μονοπάτια (μετά-ευρεστική συνάρτηση).

Στην κλασσική μορφή του αλγορίθμου ο Marco Dorigo εξέδωσε το 1997 [20] μία νεότερη εκδοχή, με καλύτερα αποτελέσματα στις δοκιμαστικές συγκρίσεις. Η νέα εκδοχή, που είναι η αποικία μυρμηγκιών, διαφοροποιείται στον τρόπο επιλογής της διαδρομής από τα μυρμήγκια, στον κανόνα ανανέωσης της φερορμόνης στο τέλος κάθε γύρου (global updating rule) και κατά την πορεία των μυρμηγκιών κατά την διάρκεια του γύρου (local updating rule). Στην έρευνα αυτή ο τελευταίος κανόνας δεν χρησιμοποιείται.

3.2. Μορφή του αλγορίθμου στο πρόβλημα της βαρύτερης κλίκας

Έστω η φερορμόνη μεταξύ δύο κόμβων r, s είναι $\tau(r, s)$, επίσης ορίζουμε ως $J_k(r)$ το σύνολο των κόμβων που ενώνονται με το r και δεν έχουν προσπελασθεί νωρίτερα στον ίδιο γύρο από το μυρμήγκι k και cn το πλήθος των κόμβων στο $J_k(r)$.

Ο κανόνας μετάβασης έχει ως εξής:

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \tau(r,u), & if\ q < q_0 \\ R, & otherwise \end{cases}$$

Όπου S ο κόμβος που επιλέχθηκε, $q \in [0,1]$ τυχαία μεταβλητή που ανανεώνεται σε κάθε γύρο, $q_0 \in [0,1]$ παράμετρος του προγράμματος, και R το αποτέλεσμα τυχαίας επιλογής με βάρη πιθανοτήτων όπως παρακάτω:

$$p_k(r,s) = \begin{cases} \frac{1}{\sum^{cn} 1}, & \forall u \in J_k(r): if\ \tau(r,u) = 0 \\ \frac{\tau(r,s)}{\sum_{u \in J_k(r)} \tau(r,u)}, & \exists u \in J_k(r): if\ \tau(r,u) \neq 0 \\ 0, & otherwise \end{cases}$$

Ο αλγόριθμος έχει ως εξής:

Παράμετροι : rounds, αριθμός μυρμηγκιών, γράφος, decay, q_0

4. Για γύρους rounds κάνε:
    a. Για κάθε μυρμήγκι:
        i. Διάλεξε τυχαία ένα κόμβο.
        ii. Όσο η κλίκα δεν είναι τοπικά μέγιστη:
            1. Διάλεξε επόμενο κόμβο σύμφωνα με τον κανόνα μετάβασης.
        iii. Εάν το βάρος της κλίκας είναι μέγιστο:
            1. Αποθήκευσε τους κόμβους.
        iv. Πρόσθεσε φερορμόνη τις ακμές της βαρύτερης κλίκας.
        v. Ελάττωσε την φερορμόνη κατά ποσοστό decay.
5. Εκτύπωσε στατιστικά γράφου, αποτελέσματα και χρόνους.

Σχήμα 2: Ο αλγόριθμος του ACO

3.3. Ένα παράδειγμα του ACO

Έστω ότι έχουμε τον γράφο στο *figure 2* που παρουσιάστηκε στο παράδειγμα της προηγούμενης ενότητας, στο 2.5.

Για ευκολία, θεωρούμε ότι έχουμε μόνο ένα μυρμήγκι και ορίζουμε την έκκριση φερορμόνης από το μυρμήγκι ίση με 1 διά το πλήθος τον κόμβων της κλίκας και η εξάτμιση της φερορμόνης ανά γύρο είναι 15%.

Για τις παρακάτω εικόνες η νέα κλίκα του παριστάνεται με κίτρινο χρώμα, με μπλε οι γείτονες της κλίκας – πιθανοί επόμενοι κόμβοι, με πράσινο η φερορμόνη, και με θαλασσί μέσα στους κόμβους αναπαρίσταται η θέση του μυρμηγκιού.

