ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΥΓΚΡΙΣΗ ΓΕΝΙΚΩΝ ΕΥΡΙΣΤΙΚΩΝ ΔΙΑΤΑΞΗΣ ΜΕΤΑΒΛΗΤΩΝ (VARIABLE ORDERING HEURISTICS) ΓΙΑ ΔΥΑΔΙΚΑ ΠΡΟΒΛΗΜΑΤΑ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ

ΙΩΑΝΝΗΣ ΖΟΥΝΟΣ

ΠΑΝΑΓΙΩΤΗΣ ΝΤΕΜΣΙΑΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ

ΔΡ. ΚΩΝΣΤΑΝΤΙΝΟΣ ΣΤΕΡΓΙΟΥ

ΚΟΖΑΝΗ ΟΚΤΩΒΡΙΟΣ 2018

Περίληψη

Ένας σημαντικός κλάδος, στον οποίον βρίσκει εφαρμογή η σύγχρονη Τεχνητή Νοημοσύνη, είναι η επίλυση προβλημάτων. Στην κατηγορία αυτή υπάγονται τα Δυαδικά Προβλήματα Ικανοποίησης Περιορισμών (Binary Constraint Satisfaction Problems -CSPs). Η επίλυση των προβλημάτων αυτών επιτυγχάνεται με χρήση της αναζήτησης και της διάδοσης περιορισμών (Constraint Propagation). Ο αλγόριθμος MAC (Maintaining Arc Consistency) θεωρείται ως ο πιο διαδεδομένος αλγόριθμος για την επίλυση τέτοιου είδους προβλημάτων. Με την χρήση ευριστικών μηχανισμών (heuristics), ο συγκεκριμένος αλγόριθμος-επιλυτής (solver) επιτυγχάνει σημαντική βελτίωση της απόδοσής του.
Σκοπός της συγκεκριμένης διπλωματικής εργασίας είναι η σύγκριση των καλύτερων γενικών ευριστικών μηχανισμών διάταξης μεταβλητών (Variable Ordering Heuristics VOHs), ως προς τον χρόνο εκτέλεσης του αλγορίθμου MAC καθώς και ως προς τον αριθμό των κόμβων που δημιουργεί.
Πιο συγκεκριμένα, οι τέσσερεις βασικοί ευριστικοί μηχανισμοί οι οποίοι εξετάστηκαν είναι ο dom/ddeg (Domain over Dynamic Degree), ο dom/wdeg (Domain over Weighted Degree), ο ABS (Activity Based Search) και ο IBS (Impact Based Search), από τους οποίους παρατηρήθηκε μέσα από την πειραματική διαδικασία ότι ο dom/wdeg αποτελεί την βέλτιστη επιλογή. Τέλος, πραγματοποιήθηκε μελέτη του Συνδυαστικού ευριστικού μηχανισμού Common Choice, ο οποίος αποτελεί την συνδυαστική λειτουργία των παραπάνω τεσσάρων και παρουσιάζει δυναμική συμπεριφορά.

Λέξεις Κλειδιά: Δυαδικά Προβλήματα Ικανοποίησης Περιορισμών, Διάδοση Περιορισμών, Αλγόριθμος Διατήρησης Συνέπειας Τόξου, Ευριστικός Μηχανισμός Διάταξης Μεταβλητών

Abstract

An important branch of modern Artificial Intelligence is the problem solving. This area of AI includes Binary Constraint Satisfaction Problems –CSPs. The solution of these problems, is achieved through search and Constraint Propagation. MAC (Maintaining Arc Constistency) algorithm, is considered as the most popular algorithm, to resolve such kind of problems. Using heuristics, the specific algorithm-solver, improves significantly its performance.
The purpose of this paper, is the comparison of the best general Variable Ordering Heuristics VOHs in terms of MAC algorithm's execution time and in terms of the number of nodes it creates.
Specifically, the four basic heuristics which we examine are dom/ddeg (Domain over Dynamic Degree), dom/wdeg (Domain over Weighted Degree), ABS (Activity Based Search) and IBS (Impact Based Search), from which dom/wdeg is considered as the best choice through the experimental process. Finally, Common Choice heuristic has been studied, which is the combined function of the above four heuristics and it presents dynamic behavior.

Key Words: Binary Constraint Satisfaction Problems, Constraint Propagation, Maintaining Arc Consistency, Variable Ordering Heuristics.

Πίνακας Περιεχομένων

1. Εισαγωγή

1.1 Τεχνητή Νοημοσύνη

Η Τεχνητή Νοημοσύνη (ΤΝ) είναι ένα από τα πιο νέα ερευνητικά πεδία και αναφέρεται στον κλάδο εκείνο της πληροφορικής που ασχολείται με την ανάπτυξη ευφυών συστημάτων. Ο όρος της αποδίδεται στον John McCarthy [15] ο οποίος τον εισήγαγε για πρώτη φορά το 1956 στη διάσκεψη του Dartmouth. Ως ευφυή συστήματα θεωρούμε τα υπολογιστικά εκείνα συστήματα, τα οποία σκέφτονται και ενεργούν όπως ο άνθρωπος.

1.2 Ο ορισμός της Τεχνητής Νοημοσύνης

Με μια αναζήτηση στη βιβλιογραφία, μπορεί κανείς να διαπιστώσει μια σειρά από ορισμούς για τη τεχνητή νοημοσύνη. Παρόλα αυτά, ένας γενικός ορισμός της έννοιας θα μπορούσε να είναι ο παρακάτω [21]:

Η Τεχνητή Νοημοσύνη είναι ο τομέας της Επιστήμης των Υπολογιστών που ασχολείται με τη σχεδίαση και την υλοποίηση προγραμμάτων τα οποία είναι ικανά να μιμηθούν τις ανθρώπινες γνωστικές ικανότητες, εμφανίζοντας έτσι χαρακτηριστικά που αποδίδουμε συνήθως σε ανθρώπινες συμπεριφορά, όπως η επίλυση προβλημάτων, η αντίληψη μέσω της όρασης, η μάθηση, η εξαγωγή συμπερασμάτων, η κατανόηση φυσικής γλώσσας, κλπ.

1.3 Η εξέλιξη της Τεχνητής Νοημοσύνης

Μελετώντας την εξέλιξη της τεχνητής νοημοσύνης, μπορούμε να διαπιστώνουμε ότι διακρίνεται σε δύο κατηγορίες, την *κλασική* ή *συμβολική* και τη *συνδετική(connectionist approach)* ή *μη-συμβολική*. Συγκεκριμένα, η ***συμβολική*** βασίζεται στην κατανόηση των νοητικών διεργασιών και ασχολείται με τη προσομοίωση της ανθρώπινης νοημοσύνης, προσεγγίζοντάς την με αλγορίθμους και συστήματα που βασίζονται στη γνώση. Από την άλλη πλευρά, η ***μη-συμβολική*** βασίζεται στη μίμηση της βιολογικής λειτουργίας του εγκεφάλου προσεγγίζοντας το θέμα με τα νευρομορφικά ή νευρωνικά δίκτυα (neural networks).

1.4 Επίλυση προβλημάτων

Η επίλυση προβλημάτων είναι κλάδος της τεχνητής νοημοσύνης (ΤΝ) ο οποίος αφορά τον σχεδιασμό κατάλληλων ενεργειών με στόχο την μετάβαση ενός ελέγξιμου συστήματος σε μία αποδεκτή τελική κατάσταση, εκκινώντας από κάποια προκαθορισμένη αρχική κατάσταση. Συνήθως επιτυγχάνεται μέσω ενός αλγορίθμου ο οποίος λαμβάνει ως είσοδο το δοθέν πρόβλημα και επιστρέφει ως έξοδο μία λύση του προβλήματος, αφού αξιολογήσει πρώτα μία ομάδα υποψηφίων λύσεων. Πρόκειται για ένα θεμελιώδες γνωστικό πεδίο της τεχνητής νοημοσύνης το οποίο γνώρισε μεγάλη ανάπτυξη ήδη από τη δεκαετία του 1950. Αποτέλεσε μία προσπάθεια αλγοριθμικής εξομοίωσης της διαδικασίας της σκέψης, ενώ με τον καιρό ενσωμάτωσε μεθοδολογίες από τη θεωρία βελτιστοποίησης και τη θεωρία γράφων.

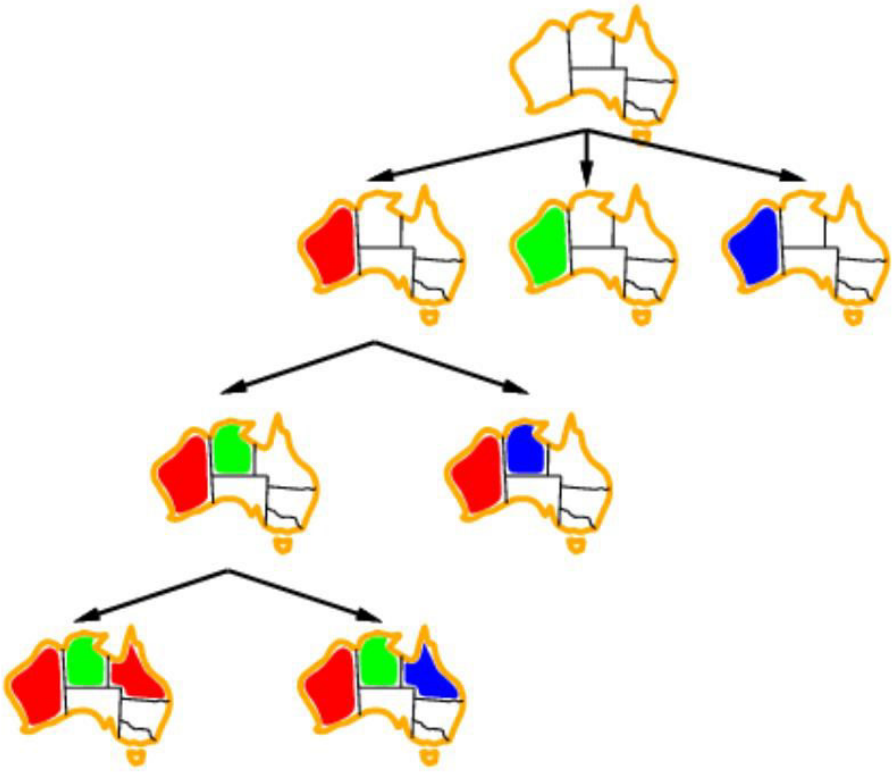
Ένα πρόβλημα είναι ένα σύνολο αντικειμένων, ιδιοτήτων και σχέσεων το οποίο ορίζεται από μία αρχική κατάσταση, μία επιθυμητή τελική κατάσταση και τις επιτρεπτές ενέργειες στα αντικείμενα του προβλήματος. Στόχος είναι, ξεκινώντας από την αρχική κατάσταση, να γίνει μία κατάλληλη ακολουθία ενεργειών η οποία θα καταλήγει στην τελική κατάσταση. Αυτή η διαδικασία ονομάζεται επίλυση του προβλήματος (π.χ. η διεξαγωγή μίας παρτίδας σκάκι). Η επίλυση προβλημάτων έχει προφανώς σπουδαίες εφαρμογές στην απόδειξη θεωρημάτων, στο χρονοπρογραμματισμό ενεργειών, στη διεξαγωγή παιγνίων κλπ, ενώ θεωρείται κεντρικό χαρακτηριστικό της ευφυΐας. Βασικό στοιχείο στην επίλυση προβλημάτων είναι η αναπαράσταση τους. Για το σκοπό αυτό, προτεινόμενη μέθοδος είναι η αναπαράσταση με χώρο καταστάσεων.

Στη μέθοδο του χώρου καταστάσεων βασική δομική μονάδα είναι η κατάσταση, το σύνολο δηλαδή των αντικειμένων που εμπλέκονται στο πρόβλημα μαζί με τις ιδιότητες και τις μεταξύ τους σχέσεις. Η κατάσταση ορίζεται ως ένα απλουστευμένο, αφαιρετικό μοντέλο του κόσμου ενώ το σύνολο των καταστάσεων (στιγμιότυπων) στις οποίες μπορεί να βρεθεί αυτός ο κόσμος του προβλήματος ονομάζεται χώρος καταστάσεων. Το ίδιο το πρόβλημα ορίζεται με βάση την αρχική κατάσταση από την οποία ξεκινάμε, την επιθυμητή τελική κατάσταση στην οποία πρέπει να καταλήξουμε (ή πολλές δυνατές τελικές) και το σύνολο των τελεστών μετάβασης, δηλαδή των επιτρεπτών πράξεων που μπορούν να εκτελεστούν στα αντικείμενα μίας κατάστασης οδηγώντας σε μια άλλη (π.χ. στην αναπαράσταση μίας παρτίδας σκάκι τελεστής είναι η έγκυρη μετακίνηση ενός πιονιού). Λύση του προβλήματος είναι μία ακολουθία διαδοχικών τελεστών μετάβασης και καταστάσεων που ξεκινά από μία αρχική κατάσταση και καταλήγει σε μία τελική.

Τέλος, ως ένας παραστατικός και συγχρόνως κατανοητός τρόπος αναπαράστασης της διαδικασίας επίλυσης προβλημάτων, μπορεί να θεωρηθεί ο δενδρικός γράφος. Κατά τον τρόπο αυτό, θεωρούμε τη ρίζα του δέντρου ως αρχική κατάσταση, τους κόμβους ως ενδιάμεσες καταστάσεις, τα κλαδιά ως τελεστές μετάβασης και τέλος, τα φύλλα

του δέντρου ως τελικές ή/και αδιέξοδες καταστάσεις. Σε αυτό το σημείο πρέπει να τονιστεί ότι σε προβλήματα ικανοποίησης περιορισμών η αναζήτηση για εύρεση κάποιας λύσης μπορεί να οδηγήσει σε μια κατάσταση αδιεξόδου οπότε σε αυτή την περίπτωση δεν μπορεί να συνεχιστεί η αναζήτηση περαιτέρω (για αυτό τον λόγο και κάποιο φύλλο του δέντρου μπορεί να αναπαριστά και αδιέξοδα εκτός από τελικές καταστάσεις-λύσεις). Στη περίπτωση λοιπόν κάποιας λύσης του προβλήματος θεωρούμε ως λύση το μονοπάτι που δημιουργείται από τη ρίζα του δέντρου ως το φύλλο εκείνο που αντιστοιχεί σε τελική κατάσταση. Βέβαια σε πραγματικά προβλήματα το μέγεθος αυτού του δένδρου γίνεται εξαιρετικά μεγάλο μετά την επέκταση λίγων μόλις κόμβων και επομένως η αναζήτηση λύσεων σε ένα τέτοιο δένδρο καθίσταται εξαιρετικά χρονοβόρα. Αυτό το ζήτημα στη βιβλιογραφία της ΤΝ αναφέρεται ως ***συνδυαστική έκρηξη***.

Ένα παράδειγμα δενδρικού γράφου είναι το παρακάτω στο οποίο φαίνεται η μορφή αναζήτησης που θα έπαιρνε ένα πρόβλημα χρωματισμού γράφου και στη περίπτωσή μας ο χρωματισμός του χάρτη των πολιτειών της Αυστραλίας με τη χρήση τριών χρώματα (κόκκινο, μπλε, πράσινο) :



1.1 Παράδειγμα Δενδρικού Γράφου

2. Προβλήματα Ικανοποίησης Περιορισμών

Μία σημαντική κατηγορία προβλημάτων αναζήτησης είναι τα προβλήματα ικανοποίησης περιορισμών (constraint satisfaction problems - CSP). Σε αυτά δεν είναι πλήρως γνωστές οι τελικές καταστάσεις, μόνο κάποιες ιδιότητες τους καθώς και οι περιορισμοί τους οποίους πρέπει να ικανοποιούν. Επίσης δε μας ενδιαφέρει το μονοπάτι που οδηγεί στη λύση: το μόνο που επιθυμούμε να βρούμε είναι η πλήρης μορφή μίας τελικής κατάστασης.

Επιπρόσθετα, τα προβλήματα ικανοποίησης περιορισμών (constraint satisfaction problems - CSP) είναι προβλήματα τα οποία παρουσιάζουν απλή δομή και επιδέχονται μια απλή αναπαράσταση.

Για το είδος αυτών των προβλημάτων, υπάρχουν αλγόριθμοι αναζήτησης οι οποίοι εκμεταλλεύονται αυτή την απλή αναπαράσταση και χρησιμοποιούν απλούς ευριστικούς μηχανισμούς (και όχι ειδικούς για το συγκεκριμένο πρόβλημα) με σκοπό την επίλυση μεγάλων προβλημάτων.

Ένα ακόμη στοιχείο το οποίο αξίζει να αναφέρουμε για τα CSP, είναι το γεγονός ότι η απλή δομή τους, μας δίνει επιπλέον τη δυνατότητα, να ορίσουμε μεθόδους αποσύνθεσης προβλημάτων (problem decomposition) και μας προσφέρει μια βαθύτερη κατανόηση της σχέσης μεταξύ της δομής ενός προβλήματος και της δυσκολίας επίλυσής του.

2.1 Ορισμός CSP

Ένα πρόβλημα ικανοποίησης περιορισμών (CSP) ορίζεται ως :

- Ένα σύνολο από μεταβλητές X_1 ,…, X_n. Κάθε μεταβλητή X_i έχει ένα μη κενό πεδίο (domain) δυνατών τιμών D_i.
- Ένα σύνολο από περιορισμούς C_i ,…, C_m. Ένας περιορισμός καθορίζει τους επιτρεπτούς συνδυασμούς τιμών για ένα υποσύνολο του συνόλου των μεταβλητών.
- Τυπικά, ένας k-αδικός περιορισμός C με μεταβλητές X_1 ,…, X_k είναι ένα υποσύνολο του καρτεσιανού γινομένου D_1 x…xD_k .

Σε κάθε πρόβλημα ικανοποίησης περιορισμών, μπορεί να δοθεί μια αυξητική διατύπωση (incremental formulation), όπως σε ένα συνηθισμένο πρόβλημα αναζήτησης, με τον παρακάτω τρόπο:

- **Αρχική κατάσταση:** Η κενή ανάθεση τιμών { }, όπου δεν έχει δοθεί τιμή σε καμιά από τις μεταβλητές.
- **Συνάρτηση διαδοχών:** Μπορεί να δοθεί τιμή σε οποιαδήποτε μεταβλητή δεν έχει ήδη δοθεί, εφόσον αυτή (η ανάθεση) δεν συγκρούεται με τις προηγούμενες αναθέσεις τιμών στις μεταβλητές.
- **Έλεγχος στόχου:** Η τρέχουσα ανάθεση τιμών είναι πλήρης. Δηλαδή, όλες η μεταβλητές του προβλήματος να έχουν λάβει τιμή.
- **Κόστος διαδρομής:** Ένα σταθερό κόστος για κάθε βήμα.

Μία **λύση** ενός προβλήματος ικανοποίησης περιορισμών είναι μία ανάθεση τιμών σε όλες τις μεταβλητές, η οποία ικανοποιεί όλους τους περιορισμούς. Ένα CSP ονομάζεται **συνεπές (consistent)** αν έχει μία τουλάχιστον λύση, διαφορετικά ονομάζεται **ασυνεπές (inconsistent)**.

Ένα παράδειγμα προβλήματος ικανοποίησης περιορισμών είναι αυτό των n-queens στο οποίο σκοπός είναι να τοποθετήσουμε τις n (πλήθος) βασίλισσες queens σε μία σκακιέρα διαστάσεων n x n με τέτοιο τρόπο ώστε 2 βασίλισσες να μη βρίσκονται ταυτόχρονα στην ίδια γραμμή , στήλη ή διαγώνιο.

Για n = 4 προκύπτει ένα πρόβλημα P με 4 μεταβλητές X_i (i = 1,2,3,4), όπου η κάθε μία παριστάνει τη στήλη που καταλαμβάνει η i-οστή βασίλισσα στη i-οστή γραμμή.

Αν οι στήλες παριστάνονται από τους αριθμούς 1,...,4, τότε το πεδίο κάθε μεταβλητής X_i είναι :

$$D_i = \{1,2,3,4\}.$$

Για κάθε ζεύγος μεταβλητών X_i και X_j, έχουμε δύο δυαδικούς περιορισμούς :

$$X_i \neq X_j \quad (\text{δεν υπάρχει ζευγάρι βασιλισσών στην ίδια γραμμή})$$
$$|i-j| \neq |X_i - X_j| \quad (\text{δεν υπάρχει ζευγάρι βασιλισσών στην ίδια διαγώνιο})$$

2.1.1 Είδη περιορισμών

Ανάλογα με το πόσες μεταβλητές περιλαμβάνει ένας περιορισμός, αυτός χαρακτηρίζεται ως :

- **Μοναδιαίος (unary)** → Όταν περιλαμβάνει μία μεταβλητή : π.χ. $X_1 > 0$
- **Δυαδικός (binary)** → Όταν περιλαμβάνει δύο μεταβλητές : π.χ. $X_1 > X_2$
- **Ανώτερης τάξης (higher order)** → Όταν περιλαμβάνει περισσότερες από δύο μεταβλητές : π.χ. $X_1 + X_2 + X_3 > 0$

2.2 Μέθοδοι αναζήτησης

Στόχος της τεχνητής νοημοσύνης είναι (ιδιαίτερα σε πολύπλοκα προβλήματα) να ελαχιστοποιηθεί το κόστος της διαδρομής-επίλυσης. Τα χαρακτηριστικά των διάφορων μεθόδων αναζήτησης, διακρίνονται στα τέσσερα παρακάτω :

- Εάν επιτυγχάνεται η εύρεση της πιο οικονομικής (βέλτιστης) λύσης (optimal search) ή απλά αναζητείται κάποια διαδρομή (non-optimal search).
- Εάν η αναζήτηση πραγματοποιείται τυφλά (blind search) ή είναι καθοδηγούμενη (informed search), δηλαδή εάν η κάθε μετάβαση πραγματοποιείται ανεξάρτητα ή βάση της ποιότητας της νέας κατάστασης (πόσο πιο κοντά εκτιμάται ότι νέα η κατάσταση πλησιάζει στην τελική κατάσταση) αντίστοιχα.
- Η **τυφλή** αναζήτηση είναι πιο απλή καθώς, δεν εμπεριέχεται πληροφορία για τον χώρο του προβλήματος, οπότε η μετάβαση γίνεται βάση κάποιο συστηματικού κριτηρίου. Αντίθετα, η **καθοδηγούμενη** αναζήτηση χρησιμοποιεί ευριστικές μεθόδους (heuristics) οι οποίες αφορούν την ποιότητα της μετάβασης.
- Εάν η μέθοδος είναι πλήρης (complete), δηλαδή εγγυάται ότι θα βρεθεί μία λύση, εφόσον υπάρχει.
- Εάν χρησιμοποιείται οπισθοδρόμηση σε περίπτωση μη εύρεσης λύσης.

2.3 Αλγόριθμοι αναζήτησης

Για προβλήματα ικανοποίησης περιορισμών έχουν προταθεί αρκετοί αλγόριθμοι όπως ο DFS (αναζήτηση πρώτα κατά βάθος), καθώς και παραλλαγές του αλγορίθμου οι οποίοι αναφέρονται ως αλγόριθμοι με οπισθοδρόμηση. Η βασική ιδέα γύρω από τη κατηγορία αυτών των αλγορίθμων είναι ότι επιλέγουμε τιμές για μία μεταβλητή και έπειτα ελέγχουμε αν οι περιορισμοί ικανοποιούνται. Σε περίπτωση όπου δεν υπάρχουν άλλες επιτρεπτές τιμές για να επιλεχθούν εκτελούμε οπισθοδρόμηση.

Η συγκεκριμένη διπλωματική επικεντρώνεται σε προβλήματα ικανοποίησης δυαδικών περιορισμών. Για το λόγο αυτό, ο αλγόριθμος που αναλύεται στις επόμενες ενότητες είναι ο αλγόριθμος “Διατήρησης Συνέπειας Τόξου” (maintaining arch consistency – MAC), ο οποίος και προτείνεται ως ο καταλληλότερος για την επίλυση δυαδικών προβλημάτων.

2.3.1 Διάδοση περιορισμών

Πριν την ανάλυση του MAC θα πρέπει να αναφέρουμε την έννοια της διάδοσης περιορισμών, καθώς σχετίζεται με αυτή.

Οι τεχνικές και οι αλγόριθμοι για την επίλυση των CSP ανήκουν κυρίως σε δύο κύριες κατηγορίες: συμπερασματικούς και αναζήτησης [18, 15]. Οι μέθοδοι συμπερασμάτων αποσκοπούν στην απλοποίηση ενός προβλήματος έτσι ώστε να διευκολύνεται η επίλυσή του, διατηρώντας παράλληλα τη σημασιολογία του, δηλαδή το σύνολο των λύσεων. Η απλούστευση μπορεί να επιτευχθεί μετασχηματίζοντας το σύνολο των μεταβλητών και των περιορισμών ή απορρίπτοντας τους ασύμβατους συνδυασμούς τιμών.

Πιο συγκεκριμένα, η κεντρική ιδέα της διάδοσης περιορισμών (constraint propagation) είναι να λάβουμε υπόψη μας τους δοθέντες περιορισμούς όσο πιο νωρίς μπορούμε κατά τη διάρκεια της αναζήτησης ή ακόμα και πριν αρχίσει η αναζήτηση.

Για παράδειγμα, σε κάθε βήμα της αναζήτησης μπορούμε να κλαδεύουμε τμήματα του χώρου αναζήτησης που απομένει να εξερευνηθεί, εξετάζοντας τις συνέπειες των μερικών αναθέσεων. Οι αλγόριθμοι που έχουν προταθεί για τη διάδοση περιορισμών ονομάζονται συχνά αλγόριθμοι που βλέπουν μπροστά (look-ahead).

2.3.2 Συνέπεια Τόξου (AC – Arc Consistency)

Συστατικό στοιχείο του MAC αποτελούν οι αλγόριθμοι Συνέπειας Τόξου Arc Consistency AC. Η συγκεκριμένη κατηγορία αλγορίθμων αποτελεί μια μορφή διάδοσης περιορισμών και θεωρείται η πλέον διαδεδομένη κατηγορία αλγορίθμων διάδοσης περιορισμών για CSP. Οι αλγόριθμοι της συνέπειας τόξου, εκμεταλλεύονται την πληροφορία που υπάρχει στους περιορισμούς για μείωση του χώρου αναζήτησης. Η βασική ιδέα αυτής της κατηγορίας αλγορίθμων είναι ότι εκτελεί απαλοιφή (διαγραφή) από τα αρχικά πεδία των μεταβλητών, κάποιων από τις τιμές οι οποίες δεν μπορούν να συμμετέχουν στην τελική λύση του προβλήματος.

Ένας γενικός ορισμός που προκύπτει είναι ο εξής :

Μια μεταβλητή X είναι συνεπής (arc consistent) αν για κάθε άλλη μεταβλητή Y ισχύει το εξής : Για κάθε τιμή a της X υπάρχει τουλάχιστον μια τιμή b της Y τέτοια ώστε η b να υποστηρίζει την a (δηλαδή, η a και b να είναι συμβατές).

Με άλλα λόγια, ο έλεγχος συνέπειας έγκειται στη διαδικασία κατά την οποία γίνεται διαγραφή από το πεδίο κάθε μεταβλητής, εκείνων των τιμών οι οποίες είναι ασυνεπείς ως προς κάποια άλλη μεταβλητή.
Η κύρια δυσκολία η οποία παρουσιάζεται σε αυτή την κατηγορία αλγορίθμων έγκειται στο γεγονός ότι η διαγραφή μιας τιμής από το πεδίο τιμών μιας μεταβλητής, οδηγεί σε αλλαγές στα πεδία άλλων μεταβλητών. Αυτό σημαίνει, ότι μετά από κάθε διαγραφή ασυνεπούς τιμής πρέπει να επανεξεταστούν τα πεδία των "άμεσα" συνδεδεμένων (γειτονικών) μεταβλητών.

2.3.3 AC – 3

Ο απλούστερος αποδοτικός αλγόριθμος στην κατηγορία της διάδοσης περιορισμών προτείνεται από τον Mackworth [6] και είναι ο AC-3, ο οποίος μπορεί να χειριστεί μόνο δυαδικούς περιορισμούς (στους οποίους συμμετέχουν το πολύ δύο μεταβλητές) και για αυτό το λόγο προτάθηκε αρχικά για δυαδικά δίκτυα. Τέλος, ο αλγόριθμος AC-3 καθίσταται ως ο πιο ευρέως χρησιμοποιούμενος αλγόριθμος, εξαιτίας της απλής και φυσικής δομής του.

Ο συγκεκριμένος αλγόριθμος περιγράφεται ως εξής :

Έστω ένα πρόβλημα ικανοποίησης περιορισμών P = <X,D,C> όπου :

X : Το σύνολο μεταβλητών του προβλήματος.

D : Το σύνολο των πεδίων τιμών της κάθε μεταβλητής του προβλήματος.

