ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χωροθέτηση και Ενοικίαση Κοινόχρηστων Πόρων: Πειραματική Αξιολόγηση Αλγορίθμων

Ιωάννης Ν. Σιγάλας

Επιβλέπων: Βασίλειος Ζησιμόπουλος, Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2013

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χωροθέτηση και Ενοικίαση Κοινόχρηστων Πόρων: Πειραματική Αξιολόγηση Αλγορίθμων

Ιωάννης Ν. Σιγάλας
Α.Μ.: Μ1170

ΕΠΙΒΛΕΠΩΝ: **Βασίλειος Ζησιμόπουλος,** Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:
Β. Ζησιμόπουλος, Καθηγητής, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Δ. Φωτάκης, Επίκουρος Καθηγητής, Εθνικό Μετσόβιο Πολυτεχνείο

Σεπτέμβριος 2013

ΕΥΧΑΡΙΣΤΙΕΣ

Για την εκπόνηση της παρούσας εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα, καθ. κ. Βασίλειο Ζησιμόπουλο για τη συνεργασία, την καθοδήγηση και τη συμβολή του. Επίσης τον κ. Φωτάκη για τη συμμετοχή του στην εξέταση της εργασίας και τις χρήσιμες παρατηρήσεις του.

ΠΕΡΙΛΗΨΗ

Στην εργασία αυτή μελετάται το πρόβλημα της χωροθέτησης (facility location) και ενοικίασης (facility leasing) κοινόχρηστων πόρων και συγκεκριμένα η εκδοχή του προβλήματος που απαιτεί ανοχή σε σφάλματα των κοινόχρηστων πόρων μέσω πλεονασμού (fault tolerant facility location). Εξετάζεται πειραματικά η δυσκολία του προβλήματος της χωροθέτησης καθώς και της ενοικίασης. Επίσης υλοποιούνται και αξιολογούνται δύο ευρέως αποδεκτοί αλγόριθμοι και εξετάζεται αναλυτικά η προσέγγιση τους στις βέλτιστες τιμές και η διακύμανση της με διάφορους παράγοντες. Με βάση τα αποτελέσματα προτείνονται δύο νέοι αλγόριθμοι με καλύτερη συμπεριφορά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αλγοριθμική Επιχειρησιακή Έρευνα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Χωροθέτηση πόρων, προσεγγιστικοί αλγόριθμοι, πειραματική μελέτη, ενοικίαση πόρων, ανοχή σε σφάλματα.

ABSTRACT

In this work, we deal with facility location problems and especially with fault tolerant facility location and offline facility leasing. We perform an experimental study of the difficulty of these two problems. Also two well known algorithms are implemented and evaluated and their approximation to the optimal solution is being studied as it fluctuates when changing various factors. Based on the results we propose two new algorithms which perform even better.

SUBJECT AREA: Algorithmical Operational Research

KEYWORDS: Facility location, approximation algorithms, facility leasing, experimental study, fault tolerant.

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

Πολλά προβλήματα που ανήκουν στην κατηγορία των «δύσκολων» προβλημάτων έχουν εκθετική πολυπλοκότητα και αυτό σημαίνει ότι καθώς μεγαλώνει η διάσταση του προβλήματος πολύ γρήγορα γίνεται αδύνατη η λύση τους σε εύλογο χρόνο. Το πρόβλημα της χωροθέτησης κοινόχρηστων πόρων (facility location FL) ανήκει σε αυτή την κατηγορία.

Για την αντιμετώπιση τέτοιου είδους προβλημάτων έχουν προταθεί προσεγγιστικοί αλγόριθμοι οι οποίοι σε πολυωνυμικό χρόνο δίνουν μία πολύ καλή λύση μέσα σε συγκεκριμένα όρια θεωρητικής απόκλισης. Επίσης υπάρχουν και ευριστικοί αλγόριθμοι οι οποίοι δεν παρέχουν κάποια θεωρητική μέγιστη απόκλιση αλλά συχνά λειτουργούν πολύ καλά και υπερέχουν των πρώτων.

Το πρόβλημα FL έχει μελετηθεί εκτενώς στην επιχειρησιακή έρευνα. Ωστόσο υπάρχουν μορφές του προβλήματος όπως η χωροθέτηση πόρων με ανοχή σε σφάλματα (fault tolerant facility location στο εξής FTFL) και η ενοικίαση κοινόχρηστων πόρων (offline facility leasing στο εξής OFL) που προκύπτουν από τις πραγματικές ανάγκες που εμφανίζονται σε τομείς της επιχειρηματικής δραστηριότητας. Αυτές οι εκδοχές του προβλήματος αν και έχουν μελετηθεί θεωρητικά και μπορεί να βρει κανείς αρκετούς προσεγγιστικούς αλγορίθμους, δεν γνωρίζουμε πόσο καλοί είναι σε πραγματικά προβλήματα.