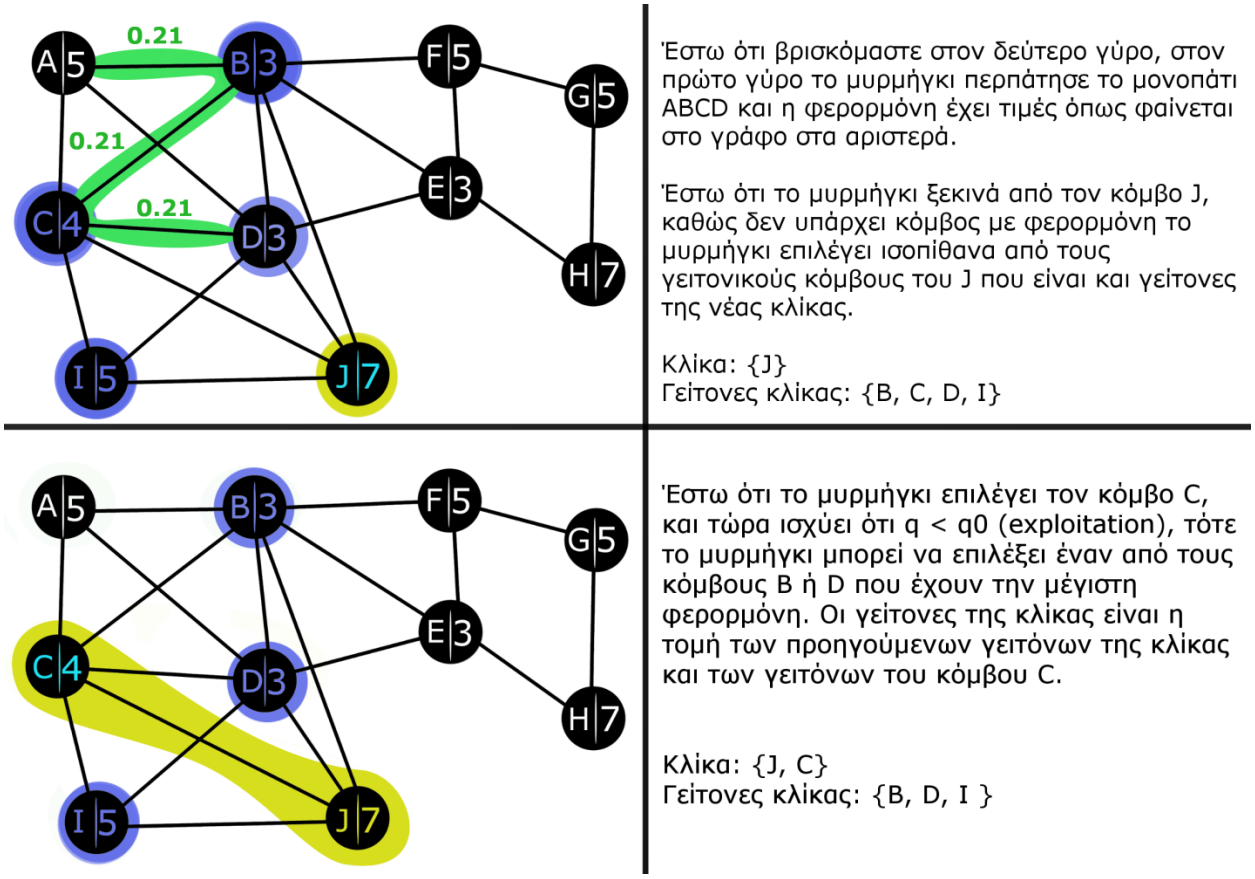


Figure 3: Τα δύο πρώτα βήματα του ACO

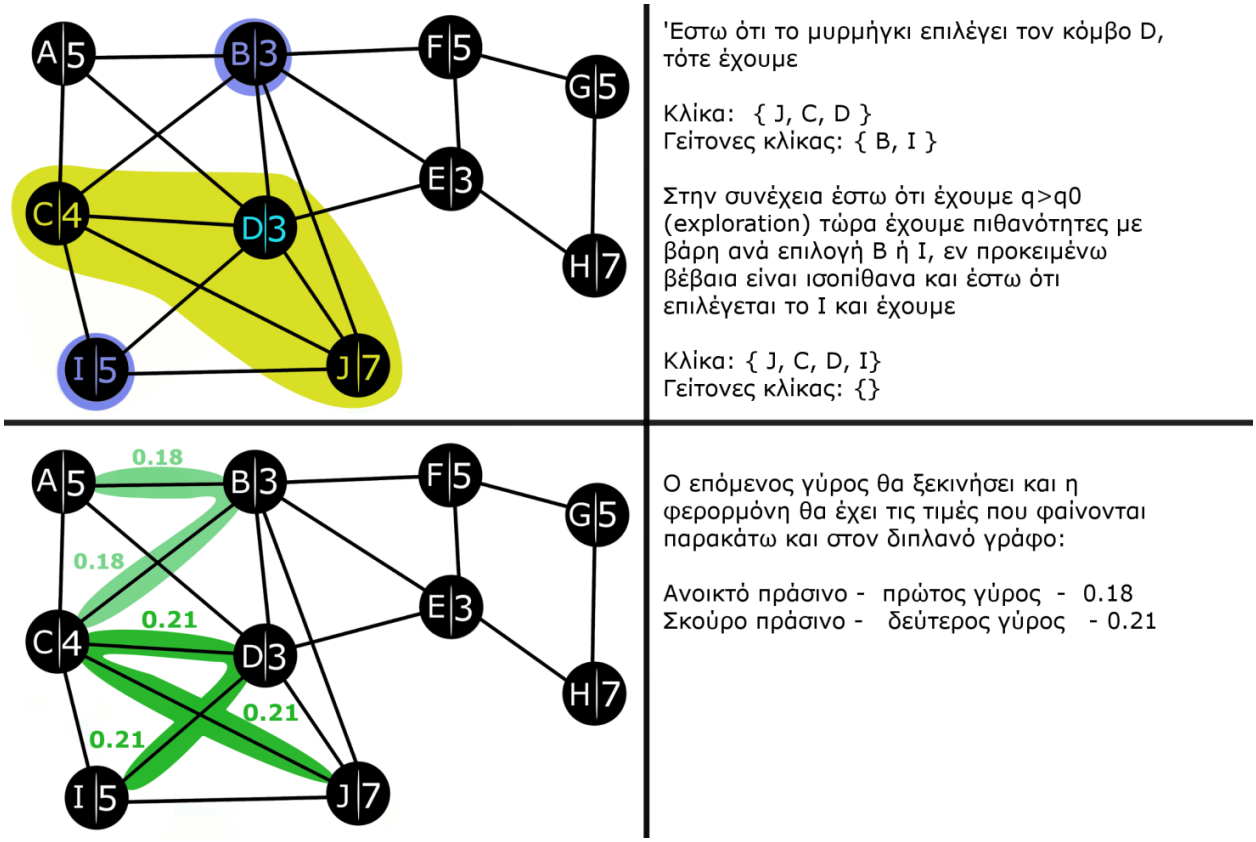


Figure 4: Τα δύο τελευταία βήματα του ACO

3.4. Δομές του προγράμματος του ACO

Προκειμένου τα αποτελέσματα να είναι σωστά – δίκαια στην σύγκριση, χρησιμοποιήθηκε η ίδια κλάση του γράφου και αρχείων που περιγράφθηκε στην ενότητα 2.4.1. και 2.4.2. Παρακάτω περιγράφονται οι διαφορετικές δομές που χρησιμοποιήθηκαν για το ACO.

3.4.1. Η κλάση της ακμής (Edge)

Η κλάση της ακμής υλοποιήθηκε για την καλύτερη διαχείριση και διαχωρισμό των δεδομένων και των λειτουργιών του προγράμματος. Όταν και μόνο όταν ένα μυρμήγκι εκκρίνει φερορμόνη σε μία ακμή του γράφου δημιουργείται η κλάση της ακμής για την καλύτερη διαχείριση της. Οι λειτουργίες που επιτελεί η ακμή είναι η προσπέλαση τιμών φερορμόνης, ανανέωση τιμών φερορμόνης και η εξάτμιση.

3.4.2. Η δομές και λειτουργίες της κλάσης του γράφου φερορμόνης

Η κλάση του γράφου φερορμόνης (**FerermoneGraph**) επεκτείνει τον γράφο του NPSO, που περιγράφεται στην ενότητα 2.4.2., ώστε να αποθηκεύει τις ακμές που περιγράφθηκαν στην 3.4.1. σε ένα λεξικό (dictionary). Προκειμένου να γίνει εξοικονόμηση πόρων, δημιουργούνται και αποθηκεύονται μόνο οι ακμές που έχουν φερορμόνη μεγαλύτερη από το μηδέν.

3.4.3. Η κλάση του μυρμηγκιού

Η κλάση του μυρμηγκιού έχει την εξής δομή:

1. Μία λίστα κόμβων που έχουν προσπελαστεί από το μυρμήγκι – το μονοπάτι.
2. Μία λίστα με τους επόμενους πιθανούς κόμβους.