C : Το σύνολο των (δυαδικών) περιορισμών του προβλήματος.

Επίσης, Q μία ουρά στην οποία εισάγονται μεταβλητές προκειμένου να ελεγχθεί η συνέπειά τους.

Τότε, ο αλγόριθμος περιγράφεται ως εξής :

- Εισαγωγή όλων των μεταβλητών του συνόλου X στην ουρά Q.
- Όσο η Q δεν είναι άδεια επανέλαβε την παρακάτω διαδικασία :
    - Πάρε τη πρώτη μεταβλητή x_i (και αφαίρεσε την) από την ουρά Q.
    - Για κάθε περιορισμό $c_i=(x_i, x_j)$ της μεταβλητής x_i έλεγξε αν για κάθε τιμή d_j από το πεδίο τιμών της μεταβλητής x_j υπάρχει τουλάχιστον μία τιμή d_i από το πεδίο τιμών της μεταβλητής x_i τέτοια ώστε να ικανοποιείται ο περιορισμός $c_i=(x_i, x_j)$.
    - Αν δεν υπάρχει τέτοια τιμή d_i στο πεδίο τιμών της μεταβλητής x_i τότε διέγραψε τη τιμή d_j από το πεδίο τιμών της μεταβλητής x_j .
        - Αν το πεδίο τιμών της μεταβλητής x_j :
            - Μείνει κενό (αφαιρεθούν όλες οι τιμές του) επέστρεψε αποτυχία.
            - Δεν μείνει κενό αλλά έχει επηρεαστεί (αφαιρέθηκε τουλάχιστον μία τιμή) τότε, πρόσθεσε αν δεν υπάρχει ήδη την x_j στην ουρά Q.
    - Στη συνέχεια επαναλαμβάνεται η διαδικασία ελέγχου της Q, όπως ακριβώς περιγράφηκε στα παραπάνω βήματα.
- Αν η Q μείνει κενή, τότε επιστρέφει το σύνολο X.

Η παραπάνω διαδικασία μπορεί να περιγραφεί και σε μορφή ψευδοκώδικα ως εξής :

```
1.  function AC3 (X,C,D) {
2.          Q ← X
3.          while Q {
4.                  X_i ← Q[0]
5.                  Q.remove(X_i)
6.                  for each X_j in each constraint(X_i, X_j) {
7.                          if REVISE (X_i, X_j, D_i, D_j, constraint(X_i, X_j)) then {
8.                                  if D_j == 0 then return FAIL
9.                                  else {
10.                                         if X_j not in Q then Q.add(X_j)
11.                                 }
12.                         }
13.                 }
14.         }
15.         return SUCCESS, X
16. }
17.
18. function REVISE (X_i, X_j, D_i, D_j, constraint(X_i, X_j)) {
19.         DELETE ← 1
20.         for each value_j in D_j {
21.             for each value_i in D_i {
22.                 if (value_j, value_i) satisfies constraint(X_i, X_j) or  constraint(X_j, X_i) then {
23.                     DELETE ← 0
24.                 }
25.             }
26.             if DELETE == 1 then D_j.remove(value_j)
27.         }
28.         return DELETE
29. }
```

• Όπως φαίνεται στη γραμμή 2 του παραπάνω κώδικα η ουρά Q αρχικοποιείται με το σύνολο X, των μεταβλητών του.

• Όσο η Q δεν είναι κενή, αποθηκεύει σε μια προσωρινή μεταβλητή X_i την πρώτη μεταβλητή της ουράς και παράλληλα την αφαιρεί από την Q (γραμμές 4 και 5 αντίστοιχα).

• Στη συνέχεια, για κάθε περιορισμό παίρνει τη γειτονική μεταβλητή X_j της X_i, (δηλαδή, η μεταβλητή X_j συμμετέχει σε κάποιον περιορισμό με την X_i) και εκτελείται η συνάρτηση REVISE (γραμμή 7), η οποία εκτελεί τον έλεγχο συνέπειας τόξου, διαγράφοντας τιμές οι οποίες είναι ασυνεπείς.

• Πιο συγκεκριμένα, στη REVISE χρησιμοποιούμε μία μεταβλητή DELETE με αρχική τιμή 1 (γραμμή 19).

• Έπειτα, για κάθε τιμή value_j του πεδίου τιμών (D_j) της X_j γίνεται έλεγχος αν υπάρχει τουλάχιστον μία τιμή value_i στο πεδίο τιμών (D_i) της X_i, ώστε το ζεύγος τιμών (value_j, value_i) να ικανοποιεί τον περιορισμό $C(X_i, X_j)$ ή $C(X_j, X_i)$.

- Αν υπάρχει κάποια τιμή τότε η μεταβλητή DELETE γίνεται 0 (γραμμή 23).
- Αν η DELETE δεν αλλάξει τιμή και παραμείνει 1 τότε σημαίνει ότι δεν υπάρχει τιμή στο D_i για την οποία να ικανοποιείται ο περιορισμός, οπότε διαγράφεται η συγκεκριμένη τιμή από το πεδίο τιμών D_j της μεταβλητής X_j (γραμμή 26).
- Μόλις, τελειώσει ο έλεγχος συνέπειας, τότε επιστρέφεται η τιμή της DELETE (γραμμή 28).
- Επιστρέφοντας στον AC-3 γίνεται έλεγχος της επιστρεφόμενης τιμής, της συνάρτησης REVISE (γραμμή 7).
    - Αν είναι 0 τότε προκύπτει το συμπέρασμα ότι υπάρχει τουλάχιστον μία τιμή στο πεδίο τιμών D_i για κάθε τιμή στο πεδίο τιμών D_j, ώστε να ικανοποιείται ο αντίστοιχος περιορισμός $C(X_i, X_j)$ ή $C(X_j, X_i)$, ή με άλλα λόγια δεν έχει γίνει κάποια διαγραφή στο πεδίο τιμών της X_j.
- Στη συνέχεια, επιλέγουμε την επόμενη μεταβλητή με την οποία συμμετέχει η X_i στον επόμενο περιορισμό (γραμμή 6).
- Σε περίπτωση όμως, που η επιστρεφόμενη τιμή της DELETE είναι 1 τότε, σημαίνει ότι το πεδίο τιμών της X_j έχει μεταβληθεί (μειωθεί), οπότε ελέγχεται (γραμμή 8) αν το πεδίο τιμών της X_j είναι κενό.
    - Αν το πεδίο είναι κενό τότε η εκτέλεση του AC-3 τερματίζει και επιστρέφουμε αποτυχία, σε άλλη περίπτωση (αν το πεδίο δεν είναι κενό), ελέγχουμε αν η μεταβλητή X_j υπάρχει ήδη μέσα στην ουρά Q (γραμμή 10).
        - Αν δεν υπάρχει τότε προστίθεται, ενώ σε άλλη περίπτωση επιστρέφουμε στην αρχή του βρόχου (γραμμή 6) για την επιλογή της επόμενης μεταβλητής από τον επόμενο περιορισμό της X_i.
- Μόλις, ελέγξουμε όλες τις γειτονικές μεταβλητές της X_i παίρνουμε την επόμενη μεταβλητή από την Q και επαναλαμβάνεται η παραπάνω διαδικασία.
- Όταν η Q μείνει κενή τότε επιστρέφουμε επιτυχία και το σύνολο μεταβλητών X του προβλήματος με τα μειωμένα (ή όχι) πεδία τιμών τους.

Ο έλεγχος της συνέπειας τόξου, μπορεί να εφαρμοστεί είτε ως στάδιο προεπεξεργασίας πριν τη διαδικασία της αναζήτησης είτε ως στάδιο διάδοσης μετά από κάθε ανάθεση τιμής κατά την αναζήτηση. Και στις δύο περιπτώσεις, ο έλεγχος αυτός θα πρέπει να εφαρμόζεται επανειλημμένα μέχρι να μην απομείνουν άλλες ασυνέπειες.

Η πολυπλοκότητα του ελέγχου της συνέπειας τόξου μπορεί να αναλυθεί με τον εξής τρόπο: Ένα δυαδικό πρόβλημα ικανοποίησης περιορισμών (binary CSP) έχει το πολύ $O(n^2)$ τόξα, όπου ως τόξο θεωρούμε ένα περιορισμό ανάμεσα σε δύο μεταβλητές. Κάθε τόξο (X_j, X_i) μπορεί να εισαχθεί στην ουρά μόνο d φορές, επειδή η X_i έχει το πολύ d τιμές που μπορούν να διαγραφούν. Ο έλεγχος της συνέπειας τόξου μπορεί να γίνει σε χρόνο $O(d^2)$. Επομένως, ο ολικός χρόνος της χειρότερης περίπτωσης είναι $O(n^2d^3)$. Αν και η μέθοδος αυτή είναι σημαντικά δαπανηρή, συνήθως αξίζει τον κόπο παρά το επιπλέον κόστος.

2.3.4 Αλγόριθμος αναζήτησης με υπαναχώρηση-οπισθοδρόμηση (Backtracking Search)

Μια προσπάθεια για την επίλυση προβλημάτων ικανοποίησης περιορισμών, γενικά απαιτεί κάποια μορφή αναζήτησης. Ο όρος *αναζήτηση με υπαναχώρηση-οπισθοδρόμηση (backtracking search)*, χρησιμοποιείται για αναζήτηση πρώτα σε βάθος, κάνοντας επιλογή τιμών για μία μόνο μεταβλητή τη φορά και υπαναχωρεί όταν μια μεταβλητή δεν έχει άλλες επιτρεπτές τιμές που μπορούν να ανατεθούν. Γενικά, μπορούμε να πούμε ότι τέτοιου είδους μέθοδοι εξερευνούν το δέντρο αναζήτησης με ένα συστηματικό τρόπο, και μπορούν να εγγυηθούν ότι μια λύση θα βρεθεί εφόσον υπάρχει στο πρόβλημα, ή το πρόβλημα θα αποδειχθεί ως μη επιλύσιμο εφόσον δεν υπάρχει κάποια λύση.

Ο αλγόριθμος, ουσιαστικά χρησιμοποιεί την μέθοδο της αυξητικής παραγωγής ενός διαδόχου τη φορά. Η παραγωγή ενός διαδόχου, γίνεται με βάση την τρέχουσα ανάθεση τιμής. Επειδή η αναπαράσταση των προβλημάτων ικανοποίησης περιορισμών είναι τυποποιημένη, δεν είναι ανάγκη να παρέχουμε στον αλγόριθμο **BACKTRACKING SEARCH** μια ειδική για το συγκεκριμένο πρόβλημα αρχική κατάσταση, συνάρτηση διαδοχών ή έλεγχο στόχου. Η μοναδική πληροφορία η οποία απαιτείται για την εκτέλεση του αλγορίθμου είναι η αποθήκευση του τρέχοντος μονοπατιού αναζήτησης (του μονοπατιού που εξερευνήθηκε τελευταία) καθώς ο αλγόριθμος αναζητά μία λύση τη φορά.

Θεωρείται ως ο καταλληλότερος αλγόριθμος για την επίλυση προβλημάτων ικανοποίησης περιορισμών, κυρίως γιατί απαιτεί χώρο αναζήτησης πολυωνυμικού βαθμού.

Παρακάτω παρουσιάζεται ο ψευδοκώδικας του αλγορίθμου Backtracking :

```
1.  function BACKTRACKING-SEARCH (X, D, C) {
2.      while a assignment is not complete {
3.          variable ← SELECT-UNASSIGNED-VARIABLE (X, D, C)
4.          return ← 1
5.          for each value in ORDER-DOMAIN-VALUES (D (variable)) {
6.              if value is consistent with assignment according to Constraints (X, D) then{
7.                  add {variable = value} to assignment
8.                  result ← 0
9.                  break
10.             }
11.         }
12.         if result == 1 and assignment is not empty then {
13.             remove {previous-variable = value} from assignment
14.             variable ← previous-variable
15.         }
16.         else if variable is start-variable and assignment is empty then {
17.             return failure
```

```
18.                 }
19.         }
20.         return solution
21.  }
```

Ο αλγόριθμος δέχεται ως είσοδο το πρόβλημα (σύνολο μεταβλητών X, σύνολο πεδίων τιμών D των μεταβλητών και σύνολο περιορισμών C). Όσο όλες οι μεταβλητές του συστήματος δεν έχουν λάβει τιμή, επαναλαμβάνεται η ακόλουθη διαδικασία (γραμμή 2):

- Πάρε μία μεταβλητή , η οποία δεν έχει λάβει τιμή. (γραμμή 3)
- Σε μία προσωρινή (βοηθητική) μεταβλητή **"result"**, θέσε την τιμή 1 (γραμμή 4).
- Για κάθε τιμή **"value"** από το ταξινομημένο πεδίο ορισμού της μεταβλητής, έλεγξε αν η συγκεκριμένη τιμή ικανοποιεί το σύνολο των περιορισμών (γραμμή 5-6).
  - Σε περίπτωση όπου ικανοποιείται το σύνολο των περιορισμών, πρόσθεσε την ανάθεση της τιμής, που έγινε στην μεταβλητή (**add {variable = value} to assignment**), στο σύνολο αναθέσεων (γραμμή 7). Επίσης, θέσε το **"result"** ίσο με 0, ώστε να μην γίνει οπισθοδρόμηση και σπάσε τον βρόχο (γραμμή 8) πηγαίνοντας στην επόμενη (ελεύθερη) μεταβλητή.
  - Αν δεν συμβαίνει το παραπάνω τότε έλεγξε την επόμενη τιμή για ανάθεση, από το πεδίο τιμών της μεταβλητής.
- Όταν ελεγχθούν όλες οι τιμές ή σπάσει ο βρόχος τότε, αν το **"result"** είναι ίσο με 1 και το σύνολο αναθέσεων δεν είναι κενό τότε, αφαίρεσε την προηγούμενη ανάθεση τιμής από το σύνολο αναθέσεων (**remove {variable = value} from assignment** (γραμμή 13)) και πάρε την μεταβλητή της οποίας μόλις αφαίρεσες την ανάθεση που αναφέρθηκε παραπάνω (γραμμή 14). Έπειτα, έλεγξε την ανάθεση της αμέσως επόμενης τιμής (γραμμή 5) για την μεταβλητή που μόλις πήρες.
- Αν η μεταβλητή που ελέγχεις τη δεδομένη στιγμή είναι η πρώτη μεταβλητή, δηλαδή η ρίζα κόμβος και το σύνολο αναθέσεων είναι κενό, τότε σημαίνει ότι το πρόβλημα δεν έχει λύση οπότε επέστρεψε **"αποτυχία" (failure)** (γραμμή 16).
- Αν το σύνολο των αναθέσεων έχει συμπληρωθεί, δηλαδή όλες οι μεταβλητές του προβλήματος έχουν λάβει από μία νόμιμη (επιτρεπτή) τιμή, τότε επέστρεψε τη λύση του προβλήματος (γραμμές 2 και 18).

2.3.5 Αλγόριθμος Διατήρησης Συνέπειας Τόξου (MAC - Maintaining Arch Consistency)

Η απλή οπισθοδρόμηση είναι ένας αλγόριθμος χωρίς πληροφόρηση, και έτσι δεν αναμένουμε να είναι πολύ αποτελεσματική για μεγάλα προβλήματα. Προκειμένου να

υπάρξει κάποια βελτίωση της απόδοσης των αλγορίθμων απληροφόρητης αναζήτησης γίνεται χρήση ειδικών ευριστικών μηχανισμών (Heuristics) τους οποίους θα αναλύσουμε αργότερα στην παρούσα διπλωματική. Οι μηχανισμοί αυτοί, προκύπτουν από τη γνώση που έχουμε για το πρόβλημα. Ωστόσο όπως προκύπτει, μπορούμε να επιλύσουμε προβλήματα ικανοποίησης περιορισμών αποδοτικά και χωρίς να έχουμε ειδική γνώση του προβλήματος. Για το λόγο αυτό βρίσκουμε μεθόδους γενικής χρήσης οι οποίοι απαντούν στα παρακάτω ερωτήματα:

1. Ποια είναι η επόμενη μεταβλητή στην οποία θα πρέπει να γίνει ανάθεση τιμής και με ποια σειρά θα πρέπει να δοκιμαστούν οι τιμές της;
2. Τι συνέπειες έχουν οι τρέχουσες αναθέσεις τιμών των μεταβλητών για τις άλλες μεταβλητές στις οποίες δεν έχει γίνει ακόμα κάποια ανάθεση τιμής;
3. Όταν μια διαδρομή αποτύχει, δηλαδή φτάσει σε μια κατάσταση στην οποία μια μεταβλητή δεν έχει νόμιμες τιμές (τιμές που να ικανοποιούν τους περιορισμούς), μπορεί η αναζήτηση να αποφύγει να επαναλάβει αυτή την αποτυχία στις επόμενες διαδρομές;

Ο αλγόριθμος οπισθοδρόμησης, ελέγχει μόνο τους περιορισμούς μεταξύ της τρέχουσας μεταβλητής και των παρελθουσών μεταβλητών.

Μια σημαντική τεχνική για τη βελτίωση της αποδοτικότητας είναι η διατήρηση ενός επιπέδου τοπικής συνέπειας κατά την αναζήτηση προς τα πίσω, με την εκτέλεση διάδοσης περιορισμών σε κάθε κόμβο του δέντρου αναζήτησης. Κάθε φορά που δημιουργείται ένα νέο υπό-πρόβλημα, αφαιρώντας τιμές από τα πεδία τιμών μελλοντικών μεταβλητών που είναι ασυμβίβαστες με την τρέχουσα ανάθεση, το υπό-πρόβλημα γίνεται arc consistent. Συνέπεια αυτού είναι η διαγραφή περαιτέρω τιμών από τα πεδία τιμών μελλοντικών μεταβλητών.

Η παραπάνω διαδικασία έχει δύο σημαντικά οφέλη. Κατ 'αρχάς, η αφαίρεση ασυνεπειών κατά την αναζήτηση μπορεί να κλαδεύει δραματικά το δέντρο αναζήτησης, αφαιρώντας πολλά αδιέξοδα και απλουστεύοντας το υπόλοιπο υπό-πρόβλημα. Σε ορισμένες περιπτώσεις, μια μεταβλητή θα έχει ένα κενό πεδίο τιμών μετά τη διάδοση του περιορισμού. Δηλαδή, καμία τιμή δεν θα ικανοποιεί τους περιορισμούς στους οποίους συμμετέχει η μεταβλητή αυτή. Σε αυτήν την περίπτωση, η οπισθοδρόμηση μπορεί να ξεκινήσει καθώς δεν υπάρχει λύση κατά μήκος αυτού του κλαδιού του δέντρου αναζήτησης. Σε άλλες περιπτώσεις, οι μεταβλητές θα έχουν υποστεί σημαντική μείωση των τιμών τους. Εάν ένα πεδίο τιμών μειωθεί σε μία μόνο τιμή, η τιμή αυτή επιβάλλεται στη μεταβλητή και δεν χρειάζεται να διακλαδωθεί στο μέλλον. Έτσι, μπορεί να είναι πολύ πιο εύκολο να βρεθεί λύση σε ένα CSP μετά από διάδοση περιορισμών ή να αποδειχθεί αντίστοιχα ότι ένα CSP δεν έχει λύση. Δεύτερον, μερικά από τα σημαντικότερα VOHs (Variable Ordering Heuristics) χρησιμοποιούν τις πληροφορίες που συλλέγονται από τη διάδοση των περιορισμών για την λήψη αποτελεσματικών αποφάσεων διάταξης μεταβλητών. Ως εκ τούτου,

είναι πλέον απαραίτητο για έναν αλγόριθμο οπισθοδρόμησης να ενσωματώσει κάποια μορφή διάδοσης περιορισμών.

Ο αλγόριθμος Διατήρησης Συνέπειας Τόξου **(Maintaining Arc Consistency - MAC)** [8], είναι αλγόριθμος αναζήτησης οπισθοδρόμησης ο οποίος, διατηρεί συνέπεια τόξου κατά τη διάρκεια της αναζήτησης και θεωρείται επί του παρόντος ως η πιο αποτελεσματική ολοκληρωμένη προσέγγιση γενικής χρήσης για την επίλυση προβλημάτων δυαδικών CSP.

Ο αλγόριθμος MAC καθίσταται ως ένας αλγόριθμος επίλυσης δυαδικών προβλημάτων ικανοποίησης περιορισμών και αποτελείται επί της ουσίας από τρία συστατικά μέρη :

- Τον αλγόριθμο Backtracking, οποίος αποτελεί και το σκελετό του αλγορίθμου MAC.
- Μια ευριστική συνάρτηση, η οποία είναι υπεύθυνη στο να εντοπίζει βάση κάποιου κριτηρίου την επόμενη μεταβλητή που θα λάβει μια τιμή.
- Έναν αλγόριθμο διάδοσης περιορισμών όπως ο AC και συγκεκριμένα η έκδοση AC-3, προκειμένου να επιτευχθεί όσο το δυνατόν περισσότερο μείωση του χώρου αναζήτησης του προβλήματος και άρα δημιουργία λιγότερων κόμβων στο δέντρο αναζήτησης.

Συνοπτικά ο τρόπος λειτουργίας του αλγορίθμου περιγράφεται ως εξής :

- Κάθε φορά μια ελεύθερη μεταβλητή του συνόλου μεταβλητών του προβλήματος(η οποία επιλέγεται βάση κάποιου κριτηρίου από μια ευριστική συνάρτηση) αρχικοποιείται με μία τιμή από το πεδίο τιμών της.
- Έπειτα, εκτελώντας τον AC-3 πραγματοποιείται η διάδοση περιορισμών όπως περιγράφεται σε προηγούμενη ενότητα.
  - Αν ο AC-3 επιστρέψει επιτυχία τότε η μεταβλητή προστίθεται στη λύση του προβλήματος και ακολουθείται ξανά η διαδικασία επιλογής της επόμενης μεταβλητής για ανάθεση.
  - Αν ο AC-3 αποτύχει εκτελείται αναίρεση όλων εκείνων των αλλαγών που προκλήθηκαν από την εν λόγω ανάθεση. Στη συνέχεια η ίδια μεταβλητή λαμβάνει την αμέσως επόμενη τιμή από το πεδίο τιμών της και ακολουθείται και πάλι η διαδικασία διάδοσης περιορισμών.
    - Στη περίπτωση βέβαια, όπου εξαντληθούν όλες οι τιμές του πεδίου τιμών της δηλαδή ο AC-3 αποτυγχάνει για κάθε ανάθεση τιμής στη συγκεκριμένη μεταβλητή τότε επιλέγεται η αμέσως προηγούμενη μεταβλητή που έλαβε κάποια τιμή και αναιρώντας τις αλλαγές που προκλήθηκαν από τη συγκεκριμένη ανάθεση, θέτουμε στη μεταβλητή αυτή την αμέσως επόμενη τιμή του πεδίου τιμών της και εκτελείται και πάλι ο AC-3.

- Στη περίπτωση όπου πρέπει να πάρουμε τη προηγούμενη μεταβλητή και αυτή δεν υπάρχει τότε αυτό σημαίνει ότι βρισκόμαστε στη ρίζα του δέντρου αναζήτησης και επίσης ότι το συγκεκριμένο πρόβλημα δεν διαθέτει λύση, οπότε και ο MAC τερματίζει επιστρέφοντας "ΑΠΟΤΥΧΙΑ".

- Αν δεν υπάρχει άλλη ελεύθερη μεταβλητή στο σύνολο μεταβλητών του προβλήματος τότε σημαίνει ότι έχουμε φτάσει σε φύλλο-λύση του προβλήματος οπότε ο αλγόριθμος MAC επιστρέφει τη λύση του προβλήματος.

Επί της ουσίας, η διαδικασία επίλυσης τερματίζει όταν βρεθεί μια λύση, δηλαδή μια τιμή να έχει ανατεθεί σε κάθε μεταβλητή ή όταν επιτυγχάνεται μια από τις ακόλουθες συνθήκες:

- Το δέντρο αναζήτησης έχει διερευνηθεί πλήρως χωρίς να βρεθεί λύση (δηλαδή, η οπισθοδρόμηση να οδηγήσει πίσω στη ρίζα του δέντρου, όπου βρίσκεται η αρχική κατάσταση του προβλήματος).
- Ο χρόνος εκτέλεσης της αναζήτησης ή το όριο οπισθοδρόμησης έχει ξεπεραστεί, εφόσον τα όρια αυτά έχουν τεθεί πριν από την εκτέλεση του κάθε προβλήματος.

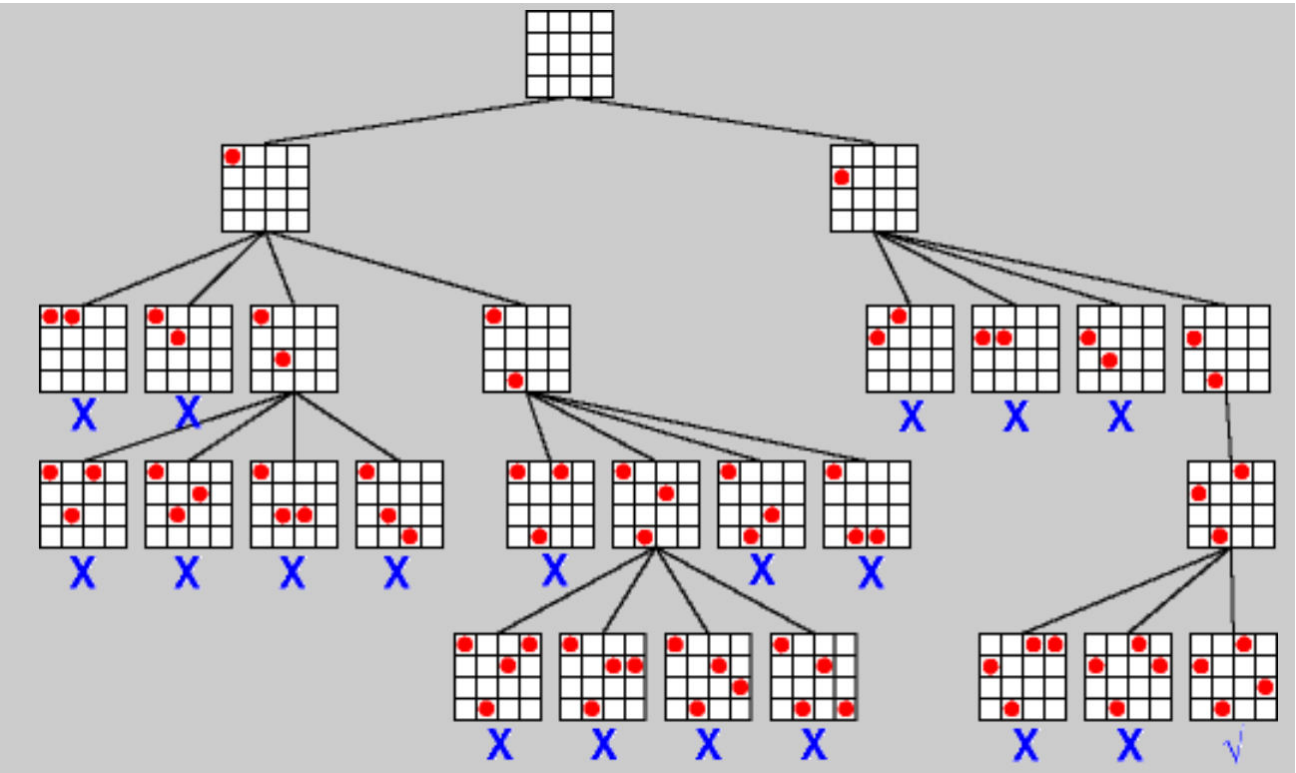
Η παραπάνω διαδικασία μπορεί να περιγραφεί σε μορφή ψευδοκώδικα, όπως εικονίζεται παρακάτω :

```
1.  function MAC (X, D, C) {
2.      call AC-3 (X, D, C) for constraint propagation
3.      while assignment is not complete {
4.          variable <- SELECT-UNASSIGMENT-VARIABLE (X, assignment)
5.          result <- 1
6.          for each value in ORDER-DOMAIN-VALUES (D (variable)) {
7.              if value is consistent with assignment according to AC-3 (variable, D (variable), C (variable)) then{
8.                  add { variable = value } to assignment
9.                  result <- 0
10.                 break
11.             }
12.         }
13.         if result == 1 and assignment is not empty then {
14.             remove { previous-variable = variable } from assignment
15.             variable <- previous-variable
16.         }
17.         else if variable is start-variable and assignment is empty then return failure
18.     }
19.     return solution
20. }
```

Ως δομή κώδικα δεν διαφέρει σημαντικά από τον αλγόριθμο Backtracking τον οποίο αναλύσαμε σε προηγούμενη ενότητα. Μία από τις διαφορές που εντοπίζουμε είναι στη γραμμή 2 του κώδικα, στην οποία γίνεται εκτέλεση του AC-3 για το σύνολο των μεταβλητών προκειμένου, πρωτίστως να πραγματοποιηθεί διάδοση περιορισμών και αφετέρου να μειωθεί ο χώρος αναζήτησης του προβλήματος. Επιπλέον, μια διαφορά

εντοπίζεται και στη γραμμή 7 του κώδικα, όπου ο AC-3 εκτελείται ξανά, για την συγκεκριμένη μεταβλητή την οποία εξετάζουμε.