Η αλματώδης ανάπτυξη του internet προκάλεσε την εμφάνιση τέτοιων προβλημάτων, που ακόμη και αν υπήρχαν παλαιότερα, εμφανίζονταν σε πολύ μικρότερα μεγέθη και έτσι μπορούσαν να επιλύονται βέλτιστα. Όμως με δεδομένο το συνεχώς αυξανόμενο μέγεθος του internet και την ανάγκη συχνά για αποφάσεις σε πραγματικό χρόνο (real time) πχ cognitive radios δεν είναι δυνατό να περιμένουμε την επίλυση ενός προβλήματος στην βέλτιστη τιμή, όταν μπορούμε να πάρουμε το 1,01% της βέλτιστης σε χρόνο δύο ή τρείς τάξεις μεγέθους μικρότερο.

Το UFL έχει πολλές εφαρμογές τόσο σε φυσικές εγκαταστάσεις, πχ. μία βιομηχανία που λειτουργεί με τρία εργοστάσια και θέλει να είναι όσο το δυνατόν πιο κοντά στους προμηθευτές και τους πελάτες της.) όσο και στο internet. Συχνά σε τέτοιες εφαρμογές απαιτείται και η αντοχή σε σφάλματα, λόγω της ευπάθειας του πρωτοκόλλου IP. πχ αν μία σημαντική ιστοσελίδα παρέχεται από 2 ή 3 εξυπηρετητές τότε μειώνεται ο χρόνος εξυπηρέτησης (καθώς ένας εξυπηρετητής από όλους θα είναι πιο κοντά στον κάθε πελάτη συγκριτικά με ένα σύστημα με έναν εξυπηρετητή) και είμαστε σίγουροι ότι ακόμα και αν τεθεί εκτός λειτουργίας ένας εξυπηρετητής ή κάποιο τμήμα του δικτύου, δεν θα έχει σημαντική επίδραση στην ιστοσελίδα μας.

Μία τάση που έχει παρουσιαστεί πρόσφατα είναι η δημιουργία επιχειρήσεων που δεν έχουν ίδιους πόρους αλλά ενοικιάζουν πόρους και τους παρέχουν στους χρήστες τους. Για να λειτουργεί αποδοτικά μια τέτοια επιχείρηση θα πρέπει να έχει έναν αποτελεσματικό τρόπο καθορισμού των πόρων που ενοικιάζει. Αυτό δημιουργεί την ανάγκη για αξιόπιστους αλγόριθμους OFL. Ένα παράδειγμα τέτοιας επιχείρησης θα μπορούσε να είναι μία εταιρία ενοικίασης αυτοκινήτων σε ένα τουριστικό προορισμό. Λόγω της εποχικότητας της πελατείας της δεν είναι συμφέρον να διαθέτει το δικό της στόλο αυτοκινήτων αλλά μπορεί να ενοικιάζει μακροπρόθεσμα (leasing) κάποια αυτοκίνητα και να τα διαθέτει στη συνέχεια με χρέωση ανά ημέρα. Αυτό προϋποθέτει την ορθή πρόβλεψη των απαιτούμενων οχημάτων ώστε να καλύψει την ζήτηση αλλά να μην χρεώνεται για οχήματα που δεν ενοικιάζονται και προκαλούν έτσι ζημιά.

Υπάρχουσες πειραματικές μελέτες για το Uncapacitated Facility Location (UFL), όπως του M. Hoefer στο [4] και των M. Resend, R Werneck στο [5], έχουν

σκοπό τη μελέτη και σύγκριση διαφόρων αλγορίθμων μεταξύ τους. Ο M. Hoefer συγκρίνει τους αλγορίθμους JMS (από τα αρχικά των Jain, Mahdian, και Saber) με θεωρητική προσέγγιση 1,61, MYZ (ομοίως από τους Mahdian, Ye και Zhang) με προσέγγιση 1,52, καθώς και τους ευριστικούς αλγόριθμους Tabu search, Local search, Volume algorithm. Συνοπτικά τα αποτελέσματα είναι ότι ο αλγόριθμος Tabu search δίνει το μικρότερο σφάλμα ενώ οι MYZ,JMS είναι πιο γρήγοροι. Με βάση αυτά τα συμπεράσματα, οι M. Resend, R Werneck συγκρίνουν τον Tabu με έναν νέο ευριστικό αλγόριθμο που ονομάζουν hybrid multistart και δείχνουν ότι αποδίδει ακόμα καλύτερα. Σε κάθε περίπτωση το σφάλμα που δίνουν οι προσεγγιστικοί αλγόριθμοι είναι μικρότερο από 2% της βέλτιστης λύσης.