Σε κάθε γύρο του προγράμματος κάθε μυρμήγκι επιστρέφει μία τοπικά μέγιστη κλίκα, αυτή είναι η δουλειά της συνάρτησης εύρεσης κλίκας (**findAClique**) η οποία εσωτερικά καλεί την συνάρτηση του γράφου **findAvailableNodesForClique**.

Επίσης υπάρχει συνάρτηση επαναφοράς (**reset**) που κενώνει τις λίστες ώστε το μυρμήγκι να ξεκινήσει τον επόμενο γύρο.

3.4.4. Η κλάση της Αποικίας

Τέλος, η κλάση της αποικίας είναι η υπερκλάση του προγράμματος που διαχειρίζεται τα μυρμήγκια, τους γράφους, και διατηρεί την καλύτερη λύση και της ανανεώνει την φερορμόνη.

4. ΣΥΓΚΡΙΣΕΙΣ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

4.1. Πληροφορίες συστήματος

Στον παρακάτω πίνακα είναι τα στοιχεία του συστήματος εκτέλεσης των δοκιμών

Πίνακας 1: Πληροφορίες συστήματος

Μοντέλο υπολογιστή	Asus ROG GL552VW
Τύπος Συστήματος	64-bit
Επεξεργαστής	Intel core i7-6700HQ @ 2.60 GHz
RAM	16 GB
Λειτουργικό σύστημα	Windows 10 Home

4.2. Συγκριτικοί πίνακες

Για τις συγκρίσεις χρησιμοποιήθηκαν γράφοι δύο κατηγοριών. Η πρώτη κατηγορία είναι γράφοι με βάρη που δημιουργήθηκαν αυτόματα και χαρακτηρίζονται από το πρόθεμα «Graph-» και έχουν πυκνότητα περίπου ίση με 0,5. Η δεύτερη ομάδα γράφων είναι από την DIMACS[1], αυτοί οι γράφοι δεν έχουν βάρη οπότε ορίστηκε σαν βάρος η τιμή 1 για όλους τους κόμβους. Το ACO και NPSO δεν αναπτύσσονται – επιλέγουν με βάση το βάρος των κόμβων, συνεπώς μπορούν να χρησιμοποιηθούν σε γράφους χωρίς βάρη για να βρουν την μεγαλύτερη κλίκα του γράφου.