Εφαρμόζοντας τον αλγόριθμο MAC για την επίλυση του προβλήματος των τεσσάρων βασιλισσών, το οποίο αναφέραμε προηγουμένως, προκύπτει το παρακάτω δέντρο αναζήτησης :



2.1 Πρόβλημα 4X4 Queens

3. Ευριστικές Συναρτήσεις για CSPs

Προκειμένου να μειωθεί ο τεράστιος για ρεαλιστικά προβλήματα χώρος αναζήτησης και ο απαιτούμενος για την εύρεση της λύσης χρόνος, μπορούν να χρησιμοποιηθούν αλγόριθμοι οι οποίοι εκμεταλλεύονται ευριστικούς μηχανισμούς, δηλαδή στρατηγικές (συνήθως συναρτήσεις που εξαρτώνται από το εκάστοτε πρόβλημα) οι οποίες αξιολογούν προσεγγιστικά τις ενδιάμεσες καταστάσεις ως προς την εκτιμώμενη απόσταση τους από μία τελική κατάσταση, επεκτείνοντας πρώτα αυτές με τη βέλτιστη ευριστική τιμή (οι οποίες αναμένεται να οδηγήσουν συντομότερα σε λύση) ή/και "κλαδεύοντας" τις υπόλοιπες καταστάσεις. Οι ευριστικοί μηχανισμοί δεν είναι πάντα εύστοχοι και παρόλο που κωδικοποιούνται αλγοριθμικά υπό τη μορφή της ευριστικής συνάρτησης, δεν μπορούν να θεωρηθούν αλγόριθμοι. Αυτό οφείλεται στο γεγονός ότι προκειμένου να μειώσουν το χώρο αναζήτησης ή να επιταχύνουν την εύρεση της λύσης, λειτουργούν προσεγγιστικά και «διαισθητικά» (περίπου όπως οι άνθρωποι), ενώ οι αλγόριθμοι είναι ακριβείς και λειτουργούν πάντα ορθά. Στην πλειονότητα των περιπτώσεων πάντως οι ευριστικές στρατηγικές οδηγούν σε πολύ καλά αποτελέσματα (αναλόγως βέβαια του προβλήματος), ωστόσο απέχουν πολύ από το να προσομοιώνουν τους μηχανισμούς της ανθρώπινης σκέψης.

3.1 Ευριστικοί Μηχανισμοί

Ένας γενικός ορισμός για τον όρο "ευριστικός μηχανισμός" είναι ο ακόλουθος:

Ευριστικός μηχανισμός (heuristic) είναι μία στρατηγική, βασισμένη στη γνώση για το συγκεκριμένο πρόβλημα, που χρησιμοποιείται σαν βοήθημα για τη γρηγορότερη επίλυσή του.

Αναλυτικότερα, ένας ευριστικός μηχανισμός υλοποιείται, όπως αναφέρθηκε πιο πάνω υπό τη μορφή ευριστικής συνάρτησης (heuristic function). Η ευριστική συνάρτηση λοιπόν, παράγει μία τιμή, τη λεγόμενη ευριστική τιμή (heuristic value), για μία συγκεκριμένη κατάσταση του προβλήματος, η οποία τιμή εκφράζει κατά πόσο κοντά βρίσκεται η κατάσταση αυτή σε μία τελική κατάσταση. Εδώ πρέπει να τονίσουμε ότι, η ευριστική τιμή δεν είναι η πραγματική τιμή της απόστασης από μία τερματική κατάσταση, αλλά μία εκτίμηση η οποία, πολλές φορές μπορεί να είναι και λανθασμένη.

Ένας αλγόριθμος αναζήτησης για την επίλυση προβλημάτων ικανοποίησης περιορισμών, απαιτεί τη σειρά με την οποία οι μεταβλητές του προβλήματος πρέπει να επιλεχθούν για ανάθεση τιμής, καθώς επίσης και τη σειρά των τιμών που αποδίδονται στις μεταβλητές από τα πεδία τιμών τους. Επιλέγοντας τη σωστή σειρά με την οποία θα γίνει επιλογή της επόμενης μεταβλητής (ή τιμής), μπορεί να βελτιώσει αισθητά την απόδοση επίλυσης προβλημάτων ικανοποίησης περιορισμών.

Από τα παραπάνω εντοπίζουμε δύο κατηγορίες ευριστικών μηχανισμών :

- Ευριστικοί μηχανισμοί διάταξης μεταβλητών (**Variable Ordering Heuristics – VOH**)
- Ευριτικοί μηχανισμοί διάταξης τιμών (**Value Ordering Heuristics**)

Στη παρούσα διπλωματική γίνεται σύγκριση ευριστικών μηχανισμών διάταξης μεταβλητών, οπότε και παρακάτω αναλύεται κυρίως, η συγκεκριμένη κατηγορία. Ωστόσο, κάποια στοιχεία αναφέρονται και στη δεύτερη κατηγορία ευριστικών μηχανισμών.

3.1.1 Ευριστικοί μηχανισμοί διάταξης μεταβλητών (Variable Ordering Heuristics – VOH)

Τα πειράματα και οι αναλύσεις πολλών ερευνητών έχουν δείξει πως η σειρά με την οποία επιλέγεται μία μεταβλητή για ανάθεση τιμής μπορεί να έχει ουσιαστική επίδραση στην πολυπλοκότητα ενός αλγορίθμου αναζήτησης με οπισθοδρόμηση (backtrack search). Η διάταξη (σειρά επιλογής) μπορεί να είναι είτε ***στατική***, είτε ***δυναμική*** :

- Στη ***στατική διάταξη*** *(static ordering)*, η σειρά με την οποία επιλέγεται η (επόμενη) μεταβλητή, έχει ήδη προκαθοριστεί πριν την εκκίνηση εκτέλεσης του αλγορίθμου αναζήτησης και δεν μεταβάλλεται καθ' όλη την διάρκεια εκτέλεσης του.
- Στη ***δυναμική διάταξη*** *(dynamic ordering)*, η σειρά με την οποία επιλέγεται η (επόμενη) μεταβλητή, δεν είναι προκαθορισμένη και αυτό σημαίνει ότι μπορεί να αλλάξει σε οποιοδήποτε στιγμή εκτέλεσής του αλγορίθμου, ανάλογα με την τρέχουσα κατάσταση που βρίσκεται η αναζήτηση.

Θα πρέπει να διευκρινίσουμε ότι η δυναμική διάταξη δεν είναι εφικτή για όλους τους αλγόριθμους αναζήτησης, ανάλογα με τον αλγόριθμο μπορεί να είναι διαθέσιμες περισσότερες ή λιγότερες πληροφορίες.

Επίσης, πρέπει να αναφερθεί ότι από τους διάφορους ευριστικούς μηχανισμούς που έχουν αναπτυχθεί, οι πιο κοινοί είναι αυτοί οι οποίοι βασίζονται στην αρχή, την οποία διατύπωσαν οι Haralick και Elliott, του **"να αποτύχεις πρώτα"** (**"fail-first"**) [9], η οποία μπορεί να εξηγηθεί ως : "*Για να πετύχεις, δοκίμασε πρώτα εκεί που είναι πιο πιθανόν να αποτύχεις*".

Όσον αφορά τους ευριστικούς μηχανισμούς διάταξης τιμής, προκύπτει πως μια διαφορετική διάταξη τιμών, θα προκαλούσε αναδιάταξη στα κλαδιά που προκύπτουν από κάθε κόμβο του δέντρου αναζήτησης. Αυτό είναι ένα πλεονέκτημα, καθώς εξασφαλίζει ότι ένας κλάδος που οδηγεί σε μια λύση αναζητείται νωρίτερα από τους

κλάδους που οδηγούν σε αδιέξοδο, υπό την προϋπόθεση ότι απαιτείται μόνο μία λύση. Εάν απαιτούνται όλες οι λύσεις ή εάν πρέπει να γίνει διάσχιση (αναζήτηση) σε ολόκληρο το δέντρο επειδή δεν υπάρχουν λύσεις, τότε η σειρά με την οποία διερευνώνται τα κλαδιά είναι αδιάφορη.

Είναι σύνηθες σε προβλήματα ικανοποίησης περιορισμών, να δίνεται μεγάλη έμφαση στην χρήση μηχανισμών διάταξης μεταβλητών (**Variable Ordering Heuristics**) και όχι στη χρήση μηχανισμών διάταξης τιμών (**Value Ordering Heuristics**). Για την ανάθεση τιμών, προτιμάται συνήθως η λεξικογραφική διάταξη (**lexicographic order**).

Θα πρέπει να διευκρινιστεί πως οι ευριστικοί μηχανισμοί δεν αποτελούν αλγορίθμους από μόνοι τους αλλά πρέπει να υλοποιούνται σε συνδυασμό με κάποιον αλγόριθμο – επιλυτή (**solver**). Ένας τέτοιος αλγόριθμος είναι ο αλγόριθμος **Διατήρησης Συνέπειας Τόξου** (ή **MAC**). Ο συγκεκριμένος λοιπόν, αλγόριθμος αποτελείται στην ουσία από τρία μέρη: Τον αλγόριθμο οπισθοδρόμησης (**Backtracking Search**), έναν αλγόριθμο συνέπειας τόξου (**AC-3**) και έναν ευριστικό μηχανισμό. Οπότε η συνάρτηση η οποία υλοποιεί τον ευριστικό μηχανισμό θα πρέπει να καλείται στο σημείο εκείνο όπου γίνεται επιλογή της επόμενης μεταβλητής. Σε επίπεδο κώδικα η ευριστική συνάρτηση ταυτίζεται με τη συνάρτηση SELECT-UNASSIGNED-VARIABLE (γραμμή 4 στον κώδικα MAC).

Στις επόμενες ενότητες περιγράφονται και αναλύονται οι ευριστικοί μηχανισμοί τους οποίους συγκρίνουμε (είτε ως απλές, είτε ως συνδυαστικές υλοποιήσεις) βάση πειραμάτων που παρουσιάζονται στην ενότητα 4. Θα πρέπει επίσης, να τονιστεί πως οι παρακάτω υλοποιήσεις περιγράφουν τη διαδικασία επιλογής της πρώτης καλύτερης μεταβλητής, βάση των κριτηρίων του κάθε heuristic. Παρόλα αυτά, δίνεται η δυνατότητα στον χρήστη να απαιτήσει από τον μηχανισμό να βρίσκει τις πρώτες k καλύτερες μεταβλητές, όπου το k είναι μία παράμετρος την οποία μπορεί να ορίσει ο χρήστης, με αποτέλεσμα η επόμενη μεταβλητή να επιλέγεται μέσα από αυτό το σύνολο που δημιουργείται με τυχαίο τρόπο.

3.2 Ευριστική Συνάρτηση Domain

Στα προβλήματα ικανοποίησης περιορισμών γνωρίζουμε ότι κάθε μεταβλητή περιλαμβάνει ένα πεδίο τιμών. Το κάθε πεδίο τιμών (domain), εκφράζει ένα μέγεθος, το οποίο προκύπτει στην ουσία ως το πλήθος των τιμών που περιλαμβάνει. Ένας λοιπόν, εύκολος και απλός ευριστικός μηχανισμός για διάταξη μεταβλητών θεωρείται το domain (ή dom) [9]. Σκοπός του συγκεκριμένου heuristic είναι να βρίσκει κάθε φορά τη μεταβλητή εκείνη με το μικρότερο σε μέγεθος πεδίο τιμών.

Επιπλέον πρέπει, να αναφερθεί πως στη παρούσα διπλωματική, ο παραπάνω μηχανισμός (domain) δεν χρησιμοποιήθηκε μόνος του για την διεξαγωγή των

πειραμάτων, αλλά σε συνδυασμό με τα επόμενα heuristics τα οποία θα αναλυθούν στις επόμενες ενότητες.

3.3 Ευριστική Συνάρτηση DOM/DDEG

Η ευριστική συνάρτηση **DDEG** λειτουργεί κρατώντας για κάθε μεταβλητή ενός προβλήματος έναν μετρητή που δείχνει το πλήθος των περιορισμών στους οποίους συμμετέχει η κάθε μεταβλητή με μια άλλη ελεύθερη μεταβλητή, δηλαδή με μια μεταβλητή η οποία δεν έχει λάβει ακόμη τιμή. Συγκεκριμένη η παραπάνω τεχνική συνοψίζεται βάση του παρακάτω τύπου ως εξής:

$$ddeg(x)=\sum_{c\in C} c(x) \quad \alpha\nu \quad x,y\in c(x) \quad \kappa\alpha\iota\ y\in Free\ \tau\acute{o}\tau\varepsilon\ c(x)=1\ \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\acute{\alpha}\ c(x)=0 \qquad (3.1)$$

Όπου :

- **ddeg(x)** : Ο μετρητής της μεταβλητής **x**.
- **y** : Μια μεταβλητή που ανήκει στο σύνολο των ελεύθερων μεταβλητών του προβλήματος και συμμετέχει στον περιορισμό **c(x)** μαζί με τη μεταβλητή x.
- **Free** : Το σύνολο των ελεύθερων μεταβλητών του προβλήματος.

Με άλλα λόγια ο παραπάνω τύπος είναι το άθροισμα όλων των περιορισμών εκείνων που συμμετέχει η μεταβλητή **x** με μία άλλη ελεύθερη μεταβλητή **y**.

Από τα παραπάνω καταλαβαίνει κανείς ότι πρόκειται για ευριστική συνάρτηση διάταξης μεταβλητών καθώς η επιλογή της επόμενης μεταβλητής δεν εξαρτάται από επιλογή κάποιας τιμής (για ανάθεση).

Στην παρούσα διπλωματική, υλοποιούμε το συνδυαστικό heuristic dom/ddeg το οποίο βασίζεται στη λειτουργία του ddeg. Ο τρόπος λειτουργίας του συγκεκριμένου heuristic (ευριστικού μηχανισμού) παρουσιάζεται ως εξής :

- Κάθε φορά επιλέγεται η μεταβλητή με το μικρότερο λόγο:

$$\left|\frac{D(x)}{ddeg(x)}\right| \qquad (3.2)$$

όπου **D(x)** το μέγεθος του πεδίου τιμών της μεταβλητής **x**.

- Όλα τα ddeg των μεταβλητών αρχικοποιούνται σε μηδέν.
- Έπειτα, το ddeg κάθε μεταβλητής υπολογίζεται, λαμβάνοντας τιμή βάση του συνόλου των περιορισμών στους οποίους συμμετέχει η μεταβλητή με άλλες

ελεύθερες μεταβλητές. Η διαδικασία αυτή επαναλαμβάνεται, πριν από την επιλογή κάθε μεταβλητής.

Η υλοποίηση σε ψεύδοκώδικα του παραπάνω heuristic και συγκεκριμένα του dom/ddeg δίνεται παρακάτω :

```
1.  function DOM/DDEG ( X, assigment ) {
2.          if there is not any free variable in X {
3.             best_var ← X [first variable which is not in assigment]
4.             for each var in X where var not in assigment {
5.                 if dom (var)/ddeg (var) < dom (best_var)/ddeg (best_var) then {
6.                    best_var ← var
7.                 }
8.             }
9.            return best_var
10.         }
11.         return 0
12. }
```

Επεξήγηση παραμέτρων/μεταβλητών

X : Το σύνολο (λίστα) των μεταβλητών του προβλήματος.

assigment : Το σύνολο των μεταβλητών που ανήκουν στο μονοπάτι της λύσης (δηλαδή έχουν λάβει κάποια τιμή).

best_var : Μεταβλητή όπου αποθηκεύεται η καλύτερη επιλογή μεταβλητής βάση του heuristic.

var : Βοηθητική μεταβλητή για την εύρεση της καλύτερης μεταβλητής.

Επεξήγηση ψευδοκώδικα

Στη **γραμμή 2** του παραπάνω ψευδοκώδικα γίνεται έλεγχος για το αν υπάρχει κάποια ελεύθερη μεταβλητή στο σύνολο **X**. Αν δεν υπάρχει κάποια ελεύθερη μεταβλητή τότε η συνάρτηση επιστρέφει **0 (γραμμή 11)** διαφορετικά, αρχικοποιούμε **(γραμμή 3)** τη μεταβλητή **best_var** με τη πρώτη ελεύθερη μεταβλητή από το σύνολο **X**. Έπειτα, στη **γραμμή 4** ξεκινά μια δομή επανάληψης η οποία για κάθε ελεύθερη μεταβλητή **var** του **X** ελέγχει στη **γραμμή 5** αν ο λόγος **dom(var)/ddeg(var)** είναι μικρότερος από τον αντίστοιχο της **best_var** τότε η μεταβλητή var ανατίθεται στη best_var **(γραμμή 6)**. Μόλις, ελεγχθούν όλες οι ελεύθερες μεταβλητές τότε **(γραμμή 9)** επιστρέφει τη καλύτερη μεταβλητή που βρέθηκε από τη παραπάνω διαδικασία αναζήτησης.

Χρήση παραμέτρου k

Προηγουμένως, παρουσιάστηκε ο ψευδοκώδικας της ευριστικής συνάρτησης. Σκοπός αυτής αποτελεί η εύρεση της μεταβλητής με το μικρότερο λόγο dom/ddeg. Στη πειραματική διαδικασία που ακολουθήθηκε χρησιμοποιήσαμε μια παράμετρος k η οποία ανάλογα με τη τιμή που της δίνει ο χρήστης αυτή ορίζει πόσες k καλύτερες μεταβλητές θέλει να εντοπίσει η συνάρτηση. Έπειτα, η επιλογή της μεταβλητής του προβλήματος γίνεται με τυχαίο τρόπο μέσα από το σύνολο των k καλύτερων μεταβλητών.

Επομένως, μια διαφορετική έκδοση του προηγούμενου ψευδοκώδικα κάνοντας πλέον χρήση και της παραμέτρου k είναι η παρακάτω :

```
1. function DOM/DDEG_K ( X, assigment, k ) {
2.         if there is not any free variable in X {
3.                 Free_X ← {each var which is in X and not in assigment}
4.                 sort_min_to_max (Free_X)
5.                 best_var ← choose random one var from Free_X[0:k]
6.                 return best_var
7.         }
8.         return 0
9. }
```

Παρατηρούμε λοιπόν πως για να πάρουμε εύκολα περισσότερες από μία μεταβλητές θα πρέπει να χρησιμοποιήσουμε έναν οποιοδήποτε αλγόριθμο ταξινόμησης τύπου **sort_min_to_max (γραμμή 4)** και να τον εφαρμόσουμε σε ένα σύνολο **Free_X** το οποίο έχουμε ήδη αποθηκεύσει **(γραμμή 3)** μόνο τις ελεύθερες μεταβλητές. Να τονίσουμε τέλος που η ταξινόμηση γίνεται με κριτήριο το λόγο **dom/ddeg** της κάθε μεταβλητής και μάλιστα από τον μικρότερο στο μεγαλύτερο λόγο.

Υπενθυμίζουμε επίσης, στην ενότητα 2 δόθηκε ο παρακάτω ψευδοκώδικας του αλγορίθμου MAC με τα αντίστοιχα σχόλια :

```
1.  function MAC (X, D, C) {
2.      call AC-3 (X, D, C) for constraint propagation
3.      while assignment is not complete {
4.          variable <- SELECT-UNASSIGMENT-VARIABLE (X, assignment)
5.          result <- 1
6.          for each value in ORDER-DOMAIN-VALUES (D (variable)) {
7.              if value is consistent with assignment according to AC-3 (variable, D (variable), C (variable)) then{
8.              add { variable = value } to assignment
9.              result <- 0
10.             break
11.             }
12.         }
13.         if result == 1 and assignment is not empty then {
14.             remove { previous-variable = variable } from assignment
15.             variable <- previous-variable
```

```
16.                 }
17.                 else if variable is start-variable and assignment is empty then return failure
18.         }
19.         return solution
20.   }
```

Όπως παρατηρούμε παραπάνω στη γραμμή 4 καλείται η συνάρτηση SELECT-UNASSIGMENT-VARIABLE (X, assigment) σε αυτή λοιπόν τη θέση καλούμε την ευριστική συνάρτηση που αναλύσαμε προηγουμένως , δηλαδή τη DOM/DDEG (X, assigment) ή αν θέλουμε τη χρήση της παραμέτρου k τότε καλούμε τη DOM/DDEG_K (X, assigment, k).

3.4 Ευριστική Συνάρτηση DOM/WDEG

Η ευριστική συνάρτηση **WDEG** όπως αναφέρεται και στο [14] διατηρεί για κάθε περιορισμό έναν μετρητή (βάρος) ο οποίος αναπαριστά πόσες φορές ο περιορισμός οδήγησε σε αποτυχία, δηλαδή την αφαίρεση της τελευταίας τιμής από το πεδίο τιμών μιας μεταβλητής κατά τη διαδικασία της διάδοσης των περιορισμών. Ο σταθμισμένος βαθμός (**weighted degree**) κάθε μεταβλητής ορίζεται ως :

$$weight(x) = \sum_{c \in C} weight(c) \qquad (3.3)$$

Όπου :

- **weight(x) :** Ο σταθμισμένος βαθμός της μεταβλητής $x \in Free\_Vars$ (σύνολο ελεύθερων μεταβλητών).
- **weight(c) :** Το βάρος του περιορισμού $c \in C$ (όπου **C** είναι το σύνολο των περιορισμών εκείνων στους οποίους συμμετέχει η μεταβλητή x με μεταβλητές που δεν έχουν λάβει κάποια τιμή).

Με άλλα λόγια ο σταθμισμένος βαθμός κάθε μεταβλητής ορίζεται ως το άθροισμα των επιμέρους σταθμισμένων βαθμών των περιορισμών εκείνων στους οποίους συμμετέχει η ίδια η μεταβλητή με άλλες ελεύθερες μεταβλητές.

Στην παρούσα διπλωματική, χρησιμοποιούμε το heuristic dom/wdeg. Ο τρόπος που λειτουργεί το συγκεκριμένο heuristic είναι ο εξής :

• Κάθε φορά επιλέγεται η μεταβλητή που έχει το μικρότερο λόγο:

$$\frac{|D(x)|}{weight(x)} \qquad (3.4)$$

όπου D(x) το μέγεθος του πεδίου τιμών της μεταβλητής x.

• Όλα τα βάρη των περιορισμών αρχικοποιούνται σε 1 και κάθε φορά που ένα περιορισμός αποτυγχάνει (οδηγεί σε διαγραφή της τελευταίας τιμής του πεδίου τιμών μιας μεταβλητής), τότε το βάρος του συγκεκριμένου περιορισμού αυξάνει κατά μία μονάδα.

Σε ψευδοκώδικα το παραπάνω heuristic υλοποιήθηκε ως εξής :

```
1.  function DOM/WDEG ( X, assigment ) {
2.        if there is not any free variable in X {
3.          best_var ← X[first variable which is not in assigment]
4.          for each var in X where var not in assigment {
5.                  if dom (var)/wdeg (var) < dom (best_var)/wdeg (best_var) then {
6.                          best_var ← var
7.                  }
8.          }
9.          return best_var
10.       }
11.       return 0
12. }
```

Επεξήγηση παραμέτρων/μεταβλητών

X : Το σύνολο (λίστα) των μεταβλητών του προβλήματος.

assigment : Το σύνολο των μεταβλητών που ανήκουν στο μονοπάτι της λύσης (δηλαδή έχουν λάβει κάποια τιμή).

best_var : Μεταβλητή όπου αποθηκεύεται η καλύτερη επιλογή μεταβλητής βάση του heuristic.

var : Βοηθητική μεταβλητή για την εύρεση της καλύτερης μεταβλητής.

Επεξήγηση ψευδοκώδικα

Στη **γραμμή 2** του παραπάνω ψευδοκώδικα γίνεται έλεγχος για το αν υπάρχει κάποια ελεύθερη μεταβλητή στο σύνολο **X**. Αν δεν υπάρχει κάποια ελεύθερη μεταβλητή τότε η συνάρτηση επιστρέφει **0 (γραμμή 11)** διαφορετικά, αρχικοποιούμε **(γραμμή 3)** τη μεταβλητή **best_var** με τη πρώτη ελεύθερη μεταβλητή από το σύνολο **X**. Έπειτα, στη **γραμμή 4** ξεκινά μια δομή επανάληψης η οποία για κάθε ελεύθερη μεταβλητή **var** του **X** ελέγχει στη **γραμμή 5** αν ο λόγος **dom(var)/wdeg(var)** είναι μικρότερος από τον αντίστοιχο της **best_var** τότε η μεταβλητή var ανατίθεται στη best_var **(γραμμή 6)**. Μόλις, ελεγχθούν όλες οι ελεύθερες μεταβλητές τότε **(γραμμή 9)** επιστρέφει τη καλύτερη μεταβλητή που βρέθηκε από τη παραπάνω διαδικασία αναζήτησης.

Χρήση παραμέτρου k

Προηγουμένως, παρουσιάστηκε ο ψευδοκώδικας της ευριστικής συνάρτησης. Σκοπός αυτής αποτελεί η εύρεση της μεταβλητής με το μικρότερο λόγο dom/wdeg. Στη πειραματική διαδικασία που ακολουθήθηκε χρησιμοποιήσαμε μια παράμετρο k η οποία ανάλογα με τη τιμή που της δίνει ο χρήστης αυτή ορίζει πόσες k καλύτερες μεταβλητές θέλει να εντοπίσει η συνάρτηση. Έπειτα, η επιλογή της μεταβλητής του προβλήματος γίνεται με τυχαίο τρόπο μέσα από το σύνολο των k καλύτερων μεταβλητών.

Επομένως, μια διαφορετική έκδοση του προηγούμενου ψευδοκώδικα κάνοντας πλέον χρήση και της παραμέτρου k είναι η παρακάτω :

```
1. function DOM/WDEG_K ( X, assigment, k ) {
2.         if there is not any free variable in X {
3.                 Free_X ← {each var which is in X and not in assigment}
4.                 sort_min_to_max (Free_X)
5.                 best_var ← choose random one var from Free_X[0:k]
6.                 return best_var
7.         }
8.         return 0
9. }
```

Παρατηρούμε λοιπόν πως για να πάρουμε εύκολα περισσότερες από μία μεταβλητές θα πρέπει να χρησιμοποιήσουμε έναν οποιοδήποτε αλγόριθμο ταξινόμησης τύπου **sort_min_to_max (γραμμή 4)** και να τον εφαρμόσουμε σε ένα σύνολο **Free_X** το οποίο έχουμε ήδη αποθηκεύσει **(γραμμή 3)** μόνο τις ελεύθερες μεταβλητές. Να τονίσουμε τέλος που η ταξινόμηση γίνεται με κριτήριο το λόγο **dom/wdeg** της κάθε μεταβλητής και μάλιστα από τον μικρότερο στο μεγαλύτερο λόγο.

Υπενθυμίζουμε επίσης, στην ενότητα 2 δόθηκε ο παρακάτω ψευδοκώδικας του αλγορίθμου MAC με τα αντίστοιχα σχόλια :

```
1.  function MAC (X, D, C) {
2.      call AC-3 (X, D, C) for constraint propagation
3.      while assignment is not complete {
4.          variable <- SELECT-UNASSIGMENT-VARIABLE (X, assignment)
5.          result <- 1
6.          for each value in ORDER-DOMAIN-VALUES (D (variable)) {
7.              if value is consistent with assignment according to AC-3 (variable, D (variable), C (variable)) then{
8.                  add { variable = value } to assignment
9.                  result <- 0
10.                 break
11.             }
12.         }
13.         if result == 1 and assignment is not empty then {
14.           remove { previous-variable = variable } from assignment
15.           variable <- previous-variable
16.         }
17.       else if variable is start-variable and assignment is empty then return failure
18.     }
19.     return solution
20. }
```

Όπως παρατηρούμε παραπάνω στη γραμμή 4 καλείται η συνάρτηση SELECT-UNASSIGMENT-VARIABLE (X, assigment) σε αυτή λοιπόν τη θέση καλούμε την ευριστική συνάρτηση που αναλύσαμε προηγουμένως, δηλαδή τη DOM/WDEG (X, assigment) ή αν θέλουμε τη χρήση της παραμέτρου k τότε καλούμε τη DOM/WDEG_K (X, assigment, k).