Αυτή η πειραματική αξιολόγηση εξετάζει κατ' αρχήν την πραγματική δυσκολία του προβλήματος και τις συνθήκες υπό τις οποίες αξίζει να χρησιμοποιήσει κανείς τέτοιους προσεγγιστικούς αλγορίθμους, τόσο για το FTFL όσο και για το OFL. Στη συνέχεια εξετάζονται σε τυχαία μετρικά στιγμιότυπα οι αλγόριθμοι των Swamy & Shmoys (SS) και Jain & Vazirani (JV). Επίσης εξετάζονται δύο ευριστικοί αλγόριθμοι που αναπτύχθηκαν με βάση παρατηρήσεις από τις επιδόσεις των δύο πρώτων. Ο πρώτος είναι ένας συνδυασμός των δύο προηγουμένων και θα τον αναφέρουμε ως Swamy & Shmoys-Jain & Vazirani (SSJV) ενώ ο άλλος επαναλαμβάνει πολλές φορές το χαλαρωμένο πρόβλημα και αναφέρεται ως repeated relaxed (RR). Τελικά καταλήγουμε σε προτάσεις για την επιλογή της καταλληλότερης μεθόδου επίλυσης του προβλήματος ανάλογα με τις απαιτήσεις σε χρόνο και εγγύτητα στη βέλτιστη λύση, καθώς και το μέγεθος και τη δυσκολία του στιγμιοτύπου.

.

.

2. ΤΟ ΠΡΟΒΛΗΜΑ ΧΩΡΟΘΕΤΗΣΗΣ ΠΟΡΩΝ (UFL)

Το πρόβλημα χωροθέτησης στη γενική του μορφή αποτελείται από ένα σύνολο από κοινόχρηστους πόρους και ένα σύνολο από πελάτες. Στη βιβλιογραφία οι πόροι αναφέρονται ως facilities ή servers και οι πελάτες ως clients ή cities. Στο παρόν θα χρησιμοποιήσουμε τους όρους εξυπηρετητής και πελάτης, υπονοώντας οποιοδήποτε από τα προηγούμενα, ανάλογα με την εφαρμογή που θέλουμε να επιλύσουμε. Κάθε εξυπηρετητής έχει ένα κόστος εγκατάστασης (opening cost) και κάθε πελάτης μπορεί να εξυπηρετηθεί από έναν εξυπηρετητή με ένα κόστος εξυπηρέτησης (service cost). Το ζητούμενο είναι ο προσδιορισμός ενός υποσυνόλου των εξυπηρετητών οι οποίοι θα ανοίξουν και ποιους πελάτες θα εξυπηρετήσει ο καθένας από αυτούς ώστε το συνολικό κόστος να είναι ελάχιστο και κάθε πελάτης να εξυπηρετείται από έναν εξυπηρετητή. Συνολικό κόστος εννοείται το άθροισμα του κόστους εγκατάστασης για τους εξυπηρετητές που είναι ανοιχτοί και το άθροισμα του κόστους εξυπηρέτησης για τους πελάτες που εξυπηρετούνται από αυτούς. Αυτό είναι γνωστό και σαν Uncapacitated Facility Location (UFL) καθώς οι εξυπηρετητές δεν έχουν περιορισμό ως προς τους πελάτες που μπορούν να εξυπηρετήσουν και τις ποσότητες που αυτοί επιθυμούν. Η μαθηματική διατύπωση του προβλήματος είναι:

$$\min \sum_i f_i y_i + \sum_i \sum_j x_{ij} c_{ij} \quad (1)$$

$$\sum_i x_{ij} \geq 1 \qquad \forall j \qquad (2)$$

$$x_{ij} \leq y_i \qquad \forall\, i,j \qquad (3)$$

$$y_i \leq 1 \qquad \forall\, i \qquad (4)$$

$$x_{ij}, y_i \geq 0 \qquad \forall\, i,j \qquad (5)$$

Όπου

f_i το κόστος εγκατάστασης του εξυπηρετητή i.

c_{ij} το κόστος εξυπηρέτησης από τον εξυπηρετητή i στον πελάτη j.

x_{ij} ακέραιος αριθμός που περιγράφει αν ο πελάτης j εξυπηρετείται από τον εξυπηρετητή i.

y_i ακέραιος αριθμός που εκφράζει αν ο εξυπηρετητής i είναι ανοικτός.