Παρακάτω δίδονται τα χαρακτηριστικά των γράφων που χρησιμοποιήθηκαν:

Πίνακας 2: Χαρακτηριστικά των γράφων δοκιμών

Γράφος	Κόμβοι	Ακμές	Πυκνότητα	Γύροι	Βάρος Βαρύτερης κλίκας
Graph-100	100	2305	0,465656	50	1040
Graph-150	150	5711	0,511051	75	1721
Graph-200	200	9260	0,4653	100	2973
Graph-500	500	62468	0,50074	250	11407
Graph-1000	1000	245675	0,49184	400	31404
C125.9	125	6963	0,89845	100	34*[2]
C250.9	250	27984	0,89845	150	44*
C500.9	500	112332	0,90045	250	57
C1000.9	1000	450079	0,90106	400	68

[1] http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark
[2] * : Έχει αποδειχθεί ως βέλτιστη λύση με χρήση άλλων αλγορίθμων

Τα αποτελέσματα στους παρακάτω πίνακες είναι μέσοι όροι από 25 επαναλήψεις σε κάθε γράφο. Οι χρόνοι είναι μετρημένοι σε δευτερόλεπτα.

Πίνακας 3: Αποτελέσματα ACO και NPSO σε γράφους πυκνότητας 5

	Graph-100		**Graph-150**		**Graph-200**		**Graph-500**		**Graph-1000**	
	ACO	**NPSO**	**ACO**	**NPSO**	**ACO**	**NPSO**	**ACO**	**NPSO**	**ACO**	**NPSO**
Min Clique	*964*	791	*1560*	1457	*2651*	2372	*9900*	8653	*26077*	24601
AVG Clique	*1026*	927	*1676*	1568	*2793*	2685	*10619*	9985	*29628*	27594
Max Clique	*1040*	1030	*1721*	1683	*2973*	2937	*11298*	10872	*32692*	32395
LII [3]	*26*	28	*48*	52	*53*	80	*160*	180	*313*	347
Time of LII	0.471	*0.168*	2.318	*0.935*	3.17	*2.433*	55.51	*46.94*	*450.9*	551.9
Total Time	0.878	*0.315*	3.738	*1.374*	6.31	*3.031*	89.66	*75.55*	599.2	*676.9*

Πίνακας 4: Αποτελέσματα ACO και NPSO σε γράφους πυκνότητας 9

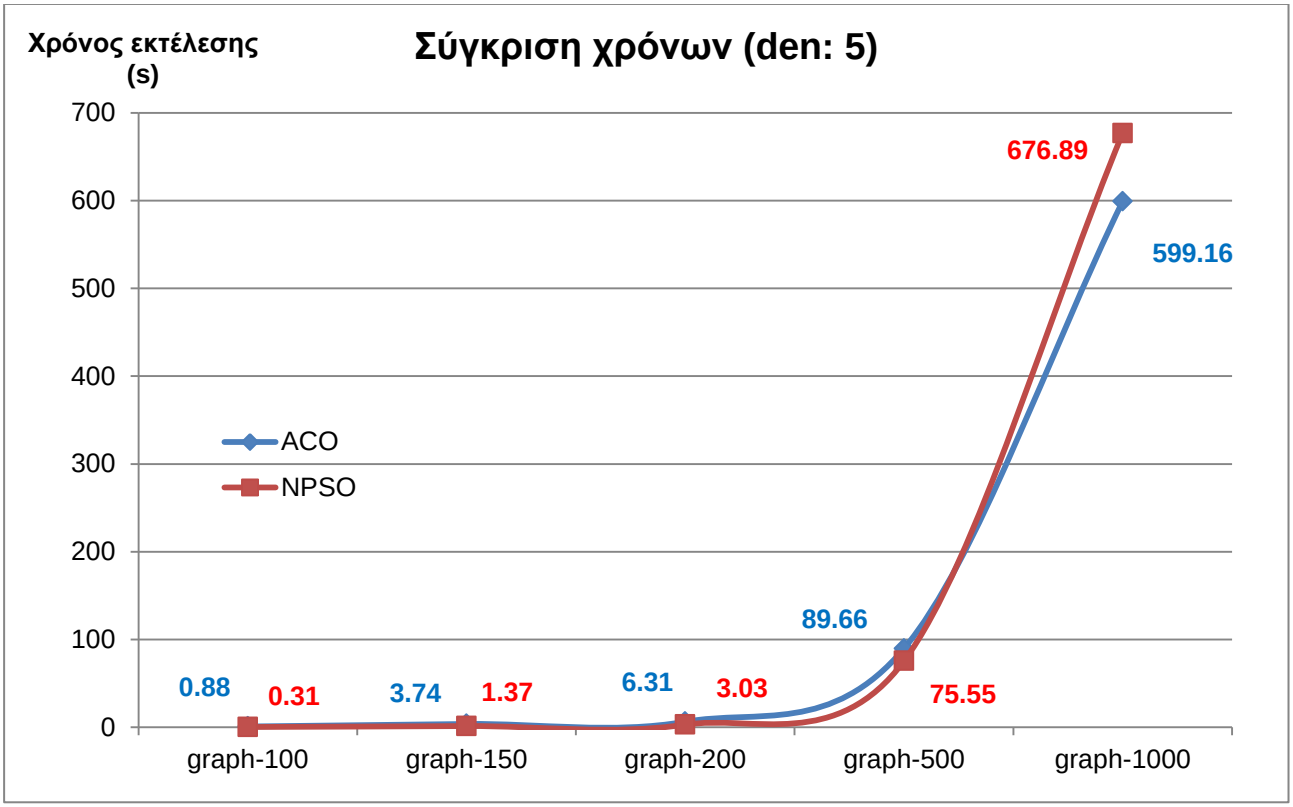
	C125.9		**C250.9**		**C500.9**		**C1000.9**	
	ACO	**NPSO**	**ACO**	**NPSO**	**ACO**	**NPSO**	**ACO**	**NPSO**
Min Clique	*31*	26	*39*	34	*47*	37	*55*	51
AVG Clique	*34*	32	*41*	39	*50*	47	*57*	55
Max Clique	34	34	*44*	43	*52*	51	*61*	60
LII	*32*	37	*60*	76	*114*	121	*249*	289
Time of LII	5.0226	*1.4548*	26.6637	*10.8432*	148.445	*88.482*	*913.72*	943.64
Total Time	18.4277	*3.0002*	72.429	*20.611*	347.43	*159.70*	1857.6	*1632.2*