3.5 Ευριστική Συνάρτηση ACTIVITY BASED SEARCH (ABS)

Το ABS heuristic, σύμφωνα με το [14] αποτελεί έναν ευριστικό μηχανισμό ο οποίος σχετίζεται άμεσα με την διάδοση των περιορισμών. Ο κύριος σκοπός του μηχανισμού αυτού, είναι να υπολογίζει (προβλέπει) κατά πόσο μια μεταβλητή από το σύνολο των μεταβλητών του προβλήματος, επηρεάζεται από την διάδοση των περιορισμών, δηλαδή κατά πόσο το πεδίο τιμών της μειώνεται στη φάση της αναζήτησης. Αποτελεί έναν εύκολο σε εφαρμογή ευριστικό μηχανισμό ο οποίος δεν απαιτεί κάποιες ιδιαίτερες τεχνικές πάνω στη διάδοση περιορισμών, καθώς επίσης έχει την δυνατότητα να εφαρμοστεί και σε προβλήματα, των οποίων τα πεδία τιμών των μεταβλητών είναι τεράστια ως προς το μέγεθος.

3.5.1 Βασική ιδέα λειτουργίας.

Ο τρόπος λειτουργίας του βασίζεται στην ιδέα πως για κάθε μεταβλητή κρατάμε έναν μετρητή, τον οποίο ονομάζουμε activity και μετρά τη δραστηριότητα της κάθε μεταβλητής κατά τη διαδικασία της διάδοσης των περιορισμών. Στην ουσία, η δραστηριότητα αυτή αναφέρεται στο πόσο συχνά (αλλά όχι σε τι βαθμό) επηρεάζεται

μια μεταβλητή (δηλαδή αν γίνονται διαγραφές στο πεδίο τιμών της) από τη διάδοση των περιορισμών.

Βέβαια, πρέπει να επισημανθεί πως η συγκεκριμένη μετρική (το activity) ενημερώνεται συστηματικά κατά τη διαδικασία της αναζήτησης. Επιπλέον, περιλαμβάνεται ένα στάδιο αρχικοποίησης κάθε activity μεταβλητής το οποίο ονομάζεται probing και είναι στην ουσία ένα πλήθος τυχαίων δοκιμών επιλογής μεταβλητής και ανάθεσης τιμής, το οποίο θα αναλυθεί εκτενέστερα παρακάτω.

3.5.2 Περιγραφή υπολογισμού του activity.

Δεδομένου ενός δοθέντος προβλήματος ικανοποίησης περιορισμών (CSP) με ένα σύνολο μεταβλητών X, ένα σύνολο περιορισμών C και ένα σύνολο πεδίων τιμών D εφαρμόζεται έπειτα από κάθε ανάθεση τιμής ένας αλγόριθμος F (στην παρόν υλοποίηση ο AC-3) ο οποίος παράγει ένα νέο σύνολο X΄ που περιλαμβάνει τις μεταβλητές εκείνες που με τη διάδοση περιορισμών επηρεάστηκαν (υπήρξε μείωση στα πεδία τιμών τους).

Στη συνέχεια, το activity (με αρχική τιμή 0) της κάθε μεταβλητής ασχέτως αποτελέσματος του αλγορίθμου F (αποτυχία ή επιτυχία) διαμορφώνεται βάση των παρακάτω κανόνων:

$$\begin{aligned} &\forall x \in X \quad \mu\varepsilon \ |D(x)| > 1 : A(x) = A(x) \cdot \gamma \\ &\forall x \in X' \qquad\qquad\quad : A(x) = A(x) + 1 \end{aligned} \tag{3.5}$$

Όπου :

- A(x) : Συμβολίζεται το activity της μεταβλητής x.
- γ : Το γ αποτελεί μία παράμετρο αποσύνθεσης (decay parameter) με εύρος τιμών.
  - Η συγκεκριμένη παράμετρος επηρεάζει μόνο τις ελεύθερες μεταβλητές διαφορετικά θα προκαλούσε διαγραφή του activity όλων των μεταβλητών που έχουν ήδη λάβει τιμή κατά τη διάρκεια της αναζήτησης.
- |D(x)| : Το μέγεθος του πεδίου ορισμού της μεταβλητής x.

3.5.3 Περιγραφή διαδικασίας αναζήτησης.

Κατά τη διαδικασία της αναζήτησης η μεταβλητή που επιλέγεται για ανάθεση τιμής βρίσκεται συγκρίνοντας το λόγο A(x) / |D(x)| για κάθε μεταβλητή και τελικά επιλέγεται αυτή με το μεγαλύτερο λόγο αφού θεωρείται και η πιο δραστήρια.

3.5.4 Περιγραφή διαδικασίας αρχικοποίησης των activities.

Η αναζήτηση βασισμένη στο activity εφαρμόζει το λεγόμενο probing, το οποία είναι στην ουσία μια σειρά από τυχαία μονοπάτια αναζήτησης με τυχαία επιλογή επόμενης μεταβλητής και ανάθεσης τιμής, με σκοπό να αρχικοποιήσει τα activities των μεταβλητών και ως συνέπεια να δημιουργήσει έναν οδηγό για την διαδικασία της αναζήτησης στη συνέχεια.

Ο τρόπος λειτουργίας του probing υλοποιείται ακολουθώντας τους παρακάτω εξής κανόνες :

- Για κάθε probe (δοκιμή) ενός συνόλου probes ξεκινάμε επιλέγοντας τυχαία μια μεταβλητή x από το σύνολο των ελεύθερων μεταβλητών X του προβλήματος και αναθέτουμε σε αυτή μία τιμή από το πεδίο τιμών της τυχαία.
- Έπειτα, εφαρμόζουμε έναν αλγόριθμο διάδοσης περιορισμών (στη περίπτωσή μας τον AC-3), οποίος ανάλογα με το αποτέλεσμα που θα επιστρέψει (αποτυχία ή επιτυχία) συμβαίνουν τα εξής δύο σενάρια :
    - Αν η διάδοση των περιορισμών ήταν επιτυχής (δηλαδή δεν έγινε διαγραφή όλων των τιμών κάποιου πεδίου ορισμού) συνεχίζουμε επιλέγοντας την επόμενη μεταβλητή με τυχαία επιλογή και ανάθεση τιμής μέχρι να προκύψει κάποια αποτυχία ή να οδηγήσει ακόμη και σε λύση του προβλήματος. Οπότε προκύπτει πως κάθε μεταβλητή που πήρε τιμή και πέρασε με επιτυχία τη διάδοση των περιορισμών προστίθεται στο μονοπάτι αναζήτησης π.
    - Αν η διάδοση των περιορισμών καταλήξει σε αποτυχία τότε όλες οι αλλαγές που έχουν προκύψει μέχρι εκείνο το σημείο αναιρούνται και αν υπάρχει κι άλλο probe τότε ξεκινάει και πάλι η αναζήτηση εύρεσης νέου μονοπατιού που να οδηγεί στη λύση του προβλήματος. Επιπλέον, στη περίπτωση που παρατηρηθεί στη ρίζα του δέντρου αναζήτησης η τιμή που ανατέθηκε στη μεταβλητή να οδηγεί σε αποτυχία τότε αυτή διαγράφεται μόνιμα από το πεδίο τιμών της.

Όσον αφορά τα activities αυτά τροποποιούνται ως εξής :

$A_0^{\pi}(x)=0 \Leftrightarrow$ η μεταβλητή x δεν συμμετείχε στη διαδικασία της διάδοσης περιορισμών.
$A_j^{\pi}(x)=A_{j-1}^{\pi}(x)+1 \Leftrightarrow$ η μεταβλητή x συμμετείχε στη $j-$οστή διαδικασία διάδοσης περιορισμών.
$A_j^{\pi}(x)=A_{j-1}^{\pi}(x) \Leftrightarrow$ η μεταβλητή x συμμετείχε σε κάποια διαδικασία διάδοσης περιορισμών.

Όπου :

- π : Το μονοπάτι που προκύπτει σε κάθε probe.
- j : Ο j-οστός κόμβος στον οποία εκτελέστηκε η διάδοση περιορισμών και j-1 είναι ο αμέσως προηγούμενος κόμβος.
- A(x) : Συμβολίζουμε το activity της μεταβλητής x.

Πιο αναλυτικά, αν μια μεταβλητή x δεν συμμετείχε σε κανένα probe τότε το activity του μένει 0. Παράλληλα, σε μια δεδομένη διάδοση περιορισμών j οι μεταβλητές που επηρεάστηκαν αυξάνουν το τρέχων activity ($A_j^{\pi}(x)$) τους κατά μία μονάδα και τέλος, οι μεταβλητές που δεν επηρεάστηκαν στη j διάδοση αλλά σε κάποια προηγούμενη παραμένουν τα activities τους ως έχουν.

Στη διαδικασία του probing το γ είναι ίσο με ένα και τα activities δεν μειώνονται. Επίσης, ο αριθμός των probes δίνεται από τον χρήστη, ενώ μια καλή μέθοδος, την οποία όμως δεν εξετάζουμε στη παρούσα διπλωματική θα ήταν η επιλογή κάποιου διαστήματος εμπιστοσύνης προκειμένου να εκτελεστούν τόσα probes ώστε οι τιμές των activities να συγκλίνουν στις πραγματικές τους τιμές.

Τέλος, στην αρχή κάθε probe οι τιμές των activities δεν αρχικοποιούνται πάλι στο μηδέν αλλά συνεχίζουν βάση των τιμών που απέκτησαν από προηγούμενα probes. Να τονιστεί επίσης, πως κατά τη διαδικασία του probing μπορεί να βρεθεί ακόμη και λύση για το πρόβλημα οπότε η αναζήτηση σταματάει επιστρέφοντας τη συγκεκριμένη λύση.

Η παραπάνω ευριστική συνάρτηση (heuristic) μπορεί να υλοποιηθεί σε ψεύδοκωδικα ως εξής :

```
1.  function ACTIVITY ( X, assigment ) {
2.      if there is not any free variable in X {
3.        best_var ← X [first variable which is not in assigment]
4.        for each var in X where var not in assigment {
5.              if activity (var)/dom (var) > activity (best_var)/dom (best_var) then{
6.                best_var ← var
7.              }
8.        }
9.        return best_var
10.     }
11.     return 0
12. }
```

Επεξήγηση παραμέτρων/μεταβλητών

X : Το σύνολο (λίστα) των μεταβλητών του προβλήματος.

assigment : Το σύνολο των μεταβλητών που ανήκουν στο μονοπάτι της λύσης (δηλαδή έχουν λάβει κάποια τιμή).

best_var : Μεταβλητή όπου αποθηκεύεται η καλύτερη επιλογή μεταβλητής βάση του heuristic.

var : Βοηθητική μεταβλητή για την εύρεση της καλύτερης μεταβλητής.

Επεξήγηση ψευδοκώδικα

Στη **γραμμή 2** του παραπάνω ψευδοκώδικα γίνεται έλεγχος για το αν υπάρχει κάποια ελεύθερη μεταβλητή στο σύνολο **X**. Αν δεν υπάρχει κάποια ελεύθερη μεταβλητή τότε η συνάρτηση επιστρέφει **0 (γραμμή 11)** διαφορετικά, αρχικοποιούμε **(γραμμή 3)** τη μεταβλητή **best_var** με τη πρώτη ελεύθερη μεταβλητή από το σύνολο **X**. Έπειτα, στη **γραμμή 4** ξεκινά μια δομή επανάληψης η οποία για κάθε ελεύθερη μεταβλητή **var** του **X** ελέγχει στη **γραμμή 5** αν ο λόγος **activity(var)/dom(var)** είναι μεγαλύτερος από τον αντίστοιχο της **best_var** τότε η μεταβλητή var ανατίθεται στη best_var **(γραμμή 6)**. Μόλις, ελεγχθούν όλες οι ελεύθερες μεταβλητές τότε **(γραμμή 9)** επιστρέφει τη καλύτερη μεταβλητή που βρέθηκε από τη παραπάνω διαδικασία αναζήτησης.

Χρήση παραμέτρου k

Προηγουμένως, παρουσιάστηκε ο ψευδοκώδικας της ευριστικής συνάρτησης. Σκοπός αυτής αποτελεί η εύρεση της μεταβλητής με το μεγαλύτερο λόγο activity/dom. Στη πειραματική διαδικασία που ακολουθήθηκε χρησιμοποιήσαμε μια παράμετρο k η οποία ανάλογα με τη τιμή που της δίνει ο χρήστης αυτή ορίζει πόσες k καλύτερες μεταβλητές θέλει να εντοπίσει η συνάρτηση. Έπειτα, η επιλογή της μεταβλητής του προβλήματος γίνεται με τυχαίο τρόπο μέσα από το σύνολο των k καλύτερων μεταβλητών.

Επομένως, μια διαφορετική έκδοση του προηγούμενου ψευδοκώδικα κάνοντας πλέον χρήση και της παραμέτρου k είναι η παρακάτω :

```
1. function ACTIVITY_K ( X, assigment, k ) {
2.         if there is not any free variable in X {
3.                 Free_X ← {each var which is in X and not in assigment}
4.                 sort_max_to_min (Free_X)
5.                 best_var ← choose random one var from Free_X [0:k]
6.                 return best_var
7.         }
8.         return 0
9. }
```

Παρατηρούμε λοιπόν πως για να πάρουμε εύκολα περισσότερες από μία μεταβλητές θα πρέπει να χρησιμοποιήσουμε έναν οποιοδήποτε αλγόριθμο ταξινόμησης τύπου **sort_max_to_min (γραμμή 4)** και να τον εφαρμόσουμε σε ένα σύνολο **Free_X** το οποίο έχουμε ήδη αποθηκεύσει **(γραμμή 3)** μόνο τις ελεύθερες μεταβλητές. Να τονίσουμε τέλος που η ταξινόμηση γίνεται με κριτήριο το λόγο **activity/dom** της κάθε μεταβλητής και μάλιστα από τον μεγαλύτερο στο μικρότερο λόγο.

Υπενθυμίζουμε επίσης, στην ενότητα 2 δόθηκε ο παρακάτω ψευδοκώδικας του αλγορίθμου MAC με τα αντίστοιχα σχόλια :

```
1.  function MAC (X, D, C) {
2.      call AC-3 (X, D, C) for constraint propagation
3.      while assignment is not complete {
4.          variable <- SELECT-UNASSIGMENT-VARIABLE (X, assignment)
5.          result <- 1
6.          for each value in ORDER-DOMAIN-VALUES (D (variable)) {
7.              if value is consistent with assignment according to AC-3 (variable, D (variable), C (variable)) then{
8.                  add { variable = value } to assignment
9.                  result <- 0
10.                 break
11.             }
12.         }
13.         if result == 1 and assignment is not empty then {
14.             remove { previous-variable = variable } from assignment
15.             variable <- previous-variable
16.         }
17.         else if variable is start-variable and assignment is empty then return failure
18.     }
19.     return solution
20. }
```

Όπως παρατηρούμε παραπάνω στη γραμμή 4 καλείται η συνάρτηση SELECT-UNASSIGMENT-VARIABLE (X, assigment) σε αυτή λοιπόν τη θέση καλούμε την ευριστική συνάρτηση που αναλύσαμε προηγουμένως, δηλαδή τη ACTIVITY (X, assigment) ή αν θέλουμε τη χρήση της παραμέτρου k τότε καλούμε τη ACTIVITY_K (X, assigment, k).

3.6 Ευριστική Συνάρτηση IMPACT BASED SEARCH (IBS)

Σε ένα πρόβλημα ικανοποίησης περιορισμών, η ανάθεση μιας τιμής σε κάποια μεταβλητή του προβλήματος εξαιτίας της διάδοσης περιορισμών, οδηγεί σε διαγραφές τιμών από τα πεδία τιμών άλλων μεταβλητών. Η διαγραφή αυτή των τιμών, οδηγεί επομένως σε κάποια μείωση του χώρου αναζήτησης.
Σύμφωνα με το [13], ως impact μιας ανάθεσης τιμής σε μια μεταβλητή, θεωρούμε την μείωση την οποία προκαλεί στο χώρο αναζήτησης του προβλήματος. Σαν μια εκτίμηση του χώρου αναζήτησης, μπορούμε να θεωρήσουμε το καρτεσιανό γινόμενο το οποίο προκύπτει από των πολλαπλασιασμό του μεγέθους του πεδίου τιμών κάθε μεταβλητής του προβλήματος. Πιο αναλυτικά:

$$P=|D_{x1}|X\ldots X|D_{xn}| \qquad (3.6)$$

3.6.1 Υπολογισμός impact ανάθεσης τιμής.

Πιο αναλυτικά, το IBS Heuristic υπολογίζει το μέγεθος του χώρου αναζήτησης πριν και μετά την ανάθεση κάποιας τιμής (x = a) σε μία μεταβλητή χρησιμοποιώντας μια συνάρτηση όπως η παρακάτω :

$$I(x=a)=1-\frac{P_{after}}{P_{before}} \qquad (3.7)$$

Όπου :

- I : Το impact που έχει η συγκεκριμένη ανάθεση (x = a).
- P_{after} : Αποτελεί το γινόμενο όλων των πεδίων τιμών των μεταβλητών του προβλήματος μετά την ανάθεση της τιμής a στη μεταβλητή x και υπολογίζεται ως εξής:

$$P_{after}=|D'_1|\,X\,|D'_2|\,X\,\ldots\,X\,|D'_k| \quad \text{όπου } D'_i \text{ το πεδίο ορισμού μεταβλητής μετά την ανάθεση, με } i=1,2,\ldots,k \text{ και } k \text{ το πλήθος των μεταβλητών του προβλήματος.}$$

- P_{before} : Αποτελεί το γινόμενο όλων των πεδίων τιμών των μεταβλητών του προβλήματος πριν την ανάθεση της τιμής a στη μεταβλητή x και υπολογίζεται ως εξής :

$$P_{before}=|D_1|\,X\,|D_2|\,X\,\ldots\,X\,|D_k| \quad \text{όπου } D_i \text{ το πεδίο ορισμού μεταβλητής πριν την ανάθεση, με } i=1,2,\ldots,k \text{ και } k \text{ το πλήθος των μεταβλητών του προβλήματος.}$$

Ένα συμπέρασμα που προκύπτει είναι ότι όσο πιο υψηλό είναι το impact τόσο πιο μεγάλη είναι η μείωση που υφίσταται ο χώρος αναζήτησης. Συνεπώς, μία ανάθεση τιμής που οδηγεί σε αποτυχία (δηλαδή, διαγραφή όλων των τιμών ενός πεδίου τιμών μιας μεταβλητής) έχει impact ίσο με 1.

Επίσης, έχει παρατηρηθεί ότι οι τιμές των impacts οι οποίες προκύπτουν από την ανάθεση μιας συγκεκριμένης τιμής σε μια συγκεκριμένη μεταβλητή κατά την διάσχιση του δέντρου αναζήτησης, συγκλίνουν γύρω από μια συγκεκριμένη τιμή. Επομένως, για λόγους ευκολίας κατά την διαδικασία υπολογισμού των impacts των μεταβλητών του προβλήματος, θεωρούμε ότι το impact, το οποίοι προκύπτει από την ανάθεση μιας συγκεκριμένης τιμής σε μια συγκεκριμένη μεταβλητή (κόμβος), ισούται με τον μέσο όρο όλων των προηγούμενων impacts, τα οποία έχουν προκύψει από την ανάθεσης της τιμής αυτής στην ίδια μεταβλητή σε προηγούμενες χρονικές στιγμές κατά την διάσχιση του δέντρου αναζήτησης.

Ο υπολογισμός του μέσου όρου των impacts για μια συγκεκριμένη ανάθεση δίνεται από τον παρακάτω τύπο :

$$\bar{I}(x_i=a)=\frac{\sum_{k\in K} I^k(x_i=a)}{|K|} \qquad (3.8)$$

Όπου:

- $\bar{I}$: Ο μέσος όρων όλων των impacts που παρατηρήθηκαν για τη συγκεκριμένη ανάθεση τιμής ($x_i = a$, όπου $x_i \in X$ και X το σύνολο τον μεταβλητών του προβλήματος) μέχρι τον κόμβο k.
- K : Το σύνολο των κόμβων στους οποίους έγινε η συγκεκριμένη ανάθεση τιμής.
- $I^k(x_i=a)$: Το impact της ανάθεσης τιμής $x_i = a$ στο κόμβο k , όπου $k \in K$.

3.6.2 Υπολογισμός impact μεταβλητής.

Αφού λοιπόν μπορούμε να υπολογίσουμε το impact μιας ανάθεσης τιμής σύμφωνα με τους προηγούμενους τύπους θα πρέπει να υπολογίσουμε και το impact μιας μεταβλητής, εφόσον μελετάμε ευριστικές συναρτήσεις διάταξης μεταβλητών. Για τον συγκεκριμένο υπολογισμό έχουν προταθεί κάποιοι τύποι όπως ο παρακάτω :

$$\tilde{I}(x_i)=\frac{\sum_{a\in D'_{x_i}} \bar{I}(x_i=a)}{D'_{x_i}} \qquad (3.9)$$

Στην ουσία λοιπόν ο παραπάνω τύπος υπολογίζει τον μέσο όρο του αθροίσματος των μέσο όρων των impacts των εναπομεινάντων τιμών του πεδίου τιμών $D'x_i$ της μεταβλητής x_i δια το σύνολό τους. Παρόλα αυτά, ο παραπάνω τύπος δεν είναι τόσο ακριβής σε σχέση με τον παρακάτω, τον οποίο και χρησιμοποιήσαμε για την διεξαγωγή των πειραμάτων μας. Συγκεκριμένα λοιπόν, το impact μιας μεταβλητής, δίνεται :

$$I(x_i)=\sum_{a\in D'_{x_i}} 1-\bar{I}(x_i=a) \qquad (3.10)$$

Όπου :

- $I(x_i)$: Συμβολίζεται το impact της μεταβλητής x_i .
- $\bar{I}(x_i=a)$: Ο μέσος όρος του impact μιας ανάθεσης τιμής $a \in D'x_i$.

3.6.3 Αρχικοποίηση impacts.

Προκειμένου να ξεκινήσει η αναζήτηση όσο πιο αποδοτικά γίνεται θα πρέπει να επιλεγεί η σωστή μεταβλητή εκείνη που θα έχει και το μεγαλύτερο impact. Για να γίνει αυτό θα πρέπει να υπολογίσουμε αρχικά για κάθε τιμή κάθε μεταβλητής το αντίστοιχο impact που έχει στη μείωση του χώρου αναζήτησης του προβλήματος. Επειδή όμως, αυτό θα απαιτούσε αρκετό χρόνο αυτό που προτείνεται είναι ο χωρισμός του πεδίου τιμών της κάθε μεταβλητής σε μικρότερα υπό-πεδία τα οποία θα έχουν για τις τιμές που περιλαμβάνει το καθένα μία αντιπροσωπευτική τιμή impact για κάθε τιμή. Αυτή η αντιπροσωπευτική τιμή αποδίδεται στις υπόλοιπες τιμές του υπό-πεδίου υπολογίζοντας, σύμφωνα με τους προηγούμενους τύπους, μόνο το impact τις πρώτης τιμής του κάθε υπό-πεδίου και έπειτα υπολογίζεται το impact της μεταβλητής σύμφωνα με τον τύπο :

$$I'(x_i) = \sum_{j \in W} 1 - I(x_i = a^j) \qquad (3.11)$$

Όπου:

- $I'(x_i)$: Το impact της μεταβλητής κατά τη φάση της αρχικοποίησης
- $I(x_i=a_j)$: Το impact της πρώτης τιμής a του υπό-πεδίου j από το σύνολο των υπό-πεδίων W.

Πιο αναλυτικά, ο υπολογισμός της πρώτης τιμής του κάθε υπό-πεδίου πραγματοποιείται ως εξής :

- Η μεταβλητή που εξετάζουμε λαμβάνει τη πρώτη τιμή του υπό-πεδίου.
- Έπειτα εκτελείται κάποιος αλγόριθμος διάδοσης περιορισμών (στη παρούσα υλοποίηση ο AC-3). Στη συνέχεια υπολογίζεται βάση του τύπου (3.7) το impact για τη συγκεκριμένη ανάθεση (τιμή) το οποίο αποδίδεται εξίσου σε όλες τις υπόλοιπες τιμές του ίδιου υπό-πεδίου.
- Η παραπάνω διαδικασία εφαρμόζεται για κάθε μεταβλητή του προβλήματος και για κάθε υπό-πεδίο που διαθέτει.
- Τέλος, αφού εξεταστούν όλες οι μεταβλητές του προβλήματος το impact της κάθε μεταβλητής υπολογίζεται βάση του τύπου (3.11) που αναφέραμε προηγουμένως.

Η παραπάνω λοιπόν στρατηγική μένει στο γεγονός ότι το impact μιας μεταβλητής x_i διαμοιράζεται ισότιμα σε όλες τις τιμές του πεδίου τιμών της. Αυτό που προκύπτει τελικά είναι, πως για την επιλογή μιας μεταβλητής μεταξύ μεταβλητών ίδιου impact θα επιλεγεί η μεταβλητή εκείνη που έχει και το μεγαλύτερο πεδίο τιμών αφού το impact που προκαλεί μείωση του χώρου αναζήτησης ενός προβλήματος είναι σαφώς μεγαλύτερο όταν η ίδια μείωση μπορεί να προκύψει από περισσότερες τιμές.

Τέλος, για το χωρισμό των πεδίων τιμών μιας μεταβλητής σε υπό-πεδία χρησιμοποιείται μια μεταβλητή s η οποία χωρίζει το αρχικό πεδίο σε 2^s υπό-πεδία. Για τα πειράματά υπολογίζεται το s αυξάνοντάς το κατά 1 με αρχική τιμή το μηδέν, μέχρι να μην ικανοποιείται η συνθήκη $|Dx_i| / 2^s >= 2^s$.

Βέβαια, δεν πρέπει να παραληφθεί πως όταν υπολογιστεί τελικά το impact μιας ανάθεσης τιμής, η οποία έχει λάβει από τη φάση της αρχικοποίησης κάποια αντιπροσωπευτική τιμή, τότε το πραγματικό πλέον impact αντικαθιστά την αντιπροσωπευτική αυτή τιμή.

Παρακάτω παρέχεται ο ψευδοκώδικας του παραπάνω heuristic :

```
1.  function DOM/WDEG ( X, assigment ) {
2.          if there is not any free variable in X {
3.                  best_var ← X [first variable which is not in assigment]
4.                  for each var in X where var not in assigment {
5.                          if var (impact, dom) > best_var (impact, dom) then {
6.                                  best_var ← var
7.                          }
8.                  }
9.                  return best_var
10.         }
11.         return 0
12. }
```

Επεξήγηση παραμέτρων/μεταβλητών

X : Το σύνολο (λίστα) των μεταβλητών του προβλήματος.

assigment : Το σύνολο των μεταβλητών που ανήκουν στο μονοπάτι της λύσης (δηλαδή έχουν λάβει κάποια τιμή).

best_var : Μεταβλητή όπου αποθηκεύεται η καλύτερη επιλογή μεταβλητής βάση του heuristic.

var : Βοηθητική μεταβλητή για την εύρεση της καλύτερης μεταβλητής.

Επεξήγηση ψευδοκώδικα

Στη **γραμμή 2** του παραπάνω ψευδοκώδικα γίνεται έλεγχος για το αν υπάρχει κάποια ελεύθερη μεταβλητή στο σύνολο **X**. Αν δεν υπάρχει κάποια ελεύθερη μεταβλητή τότε η συνάρτηση επιστρέφει **0 (γραμμή 11)** διαφορετικά, αρχικοποιούμε **(γραμμή 3)** τη μεταβλητή **best_var** με τη πρώτη ελεύθερη μεταβλητή από το σύνολο **X**. Έπειτα, στη **γραμμή 4** ξεκινά μια δομή επανάληψης η οποία για κάθε ελεύθερη μεταβλητή **var**

του **X** ελέγχει στη **γραμμή 5** αν το **impact** της var είναι μεγαλύτερο από το αντίστοιχο της **best_var** τότε η μεταβλητή var ανατίθεται στη best_var **(γραμμή 6)**. Στη περίπτωση όμως όπου το impact και των δύο είναι ίδιο τότε η ανάθεση γίνεται μόνο αν και η var έχει το μεγαλύτερο **dom (σύνολο τιμών)**. Μόλις, ελεγχθούν όλες οι ελεύθερες μεταβλητές τότε **(γραμμή 9)** επιστρέφει τη καλύτερη μεταβλητή που βρέθηκε από τη παραπάνω διαδικασία αναζήτησης.

Χρήση παραμέτρου k

Προηγουμένως, παρουσιάστηκε ο ψευδοκώδικας της ευριστικής συνάρτησης. Σκοπός αυτής αποτελεί η εύρεση της μεταβλητής με το μεγαλύτερο impact και ως δεύτερο κριτήριο το μεγαλύτερο πεδίο τιμών. Στη πειραματική διαδικασία που ακολουθήθηκε χρησιμοποιήσαμε μια παράμετρο k η οποία ανάλογα με τη τιμή που της δίνει ο χρήστης αυτή ορίζει πόσες k καλύτερες μεταβλητές θέλει να εντοπίσει η συνάρτηση. Έπειτα, η επιλογή της μεταβλητής του προβλήματος γίνεται με τυχαίο τρόπο μέσα από το σύνολο των k καλύτερων μεταβλητών.