[3] LII: Last Important Iteration – Τελευταίος σημαντικός γύρος – Ο γύρος που δόθηκε η τελική λύση

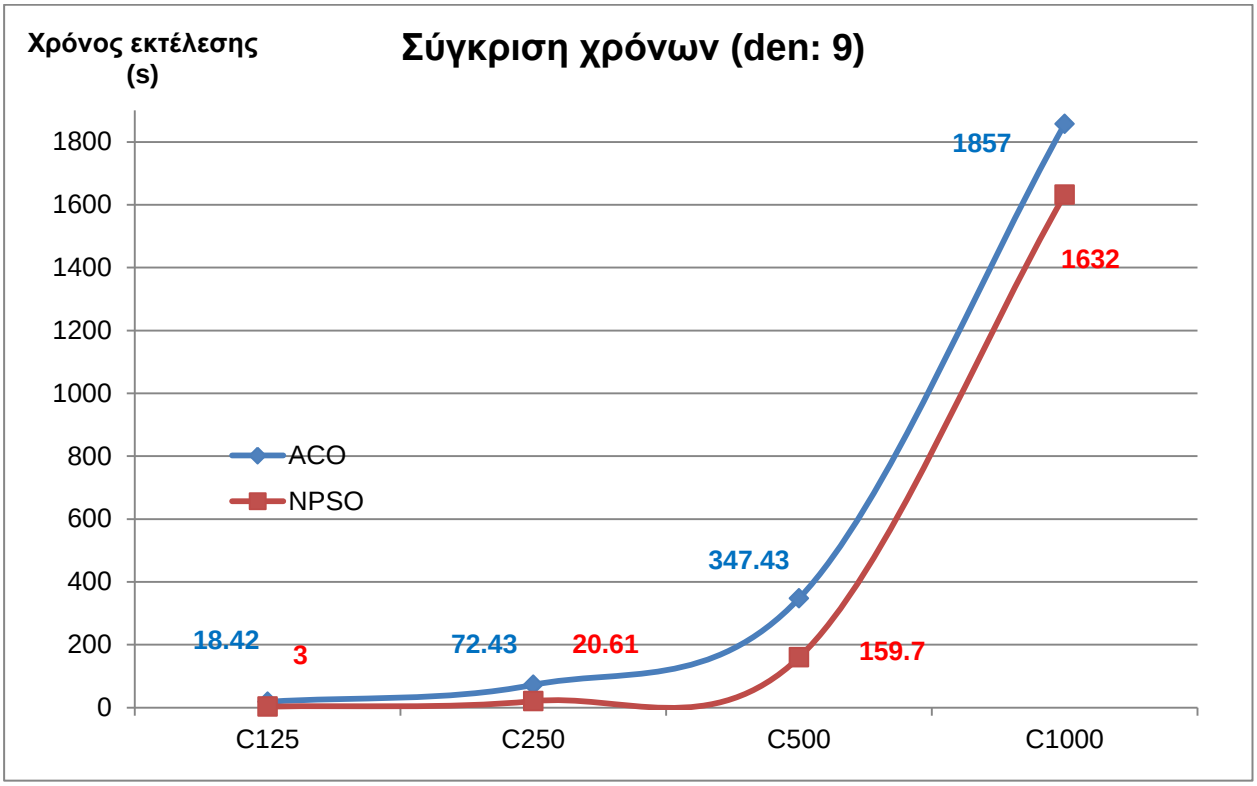
5. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΙΘΑΝΕΣ ΕΠΕΚΤΑΣΕΙΣ

Από τους πίνακες της προηγούμενης ενότητας μπορούμε να συμπεράνουμε ασφαλώς ότι ο αλγόριθμος του ACO είναι πιο αποτελεσματικός σε θέματα εύρεσης λύσεων, καθώς σε όλες τις περιπτώσεις βρήκε ίση ή καλύτερη λύση από τον NPSO. Παρόλα αυτά αξίζει να σημειωθεί ότι εν πολλοίς ο NPSO δεν βρήκε «πολύ κακές» λύσεις συγκρινόμενος με τον ACO ενώ παράλληλα είναι ταχύτερος σε συνολικό χρόνο για τους ίδιους γύρους, σε γράφους μέχρι περίπου 1000 κόμβους. Επιπλέον, η NPSO επίσης είναι λιγότερο απαιτητική σε μνήμη, καθώς χρειάζεται να αποθηκεύσει, εκτός από τον αρχικό γράφο, μόνο $n \cdot k$ bits όπου n ο αριθμός των particles και k ο αριθμός των κόμβων του γράφου.

Δίνοντας λίγο παραπάνω έμφαση στον χρόνο εκτέλεσης των δύο αλγορίθμων, είναι σημαντικό να επισημανθεί ότι ενώ αρχικά η NPSO είναι 3 και 6 φορές ταχύτερη συγκρινόμενη με την ACO σε γράφους πυκνότητας 5 και 9 αντίστοιχα, όσο ο γράφος μεγαλώνει, αυτή η διαφορά ελαττώνεται και φαίνεται ότι η ACO θα είναι ταχύτερη από την NPSO για γράφους μερικών χιλιάδων κόμβων. Αυτό φαίνεται από τα δύο παρακάτω διαγράμματα, ειδικά στην περίπτωση του γράφου graph-1000.



Σχήμα 3: Σύγκριση χρόνων ACO και NPSO (πυκνότητα 5)



Σχήμα 4: Σύγκριση χρόνων ACO και NPSO (πυκνότητα 9)

Και οι δύο αλγόριθμοι σε βάθος χρόνου μπορούν να βρουν βέλτιστες λύσεις, αλλά τα αποτελέσματα μπορούν να επιταχυνθούν με χρήση «έξυπνων» ευρεστικών συναρτήσεων που να υπολογίζουν το βάρος και το βαθμό κάθε κόμβου, το οποίο δεν εξετάστηκε ιδιαιτέρως στην συγκεκριμένη έρευνα. Επίσης μία ίσως διαφορετική αναπαράσταση του χώρου των λύσεων στο NPSO ίσως να έδινε πολύ καλύτερες λύσεις, αν παραδείγματος χάριν έλλειπαν οι μη πραγματικές κλίκες.

Στην παρακάτω εικόνα (*figure 5*) φαίνεται ο χώρος στον οποίο κινούνται τα στοιχεία με βάση τον δοκιμαστικό γράφο 10 κόμβων – δυαδικών διαστάσεων. Κάθε κύβος είναι μία υποψήφια λύση, είτε έγκυρη κλίκα είτε όχι. Υπάρχουν $2^{10} = 1024$ υποψήφιες λύσεις λόγω του υπέρ-κύβου, όταν ο γράφος έχει 8 τοπικά μέγιστους γράφους. Ο κόκκινος κύβος είναι μία τοπικά μέγιστη κλίκα, ο λευκός είναι η βέλτιστη λύση στο συγκεκριμένο πρόβλημα (C, D, I, J) και οι γκρι κύβοι είναι κάποιες μη τοπικά μέγιστες κλίκες. Οι περισσότεροι από τους πράσινους κύβους δεν είναι έγκυρες κλίκες και η συνάρτηση αξιολόγησης επιστρέφει μηδέν.