Επομένως, μια διαφορετική έκδοση του προηγούμενου ψευδοκώδικα κάνοντας πλέον χρήση και της παραμέτρου k είναι η παρακάτω :

```
1. function IMPACT_K ( X, assigment, k ) {
2.         if there is not any free variable in X {
3.                 Free_X ← { each var which is in X and not in assignment }
4.                 sort_max_to_min (Free_X)
5.                 best_var ← choose random one var from Free_X [0:k]
6.                 return best_var
7.         }
8.         return 0
9. }
```

Παρατηρούμε λοιπόν πως για να πάρουμε εύκολα περισσότερες από μία μεταβλητές θα πρέπει να χρησιμοποιήσουμε έναν οποιοδήποτε αλγόριθμο ταξινόμησης τύπου **sort_max_to_min (γραμμή 4)** και να τον εφαρμόσουμε σε ένα σύνολο **Free_X** το οποίο έχουμε ήδη αποθηκεύσει **(γραμμή 3)** μόνο τις ελεύθερες μεταβλητές. Να τονίσουμε τέλος που η ταξινόμηση γίνεται βάση 2 κριτηρίων :

- $1^{ο}$ κριτήριο το **impact** της κάθε μεταβλητής
- $2^{ο}$ κριτήριο το **dom** (domain – σύνολο τιμών)

Στο τέλος της εκτέλεσης του αλγορίθμου ταξινόμησης θα πρέπει το σύνολο Free_X να έχει ταξινομηθεί από τη μεταβλητή με το μεγαλύτερο impact και domain στη μεταβλητή με το μικρότερο impact και domain.

Υπενθυμίζουμε επίσης, στην ενότητα 2 δόθηκε ο παρακάτω ψευδοκώδικας του αλγορίθμου MAC με τα αντίστοιχα σχόλια :

```
1.  function MAC (X, D, C) {
2.      call AC-3 (X, D, C) for constraint propagation
3.      while assignment is not complete {
4.          variable <- SELECT-UNASSIGMENT-VARIABLE (X, assignment)
5.          result <- 1
6.          for each value in ORDER-DOMAIN-VALUES (D (variable)) {
7.              if value is consistent with assignment according to AC-3 (variable, D (variable), C (variable)) then {
8.              add { variable = value } to assignment
9.              result <- 0
10.             break
11.             }
12.         }
13.         if result == 1 and assignment is not empty then {
14.           remove { previous-variable = variable } from assignment
15.           variable <- previous-variable
16.         }
17.       else if variable is start-variable and assignment is empty then return  failure
18.     }
19.     return solution
20. }
```

Όπως παρατηρούμε παραπάνω στη γραμμή 4 καλείται η συνάρτηση SELECT-UNASSIGMENT-VARIABLE (X, assigment) σε αυτή λοιπόν τη θέση καλούμε την ευριστική συνάρτηση που αναλύσαμε προηγουμένως, δηλαδή τη IMPACT (X, assigment) ή αν θέλουμε τη χρήση της παραμέτρου k τότε καλούμε τη IMPACT_K (X, assigment, k).

3.7 Συνδυασμός ευριστικών μηχανισμών – Κοινή επιλογή (Common Choice)

Μέχρι τώρα, τα heuristics που αναλύθηκαν παρουσίασαν κάποιες διαφορές στο κριτήριο με το οποίο επιλέγουν την επόμενη μεταβλητή για ανάθεση τιμής. Τα heuristics όπως τα dom/ddeg και dom/wdeg για παράδειγμα, σχετίζονται με τους περιορισμούς, ενώ τα activity και impact με τις μεταβλητές και πιο συγκεκριμένα με το κατά πόσο επηρεάζονται τα πεδία τιμών τους κατά την διάδοση των περιορισμών. Ένα ενδιαφέρον ερώτημα που θα μπορούσε να διατυπώσει κανείς είναι αν τα παραπάνω heuristics μπορούν να συγκλίνουν στην απόφασή τους για την επιλογή της επόμενης μεταβλητής, ή διαφορετικά, αν υπάρχει πιθανότητα η μεταβλητή που προτείνει ένας από τους παραπάνω μηχανισμούς να εμφανίζεται ως επιλογή και των υπολοίπων ή έστω κάποιων από αυτούς τους μηχανισμούς.

Την απάντηση στο παραπάνω ερώτημα έρχεται να δώσει το heuristic της **κοινής επιλογής** (**common choice**). Αυτό λοιπόν, το heuristic έχει ως κριτήριο την επιλογή εκείνης της μεταβλητής, η οποία εμφανίζεται στα περισσότερα heuristics. Σε περίπτωση που καμία μεταβλητή εκ των τεσσάρων ευριστικών μηχανισμών δεν εμφανίζεται πάνω από μία φορά τότε επιλέγεται η μεταβλητή προτίμησης του

dom/wdeg, καθώς το συγκεκριμένο heuristic εμφανίζεται σχετικά πιο αποδοτικό έναντι άλλων.

Για την υλοποίηση του common choice ισχύουν οι εξής κανόνες :

- Αφού εκτελεστούν τα τέσσερα (προηγούμενα) heuristics σύμφωνα με τον τρόπο που αυτά ορίστηκαν στις προηγούμενες ενότητες, επιστρέφει το καθένα από ένα σύνολο με τις πρώτες k καλύτερες προτάσεις τους, που ζήτησε ο χρήστης.
- Στη συνέχεια, ο μηχανισμός ελέγχει την κάθε μεταβλητή αν εμφανίζεται και στα τέσσερα σύνολα ή έστω σε τρία ή δύο, σε διαφορετική περίπτωση σημαίνει ότι η συγκεκριμένη μεταβλητή δεν είναι κοινή, καθώς εμφανίζεται μόνο στο σύνολο από το οποίο προήλθε.
- Εάν εμφανίζεται 4 φορές τότε θεωρείται και η επικρατέστερη για να επιλεγεί ως επόμενη μεταβλητή ενώ σε περίπτωση που υπάρχουν κι άλλες μεταβλητές με την ίδια συχνότητα εμφάνισης τότε επιλέγεται τυχαία μία από αυτές. Το ίδιο ισχύει και στην περίπτωση που δεν έχουμε συχνότητα εμφάνισης 4 αλλά 3 ή 2. Όμως, σε περίπτωση που καμία μεταβλητή δεν εμφανίζεται πάνω από μία φορά τότε η επιλογή της μεταβλητής γίνεται τυχαία.

Παρακάτω παρατίθεται ο ψεύδοκωδικας του συγκεκριμένου heuristic :

```
1.  function COMMONCHOICE_K (X, assigment, k) {
2.          if there is not any free variable in X {
3.             DOM/DDEG_vars ← DOM/DDEG_k (X, assigment, k)
4.             DOM/WDEG_vars ← DOM/WDEG_k (X, assigment, k)
5.             ACTIVITY_vars ← ACTIVITY_k (X, assigment, k)
6.             IMPACT_vars ← IMPACT (X, assigment, k)_k
7.             best_var ← { choose from (DOM/DDEG_vars, DOM/WDEG_vars, ACTIVITY_vars, IMPACT_vars)
   the variable which is most common or if there is  not any common variable then choose random }
8.              return best_var
9.           }
10.          return 0
11. }
```

Στο παραπάνω ψευδοκώδικα οι μεταβλητές **DOM/DDEG_vars, DOM/WDEG_vars, ACTIVITY_vars, IMPACT_vars** είναι πίνακες (ή λίστες). Θα πρέπει να τονίσουμε όμως, ότι σε αντίθεση με τους ψευδοκώδικες που δόθηκαν για τις συναρτήσεις **DOM/DDEG_k (X, assigment, k), DOM/WDEG_k (X, assigment, k), ACTIVITY_k (X, assigment, k)** και **IMPACT (X, assigment, k)_k** εδώ τροποποιούνται ως προς το δεδομένο που πρέπει να επιστρέψουν το οποίο δεν είναι η καλύτερη μεταβλητή αλλά ένα σύνολο από τις k καλύτερες μεταβλητές (Παραλείπεται δηλαδή η εκτέλεση της τυχαίας επιλογής που παρουσιάζεται στους προηγούμενους ψευδοκώδικες που παρουσιάσαμε).

Υπενθυμίζουμε επίσης, στην ενότητα 2 δόθηκε ο παρακάτω ψευδοκώδικας του αλγορίθμου MAC με τα αντίστοιχα σχόλια :

```
1.  function MAC (X, D, C) {
2.      call AC-3 (X, D, C) for constraint propagation
3.      while assignment is not complete {
4.          variable <- SELECT-UNASSIGMENT-VARIABLE (X, assignment)
5.          result <- 1
6.          for each value in ORDER-DOMAIN-VALUES (D (variable)) {
7.              if value is consistent with assignment according to AC-3 (variable, D  (variable), C (variable)) then {
8.                  add { variable = value } to assignment
9.                  result <- 0
10.                 break
11.             }
12.         }
13.         if result == 1 and assignment is not empty then {
14.           remove { previous-variable = variable } from assignment
15.           variable <- previous-variable
16.         }
17.         else if variable is start-variable and assignment is empty then return failure
18.     }
19.     return solution
20. }
```

Όπως παρατηρούμε παραπάνω στη γραμμή 4 καλείται η συνάρτηση **SELECT-UNASSIGMENT-VARIABLE (X, assigment)** σε αυτή λοιπόν τη θέση καλούμε την ευριστική συνάρτηση που αναλύσαμε προηγουμένως, δηλαδή **τη COMMONCHOICE (X, assigment)** ή αν θέλουμε τη χρήση της παραμέτρου k τότε καλούμε τη **COMMONCHOICE_K (X, assigment, k) .**

4 Πειραματική Διαδικασία

Στην παρούσα ενότητα, γίνεται παρουσίαση των αποτελεσμάτων τα οποία προέκυψαν από την εκτέλεση των διαφόρων heuristics (dom/ddeg, dom/wdeg, ABS, IBS) τα οποία αναλύθηκαν στην προηγούμενη ενότητα. Συγκεκριμένα εφαρμόσαμε τα παραπάνω Heuristics σε δυαδικά προβλήματα των εξής τριών κατηγοριών:

1. **RLFAP (Radio Radio Link Frequency Assignment Problem)**: Είναι προβλήματα, τα οποία έχουν ως σκοπό την ανάθεση συχνοτήτων σε έναν αριθμό ραδιοζεύξεων, έτσι ώστε να ικανοποιούνται ταυτόχρονα πολλοί περιορισμοί και να χρησιμοποιούνται όσο το δυνατόν λιγότερες συχνότητες. Τα προβλήματα αυτά περιλαμβάνουν ένα σύνολο μεταβλητών, ο αριθμός των οποίων κυμαίνεται από 200-916 μεταβλητές, καθώς και ένα σύνολο περιορισμών, ο αριθμός των οποίων κυμαίνεται από 648-4638. Οι περιορισμοί οι οποίοι υπάρχουν είναι δυαδικοί (δηλαδή σε κάθε περιορισμό, συμμετέχουν δύο μόνο μεταβλητές) και είναι όλοι της μορφής $|x-y|>z$ και $|x-y|=z$ (όπου z φυσικός αριθμός και x, y οι μεταβλητές του προβλήματος). Το πλήθος των πεδίων τιμών για τις μεταβλητές ενός προβλήματος, μπορεί να διαφέρει από πρόβλημα σε πρόβλημα. Το σύνολο των πεδίων, τα όποια υπάρχουν σε κάποιο πρόβλημα ξεκινούν από 2 και μπορεί να φτάσουν τα 7 στο σύνολο. Οι τιμές οι οποίες μπορεί να υπάρχουν σε κάποιο πεδίο κυμαίνονται από 6-44.
2. **GC (Graph Coloring)**: Μερικά από τα πιο ευρέως μελετημένα συνδυαστικά προβλήματα, είναι τα προβλήματα χρωματισμού γραφημάτων. Σε αυτά τα προβλήματα, δοθέντος ενός γραφήματος και ενός συνόλου χρωμάτων, σκοπός του προβλήματος είναι να χρωματιστούν όλοι οι κόμβοι του προβλήματος με τέτοιο τρόπο ώστε καμία ακμή να μην συνδέει κόμβους του ίδιου χρώματος. Στην κατηγορία αυτή υπάγονται και κάποια nxn queens προβλήματα, στα οποία έχουμε αναφερθεί σε προηγούμενη ενότητα. Τα προβλήματα αυτά περιλαμβάνουν ένα σύνολο μεταβλητών, ο αριθμός των οποίων κυμαίνεται από 25-2030 μεταβλητές, καθώς και ένα σύνολο περιορισμών, ο αριθμός των οποίων κυμαίνεται από 100-33751. Οι περιορισμοί οι οποίοι υπάρχουν είναι δυαδικοί (δηλαδή σε κάθε περιορισμό, συμμετέχουν δύο μόνο μεταβλητές) και είναι όλοι της μορφής $|x-y|\neq 0$ (όπου z φυσικός αριθμός και x, y οι μεταβλητές του προβλήματος). Σε κάθε πρόβλημα, όλες οι μεταβλητές λαμβάνουν το ίδιο πεδίο τιμών. Οι τιμές οι οποίες μπορεί να υπάρχουν σε κάποιο πεδίο κυμαίνονται από 3-9.
3. **PIGEONS**: Σκοπός των προβλημάτων αυτών, είναι η τοποθέτηση n περιστεριών σε n-1 κουτιά έτσι ώστε ένα περιστέρι να αντιστοιχεί σε ακριβώς ένα κουτί. Τα προβλήματα αυτά περιλαμβάνουν ένα σύνολο μεταβλητών, ο αριθμός των οποίων κυμαίνεται από 5-50 μεταβλητές, καθώς και ένα σύνολο περιορισμών, ο αριθμός των οποίων κυμαίνεται από 10-1225. Οι περιορισμοί οι οποίοι υπάρχουν είναι δυαδικοί (δηλαδή σε κάθε περιορισμό, συμμετέχουν δύο μόνο μεταβλητές) και είναι όλοι της μορφής $|x-y|\neq 0$ (όπου z φυσικός

αριθμός και x, y οι μεταβλητές του προβλήματος). Σε κάθε πρόβλημα, όλες οι μεταβλητές λαμβάνουν το ίδιο πεδίο τιμών. Οι τιμές οι οποίες μπορεί να υπάρχουν σε κάποιο πεδίο κυμαίνονται από 4-49.

Πιο αναλυτικά, η σύγκριση των παραπάνω heuristics πραγματοποιήθηκε βάση τριών κριτηρίων:

1. Το χρόνο εκτέλεσης (σε δευτερόλεπτα) που απαιτείται προκειμένου ο αλγόριθμος MAC να βρει (ή όχι) λύση σε καθένα από τα παραπάνω προβλήματα, με χρήση των ανωτέρω heuristics .
2. Το πλήθος των κόμβων τους οποίους δημιουργεί ο αλγόριθμος MAC κατά την προσπάθεια για εύρεση λύσης στο πρόβλημα.
3. Καθώς και το σύνολο των προβλημάτων στα οποία ο αλγόριθμος MAC καταφέρνει να εξάγει κάποιο συμπέρασμα (επίλυση του προβλήματος ή διαπίστωση ότι κάποιο πρόβλημα δεν έχει λύση).

Για την εκτέλεση κάθε πειράματος (κάθε προβλήματος), υπάρχει το χρονικό όριο της μίας ώρας (3600 δευτερόλεπτα). Στην περίπτωση όπου το χρονικό όριο αυτό ξεπεραστεί, ο αλγόριθμος τερματίζεται. Αυτό σημαίνει ότι ο αλγόριθμος δεν έχει καταφέρει μέχρι εκείνη τη στιγμή να βρει κάποια λύση ή δεν έχει καταλήξει στο συμπέρασμα ότι δεν μπορεί να βρει κάποια λύση και επομένως επιστρέφει το πλήθος των κόμβων που έχουν δημιουργηθεί μέχρι εκείνη τη χρονική στιγμή.

Επιπλέον, γίνεται χρήση μιας παραμέτρου κ, η οποία ορίζεται από τον χρήστη και δηλώνει τον αριθμό των κ καλύτερων μεταβλητών του προβλήματος, σύμφωνα με το κάθε heuristic για την επιλογή της επόμενης μεταβλητής. Η επιλογή της μεταβλητής γίνεται με τυχαίο τρόπο μέσα από το σύνολο των κ καλύτερων μεταβλητών. Για τα συγκεκριμένα πειράματα, ορίσαμε το κ με τις τιμές 1, 3 και 5.

Τέλος, θα πρέπει να αναφέρουμε ότι τα πειράματα διεξήχθησαν σε H/Y Desktop με λειτουργικό σύστημα Windows 10, CPU Intel core-I7 6700k χρονισμένο στα 4GHz και μνήμης RAM 8 GB. Επίσης, για την υλοποίηση των απαραίτητων προγραμμάτων, έγινε χρήση της γλώσσας προγραμματισμού Python 3.x.

4.1 Αποτελέσματα dom/ddeg

Τα αποτελέσματα τα οποία προέκυψαν από την εκτέλεση των πειραμάτων για το dom/ddeg heuristic για τις παραπάνω κατηγορίες προβλημάτων, παρουσιάζονται στη συνέχεια σε μορφή πινάκων.

Πιο αναλυτικά, ο κάθε πίνακας ερμηνεύεται ως εξής:

- $1^{η}$ Στήλη: Στη συγκεκριμένη στήλη αναγράφεται ο αύξων αριθμός του κάθε προβλήματος.
- $2^{η}$ Στήλη: Στη συγκεκριμένη στήλη αναγράφεται το όνομα του κάθε προβλήματος.
- $3^{η}$ Στήλη: Στη συγκεκριμένη στήλη αναγράφεται το σύνολο των κόμβων που δημιουργήθηκαν κατά την διαδικασία της αναζήτησης.
- $4^{η}$ Στήλη: Στη συγκεκριμένη στήλη αναγράφεται ο συνολικός χρόνος (σε δευτερόλεπτα) εκτέλεσης της αναζήτησης.
- $5^{η}$ Στήλη: Στη συγκεκριμένη στήλη αναγράφεται το αποτέλεσμα της αναζήτησης. Συγκεκριμένα στην περίπτωση όπου η αναζήτηση οδηγήσει σε επίλυση του προβλήματος, τότε για το συγκεκριμένο πρόβλημα αναγράφεται η λέξη **SUCCESS** (ΕΠΙΤΥΧΙΑ). Στην περίπτωση όπου η αναζήτηση οδηγήσει σε αποτυχία επίλυσης του προβλήματος, τότε για το συγκεκριμένο πρόβλημα αναγράφεται η λέξη **FAIL** (ΑΠΟΤΥΧΙΑ). Τέλος, στην περίπτωση όπου ο συνολικός χρόνος της αναζήτησης ξεπεράσει το όριο που έχει τεθεί (δηλαδή >3600 δευτερόλεπτα ή 1 ώρα), τότε η αναζήτηση τερματίζει και αναγράφεται η λέξη **NONE** (ΚΑΝΕΝΑ).

Επίσης, στην τελευταία γραμμή του πίνακα, υπολογίζεται ο Μέσος Όρος (Μ.Ο.) των κόμβων και του συνολικού χρόνου (Στήλες 3 και 4 αντίστοιχα) των προβλημάτων εκείνων, τα οποία οδήγησαν σε αποτέλεσμα SUCCESS ή FAIL.

Τέλος, η παράμετρος k με τιμές k={1, 3, 5} δίνει την επιλογή στο dom/ddeg, να επιλέξει τυχαία την επόμενη μεταβλητή μέσα από το σύνολο των 1, 3, 5 καλύτερων μεταβλητών σύμφωνα με αυτό το heuristic. Σαφώς, στην περίπτωση όπου k=1, η ευριστική συνάρτηση επιλέγει την πρώτη καλύτερη μεταβλητή.

4.1.1 RLFAP (Radio Link Frequency Assignment Problem)

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	2-f24	206	1.88	SUCCESS
2	2-f25	229091	>3600	NONE
3	3-f10	436467	>3600	NONE
4	3-f11	469089	>3600	NONE
5	6-w2	14	3.92	FAIL
6	7-w1-f4	428	4.97	SUCCESS
7	7-w1-f5	1398719	>3600	NONE
8	8-f10	233701	>3600	NONE
9	8-f11	148071	>3600	NONE
10	11	45531	1442.91	SUCCESS

11	14-f27	297204	>3600	NONE
12	14-f28	284763	>3600	NONE
M.O.	-	11546	363.42	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 4 από τα 12 προβλήματα. Στο πρόβλημα 2-f24 πέτυχε τον μικρότερο χρόνο εκτέλεσης και στο πρόβλημα 6-w2 τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	2-f24	956636	>3600	NONE
2	2-f25	208570	>3600	NONE
3	3-f10	712	12.02	SUCCESS
4	3-f11	329317	>3600	NONE
5	6-w2	14	5.30	FAIL
6	7-w1-f4	1164	21.08	SUCCESS
7	7-w1-f5	691386	>3600	NONE
8	8-f10	174010	>3600	NONE
9	8-f11	165783	>3600	NONE
10	11	83799	3004.88	SUCCESS
11	14-f27	287441	>3600	NONE
12	14-f28	378188	>3600	NONE
M.O.	-	21422	760.82	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) και πάλι σε 4 από τα 12 προβλήματα. Στο πρόβλημα 6-w2 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	2-f24	202	1.84	SUCCESS
2	2-f25	195580	>3600	NONE
3	3-f10	551295	>3600	NONE
4	3-f11	550425	>3600	NONE
5	6-w2	14	5.75	FAIL
6	7-w1-f4	427	6.81	SUCCESS
7	7-w1-f5	824158	>3600	NONE
8	8-f10	197609	>3600	NONE
9	8-f11	144101	>3600	NONE
10	11	21726	565.69	SUCCESS
11	14-f27	486489	>3600	NONE
12	14-f28	352415	>3600	NONE
M.O.	-	5592	145.02	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) και πάλι σε 4 από τα 12 προβλήματα. Στο πρόβλημα 2-f24 πέτυχε τον μικρότερο χρόνο εκτέλεσης και στο πρόβλημα 6-w2 τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=5 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=3 τις χειρότερες.

4.1.2 GC (Graph Coloring)

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	anna-5	205	1.20	FAIL
2	anna-8	149920	698.95	FAIL
3	david-5	205	0.95	FAIL
4	david-8	69280	349.86	FAIL
5	games120-5	625	1.62	FAIL
6	games120-7	2281704	>3600	NONE
7	games120-9	120	0.36	SUCCESS
8	homer-5	205	3.17	FAIL
9	huck-5	205	0.45	FAIL
10	huck-8	5187723	>3600	NONE
11	jean-5	445	1.02	FAIL
12	jean-7	1369459	1584.99	FAIL
13	miles250-6	1236	2.94	FAIL
14	miles250-7	33859	68.80	FAIL
15	miles250-8	128	0.16	SUCCESS
16	miles500-5	205	2.22	FAIL
17	miles750-5	205	5.67	FAIL
18	miles1000-5	205	9.67	FAIL
19	queen5-5-4	40	0.11	FAIL
20	queen6-6-6	57036	136.47	FAIL
21	queen7-7-6	39396	190.58	FAIL
22	1-fullins-3-3	87	0.05	FAIL
23	1-fullins-4-4	6108276	>3600	NONE
24	1-insertions-4-3	585	0.58	FAIL
25	2-fullins-3-4	66496	34.70	FAIL
26	2-fullins-4-4	897895	>3600	NONE
27	3-fullins-3-5	3735943	>3600	NONE
28	3-fullins-5-5	32275	>3600	NONE
29	4-fullins-3-6	2665246	>3600	NONE
M.O.	-	81370	145.23	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL)

σε 22 από τα 29 προβλήματα. Στο πρόβλημα queen5-5-4 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	anna-5	205	1.23	FAIL
2	anna-8	134818	649.28	FAIL
3	david-5	205	0.89	FAIL
4	david-8	118218	490.95	FAIL
5	games120-5	18751	37.06	FAIL
6	games120-7	2893957	>3600	NONE
7	games120-9	120	0.31	SUCCESS
8	homer-5	205	3.44	FAIL
9	huck-5	247	1.03	FAIL
10	huck-8	6600977	>3600	NONE
11	jean-5	357	1.03	FAIL
12	jean-7	2404724	2411.78	FAIL
13	miles250-6	1236	3.12	FAIL
14	miles250-7	13573	34.72	FAIL
15	miles250-8	3452497	>3600	NONE
16	miles500-5	205	2.84	FAIL
17	miles750-5	205	6.98	FAIL
18	miles1000-5	205	11.98	FAIL
19	queen5-5-4	40	0.11	FAIL
20	queen6-6-6	49931	171.66	FAIL
21	queen7-7-6	31428	168.09	FAIL
22	1-fullins-3-3	102	0.05	FAIL
23	1-fullins-4-4	5319635	>3600	NONE
24	1-insertions-4-3	615	0.91	FAIL
25	2-fullins-3-4	41746	29.47	FAIL
26	2-fullins-4-4	690567	>3600	NONE
27	3-fullins-3-5	3859284	>3600	NONE
28	3-fullins-5-5	42997	>3600	NONE
29	4-fullins-3-6	2882820	>3600	NONE
M.O.	-	129083	191.80	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 21 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς ενώ στο πρόβλημα queen5-5-4 τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	anna-5	220	1.23	FAIL
2	anna-8	238078	1227.80	FAIL
3	david-5	205	1.36	FAIL
4	david-8	257345	1030.66	FAIL
5	games120-5	5527	13.78	FAIL
6	games120-7	2463681	>3600	NONE
7	games120-9	120	0.41	SUCCESS
8	homer-5	205	4.05	FAIL
9	huck-5	712	2.77	FAIL
10	huck-8	4153956	>3600	NONE
11	jean-5	642	1.86	FAIL
12	jean-7	2056334	2416.30	FAIL
13	miles250-6	2117	3.58	FAIL
14	miles250-7	7592877	>3600	NONE
15	miles250-8	128	0.16	SUCCESS
16	miles500-5	205	2.06	FAIL
17	miles750-5	205	5.14	FAIL
18	miles1000-5	205	9.01	FAIL
19	queen5-5-4	40	0.12	FAIL
20	queen6-6-6	48542	137.66	FAIL
21	queen7-7-6	21614	103.59	FAIL
22	1-fullins-3-3	31	0.03	FAIL
23	1-fullins-4-4	3812004	3023.28	FAIL
24	1-insertions-4-3	476	0.83	FAIL
25	2-fullins-3-4	64995	41.88	FAIL
26	2-fullins-4-4	764207	>3600	NONE
27	3-fullins-3-5	8202551	>3600	NONE
28	3-fullins-5-5	67952	>3600	NONE
29	4-fullins-3-6	6889205	>3600	NONE
M.O.	-	285096	364.89	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) και πάλι σε 22 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=1 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=3 τις χειρότερες. Επιπλέον για k=1 και k=5 ο αλγόριθμος επιλύει ένα περισσότερο πρόβλημα σε σχέση με k=3.

4.1.3 PIGEONS

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	5	40	0.02	FAIL
2	6	205	0.03	FAIL
3	7	1236	0.22	FAIL
4	8	8659	1.72	FAIL
5	9	69280	14.70	FAIL
6	10	623529	140.95	FAIL
7	11	6235300	1486.73	FAIL
8	12	13940189	>3600	NONE
9	13	13130158	>3600	NONE
10	14	9438261	>3600	NONE
11	15	8960219	>3600	NONE
12	16	9008590	>3600	NONE
13	18	10088416	>3600	NONE
14	20	8307603	>3600	NONE
15	25	5996669	>3600	NONE
16	30	5075409	>3600	NONE
17	35	3944743	>3600	NONE
18	40	2673381	>3600	NONE
19	45	1620461	>3600	NONE
20	50	1499564	>3600	NONE
M.O.	-	991178	236.31	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	5	40	0.01	FAIL
2	6	205	0.03	FAIL
3	7	1236	0.25	FAIL
4	8	8659	1.80	FAIL
5	9	69280	15.17	FAIL
6	10	623529	173.89	FAIL
7	11	6235300	1512.00	FAIL
8	12	13766363	>3600	NONE
9	13	13812367	>3600	NONE
10	14	12538646	>3600	NONE
11	15	12115124	>3600	NONE
12	16	10862618	>3600	NONE

13	18	10023640	>3600	NONE
14	20	8432379	>3600	NONE
15	25	4045438	>3600	NONE
16	30	4624227	>3600	NONE
17	35	2828488	>3600	NONE
18	40	3228990	>3600	NONE
19	45	2779382	>3600	NONE
20	50	2234577	>3600	NONE
M.O.	-	991178	243.31	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	5	40	0.01	FAIL
2	6	205	0.03	FAIL
3	7	1236	0.23	FAIL
4	8	8659	1.78	FAIL
5	9	69280	18.06	FAIL
6	10	623529	183.12	FAIL
7	11	6235300	2477.27	FAIL
8	12	11901759	>3600	NONE
9	13	11246456	>3600	NONE
10	14	8887167	>3600	NONE
11	15	8689387	>3600	NONE
12	16	8484393	>3600	NONE
13	18	9086262	>3600	NONE
14	20	8775567	>3600	NONE
15	25	5668322	>3600	NONE
16	30	4714985	>3600	NONE
17	35	3823188	>3600	NONE
18	40	2668041	>3600	NONE
19	45	2300162	>3600	NONE
20	50	1983656	>3600	NONE
M.O.	-	991178	382.93	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=1 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=5 τις χειρότερες ως

προς τον χρόνο εκτέλεσης, καθώς και στις τρεις περιπτώσεις δημιουργείται ο ίδιος αριθμός κόμβων.