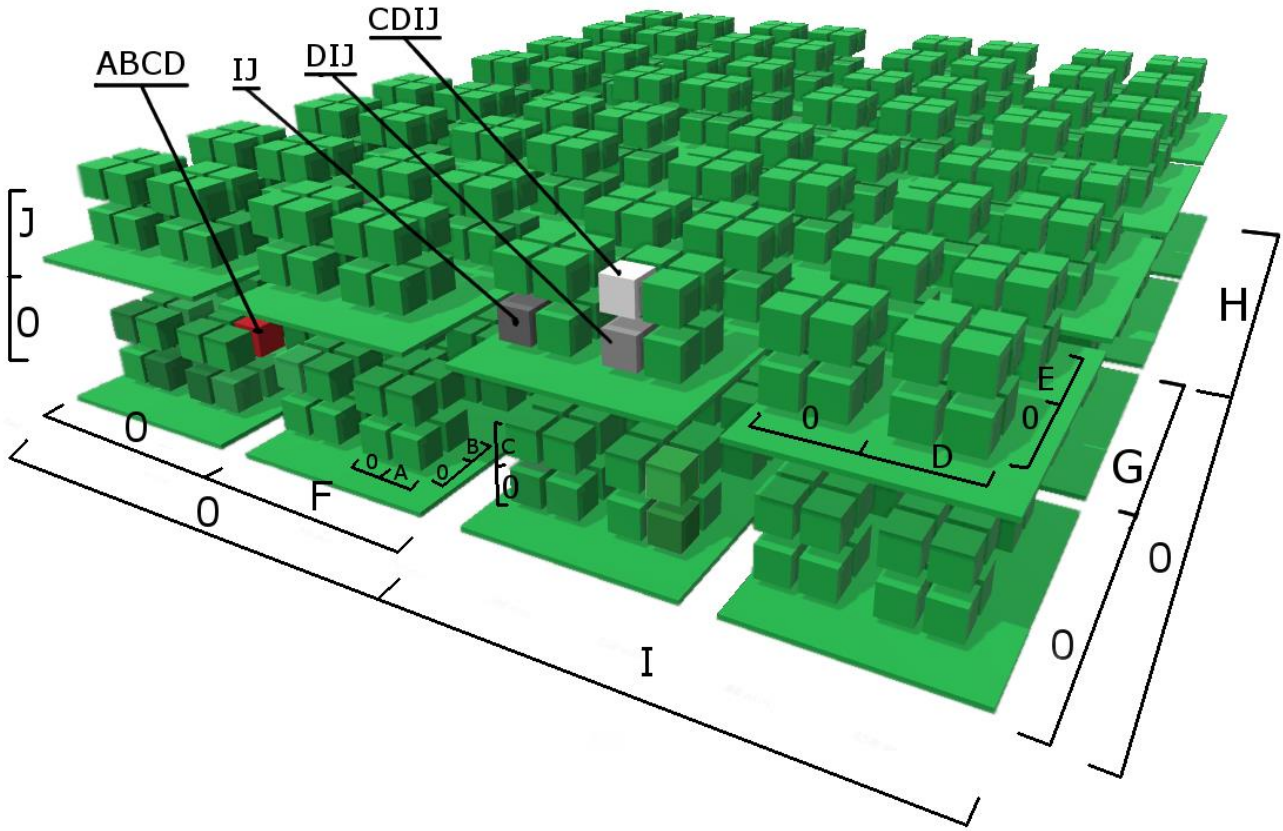


Figure 5: Χώρος λύσεων του γράφου στην Figure 2 κατά το NPSO

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός όρος**
Graph	Γράφος
Undirected graph	Μη κατευθυνόμενος γράφος
Clique	Κλίκα (γράφου)
Clique weight	Βάρος κλίκας
Maximal clique	Τοπικά μέγιστη κλίκα
Maximum clique	Ολικά μέγιστη κλίκα
Maximum weight clique	Ολικά βαρύτερη κλίκα
Maximum vertex weight clique problem	Πρόβλημα της βαρύτερης κλίκας, με βάρη στους κόμβους
Decision problem	Πρόβλημα απόφασης
Ant colony optimization	Βελτιστοποίηση αποικίας μυρμηγκιών
Particle swarm optimization	Βελτιστοποίηση σμήνους σωματιδίων
Heuristic functions	Ευρεστικές συναρτήσεις
Meta-heuristic functions	Μετα-ευρεστικές συναρτήσεις
Intensification	Εντατικοποίηση
Diversification	Διαφοροποίηση
Branch and Bound method	Μέθοδος διακλάδωσης και αποκοπής
Vertex colouring algorithm	Αλγόριθμος χρωματισμού γράφου
Backtrack tree search	Αναζήτηση με οπισθοδρόμηση σε δέντρο
Maximum Satisfiability Reasoning	Λογική της μέγιστης ικανοποίησης
Fitness function	Συνάρτηση αξιολόγησης ή κόστους
Traveling Salesman Problem	Πρόβλημα του πλανόδιου πωλητή
Tabu Search	Αναζήτηση με απαγορεύσεις

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

MCP	Maximum Clique Problem
MWCP	Maximum weight Clique Problem
MVWCP	Maximum Vertex Weight Clique Problem
ACO	Ant Colony Optimization
PSO	Particle Swarm Optimization
BPSO	Binary Particle Swarm Optimization
NPSO	A Novel Binary Particle Swarm Optimization