4.2 Αποτελέσματα dom/wdeg

Τα αποτελέσματα τα οποία προέκυψαν από την εκτέλεση των πειραμάτων για το dom/wdeg heuristic για τις παραπάνω κατηγορίες προβλημάτων, παρουσιάζονται στη συνέχεια σε μορφή πινάκων.

4.2.1 RLFAP (Radio Link Frequency Assignment Problem)

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	2-f24	201	1.81	SUCCESS
2	2-f25	14785	254.25	FAIL
3	3-f10	1264	24.59	SUCCESS
4	3-f11	8624	355.80	FAIL
5	6-w2	14	5.06	FAIL
6	7-w1-f4	429	6.02	SUCCESS
7	7-w1-f5	950	15.27	FAIL
8	8-f10	11201	353.56	SUCCESS
9	8-f11	18736	672.94	FAIL
10	11	2750	110.70	SUCCESS
11	14-f27	3709	47.78	SUCCESS
12	14-f28	8423	131.61	FAIL
M.O.	-	5924	164.95	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) και στα 12 προβλήματα. Στο πρόβλημα 2-f24 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο 6-w2 τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	2-f24	357	2.61	SUCCESS
2	2-f25	741	14.25	FAIL
3	3-f10	866	16.12	SUCCESS
4	3-f11	12487	639.11	FAIL
5	6-w2	14	3.89	FAIL
6	7-w1-f4	566	7.50	SUCCESS
7	7-w1-f5	831	11.55	FAIL
8	8-f10	15988	416.28	SUCCESS
9	8-f11	3142	85.62	FAIL

10	11	1386	46.72	SUCCESS
11	14-f27	5698	68.26	SUCCESS
12	14-f28	8669	90.14	FAIL
M.O.	-	4229	116.84	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά και στα 12 προβλήματα. Στο πρόβλημα 2-f24 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης ενώ στο 6-w2 ξανά τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	2-f24	201	1.83	SUCCESS
2	2-f25	576	12.38	FAIL
3	3-f10	2640	53.95	SUCCESS
4	3-f11	15015	733.30	FAIL
5	6-w2	14	4.17	FAIL
6	7-w1-f4	536	7.25	SUCCESS
7	7-w1-f5	1312	14.03	FAIL
8	8-f10	18514	534.64	SUCCESS
9	8-f11	462	18.42	FAIL
10	11	2813	104.36	SUCCESS
11	14-f27	19077	167.78	SUCCESS
12	14-f28	79687	1020.55	FAIL
M.O.	-	71487	222.72	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά και στα 12 προβλήματα. Στο πρόβλημα 2-f24 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης ενώ στο 6-w2 ξανά τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=3 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=5 τις χειρότερες.

4.2.2 GC (Graph Coloring)

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	anna-5	205	1.17	FAIL
2	anna-8	69321	420.92	FAIL
3	david-5	205	1.09	FAIL
4	david-8	69280	328.38	FAIL
5	games120-5	612	1.75	FAIL
6	games120-7	293395	764.26	FAIL

7	games120-9	120	0.41	SUCCESS
8	homer-5	205	3.81	FAIL
9	huck-5	205	0.64	FAIL
10	huck-8	69797	129.72	FAIL
11	jean-5	228	0.58	FAIL
12	jean-7	29403	66.33	FAIL
13	miles250-6	1236	2.89	FAIL
14	miles250-7	8928	25.84	FAIL
15	miles250-8	128	0.22	SUCCESS
16	miles500-5	205	2.55	FAIL
17	miles750-5	205	6.30	FAIL
18	miles1000-5	205	11.27	FAIL
19	queen5-5-4	40	0.11	FAIL
20	queen6-6-6	52903	199.83	FAIL
21	queen7-7-6	3783	27.83	FAIL
22	1-fullins-3-3	37	0.03	FAIL
23	1-fullins-4-4	1550	7.34	FAIL
24	1-insertions-4-3	447	0.53	FAIL
25	2-fullins-3-4	416	0.77	FAIL
26	2-fullins-4-4	508	4.95	FAIL
27	3-fullins-3-5	4034	7.81	FAIL
28	3-fullins-5-5	6760	1382.36	FAIL
29	4-fullins-3-6	58396	288.88	FAIL
M.O.	-	330642	126.23	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) και στα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	anna-5	205	1.28	FAIL
2	anna-8	69352	401.05	FAIL
3	david-5	205	1.17	FAIL
4	david-8	69280	348.62	FAIL
5	games120-5	312	0.70	FAIL
6	games120-7	112146	239.76	FAIL
7	games120-9	120	0.31	SUCCESS
8	homer-5	205	3.02	FAIL
9	huck-5	211	0.77	FAIL
10	huck-8	81311	213.84	FAIL
11	jean-5	476	1.06	FAIL
12	jean-7	18059	26.11	FAIL
13	miles250-6	1248	2.22	FAIL
14	miles250-7	8659	17.95	FAIL
15	miles250-8	128	0.16	SUCCESS

16	miles500-5	205	1.84	FAIL
17	miles750-5	205	4.67	FAIL
18	miles1000-5	205	8.61	FAIL
19	queen5-5-4	40	0.08	FAIL
20	queen6-6-6	58489	160.27	FAIL
21	queen7-7-6	1766	9.28	FAIL
22	1-fullins-3-3	45	0.03	FAIL
23	1-fullins-4-4	941	4.05	FAIL
24	1-insertions-4-3	348	0.44	FAIL
25	2-fullins-3-4	480	0.81	FAIL
26	2-fullins-4-4	568	5.23	FAIL
27	3-fullins-3-5	4402	11.41	FAIL
28	3-fullins-5-5	4699	745.19	FAIL
29	4-fullins-3-6	56039	243.62	FAIL
M.O.	-	16909	84.61	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά και στα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο queen-5-4 τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	anna-5	214	1.22	FAIL
2	anna-8	160363	643.17	FAIL
3	david-5	205	1.16	FAIL
4	david-8	69574	328.81	FAIL
5	games120-5	3270	6.78	FAIL
6	games120-7	1208176	2307.75	FAIL
7	games120-9	120	0.36	SUCCESS
8	homer-5	211	3.75	FAIL
9	huck-5	414	1.53	FAIL
10	huck-8	151213	155.73	FAIL
11	jean-5	255	0.69	FAIL
12	jean-7	9010	12.36	FAIL
13	miles250-6	1731	3.49	FAIL
14	miles250-7	25773	44.28	FAIL
15	miles250-8	131	0.17	SUCCESS
16	miles500-5	205	2.09	FAIL
17	miles750-5	205	5.42	FAIL
18	miles1000-5	205	9.77	FAIL
19	queen5-5-4	43	0.12	FAIL
20	queen6-6-6	54563	174.78	FAIL
21	queen7-7-6	4395	24.20	FAIL
22	1-fullins-3-3	59	0.06	FAIL
23	1-fullins-4-4	1068	5.44	FAIL
24	1-insertions-4-3	521	0.69	FAIL

25	2-fullins-3-4	685	1.03	**FAIL**
26	2-fullins-4-4	749	8.61	**FAIL**
27	3-fullins-3-5	16015	20.52	**FAIL**
28	3-fullins-5-5	9720	2209.78	**FAIL**
29	4-fullins-3-6	7634897	>3600	**NONE**
M.O.	-	334071	213.35	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 28 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο queen-5-4 τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=3 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=5 τις χειρότερες. Επιπλέον, για k=1 και k=3 ο αλγόριθμος οδηγείται επιλύει ένα περισσότερο πρόβλημα σε σχέση με k=5.

4.2.3 PIGEONS

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	5	40	0.02	**FAIL**
2	6	205	0.08	**FAIL**
3	7	1236	0.28	**FAIL**
4	8	8659	1.69	**FAIL**
5	9	69280	14.30	**FAIL**
6	10	623529	138.14	**FAIL**
7	11	6235300	1448.56	**FAIL**
8	12	14325091	>3600	**NONE**
9	13	13496289	>3600	**NONE**
10	14	11880029	>3600	**NONE**
11	15	10851700	>3600	**NONE**
12	16	10119740	>3600	**NONE**
13	18	10006062	>3600	**NONE**
14	20	6807733	>3600	**NONE**
15	25	5990345	>3600	**NONE**
16	30	5462219	>3600	**NONE**
17	35	3604301	>3600	**NONE**
18	40	3369363	>3600	**NONE**
19	45	2598530	>3600	**NONE**
20	50	2287273	>3600	**NONE**
M.O.	-	991178	229.01	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL)

σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	5	40	0.02	FAIL
2	6	205	0.03	FAIL
3	7	1236	0.22	FAIL
4	8	8659	1.66	FAIL
5	9	69280	14.08	FAIL
6	10	623529	165.28	FAIL
7	11	6235300	1523.64	FAIL
8	12	13711576	>3600	NONE
9	13	13848663	>3600	NONE
10	14	9922356	>3600	NONE
11	15	7506726	>3600	NONE
12	16	10581966	>3600	NONE
13	18	9969919	>3600	NONE
14	20	8828286	>3600	NONE
15	25	6737579	>3600	NONE
16	30	4684249	>3600	NONE
17	35	3526869	>3600	NONE
18	40	3053304	>3600	NONE
19	45	2621428	>3600	NONE
20	50	1442543	>3600	NONE
M.O.	-	991178	243.56	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτερόλεπτα)	Αποτέλεσμα
1	5	40	0.02	FAIL
2	6	205	0.03	FAIL
3	7	1236	0.22	FAIL
4	8	8659	1.69	FAIL
5	9	69280	14.20	FAIL
6	10	623529	137.89	FAIL
7	11	6235300	1837.09	FAIL
8	12	12757671	>3600	NONE
9	13	13210066	>3600	NONE
10	14	11721259	>3600	NONE
11	15	7925838	>3600	NONE

12	16	10473356	>3600	NONE
13	18	7114584	>3600	NONE
14	20	6231965	>3600	NONE
15	25	4756907	>3600	NONE
16	30	4057332	>3600	NONE
17	35	4371761	>3600	NONE
18	40	3061258	>3600	NONE
19	45	2259828	>3600	NONE
20	50	1716608	>3600	NONE
M.O.	-	991178	284.45	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=1 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=5 τις χειρότερες ως προς τον χρόνο εκτέλεσης, καθώς και στις τρεις περιπτώσεις δημιουργείται ο ίδιος αριθμός κόμβων.

4.3 Αποτελέσματα ABS (ACTIVITY BASED SEARCH)

Τα αποτελέσματα τα οποία προέκυψαν από την εκτέλεση των πειραμάτων για το ABS heuristic για τις παραπάνω κατηγορίες προβλημάτων, παρουσιάζονται στη συνέχεια σε μορφή πινάκων.

Επιπρόσθετα, στο ABS heuristic οι πίνακες των αποτελεσμάτων, περιλαμβάνουν μία επιπλέον στήλη ($5^{η}$ Στήλη) στην οποία καταγράφονται οι χρόνοι αρχικοποίησης των activities σε δευτερόλεπτα.

Έπειτα από δοκιμές αλλά και σύμφωνα με τις παρατηρήσεις που υπάρχουν στο [14], προέκυψε ότι ο συντελεστής γ για την τιμή 0.999 παρουσιάζει τις βέλτιστες αποδόσεις. Επίσης, ο αριθμός των probes, επιλέχθηκε εμπειρικά εφόσον έπειτα από δοκιμές καταλήξαμε ότι η τιμή 25 παρουσιάζει τις ίδιες περίπου επιδόσεις τις οποίες θα παρουσίαζαν αν επιλέγαμε αριθμό probes ίσο με 75, 100, 125, 150.

4.3.1 RLFAP (Radio Link Frequency Assignment Problem)

•k = 1 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	2-f24	200	5.58	4.89	SUCCESS
2	2-f25	1061389	>3600	11.91	NONE
3	3-f10	156602	>3600	53.38	NONE
4	3-f11	530611	>3600	31.28	NONE
5	6-w2	46492	>3600	7.92	NONE
6	7-w1-f4	410	10.50	5.16	SUCCESS
7	7-w1-f5	277914	>3600	2.41	NONE
8	8-f10	185751	>3600	31.08	NONE
9	8-f11	200849	>3600	17.27	NONE
10	11	131175	>3600	262.59	NONE
11	14-f27	276887	>3600	9.09	NONE
12	14-f28	289659	>3600	4.72	NONE
M.O.	-	305	8.04	5.03	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 2 από τα 12 προβλήματα. Στο πρόβλημα 2-f24 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	2-f24	202	17.05	14.74	SUCCESS
2	2-f25	291474	>3600	14.44	NONE
3	3-f10	164845	1081.89	50.23	SUCCESS
4	3-f11	411462	>3600	42.24	NONE
5	6-w2	52610	>3600	9.11	NONE
6	7-w1-f4	436	14.67	6.30	SUCCESS
7	7-w1-f5	529083	>3600	4.95	NONE
8	8-f10	287429	>3600	39.08	NONE
9	8-f11	137454	>3600	13.03	NONE
10	11	84068	>3600	257.66	NONE
11	14-f27	357457	>3600	12.11	NONE
12	14-f28	288322	>3600	6.03	NONE
M.O.	-	55161	371.2	23.76	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 3 από τα 12 προβλήματα. Στο πρόβλημα 7-w1-f4 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο 2-f24 τον μικρότερο αριθμό κόμβων.

•k = 5 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	2-f24	1250465	>3600	14.17	NONE
2	2-f25	855909	>3600	11.36	NONE
3	3-f10	26931	902.06	35.41	SUCCESS
4	3-f11	100279	>3600	27.34	NONE
5	6-w2	59158	>3600	7.30	NONE
6	7-w1-f4	499	12.45	4.23	SUCCESS
7	7-w1-f5	233826	>3600	2.47	NONE
8	8-f10	174116	>3600	35.09	NONE
9	8-f11	164	27.09	16.34	FAIL
10	11	78630	>3600	313.11	NONE
11	14-f27	392842	>3600	10.38	NONE
12	14-f28	274301	>3600	5.25	NONE
M.O.	-	9198	313.87	18.66	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά σε 3 από τα 12 προβλήματα. Στο πρόβλημα 7-w1-f4 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο 8-f11 τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης, αρχικοποίησης και κόμβων, προκύπτει ότι για k=1 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=3 τις χειρότερες ως προς τον χρόνο εκτέλεσης, καθώς και στις τρεις περιπτώσεις δημιουργείται ο ίδιος αριθμός κόμβων. Επιπλέον, για k=3 και k=5 ο αλγόριθμος επιλύει ένα πρόβλημα περισσότερο σε σχέση με k=1.

4.3.2 GC (Graph Coloring)

•k = 1 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	anna-5	205	2.27	1.05	FAIL
2	anna-8	69497	416.81	3.26	FAIL
3	david-5	205	2.44	1.42	FAIL
4	david-8	69280	382.20	3.76	FAIL
5	games120-5	205	2.28	1.69	FAIL
6	games120-7	2471141	>3600	4.36	NONE
7	games120-9	120	0.38	0.30	SUCCESS
8	homer-5	205	7.16	3.48	FAIL
9	huck-5	205	1.61	0.86	FAIL
10	huck-8	1604964	1637.51	2.50	FAIL
11	jean-5	249	1.16	0.67	FAIL
12	jean-7	11167	18.08	1.47	FAIL
13	miles250-6	1236	4.49	1.56	FAIL

14	miles250-7	17119	31.19	2.00	FAIL
15	miles250-8	128	2.02	1.97	SUCCESS
16	miles500-5	205	4.64	2.69	FAIL
17	miles750-5	205	10.78	5.45	FAIL
18	miles1000-5	205	19.16	9.17	FAIL
19	queen5-5-4	40	0.39	0.30	FAIL
20	queen6-6-6	56686	185.00	1.38	FAIL
21	queen7-7-6	1838	14.66	2.12	FAIL
22	1-fullins-3-3	17	0.09	0.08	FAIL
23	1-fullins-4-4	60498	157.22	0.88	FAIL
24	1-insertions-4-3	1145	1.38	0.28	FAIL
25	2-fullins-3-4	12760	10.92	0.45	FAIL
26	2-fullins-4-4	413715	1669.78	2.39	FAIL
27	3-fullins-3-5	2726372	>3600	0.88	NONE
28	3-fullins-5-5	38955	>3600	49.00	NONE
29	4-fullins-3-6	2518973	>3600	2.86	NONE
M.O.	-	92884	183.34	2.05	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 25 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3 probes = 25 $\gamma = 0.999$

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	anna-5	205	2.17	1.06	FAIL
2	anna-8	112070	730.70	3.11	FAIL
3	david-5	205	2.42	1.17	FAIL
4	david-8	69280	492.56	3.67	FAIL
5	games120-5	205	2.25	1.67	FAIL
6	games120-7	3002349	>3600	4.03	NONE
7	games120-9	120	0.31	0.25	SUCCESS
8	homer-5	205	5.33	2.58	FAIL
9	huck-5	227	1.34	0.67	FAIL
10	huck-8	671231	424.94	2.00	FAIL
11	jean-5	227	0.95	0.56	FAIL
12	jean-7	1616910	961.28	1.28	FAIL
13	miles250-6	1236	3.55	1.19	FAIL
14	miles250-7	12383	22.89	1.62	FAIL
15	miles250-8	128	0.45	0.42	SUCCESS
16	miles500-5	205	4.24	2.19	FAIL
17	miles750-5	205	9.52	4.67	FAIL
18	miles1000-5	205	16.19	7.99	FAIL
19	queen5-5-4	40	0.34	0.25	FAIL
20	queen6-6-6	56025	166.75	1.19	FAIL
21	queen7-7-6	2777	16.51	1.80	FAIL
22	1-fullins-3-3	37	0.14	0.09	FAIL

23	1-fullins-4-4	59456	159.19	1.00	FAIL
24	1-insertions-4-3	1123	1.42	0.25	FAIL
25	2-fullins-3-4	4713	4.55	0.34	FAIL
26	2-fullins-4-4	33531	184.92	2.78	FAIL
27	3-fullins-3-5	13035	27.14	1.33	FAIL
28	3-fullins-5-5	41602	>3600	46.36	NONE
29	4-fullins-3-6	1995581	>3600	2.56	NONE
M.O.	-	102153	124.69	1.74	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 26 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5 probes = 25 $\gamma = 0.999$

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	anna-5	230	2.48	1.06	FAIL
2	anna-8	414342	2820.67	3.20	FAIL
3	david-5	205	2.23	1.19	FAIL
4	david-8	70715	511.39	3.72	FAIL
5	games120-5	224	2.17	1.45	FAIL
6	games120-7	2084287	>3600	4.00	NONE
7	games120-9	120	0.31	0.25	SUCCESS
8	homer-5	205	5.38	2.64	FAIL
9	huck-5	269	1.50	0.67	FAIL
10	huck-8	1622033	1761.50	1.88	FAIL
11	jean-5	205	0.94	0.59	FAIL
12	jean-7	9522	15.67	1.28	FAIL
13	miles250-6	1834	4.45	1.08	FAIL
14	miles250-7	6022125	>3600	1.66	NONE
15	miles250-8	128	2.08	2.03	SUCCESS
16	miles500-5	205	5.83	3.33	FAIL
17	miles750-5	205	12.52	6.06	FAIL
18	miles1000-5	205	21.42	10.66	FAIL
19	queen5-5-4	40	0.48	0.33	FAIL
20	queen6-6-6	57353	247.89	1.52	FAIL
21	queen7-7-6	13939	101.47	2.39	FAIL
22	1-fullins-3-3	9	0.09	0.08	FAIL
23	1-fullins-4-4	13871	38.34	0.95	FAIL
24	1-insertions-4-3	1118	1.84	0.33	FAIL
25	2-fullins-3-4	5590	5.55	0.41	FAIL
26	2-fullins-4-4	39641	161.78	2.09	FAIL
27	3-fullins-3-5	743024	619.59	0.89	FAIL
28	3-fullins-5-5	39545	>3600	40.23	NONE
29	4-fullins-3-6	6154945	>3600	2.12	NONE
M.O.	-	119809	253.9	2	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά σε 25 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης, αρχικοποίησης και κόμβων, προκύπτει ότι για k=1 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις ως προς τον χρόνο εκτέλεσης και για k=3 ως προς το πλήθος των κόμβων που δημιουργεί, ενώ για k=5, ο αλγόριθμος παρουσιάζει τις χειρότερες επιδόσεις και για τις δύο παραμέτρους. Επιπλέον, για k=3 ο αλγόριθμος επιλύει ένα επιπλέον πρόβλημα σε σχέση με k=1 και k=5.

4.3.3 PIGEONS

•k = 1 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	5	40	0.03	0.02	FAIL
2	6	205	0.08	0.05	FAIL
3	7	1236	0.31	0.08	FAIL
4	8	8659	2.01	0.12	FAIL
5	9	69280	15.81	0.20	FAIL
6	10	623529	170.41	0.31	FAIL
7	11	6235300	1714.52	0.58	FAIL
8	12	11731045	>3600	0.66	NONE
9	13	12306254	>3600	1.14	NONE
10	14	12318849	>3600	1.64	NONE
11	15	12036364	>3600	1.67	NONE
12	16	11675969	>3600	2.19	NONE
13	18	10365229	>3600	3.56	NONE
14	20	8299597	>3600	5.58	NONE
15	25	6553408	>3600	14.53	NONE
16	30	4608704	>3600	30.91	NONE
17	35	3564591	>3600	63.16	NONE
18	40	2940528	>3600	114.72	NONE
19	45	2342702	>3600	193.52	NONE
20	50	1653177	>3600	313.66	NONE
M.O.	-	991178	271.88	0.19	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	5	40	0.03	0.02	FAIL
2	6	205	0.09	0.05	FAIL
3	7	1236	0.38	0.09	FAIL
4	8	8659	2.25	0.17	FAIL
5	9	69280	18.97	0.23	FAIL
6	10	623529	190.80	0.44	FAIL
7	11	6235300	1890.31	0.56	FAIL
8	12	14145035	>3600	0.66	NONE
9	13	10814789	>3600	1.20	NONE
10	14	11269265	>3600	1.27	NONE
11	15	11218737	>3600	1.89	NONE
12	16	10921828	>3600	2.22	NONE
13	18	9459934	>3600	3.89	NONE
14	20	8781499	>3600	5.81	NONE
15	25	6557375	>3600	14.73	NONE
16	30	4943517	>3600	31.23	NONE
17	35	3977547	>3600	60.12	NONE
18	40	3168202	>3600	104.44	NONE
19	45	2566931	>3600	175.45	NONE
20	50	2346260	>3600	277.67	NONE
M.O.	-	991178	300.4	0.22	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	5	40	0.03	0.03	FAIL
2	6	205	0.06	0.03	FAIL
3	7	1236	0.30	0.08	FAIL
4	8	8659	1.80	0.12	FAIL
5	9	69280	14.42	0.19	FAIL
6	10	623529	136.14	0.30	FAIL
7	11	6235300	1443.92	0.44	FAIL
8	12	14575392	>3600	0.62	NONE
9	13	14023355	>3600	0.88	NONE
10	14	12836023	>3600	1.16	NONE
11	15	10267180	>3600	1.73	NONE
12	16	10298526	>3600	2.91	NONE
13	18	9673923	>3600	4.53	NONE
14	20	8948186	>3600	5.31	NONE
15	25	6933574	>3600	14.45	NONE

16	30	4960241	>3600	29.12	NONE
17	35	3908947	>3600	63.59	NONE
18	40	3139086	>3600	100.26	NONE
19	45	2515910	>3600	178.48	NONE
20	50	2267784	>3600	321.50	NONE
M.O.	-	991178	228.1	0.17	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά σε 7 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=5 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=3 τις χειρότερες ως προς τον χρόνο εκτέλεσης, καθώς και στις τρεις περιπτώσεις δημιουργείται ο ίδιος αριθμός κόμβων.

4.4 Αποτελέσματα IBS (IMPACT BASED SEARCH)

Τα αποτελέσματα τα οποία προέκυψαν από την εκτέλεση των πειραμάτων για το IBS heuristic για τις παραπάνω κατηγορίες προβλημάτων, παρουσιάζονται στη συνέχεια σε μορφή πινάκων.

Επιπρόσθετα, στο IBS heuristic οι πίνακες των αποτελεσμάτων περιλαμβάνουν μία επιπλέον στήλη ($5^{η}$ Στήλη), στην οποία καταγράφονται οι χρόνοι αρχικοποίησης των impacts σε δευτερόλεπτα.

4.4.1 RLFAP (Radio Link Frequency Assignment Problem)

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	2-f24	354004	>3600	57.89	NONE
2	2-f25	268533	>3600	61.20	NONE
3	3-f10	173318	>3600	142.91	NONE
4	3-f11	223016	>3600	153.47	NONE
5	6-w2	52694	>3600	169.09	NONE
6	7-w1-f4	159300	>3600	95.38	NONE
7	7-w1-f5	137618	>3600	55.47	NONE
8	8-f10	263286	>3600	178.36	NONE
9	8-f11	167560	>3600	168.50	NONE
10	11	134379	>3600	598.33	NONE
11	14-f27	235085	>3600	85.24	NONE
12	14-f28	341451	>3600	52.52	NONE

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC δεν κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε κανένα από τα 20 προβλήματα. Επομένως, δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα όσον αφορά τον χρόνο εκτέλεσης και το πλήθος των κόμβων για τα παραπάνω προβλήματα.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	2-f24	496417	>3600	59.12	NONE
2	2-f25	459398	>3600	48.97	NONE
3	3-f10	404215	>3600	146.66	NONE
4	3-f11	413202	>3600	151.36	NONE
5	6-w2	89480	>3600	170.12	NONE
6	7-w1-f4	185723	>3600	81.61	NONE
7	7-w1-f5	421872	>3600	62.30	NONE
8	8-f10	187227	>3600	173.06	NONE
9	8-f11	217547	>3600	168.97	NONE
10	11	129410	>3600	480.17	NONE
11	14-f27	267072	>3600	81.02	NONE
12	14-f28	275217	>3600	65.94	NONE

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC δεν κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε κανένα από τα 20 προβλήματα. Επομένως, δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα όσον αφορά τον χρόνο εκτέλεσης και το πλήθος των κόμβων για τα παραπάνω προβλήματα.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	2-f24	232077	>3600	58.48	NONE
2	2-f25	449023	>3600	70.42	NONE
3	3-f10	143634	>3600	219.88	NONE
4	3-f11	192058	>3600	174.61	NONE
5	6-w2	59702	>3600	204.14	NONE
6	7-w1-f4	179517	>3600	80.14	NONE
7	7-w1-f5	700519	>3600	66.05	NONE
8	8-f10	239734	>3600	192.94	NONE
9	8-f11	186875	>3600	173.03	NONE
10	11	202238	>3600	541.00	NONE
11	14-f27	278212	>3600	93.03	NONE
12	14-f28	274848	>3600	51.30	NONE

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC δεν κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL)

σε κανένα από τα 20 προβλήματα. Επομένως, δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα όσον αφορά τον χρόνο εκτέλεσης και το πλήθος των κόμβων για τα παραπάνω προβλήματα.

4.4.2 GC (Graph Coloring)

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	anna-5	1178295	>3600	2.16	NONE
2	anna-8	1084832	>3600	2.26	NONE
3	david-5	1226913	>3600	1.62	NONE
4	david-8	1357654	>3600	1.64	NONE
5	games120-5	1036754	>3600	1.36	NONE
6	games120-7	910536	>3600	2.20	NONE
7	games120-9	120	2.97	2.61	SUCCESS
8	homer-5	291567	>3600	9.11	NONE
9	huck-5	1230494	>3600	0.83	NONE
10	huck-8	1441074	>3600	0.88	NONE
11	jean-5	1000233	>3600	0.61	NONE
12	jean-7	1137808	>3600	1.01	NONE
13	miles250-6	1163187	>3600	0.55	NONE
14	miles250-7	1092951	>3600	0.94	NONE
15	miles250-8	1689641	>3600	0.73	NONE
16	miles500-5	603514	>3600	5.09	NONE
17	miles750-5	411622	>3600	14.51	NONE
18	miles1000-5	241650	>3600	42.27	NONE
19	queen5-5-4	243	0.44	0.20	FAIL
20	queen6-6-6	1353038	>3600	0.67	NONE
21	queen7-7-6	963146	>3600	1.34	NONE
22	1-fullins-3-3	81	0.09	0.03	FAIL
23	1-fullins-4-4	1201908	>3600	1.00	NONE
24	1-insertions-4-3	1006922	2944.83	0.09	FAIL
25	2-fullins-3-4	1316051	>3600	0.22	NONE
26	2-fullins-4-4	561817	>3600	3.59	NONE
27	3-fullins-3-5	1326113	>3600	0.75	NONE
28	3-fullins-5-5	21124	>3600	408.08	NONE
29	4-fullins-3-6	1222893	>3600	1.30	NONE
M.O.	-	251842	737.08	0.73	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 4 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	anna-5	833212	>3600	2.22	NONE
2	anna-8	657720	>3600	3.14	NONE
3	david-5	1029776	>3600	2.44	NONE
4	david-8	1195722	>3600	1.83	NONE
5	games120-5	944141	>3600	1.39	NONE
6	games120-7	1137066	>3600	1.88	NONE
7	games120-9	663	3.12	2.41	SUCCESS
8	homer-5	255069	>3600	9.08	NONE
9	huck-5	1236212	>3600	1.03	NONE
10	huck-8	1480422	>3600	0.91	NONE
11	jean-5	1146006	>3600	0.61	NONE
12	jean-7	1215778	>3600	0.91	NONE
13	miles250-6	1089914	>3600	0.56	NONE
14	miles250-7	1050761	>3600	0.91	NONE
15	miles250-8	1502163	>3600	0.80	NONE
16	miles500-5	574792	>3600	5.36	NONE
17	miles750-5	440434	>3600	17.09	NONE
18	miles1000-5	255970	>3600	31.47	NONE
19	queen5-5-4	304	0.48	0.22	FAIL
20	queen6-6-6	1391265	>3600	0.69	NONE
21	queen7-7-6	1279958	>3600	1.52	NONE
22	1-fullins-3-3	103	0.12	0.05	FAIL
23	1-fullins-4-4	1092950	>3600	0.97	NONE
24	1-insertions-4-3	1061127	>3600	0.09	NONE
25	2-fullins-3-4	1235121	>3600	0.22	NONE
26	2-fullins-4-4	607187	>3600	3.69	NONE
27	3-fullins-3-5	1118881	>3600	0.88	NONE
28	3-fullins-5-5	23217	>3600	403.34	NONE
29	4-fullins-3-6	906636	>3600	1.09	NONE
M.O.	-	357	1.24	0.89	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 3 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	anna-5	1248995	>3600	2.19	NONE
2	anna-8	822471	>3600	2.30	NONE
3	david-5	1056832	>3600	2.12	NONE
4	david-8	1352584	>3600	1.70	NONE
5	games120-5	1138119	>3600	1.38	NONE
6	games120-7	1091172	>3600	2.09	NONE

7	games120-9	120	2.86	2.52	SUCCESS
8	homer-5	248680	>3600	8.83	NONE
9	huck-5	919883	>3600	1.0	NONE
10	huck-8	1061888	>3600	1.16	NONE
11	jean-5	946777	>3600	0.81	NONE
12	jean-7	1294674	>3600	0.84	NONE
13	miles250-6	1108872	>3600	0.55	NONE
14	miles250-7	1286400	>3600	0.95	NONE
15	miles250-8	1457401	>3600	0.72	NONE
16	miles500-5	658388	>3600	5.19	NONE
17	miles750-5	555127	>3600	14.55	NONE
18	miles1000-5	256074	>3600	31.59	NONE
19	queen5-5-4	210	0.41	0.22	FAIL
20	queen6-6-6	1244124	>3600	0.67	NONE
21	queen7-7-6	1150665	>3600	1.50	NONE
22	1-fullins-3-3	92	0.11	0.03	FAIL
23	1-fullins-4-4	881870	>3600	1.02	NONE
24	1-insertions-4-3	735956	>3600	0.14	NONE
25	2-fullins-3-4	1158763	>3600	0.31	NONE
26	2-fullins-4-4	543531	>3600	4.81	NONE
27	3-fullins-3-5	1109184	>3600	0.78	NONE
28	3-fullins-5-5	20217	>3600	415.27	NONE
29	4-fullins-3-6	1054762	>3600	1.09	NONE
M.O.	-	141	1.13	0.92	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) ξανά σε 3 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε ξανά τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=5 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=1 τις χειρότερες. Επιπλέον, για k=1 ο αλγόριθμος κατάφερε να επιλύσει ένα επιπλέον πρόβλημα σε σχέση με k=3 και k=5.

4.4.3 PIGEONS

•k = 1

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	5	40	0.01	0.00	FAIL
2	6	205	0.05	0.02	FAIL
3	7	1236	0.27	0.02	FAIL
4	8	8659	1.95	0.08	FAIL
5	9	69280	17.59	0.05	FAIL
6	10	623529	413.12	0.12	FAIL

7	11	1513219	>3600	0.20	NONE
8	12	1848229	>3600	0.47	NONE
9	13	2263828	>3600	0.33	NONE
10	14	2223930	>3600	0.53	NONE
11	15	2379070	>3600	0.47	NONE
12	16	2284923	>3600	0.94	NONE
13	18	2178985	>3600	2.59	NONE
14	20	2186275	>3600	4.16	NONE
15	25	2156751	>3600	8.64	NONE
16	30	1735568	>3600	23.09	NONE
17	35	1337031	>3600	58.44	NONE
18	40	1578544	>3600	78.66	NONE
19	45	1296872	>3600	108.22	NONE
20	50	1376297	>3600	211.41	NONE
M.O.	-	117158	72.17	0.05	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 6 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	5	40	0.02	0.02	FAIL
2	6	205	0.05	0.02	FAIL
3	7	1236	0.27	0.02	FAIL
4	8	8659	1.94	0.06	FAIL
5	9	69280	16.70	0.05	FAIL
6	10	623529	293.09	0.12	FAIL
7	11	2512452	>3600	0.12	NONE
8	12	1658541	>3600	0.47	NONE
9	13	2390731	>3600	0.33	NONE
10	14	1953302	>3600	0.77	NONE
11	15	1929292	>3600	0.69	NONE
12	16	2052728	>3600	1.23	NONE
13	18	2056496	>3600	3.55	NONE
14	20	2428750	>3600	5.19	NONE
15	25	2187540	>3600	8.22	NONE
16	30	2031368	>3600	23.50	NONE
17	35	1672589	>3600	43.56	NONE
18	40	1687193	>3600	97.98	NONE
19	45	1430739	>3600	116.86	NONE
20	50	1420061	>3600	207.36	NONE
M.O.	-	117158	52.01	0.05	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL)

σε 6 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. (σε δευτ.)	Αποτέλεσμα
1	5	40	0.02	0.00	FAIL
2	6	205	0.05	0.00	FAIL
3	7	1236	0.27	0.03	FAIL
4	8	8659	1.88	0.05	FAIL
5	9	69280	15.80	0.05	FAIL
6	10	623529	248.47	0.12	FAIL
7	11	2807395	>3600	0.12	NONE
8	12	2740208	>3600	0.28	NONE
9	13	1829811	>3600	0.36	NONE
10	14	2467259	>3600	0.73	NONE
11	15	2783832	>3600	0.47	NONE
12	16	2775100	>3600	0.95	NONE
13	18	2793123	>3600	2.55	NONE
14	20	2540657	>3600	3.97	NONE
15	25	2520347	>3600	9.23	NONE
16	30	2331138	>3600	22.72	NONE
17	35	2126515	>3600	54.92	NONE
18	40	1820203	>3600	85.11	NONE
19	45	1773724	>3600	117.56	NONE
20	50	1657043	>3600	205.23	NONE
M.O.	-	117158	44.42	0.04	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 6 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=5 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=1 τις χειρότερες ως προς τον χρόνο εκτέλεσης, καθώς και στις τρεις περιπτώσεις δημιουργείται ο ίδιος αριθμός κόμβων.

4.5 Σχολιασμός Αποτελεσμάτων

Σύμφωνα με τα παραπάνω αποτελέσματα, παρατηρούμε ότι για το σύνολο των περιπτώσεων, το heuristic dom/wdeg είναι εκείνο το οποίο καταφέρνει να επιλύσει τον μεγαλύτερο αριθμό προβλημάτων σε σχέση με τα υπόλοιπα 3. Αντίθετα, το heuristic IBS παρουσιάζει την χειρότερη επίδοση ως προς τον αριθμό των προβλημάτων τα οποία κατάφερε να επιλύσει. Όσον αφορά τα heuristics dom/ddeg

και ABS παρατηρείται σχεδόν παραπλήσια επίδοση ως προς τον αριθμό των προβλημάτων που επιλύουν.

Μελετώντας τους μέσους όρους για τους αριθμούς των κόμβων που δημιουργήθηκαν κατά την εκτέλεση των παραπάνω πειραμάτων, παρατηρήθηκαν τα εξής:

- Στα **RLFAP** προβλήματα, το dom/ddeg είναι ο μηχανισμός εκείνος ο οποίος επιτυγχάνει τις βέλτιστες επιδόσεις, καθώς και το μικρότερο εύρος σε σχέση με τους υπόλοιπους μηχανισμούς. Σημαντικό ωστόσο να αναφερθεί είναι το γεγονός πως το dom/wdeg είναι ο μηχανισμός ο οποίος επιτυγχάνει την επίλυση του μεγαλύτερου αριθμού προβλημάτων. Όσον αφορά τον μηχανισμό ABS, παρατηρούμε ότι καταφέρνει να επιλύσει προβλήματα, δημιουργώντας έναν ελάχιστο αριθμό κόμβων παρά το γεγονός ότι δεν θεωρείται ο αποδοτικότερος μηχανισμός ως προς τον αριθμό των κόμβων τους οποίους δημιουργεί. Αυτό οφείλεται στο γεγονός ότι ο μηχανισμός ABS επωφελείται της διαδικασίας των δοκιμών (probing) που περιλαμβάνει καθώς μπορεί να οδηγήσει σε απευθείας επίλυση ενός προβλήματος. Τέλος, σε αυτή την κατηγορία προβλημάτων, ο μηχανισμός IBS δεν κατάφερε να επιφέρει κάποιο αποτέλεσμα.
- Στα **GC** προβλήματα, το dom/wdeg είναι ο μηχανισμός εκείνος ο οποίος παρουσιάζει τις καλύτερες επιδόσεις ως προς τον αριθμό των κόμβων που δημιουργεί καθώς και το μικρότερο εύρος, ενώ είναι και ο μηχανισμός εκείνος ο οποίος επιλύει τα περισσότερα προβλήματα. Σχετικά με τον μηχανισμό ABS, παρατηρούμε ότι οι επιδόσεις του δεν είναι οι βέλτιστες, καθώς και ότι ο αριθμός των κόμβων που δημιουργεί αυξομειώνεται για τα διάφορα k, συγκλίνοντας όμως γύρω από ένα συγκεκριμένο εύρος. Τέλος, είναι εύκολο κάποιος να παρατηρήσει ότι ο μηχανισμός IBS επιτυγχάνει την δημιουργία ενός ελαχίστου αριθμού κόμβων, ωστόσο δεν θα πρέπει να θεωρηθεί ως ο πιο αποδοτικός μηχανισμός, καθώς καταφέρνει να επιλύσει τα λιγότερα με διαφορά προβλήματα σε σχέση με τους υπόλοιπους.
- Στα **PIGEONS** προβλήματα, παρατηρούμε ότι ο αριθμός των κόμβων που δημιουργείται σε όλα τα heuristics παραμένει αμετάβλητος ανεξαρτήτως k. Πιο συγκεκριμένα, για τα τρία πρώτα heuristics ο αριθμός αυτός παραμένει ο ίδιος, ενώ για το IBS heuristic μειώνεται σημαντικά, λαμβάνοντας φυσικά υπόψη το γεγονός ο μηχανισμός επιτυγχάνει την επίλυση ενός λιγότερου προβλήματος σε σχέση με τους υπόλοιπους τρεις.

Επίσης, θα πρέπει να αναφερθεί ότι όσον αφορά τους χρόνους επίλυσης και το πλήθος των παραχθέντων κόμβων, είναι αδύνατο να προκύψει κάποιο ‘’ασφαλές’’ γενικότερο συμπέρασμα, καθώς οι μέσοι όροι των παραπάνω μετρικών δεν αναφέρονται πάντα στο ίδιο σύνολο επιλυμένων προβλημάτων.

Από τις παραπάνω παρατηρήσεις, είναι φανερό ότι η ευριστική συνάρτηση IBS παρουσίασε τις χαμηλότερες επιδόσεις συγκριτικά με τα υπόλοιπα τρία heuristics. Οι

λόγοι για τους οποίους το συγκεκριμένο heuristic παρουσίασε τη συγκεκριμένη συμπεριφορά είναι η πολυπλοκότητα των υπολογισμών που εκτελούνται στο συγκεκριμένο μηχανισμό, η οποία αυξάνει τον χρόνο εκτέλεσης της αναζήτησης, καθώς επίσης και η αστοχία η οποία είναι πιθανόν να προέκυψε κατά την αρχικοποίηση των impacts των μεταβλητών και συγκεκριμένα κατά την διαίρεση των πεδίων τιμών των μεταβλητών σε μικρότερα υπό-πεδία. Όπως αναφέρεται και στο [13], ο αριθμός διάσπασης ενός πεδίου τιμών σε υπό-πεδία (συντελεστής s) δεν είναι σταθερός αλλά προσεγγιστικός. Στην παρούσα διπλωματική, ο συντελεστής αυτός υπολογίζεται αυτόματα ανάλογα με το μέγεθος του πεδίου τιμών της κάθε μεταβλητής. Συνεπώς, οι παραπάνω παράγοντες συνέβαλαν καθοριστικά στην αποτυχία του IBS.

4.6 Αποτελέσματα Common Choice (Συνδυαστικό Heuristic)

Τα αποτελέσματα τα οποία προέκυψαν από την εκτέλεση των πειραμάτων για το Common Choise (Συνδιαστικό) heuristic για τις παραπάνω κατηγορίες προβλημάτων, παρουσιάζονται στη συνέχεια σε μορφή πινάκων.

4.6.1 RLFAP (Radio Link Frequency Assignment Problem)

•k = 1 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	2-f24	289820	>3600	15,48	56,22	NONE
2	2-f25	640543	>3600	14,70	64,25	NONE
3	3-f10	219716	>3600	38,06	147,84	NONE
4	3-f11	221100	>3600	25,08	153,34	NONE
5	6-w2	260594	>3600	6,38	165,59	NONE
6	7-w1-f4	446	93,48	6,61	87,92	SUCCESS
7	7-w1-f5	468157	>3600	3,51	61,36	NONE
8	8-f10	202598	>3600	31,11	176,89	NONE
9	8-f11	184152	>3600	17,78	161,99	NONE
10	11	329651	>3600	255,02	491,42	NONE
11	14-f27	278496	>3600	9,34	79.41	NONE
12	14-f28	230407	>3600	4,55	52,26	NONE
M.O.	-	446	93,48	6,61	87,92	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε μόνο 1 από τα 12 προβλήματα. Στο πρόβλημα 7-w1-f4 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	2-f24	598	75,09	13,33	56,94	SUCCESS
2	2-f25	470213	>3600	12,55	50,66	NONE
3	3-f10	447	244,97	48,12	184,08	SUCCESS
4	3-f11	418558	>3600	30,80	195,06	NONE
5	6-w2	10	180,25	8,22	168,00	FAIL
6	7-w1-f4	433	92,91	4,58	79,61	SUCCESS
7	7-w1-f5	479818	>3600	2,81	64,44	NONE
8	8-f10	263848	>3600	32,00	169,03	NONE
9	8-f11	113318	>3600	20,25	163,59	NONE
10	11	680	861,91	288,44	539,92	SUCCESS
11	14-f27	282288	>3600	12,89	92,88	NONE
12	14-f28	291144	>3600	5,16	49,94	NONE
M.O.	-	434	291,03	62,54	205,71	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 5 από τα 12 προβλήματα. Στο πρόβλημα 2-f24 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο πρόβλημα 6-w2 τον μικρότερο αριθμό κόμβων.

•k = 5 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	2-f24	200	6,41	5.73	1,00	SUCCESS
2	2-f25	463988	>3600	12,47	50,92	NONE
3	3-f10	882	259,14	47,16	189,31	SUCCESS
4	3-f11	205560	>3600	40,5	199,25	NONE
5	6-w2	12	177,91	10,45	163,27	FAIL
6	7-w1-f4	424	98,27	5,03	83,92	SUCCESS
7	7-w1-f5	4	77,53	5,17	68,11	FAIL
8	8-f10	214963	>3600	36,08	206,78	NONE
9	8-f11	11557	558,16	11,36	164,00	FAIL
10	11	34305	2474,69	269,00	542,08	SUCCESS
11	14-f27	325191	>3600	11,12	84,6	NONE
12	14-f28	247377	>3600	5,00	57,45	NONE
M.O.	-	6769	521,73	50,56	173,10	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για κ=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 7 από τα 12 προβλήματα. Στο πρόβλημα 2-f-24 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο πρόβλημα 7-w1-f5 τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=1 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις ως προς τον χρόνο εκτέλεσης, ενώ για k=5 τις χειρότερες. Επιπλέον, για k=3 ο αλγόριθμος παρουσιάζει τις καλύτερες επιδόσεις ως προς τον αριθμό των κόμβων που δημιουργούνται, ενώ για k=5 τις χειρότερες. Ωστόσο για k=5, ο αλγόριθμος καταφέρνει να καταλήξει σε κάποιο συμπέρασμα σε μεγαλύτερο αριθμό προβλημάτων.

4.6.2 GC (Graph Coloring)

•k = 1 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	anna-5	1219	6.74	1,03	2,23	FAIL
2	anna-8	956248	>3600	3,11	2,27	NONE
3	david-5	844	4,25	0,95	1,67	FAIL
4	david-8	1129308	>3600	2,72	1,70	NONE
5	games120-5	897382	>3600	1,20	1,39	NONE
6	games120-7	1067828	>3600	3,22	1,89	NONE
7	games120-9	120	0,34	0,27	1,00	SUCCESS
8	homer-5	22151	142.53	3,17	9,42	FAIL
9	huck-5	2964	5,75	0,69	0,88	FAIL
10	huck-8	1577618	>3600	1,91	0,91	NONE
11	jean-5	4468	6,47	0,58	0,62	FAIL
12	jean-7	1285126	>3600	1,22	0,81	NONE
13	miles250-6	1141667	>3600	1,20	0,55	NONE
14	miles250-7	874529	>3600	1,64	0,94	NONE
15	miles250-8	128	0,34	0,30	1,00	SUCCESS
16	miles500-5	614892	>3600	2,50	5,42	NONE
17	miles750-5	20437	217,41	4,88	15,76	FAIL
18	miles1000-5	1966	75,62	8,59	34,45	FAIL
19	queen5-5-4	66	0,59	0,24	0,22	FAIL
20	queen6-6-6	73364	234,14	1,25	0,78	FAIL
21	queen7-7-6	8735	56,56	1,86	1,52	FAIL
22	1-fullins-3-3	9	0,19	0,11	0,03	FAIL
23	1-fullins-4-4	579554	1698,52	0,83	0,98	FAIL
24	1-insertions-4-3	1050	1,72	0,25	0,09	FAIL
25	2-fullins-3-4	20157	17,99	0,31	0,20	FAIL
26	2-fullins-4-4	35510	186,52	1,92	3,70	FAIL
27	3-fullins-3-5	1236611	>3600	0,89	0,75	NONE
28	3-fullins-5-5	51980	>3600	41,56	397.36	NONE
29	4-fullins-3-6	1082529	>3600	2,08	1,11	NONE
M.O.	-	45455	156,22	1,60	4,39	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL)

σε 17 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	anna-5	205	4,53	0,97	2,23	FAIL
2	anna-8	20948	1374,33	3,17	2,35	FAIL
3	david-5	205	3,56	0,89	1,67	FAIL
4	david-8	121249	549,78	2,72	1,75	FAIL
5	games120-5	615	4,12	1,16	1,38	FAIL
6	games120-7	914476	>3600	3,97	2,33	NONE
7	games120-9	120	0,56	0,50	1,00	SUCCESS
8	homer-5	205	15,88	2,86	9,30	FAIL
9	huck-5	205	2,17	0,67	0,88	FAIL
10	huck-8	93636	248,44	1,97	0,91	FAIL
11	jean-5	213	1,75	0,53	0,62	FAIL
12	jean-7	90883	163,73	1,23	0,84	FAIL
13	miles250-6	1236	4,67	1,20	0,58	FAIL
14	miles250-7	8994	28,08	1,67	0,95	FAIL
15	miles250-8	128	0,53	0.50	1,00	SUCCESS
16	miles500-5	5029	33,19	2,23	5,11	FAIL
17	miles750-5	3474	76,20	4,80	15,39	FAIL
18	miles1000-5	740	64,73	8,53	33,66	FAIL
19	queen5-5-4	40	0,55	0,23	0,22	FAIL
20	queen6-6-6	57424	171,33	1,16	0,72	FAIL
21	queen7-7-6	2520	17,14	1,77	1,41	FAIL
22	1-fullins-3-3	68	0,20	0,11	0,03	FAIL
23	1-fullins-4-4	72908	165,92	0,84	0,97	FAIL
24	1-insertions-4-3	658	1,22	0,22	0,11	FAIL
25	2-fullins-3-4	1331	2,33	0,31	0,20	FAIL
26	2-fullins-4-4	3392	30,20	2,08	3,80	FAIL
27	3-fullins-3-5	40298	82,20	0,94	0,78	FAIL
28	3-fullins-5-5	45977	>3600	45,19	413,19	NONE
29	4-fullins-3-6	1019175	>3600	1,97	1,06	NONE
M.O.	-	20259	117,21	1,66	3,38	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 26 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο πρόβλημα queen5-5-4 τον μικρότερο αριθμό κόμβων.

•k = 5 probes = 25 $\gamma = 0.999$

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	anna-5	208	4,61	1,00	2,19	FAIL
2	anna-8	69424	482,62	3,19	2,30	FAIL
3	david-5	205	3,77	0,89	1,67	FAIL
4	david-8	69364	359,12	2,67	1,70	FAIL
5	games120-5	1333	5,94	1,24	1,39	FAIL
6	games120-7	1131052	>3600	3,33	1,88	NONE
7	games120-9	120	0,31	0,23	1,00	SUCCESS
8	homer-5	205	15,61	2,77	9,19	FAIL
9	huck-5	215	2,31	0,70	0,86	FAIL
10	huck-8	82903	339,06	1,89	0,89	FAIL
11	jean-5	215	1,89	0,58	0,62	FAIL
12	jean-7	56310	111,17	1,20	0,83	FAIL
13	miles250-6	1239	4,70	1,19	0,56	FAIL
14	miles250-7	16372	44,08	1,64	0,94	FAIL
15	miles250-8	128	0,45	0,41	1,00	SUCCESS
16	miles500-5	205	9,55	2,30	5,03	FAIL
17	miles750-5	205	25,30	4,55	15,19	FAIL
18	miles1000-5	259	51,50	7,91	32,84	FAIL
19	queen5-5-4	40	0,59	0,25	0,22	FAIL
20	queen6-6-6	57247	178,16	1,17	0,70	FAIL
21	queen7-7-6	2127	15,89	1,80	1,41	FAIL
22	1-fullins-3-3	75	0,20	0,08	0,05	FAIL
23	1-fullins-4-4	52337	15,38	0,84	0,98	FAIL
24	1-insertions-4-3	541	1,01	0,22	0,09	FAIL
25	2-fullins-3-4	911	1,59	0,30	0,20	FAIL
26	2-fullins-4-4	15792	82,80	1,83	3,52	FAIL
27	3-fullins-3-5	47111	78,24	0,86	0,73	FAIL
28	3-fullins-5-5	26523	>3600	42,36	390,52	NONE
29	4-fullins-3-6	18273	>3600	2,08	1,20	NONE
Μ.Ο.	-	28403	70,61	1,70	22,54	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 26 από τα 29 προβλήματα. Στο πρόβλημα 1-fullins-3-3 πέτυχε τον μικρότερο χρόνο εκτέλεσης ενώ στο πρόβλημα queen5-5-4 τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=5 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις ως προς τον χρόνο εκτέλεσης, ενώ για k=1 τις χειρότερες. Επιπλέον, για k=3 ο αλγόριθμος παρουσιάζει τις καλύτερες επιδόσεις ως προς τον αριθμό των κόμβων που δημιουργούνται, ενώ για k=1 τις χειρότερες. Επίσης, για k=3 και k=5, ο αλγόριθμος καταφέρνει να καταλήξει σε κάποιο συμπέρασμα σε μεγαλύτερο αριθμό προβλημάτων σε σχέση με k=1.

4.6.3 PIGEONS

•k = 1 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	5	40	0,03	0,02	0,01	FAIL
2	6	205	0,11	0,05	0,02	FAIL
3	7	1236	0,34	0,08	0,01	FAIL
4	8	8659	2,09	0,12	0,05	FAIL
5	9	69280	17,76	0,20	0,05	FAIL
6	10	623529	284,84	0,30	0,12	FAIL
7	11	2154667	>3600	0,47	0,14	NONE
8	12	2309745	>3600	0,64	0,28	NONE
9	13	2131643	>3600	1,03	0,30	NONE
10	14	2142153	>3600	1,56	0,67	NONE
11	15	2463322	>3600	1,66	0,47	NONE
12	16	2366341	>3600	2,19	0,92	NONE
13	18	2232781	>3600	3,86	2,75	NONE
14	20	2258937	>3600	5,41	4,00	NONE
15	25	2241718	>3600	15,77	9,44	NONE
16	30	2081921	>3600	30,16	22,84	NONE
17	35	1879701	>3600	59,94	45,41	NONE
18	40	1653366	>3600	102,81	79,26	NONE
19	45	1476009	>3600	198,06	120,11	NONE
20	50	1377954	>3600	296,50	228,36	NONE
M.O.	-	117158	50,86	0,13	0,03	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=1 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 6 από τα 20 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 3 probes = 25 γ = 0.999

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	5	40	0,08	0,06	0,00	FAIL
2	6	205	0,09	0,03	0,02	FAIL
3	7	1236	0,34	0,06	0,03	FAIL
4	8	8659	2,09	0,11	0,05	FAIL
5	9	69280	17,55	0,19	0,05	FAIL
6	10	623529	286,16	0,30	0,12	FAIL
7	11	2644007	>3600	0,45	0,12	NONE
8	12	2688091	>3600	0,66	0,26	NONE
9	13	2612141	>3600	0,92	0,25	NONE

10	14	2467174	>3600	1,34	0,55	NONE
11	15	2615039	>3600	1,64	0,48	NONE
12	16	2562368	>3600	2,28	0,92	NONE
13	18	2487244	>3600	3,47	2,52	NONE
14	20	2073496	>3600	5,81	4,53	NONE
15	25	1993260	>3600	19,02	11,38	NONE
16	30	2187574	>3600	30,14	22,53	NONE
17	35	2035695	>3600	61,83	48,00	NONE
18	40	1975658	>3600	101,12	76,73	NONE
19	45	1699898	>3600	180,59	108,70	NONE
20	50	1592583	>3600	265,67	203,70	NONE
M.O.	-	117158	51,06	0,13	0,05	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=3 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL) σε 28 από τα 29 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

•k = 5 probes = 25 $\gamma = 0.999$

#	Πρόβλημα	Κόμβοι	Χρόνος (σε δευτ.)	Χρ. Αρχικ. Activity (σε δευτ.)	Χρ. Αρχικ. Impact(σε δευτ.)	Αποτέλεσμα
1	5	40	0,08	0,06	0,0	FAIL
2	6	205	0,14	0,08	0,02	FAIL
3	7	1236	0,34	0,06	0,03	FAIL
4	8	8659	2,11	0,12	0,05	FAIL
5	9	69280	17,95	0,19	0,06	FAIL
6	10	623529	271,89	0,31	0,12	FAIL
7	11	2523672	>3600	0,47	0,12	NONE
8	12	2534003	>3600	0,83	0,36	NONE
9	13	2882326	>3600	0,91	0,25	NONE
10	14	2833359	>3600	1,22	0,53	NONE
11	15	2845586	>3600	1,62	0,47	NONE
12	16	2587233	>3600	2,17	0,92	NONE
13	18	2102175	>3600	4,51	3,30	NONE
14	20	2501804	>3600	6,92	5,12	NONE
15	25	2537314	>3600	13,94	8,23	NONE
16	30	2249370	>3600	30,23	22,86	NONE
17	35	2069391	>3600	68,52	52,62	NONE
18	40	1946693	>3600	102,34	78,33	NONE
19	45	1772662	>3600	170,48	105,14	NONE
20	50	1507183	>3600	268,16	217,48	NONE
M.O.	-	117158	48,76	0,14	0,47	-

Για τα παραπάνω προβλήματα, παρατηρούμε ότι για k=5 ο αλγόριθμος MAC κατάφερε να καταλήξει σε κάποιο συμπέρασμα (λύση-SUCCESS ή αποτυχία-FAIL)

σε 28 από τα 29 προβλήματα. Στο πρόβλημα 5 πέτυχε τον μικρότερο χρόνο εκτέλεσης καθώς και τον μικρότερο αριθμό κόμβων.