ΠΑΡΑΡΤΗΜΑ

Γράφος	Ένα σύνολο κορυφών (κόμβων) και ακμών (γραμμές που τους ενώνουν)
Μη κατευθυνόμενος γράφος	Ένας γράφος όπου αν υπάρχει ακμή από τον κόμβο Α στον Β, υπάρχει και από τον κόμβο Β προς τον Α
Κλίκα γράφου	Ένα υποσύνολο κόμβων ενός γράφου όπου κάθε κόμβος ενώνεται με κάθε άλλο κόμβο του υποσυνόλου αυτού.
Βάρος Κλίκας	Το άθροισμα των βαρών των κόμβων μίας κλίκας
Τοπικά μέγιστη κλίκα	Μια κλίκα ενός γράφου, η οποία δεν μπορεί να επεκταθεί άλλο, προσθέτοντας κόμβους σε αυτήν
Ολικά μέγιστη κλίκα	Η μεγαλύτερη κλίκα από όλες τις τοπικά μέγιστες κλίκες
Ολικά βαρύτερη κλίκα	Η βαρύτερη κλίκα από όλες τις τοπικά μέγιστες κλίκες
Πρόβλημα απόφασης	Η μορφή ενός προβλήματος, όπου δοθέντος του χώρου των λύσεων (πχ. γράφος) και κάποιας πιθανής λύσης (πχ. κλίκας), το πρόγραμμα μπορεί να επιστρέψει απάντηση της μορφής σωστό ή λάθος
Ευρεστικές συναρτήσεις	Κανόνες επιλογής – αξιολόγησης των επόμενων βημάτων προς εύρεση κάποιας καλής λύσης
Μέτα-ευρεστικές συναρτήσεις	Διαφέρουν από τις ευρεστικές συναρτήσεις στο ότι η αξιολόγηση δεν ορίζεται από κάποιον προγραμματιστή, αλλά από τον ίδιο τον αλγόριθμο δυναμικά, κατά την εκτέλεση του προγράμματος.

ΑΝΑΦΟΡΕΣ

[1] R. M. Karp, «Reducibility among combinatorial problems,» σε *Complexity of Computer Computations*, Νέα Ιόρκη, 1972, pp. 85-103.

[2] B. Ballard, σε *Computer Vision*, Englewood Cliffs, New Jersey, Prentice-Hall, Inc, 1982.

[3] P. M. Pardalos και G. P. Rodgers, «A branch and bound algorithm for the maximum clique problem,» *Computers Ops res Vol 19, No. 5,* pp. 363-375, 1992.

[4] P. Östergård, «A new algorithm for the maximum-weight clique problem,» *Nordic Journal of Computing,* p. 424–436, 8 4 2001.

[5] D. Kumlander, «A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search,» *Proceedings of the 5th international conference on modelling, computation and optimization in information systems and management sciences,* pp. 202-208, 2004.

[6] E. Tomita και T. Seki, «An efficient branch-and-bound algorithm for finding a maximum clique,» *In Discrete mathematics and theoritical computer science,* pp. 278-289, 2003.

[7] J. Konc και D. Janezic, «An improved branch and bound algorithm for the maximum clique problem,» *Commnications in Mathematical and in Computer Chemistry (58),* pp. 569-590, 2007.

[8] J. H. Q. Wu, «A clique-based exact method for optimal winner determination in combinatorial auctions,» *Information Sciences (334),* p. 103–121, 2016.

[9] C.-M. L. K. X. Z. Fang, «An exact algorithm based on MaxSAT reasoning for the maximum weight clique problem,» *Journal of Artificial Intelligence Research (55) ,* p. 799–833, 2016.

[10] I. M. Bomze, M. Pelillo και V. Stix, «Approximating the maximum weigh clique using replicator dynamics,» *Neural Networks, IEEE Transactions on 11(6),* pp. 1228-1241, 2000.

[11] W. J. Pullan, «Approximating the maximum vertex/edge weighted clique using local search,» *Journal of Heuristics 14 (2),* pp. 117-134, April 2008.

[12] Q. Wu, J.-K. Hao και F. Glover, «Multi-neighborhood tabu search for the maximum weight clique problem,» *Annals of Operations Research 196(1),* pp. 611-634.

[13] Y. Wang, S. Cai και M. Yin, «Two Efficient Local Search Algorithms for Maximum Weight Clique Problem,» *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16),* pp. 805-811, 2016.

[14] A. P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, 2005.

[15] J. Kennedy και R. Eberhart, «Particle Swarm Optimization,» σε *IEEE International Conference on Neural Networks*, Perth, Australia, 1995.

[16] J. Kennedy και R. Eberhart, «A discrete binary version of the particle swarm Algorithm,» σε *IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, Florida, USA, 1997.

[17] M. F. Tasgetiren και Y.-C. Liang, «A Binary Particle Swarm Optimization Algorithm for Lot Sizing Problem,» *Journal of Economic and Social Research 5 (2),* pp. 1-20, 2003.

[18] M. A. Khanesar, M. Teshnehlab και M. A. Shoorehdeli, «A Novel Binary Particle Swarm Optimization,» σε *2007 Mediterranean Conference on Control & Automation*, Αθήνα, 2007.

[19] M. Dorigo, Optimization, learning and natural algorithms, Μιλάνο, Ιταλία, 1992.

[20] M. Dorigo, «Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem,» *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION (VOL. 1, NO. 1) ,* Απρίλιος 1997.