Από τους μέσους όρους των χρόνων εκτέλεσης και κόμβων, προκύπτει ότι για k=5 ο αλγόριθμος, παρουσιάζει τις καλύτερες επιδόσεις, ενώ για k=3 τις χειρότερες ως προς τον χρόνο εκτέλεσης, καθώς και στις τρεις περιπτώσεις δημιουργείται ο ίδιος αριθμός κόμβων.

4.6.4 Πίνακες Συχνοτήτων Επιλογής Μεταβλητής

Κατά την εκτέλεση του Συνδυαστικού heuristic, εκτός από τον χρόνο εκτέλεσης και το σύνολο των κόμβων που δημιουργήθηκαν κατά την προσπάθεια επίλυσης των προβλημάτων τα οποία προαναφέρθηκαν στην παρούσα διπλωματική, εξετάσαμε τη συχνότητα με την οποία γίνεται η επιλογή της επόμενης μεταβλητής όταν αυτή εμφανίζεται από 1, 2, 3 και 4 heuristics. Οι πίνακες οι οποίοι ακλουθούν παρουσιάζουν τις αντίστοιχες συχνότητες σε ποσοστό επί τοις 100 (%). Θα πρέπει επίσης να αναφέρουμε το γεγονός ότι στους παρακάτω πίνακες, δεν περιλαμβάνονται οι συχνότητες των προβλημάτων στα οποία ο αλγόριθμος MAC δεν κατάφερε να εξάγει κάποιο συμπέρασμα, καθώς και των προβλημάτων τα οποία επιλύθηκαν μέσω του heuristic ABS κατά την διαδικασία του probing (όπου η λύση του προβλήματος, προέκυψε με τυχαία επιλογή μεταβλητών).

Αναλυτικότερα, η μορφή των πινάκων ακολουθεί την εξής δομή:

- $1^{η}$ Στήλη: Στη στήλη αυτή αναγράφεται ο αύξων αριθμός του προβλήματος.
- $2^{η}$ Στήλη: Στη στήλη αυτή αναγράφεται το όνομα του κάθε προβλήματος.
- $3^{η}$ Στήλη: Στη στήλη αυτή αναγράφεται η συχνότητα (%) με την οποία το Συνδυαστικό heuristic, επιλέγει μεταβλητή από ένα μόνο heuristic.
- $4^{η}$ Στήλη: Στη στήλη αυτή αναγράφεται η συχνότητα (%) με την οποία το Συνδυαστικό heuristic, επιλέγει εκείνη την μεταβλητή η οποία εμφανίζεται κοινή μόνο σε δύο heuristics.
- $5^{η}$ Στήλη: Στη στήλη αυτή αναγράφεται η συχνότητα (%) με την οποία το Συνδυαστικό heuristic, επιλέγει εκείνη την μεταβλητή η οποία εμφανίζεται κοινή μόνο σε τρία heuristics.
- $6^{η}$ Στήλη: Στη στήλη αυτή αναγράφεται η συχνότητα (%) με την οποία το Συνδυαστικό heuristic, επιλέγει εκείνη την μεταβλητή η οποία εμφανίζεται κοινή και στα τέσσερα heuristics.

Τέλος, οι πίνακες εμφανίζονται με την ίδια σειρά όπως στις προηγούμενες ενότητες.

RLFAP (Radio Link Frequency Assignment Problem)

- k=1

#	Πρόβλημα	1	2	3	4
1	7-w1-f4	43,53 %	45,28 %	10,44 %	0,75 %

Για k=1, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές σε δύο μόνο heuristics.

- k=3

#	Πρόβλημα	1	2	3	4
1	2-f24	1,04 %	36,45 %	59,03 %	3,47 %
2	3-f10	0,0 %	61,44 %	35,18 %	3,34 %
3	6-w2	0,0 %	100 %	0,0 %	0,0 %
4	7-w1-f4	17,75 %	44,0 %	27,25 %	4,0 %
5	11	0,0 %	52,65 %	45,3 %	2,06 %

Για k=3, σε 4 από τα 5 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές σε δύο μόνο heuristics. Ενώ σημαντικό είναι να αναφέρουμε το γεγονός ότι σε 1 από τα 4 αυτά προβλήματα το ποσοστό αυτό έφτασε το 100%.

- k=5

#	Πρόβλημα	1	2	3	4
1	3-f10	0,34 %	22,53 %	73,28 %	3,84 %
2	7-w1-f4	19,25 %	39,5 %	35,0 %	6,25 %
3	8-f11	3,99 %	62,52 %	33,46 %	0,0 %
4	11	0,0 %	13,36 %	86,45 %	0,19 %

Για k=5, σε 2 από τα 4 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές σε δύο μόνο heuristics, ενώ στα υπόλοιπα 2, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές σε τρία μόνο heuristics. Σημαντικό να αναφερθεί είναι το γεγονός ότι στην περίπτωση όπου η επιλογή των μεταβλητών γίνεται από τις κοινές ανάμεσα σε τρία heuristics τα ποσοστά είναι μεγαλύτερα.

Για k=1 και k=3 παρατηρούμε ότι τα μεγαλύτερα ποσοστά επιλογής της επόμενης μεταβλητής εμφανίζονται σε μία από τις τέσσερεις παραπάνω περιπτώσεις (περίπτωση 2), ενώ για κ=5 σε δύο από τις τέσσερεις (περίπτωση 2 και περίπτωση 3).

GC (Graph Coloring)

- k=1

#	Πρόβλημα	1	2	3	4
1	anna-5	73,66 %	23,34 %	3,0 %	0,0 %
2	david-5	67,98 %	29,3 %	2,72 %	0,0 %
3	homer-5	87,59 %	12,25 %	0,16 %	0,0 %
4	huck-5	73,37 %	24,26 %	2,37 %	0,0 %
5	jean-5	78,7 %	21,19 %	0.11 %	0,0 %
6	miles750-5	94,77 %	5,17 %	0.06 %	0,0 %
7	miles1000-5	97,41 %	2,59 %	0,0 %	0,0 %
8	queen5-5-4	48,14 %	51,86 %	0,0 %	0,0 %
9	queen6-6-6	41,26 %	41,12 %	17,56 %	0.06 %
10	queen7-7-6	47,97 %	46,85 %	5,18 %	0,0 %
11	1-fullins-3-3	25,0 %	75,0 %	0,0 %	0,0 %
12	1-fullins-4-4	89,98 %	7,75 %	0,25 %	0,0 %
13	1-insertions-4-3	75,87 %	21,12 %	3,02 %	0,0 %
14	2-fullins-3-4	73,15 %	25,4 %	1,45 %	0,0 %
15	2-fullins-4-4	82,33 %	16,7 %	0,97 %	0,0 %

Για k=1, σε 14 από τα 15 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται μόνο σε ένα heuristic (τυχαία επιλογή). Ενώ σημαντικό είναι να αναφέρουμε το γεγονός ότι στην περίπτωση όπου η επιλογή μεταβλητής γίνεται από τις κοινές ανάμεσα σε 4 heuristics, στο σύνολο των προβλημάτων το ποσοστό εμφάνισης αγγίζει το 0,0%.

- k=3

#	Πρόβλημα	1	2	3	4
1	anna-5	6,98 %	47,67 %	45,35 %	0,0 %
2	anna-8	0,59 %	56,92 %	42,49 %	0,0 %
3	david-5	0,0 %	75,58 %	24,42 %	0,0 %
4	david-8	0,01%	48,11 %	51.88 %	0,0 %
5	games120-5	0,0 %	23,05 %	76,95 %	0,0 %
6	homer-5	0,0 %	81,4 %	18,6 %	0,0 %
7	huck-5	0,0 %	23,26 %	76,74 %	0,0 %
8	huck-8	0,2 %	56,86 %	42,94 %	0,0 %
9	jean-5	0,0 %	38,89 %	61,11 %	0,0 %
10	jean-7	0,62 %	57,65 %	41,73 %	0,0 %
11	miles250-6	0,0 %	21,28 %	78,72 %	0,0 %
12	miles250-7	0,0 %	14,61 %	85,39 %	0,0 %
13	miles500-5	57,37 %	41,74 %	0,89 %	0,0 %
14	miles750-5	48,99 %	49,56 %	1,45 %	0,0 %
15	miles1000-5	40,73 %	54,3 %	4,97 %	0,0 %
16	queen5-5-4	0,0 %	17,65 %	82,35 %	0,0 %
17	queen6-6-6	2,01 %	48,78 %	48,4 %	0,81 %
18	queen7-7-6	1,7 %	37,48 %	60,82 %	0,0 %

19	1-fullins-3-3	9,52 %	90,48 %	0,0 %	0,0 %
20	1-fullins-4-4	62,39 %	34,61 %	2,98 %	0,0 %
21	1-insertions-4-3	25,3 %	56,57 %	18,14 %	0,0 %
22	2-fullins-3-4	6,3 %	66,01 %	27,69 %	0,0 %
23	2-fullins-4-4	28,42 %	46,03 %	25,54 %	0,0 %
24	3-fullins-3-5	3,77 %	46,69 %	49,53 %	0,0 %

Για k=3, σε 13 από τα 24 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές μεταξύ δύο heuristics, σε 9 από τα 24 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές μεταξύ τριών heuristics και σε 2 από τα 24 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες δεν εμφανίζονται κοινές μεταξύ των heuristics, άρα η επιλογή τους γίνεται τυχαία . Ενώ σημαντικό είναι να αναφέρουμε το γεγονός ότι στην περίπτωση όπου η επιλογή μεταβλητής γίνεται από τις κοινές ανάμεσα σε 4 heuristics, στην πλειοψηφία των προβλημάτων το ποσοστό εμφάνισης αγγίζει το 0,0%.

- k=5

#	**Πρόβλημα**	**1**	**2**	**3**	**4**
1	anna-5	0,0 %	19,54 %	80,46 %	0,0 %
2	anna-8	0,0 %	4,94 %	95,06 %	0,0 %
3	david-5	0,0 %	16,28 %	83,72 %	0,0 %
4	david-8	0,0 %	0,81 %	99,19 %	0,0 %
5	games120-5	0,0 %	3,58 %	96,42 %	0,0 %
6	homer-5	0,0 %	26,74 %	73,26 %	0,0 %
7	huck-5	0,0 %	47,78 %	52,22 %	0,0 %
8	huck-8	0,0 %	4,12 %	95,88 %	0,0 %
9	jean-5	0,0 %	15,56 %	84,44 %	0,0 %
10	jean-7	0,0 %	6,65 %	93,35 %	0,0 %
11	miles250-6	0,0 %	0,58 %	99,42 %	0,0 %
12	miles250-7	0,0 %	0,5 %	99,5 %	0,0 %
13	miles500-5	0,0 %	87,21 %	12,79 %	0,0 %
14	miles750-5	0,0 %	39,53 %	60,47 %	0,0 %
15	miles1000-5	9,25 %	78,71 %	12,04 %	0,0 %
16	queen5-5-4	0,0 %	0,0 %	100,0 %	0,0 %
17	queen6-6-6	0,01 %	19,66 %	77,04 %	3,29 %
18	queen7-7-6	0,0 %	13,78 %	86,22 %	0,0 %
19	1-fullins-3-3	0,0 %	73,47 %	26,53 %	0,0 %
20	1-fullins-4-4	26,55 %	57,0 %	16,45 %	0,0 %
21	1-insertions-4-3	4,35 %	61,45 %	34,2 %	0,0 %
22	2-fullins-3-4	0,44 %	28,98 %	70,58 %	0,0 %
23	2-fullins-4-4	1,09 %	73,05 %	25,86 %	0,0 %
24	3-fullins-3-5	0,23 %	21,5 %	78,27 %	0,0 %
25	3-fullins-5-5	1,12 %	20,36 %	78,52 %	0,0 %
26	4-fullins-3-6	0,04 %	40,26 %	59,7 %	0,0 %

Για k=5, σε 20 από τα 26 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές μεταξύ τριών heuristics, ενώ σε 6 από τα 26 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές μεταξύ δύο heuristics. Ενώ σημαντικό είναι να αναφέρουμε το γεγονός ότι στην περίπτωση όπου η επιλογή μεταβλητής γίνεται από τις κοινές ανάμεσα σε 4 heuristics, στην πλειοψηφία των προβλημάτων το ποσοστό εμφάνισης αγγίζει το 0,0%.

Για k=3 και k=5 παρατηρούμε ότι τα μεγαλύτερα ποσοστά επιλογής της επόμενης μεταβλητής εμφανίζονται σε δύο από τις τέσσερεις παραπάνω περιπτώσεις (περίπτωση 2 και περίπτωση 3), ενώ για k=1 σε μία από τις τέσσερεις (περίπτωση 1).

PIGEONS

- k=1

#	Πρόβλημα	1	2	3	4
1	5	5,88 %	64,7 %	29,4 %	0,0 %
2	6	9,31 %	76,74 %	12,79 %	1,16 %
3	7	7,15 %	10,45 %	82,01 %	0,39 %
4	8	8,98 %	82,44 %	8,57 %	0,03 %
5	9	7,48 %	86,59 %	5,93 %	0,01 %
6	10	3,14 %	75,58 %	20,78 %	0,5 %

Για k=1, σε 5 από τα 6 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές σε δύο μόνο heuristics, σε 1 από τα 6 προβλήματα, το μεγαλύτερο ποσοστό εμφανίζεται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές σε τρία μόνο heuristics. Σημαντικό να αναφερθεί είναι το γεγονός ότι κατά πλειοψηφία, στην περίπτωση όπου η επιλογή των μεταβλητών γίνεται από τις κοινές ανάμεσα σε δύο heuristics τα ποσοστά είναι μεγαλύτερα.

- k=3

#	Πρόβλημα	1	2	3	4
1	5	0,0 %	0,0 %	5,88 %	94,12 %
2	6	0,0 %	0,0 %	9,31 %	90,7 %
3	7	0,0 %	0,0 %	11,22 %	88.78 %
4	8	0,0 %	0,28 %	11,18 %	88,54 %
5	9	0,0 %	0,2 %	10,46 %	89,35 %
6	10	0,0 %	0,15 %	10,09 %	89,77 %

Για k=3, παρατηρούμε ότι στο σύνολο των προβλημάτων τα μεγαλύτερα ποσοστά εμφανίζονται στην επιλογή μεταβλητών οι οποίες εμφανίζονται κοινές και στα τέσσερα heuristics.

- k=5

#	Πρόβλημα	1	2	3	4
1	5	0,0 %	0,0 %	0,0 %	100 %
2	6	0,0 %	0,0 %	0,0 %	100 %
3	7	0,0 %	0,0 %	0,0 %	100 %
4	8	0,0 %	0,0 %	0,0 %	100 %
5	9	0,0 %	0,0 %	0,02 %	99,98 %
6	10	0,0 %	0,0 %	0,0 %	100 %

Για k=5, και στα 6 προβλήματα, παρατηρούμε ότι τα ποσοστά επιλογής της επόμενης μεταβλητής εμφανίζονται κατά 100% στην περίπτωση όπου η επιλογή μεταβλητής γίνεται από τις κοινές ανάμεσα και στα τέσσερα heuristics.

Για k=3 και k=5 παρατηρούμε ότι τα μεγαλύτερα ποσοστά επιλογής της επόμενης μεταβλητής εμφανίζονται σε μία από τις τέσσερεις παραπάνω περιπτώσεις (περίπτωση 4), ενώ για k=1 σε δύο από τις τέσσερεις (περίπτωση 2 και περίπτωση 3).

4.6.5 Σχολιασμός Αποτελεσμάτων Common Choice

Παρατηρώντας τους πίνακες των αποτελεσμάτων του Συνδυαστικού heuristic (Common Choice), εύκολα μπορεί κανείς να διαπιστώσει πως η αύξηση του συντελεστή k, οδηγεί σε επίλυση μεγαλύτερου αριθμού προβλημάτων στις δύο πρώτες κατηγορίες προβλημάτων, ενώ στην τρίτη κατηγορία ο αριθμός αυτός παραμένει σταθερός.

Ως προς τον χρόνο εκτέλεσης, παρατηρούμε ότι η αύξηση του συντελεστή k, στην πρώτη κατηγορία οδηγεί σε αύξηση του χρόνου εκτέλεσης, ενώ στην δεύτερη κατηγορία σε μείωση αυτού. Στην τρίτη κατηγορία προβλημάτων, ο χρόνος εκτέλεσης παραμένει σε σταθερά περίπου επίπεδα ανεξαρτήτως k.

Όσον αφορά τον αριθμό των κόμβων, στην πρώτη κατηγορία προβλημάτων παρατηρούμε ότι για k=1 και k=3, ο αριθμός των κόμβων τους οποίους δημιουργεί παραμένει σχετικά σταθερός, ενώ για k=5 αυξάνεται ραγδαία. Στην δεύτερη κατηγορία προβλημάτων, για k=3 ο μηχανισμός παρουσιάζει τις βέλτιστες επιδόσεις. Τέλος, στην τρίτη κατηγορία προβλημάτων ο αριθμός των κόμβων που δημιουργούνται παραμένει σταθερός, ανεξαρτήτως k.

Συγκρίνοντας το Συνδυαστικό heuristic με τα υπόλοιπα τέσσερα παραλείποντας το κριτήριο του χρόνου εκτέλεσης, καταλήγουμε στις εξής παρατηρήσεις. Ως προς το σύνολο των προβλημάτων τα οποία επιλύει, για τις δύο πρώτες κατηγορίες παρατηρούμε ότι καθώς αυξάνεται το k, υπάρχει ραγδαία αύξηση και του αριθμού των λυμένων προβλημάτων. Για k=1 ο αριθμός αυτός ξεκινά από πολύ μικρές τιμές αποτελώντας την χειρότερη περίπτωση μεταξύ των heuristics και στη συνέχεια για

k=3 και k=5 ξεπερνά τα υπόλοιπα προσεγγίζοντας το dom/wdeg. Ως προς τον αριθμό των κόμβων τους οποίους δημιουργεί, ξανά στις δύο πρώτες κατηγορίες προβλημάτων παρατηρούμε ότι ανεξαρτήτως k οι επιδόσεις του μηχανισμού βρίσκονται σε ικανοποιητικά επίπεδα σε σχέση με τους υπόλοιπους μηχανισμούς, δεν αποτελούν όμως τις βέλτιστες. Σχετικά με την τρίτη κατηγορία προβλημάτων, παρατηρούμε ότι στο σύνολο των παραμέτρων, οι επιδόσεις του Common Choice, σχεδόν ταυτίζονται με τις επιδόσεις του IBS.

Τέλος, μελετώντας τους πίνακες συχνοτήτων το βασικό συμπέρασμα στο οποίο καταλήγουμε είναι, ότι για όλα τα είδη προβλημάτων, καθώς αυξάνεται το k αυξάνονται και τα ποσοστά επιλογής επόμενης μεταβλητής η οποία εμφανίζεται κοινή ανάμεσα σε 2, 3 ή και 4 heuristics, με την πλειονότητα των προβλημάτων η επιλογή να προκύπτει κυρίως μεταξύ δύο και τριών heuristics. Αυτό είναι λογικό καθώς όσο αυξάνεται το k, αυξάνεται και ο αριθμός των μεταβλητών τις οποίες προτείνει το κάθε heuristic, επομένως και η πιθανότητα κάποια μεταβλητή από αυτές να εμφανιστή σε παραπάνω από ένα heuristic. Ακόμα, η κοινή μεταβλητή η οποία τελικά επιλέγετε για k>1, εμφανίζεται στις πρώτες θέσεις προτίμησης του κάθε heuristic.

5. Ανακεφαλαίωση - Συμπεράσματα

Στην παρούσα διπλωματική, πραγματοποιήθηκε σύγκριση των ευριστικών μηχανισμών dom/ddeg, dom/wdeg, ABS και IBS σχετικά με το πόσο συμβάλουν στην βελτιστοποίηση του αλγορίθμου MAC κατά επίλυση δυαδικών προβλημάτων ικανοποίησης περιορισμών.

Αρχικά έγινε αναφορά στον τρόπο λειτουργίας του αλγορίθμου MAC, καθώς και στα δομικά στοιχεία τα οποία το αποτελούν. Τα στοιχεία αυτά είναι ο αλγόριθμος για την διάδοση των περιορισμών AC-3, καθώς και ο αλγόριθμος αναζήτησης με οπισθοδρόμηση Backtracking.

Στη συνέχεια, έγινε περιγραφή των προαναφερθέντων ευριστικών μηχανισμών, όπως επίσης και ενός πέμπτου μηχανισμού (Common Choice-Κοινή Επιλογή) ο οποίος συνδυάζει την λειτουργία των παραπάνω τεσσάρων μηχανισμών, επιλέγοντας την μεταβλητή εκείνη η οποία εμφανίζεται (είναι κοινή) στους περισσότερους προαναφερθέντες μηχανισμούς.

Έπειτα διεξήχθη μια σειρά πειραμάτων για την επίλυση προβλημάτων τριών κατηγοριών (RLFAP, GC, PIGEONS) εφαρμόζοντας τους παραπάνω μηχανισμούς σε συνδυασμό με τον αλγόριθμο MAC και λαμβάνοντας υπ' όψην το συνολικό χρόνο εκτέλεσης, το σύνολο των κόμβων που δημιουργήθηκαν κατά την διαδικασία της αναζήτησης και τον αριθμό των επιλυθέντων προβλημάτων, πραγματοποιήθηκε σύγκριση της απόδοσής τους. Τα συμπεράσματα τα οποία προέκυψαν από τα αποτελέσματα των παραπάνω πειραμάτων συνοψίζονται ως εξής:

- Ο ευριστικός μηχανισμός dom/wdeg, αποδείχτηκε στην πλειοψηφία των προβλημάτων ως προς το σύνολο των παραπάνω κριτηρίων ως ο αποδοτικότερος μεταξύ των υπολοίπων.
- Ο ευριστικός μηχανισμός dom/ddeg, παρουσίασε με τη σειρά του ικανοποιητικά αποτελέσματα, χωρίς ωστόσο να καταφέρουν να προσεγγίσουν τις επιδόσεις του dom/wdeg.
- Ο ευριστικός μηχανισμός ABS, παρουσιάζει μεγάλες διακυμάνσεις ως προς τα αποτελέσματα τα οποία εξάγει. Δηλαδή, είναι ένας αλγόριθμος ο οποίος μπορεί να αυξήσει ή να μειώσει την αποδοτικότητά του εξαιτίας της τυχαιότητάς η οποία εμφανίζεται κατά την διαδικασία αρχικοποίησης του.
- Ο ευριστικός μηχανισμός IBS, παρουσιάζει χαμηλή αποδοτικότητα σε σύγκριση με του υπόλοιπους τέσσερεις μηχανισμός. Αυτό οφείλεται στο γεγονός ότι αν και οι τιμές των αποτελεσμάτων τόσο ως προς τους χρόνους εκτέλεσης, όσο και ως προς τον αριθμό των κόμβων που δημιουργήθηκαν είναι αρκετά καλές, ο αριθμός των προβλημάτων τα οποία καταφέρνει να επιλύσει είναι αρκετά μικρός. Όπως αναφέρθηκε και σε προηγούμενη ενότητα,

η χαμηλή αυτή απόδοσή του, μπορεί να οφείλεται στην λανθασμένη αρχικοποίηση του.

- Τέλος, ο συνδυαστικός ευριστικός μηχανισμός Common Choice, παρ ότι δεν προκύπτει ως ο πιο αποδοτικός μεταξύ των υπολοίπων, καταφέρνει να δώσει ικανοποιητικά αποτελέσματα. Επίσης, η χρήση της παραμέτρου k φαίνεται να είναι αυτή, η οποία επηρεάζει καθοριστικά την απόδοση του συγκεκριμένου μηχανισμού, αφού για τιμές k>1 κατάφερε να ξεπεράσει σε επιδόσεις ως προς των αριθμό των παραχθέντων κόμβων, ακόμη και τον μηχανισμό dom/wdeg. Επομένως, μπορούμε να συμπεράνουμε ότι ο συγκεκριμένος μηχανισμός, παρουσιάζει δυναμική συμπεριφορά και μπορεί να προσαρμοστεί στις απαιτήσεις του κάθε προβλήματος. Τέλος, από την καταγραφή των συχνοτήτων επιλογής της επόμενης μεταβλητής, προέκυψε ότι καθώς η παράμετρος k αυξάνεται, αυξάνεται και η πιθανότητα εμφάνισης της ίδιας μεταβλητής σε περισσότερους από έναν μηχανισμούς, καθώς και ότι εμφανίζεται στις πρώτες k θέσεις προτίμησης.

6. Προοπτικές για το Μέλλον

Σε μεταγενέστερο στάδιο, σκοπός είναι η εξέλιξη της παρούσας διπλωματικής και η μελλοντική ενασχόληση με αυτή. Τα κύρια θέματα στα οποία θα πρέπει να δοθεί έμφαση για επιπλέον μελέτη είναι:

- Η εκτέλεση επιπλέον πειραμάτων για την διερεύνηση των παραπάνω ευριστικών μηχανισμών με νέες κατηγορίες προβλημάτων π.χ. the Driver problem, the Black Hole problem, the primes problem, The QCP, QWH and BQWH Problems κ.τ.λ. καθώς και μη-δυαδικών προβλημάτων.
- Η μελέτη βελτίωσης του συνδυαστικού ευριστικού μηχανισμού με νέους τρόπους συνδυασμού ή και τροποποίηση κάποιων από τους υπάρχοντες παραμέτρους.
- Περαιτέρω ανάλυση του μηχανισμού IBS κυρίως ως προς τον τρόπο αρχικοποίησης των Impacts των μεταβλητών, καθώς και την χρήση επιπλέον στρατηγικών π.χ. στρατηγική τυχαίας αναζήτησης (random search strategy), στρατηγική ελαχίστου μεγέθους πεδίου τιμών (minimum domain size strategy) και άλλες με σκοπό την βελτίωση της απόδοσης του μηχανισμού.

Βιβλιογραφία

[1] http://www.wikipedia.org

[2] Stuart Russell-Peter Norving (2003). *Τεχνητή Νοημοσύνη Μία σύγχρονη προσέγγιση*, *ΔΕΥΤΕΡΗ ΑΜΕΡΙΚΑΝΙΚΗ ΕΚΔΟΣΗ*. Εκδόσεις "ΚΛΕΙΔΑΡΙΘΜΟΣ", 2005.

[3] F.Boussemart, F.Hemery,and C.Lecoutre. Revision ordering heuristics for the Constraint Satisfaction Problem. In 10th International Conference on Principles and Practice of Constraint Programming (CP2004), Work shop on Constraint Propagation and Implementation, Toronto, Canada, 2004.

[4] A. Chmeiss and P. J´egou. Efficient path-consistency propagation. Journal on Artificial Intelligence Tools, 7(2):121–142, 1998

[5] Y.Deville and P.V.Hentenryck. An efficient arc consistency algorithm for a class of CSP problems. In Proceedings of IJCAI-91, pages 325–330, 1991.

[6] A. Mackworth. On reading sketch maps. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-1977), pages 598–606, Cambridge MA, 1977.

[7] R. Mohr and T. Henderson. Arc and Path Consistency Revisited. Artificial Intelligence, 28:225–233, 1986.

[8] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Proceedings of CP-1994, pages 10–20, 1994.

[9] R.M. Haralick and Elliott. Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence, 14:263–314, 1980.

[10] J.J. McGregor. Relational consistency algorithms and their applications in finding subgraph and graph isomorphism. Information Science, 19:229–250, 1979.

[11] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In Proceedings of ECAI-04, pages 146–150, 2004.

[12] T. Balafoutis and K. Stergiou. Adaptive branching for constraint satisfaction problems. In ECAI'10, pages 855–860, 2010.

[13] P.Refalo. Impact-based search strategies for constraint programming. In Proceedings of CP-2004, pages 556–571, 2004.

[14] L Michel and P. Van Hentenryck. Activity-Based Search for Black-Box Constraint Programming Solvers. In Proceedings of CPAIOR 2012, pages 228-243, 2012.

[15] E. Freuder and A. Mackworth. Constraint satisfaction: an emerging paradigm. In F. Rossi, P. van Beek, and T. Walsh, editors, Handbook of Constraint Programming, chapter 2. Elsevier, 2006.

[16] C. Bessi`ere and J.C. R´egin. MAC and combined heuristics: two reasons to for sake FC (and CBJ?). In Proceedings of CP-1996, pages61–75, Cambridge MA, 1996.

[17] B.M. Smith and S.A. Grant. Trying harder to fail first. In Proceedings of 13th European Conference on Artificial Intelligence (ECAI-1998), pages 249–253, 1998.

[18] R Dechter. Constraint Processing. Morgan Kaufmann, 2003.