Η σχέση 2 εξασφαλίζει ότι κάθε πελάτης εξυπηρετείται ενώ η σχέση 3 επιτρέπει στους πελάτες να εξυπηρετούνται μόνο από τους ανοικτούς εξυπηρετητές. Ο περιορισμός 4 περιορίζει την τιμή του y_i μέχρι 1 ενώ η σχέση 5 δεν επιτρέπει αρνητικές τιμές στις μεταβλητές.

2.1 Χωροθέτηση πόρων με ανοχή σε σφάλματα (FTFL)

Στην εκδοχή αυτή του προβλήματος κάθε πελάτης απαιτεί κάποιο πλεονασμό ώστε σε περίπτωση βλάβης κάποιου εξυπηρετητή να συνεχίσει απρόσκοπτα την λειτουργία του. Τέτοιου είδους πλεονασμός συναντάται συχνά σε εφαρμογές στο διαδίκτυο. Το πρόβλημα ορίζεται όπως προηγουμένως και επιπλέον κάθε πελάτης έχει μία απαίτηση (requirement) να εξυπηρετηθεί από r διαφορετικούς εξυπηρετητές. Η διαφορά του με το UFL είναι στη σχέση (2). Η μαθηματική διατύπωση του προβλήματος είναι:

$$\min \sum_i f_i y_i + \sum_i \sum_j x_{ij} c_{ij} \quad (1)$$

$$\sum_i x_{ij} \geq r_j \qquad \forall j \qquad (2)$$

$$x_{ij} \leq y_i \qquad \forall\, i,j \qquad (3)$$

$$y_i \leq 1 \qquad \forall\, i \qquad (4)$$

$$x_{ij}, y_i \geq 0 \qquad \forall\, i,j \qquad (5)$$

Όπου

f_i το κόστος εγκατάστασης του εξυπηρετητή i.

c_{ij} το κόστος εξυπηρέτησης από τον εξυπηρετητή i στον πελάτη j.

r_j η απαίτηση για r διαφορετικούς εξυπηρετητές του πελάτη j.

x_{ij} ακέραιος αριθμός που περιγράφει αν ο πελάτης j εξυπηρετείται από τον εξυπηρετητή i.

y_i ακέραιος αριθμός που εκφράζει αν ο εξυπηρετητής i είναι ανοικτός.

Το παραπάνω πρόβλημα χωρίς την απαίτηση να είναι ακέραιοι αριθμοί τα x_{ij}, y_i αναφέρεται σαν relaxed ή «χαλαρωμένο» πρόβλημα και έχει μεγάλη σημασία για κάποιους από τους αλγόριθμους που θα εξεταστούν παρακάτω. Η επίλυσή του μπορεί να γίνει εύκολα με τον αλγόριθμο simplex σε σχετικά μικρό χρόνο. Αντίθετα για το ακέραιο πρόβλημα απαιτείται η μέθοδος branch and bound (B&B) η οποία επαναλαμβάνει τον αλγόριθμο simplex εκθετικά στο πλήθος των μεταβλητών.

2.2 Ενοικίαση κοινόχρηστων πόρων (OFL)

Το πρόβλημα OFL εξελίσσεται κατά τη διάρκεια του χρόνου και για ένα γνωστό και προκαθορισμένο χρονικό διάστημα, το οποίο χωρίζουμε σε διακριτά τμήματα πχ ημέρες. Σε κάθε ημέρα ένα υποσύνολο από πελάτες απαιτεί να εξυπηρετηθεί από έναν εξυπηρετητή που θα πρέπει να έχει ενοικιαστεί για εκείνη την μέρα. Οι εξυπηρετητές μπορούν να ενοικιάζονται για διάφορα διαστήματα πχ για 2,5,8 ημέρες και κάθε μέρα να μην είναι διαθέσιμα όλα τα διαστήματα ενοικίασης. Όπως παραπάνω, κάθε πελάτης εξυπηρετείται από έναν εξυπηρετητή με ένα κόστος εξυπηρέτησης. Σκοπός μας είναι να καθορίσουμε ποιοι εξυπηρετητές θα ενοικιαστούν κάθε μέρα και για πόσες ημέρες ώστε να ικανοποιούνται όλοι οι πελάτες και το συνολικό κόστος ενοικίασης και εξυπηρέτησης να είναι ελάχιστο.

Προκειμένου να έχουν νόημα τα μεγαλύτερα διαστήματα ενοικίασης, το κόστος τους θα πρέπει να είναι διαμορφωμένο έτσι ώστε να «συμφέρουν» σε σχέση με τα μικρότερα. Δεδομένου ότι όλο και περισσότεροι πάροχοι υπηρεσιών χρησιμοποιούν πόρους που ενοικιάζουν από άλλους, το πρόβλημα OFL αποκτά όλο και μεγαλύτερο πεδίο εφαρμογών.

Η μαθηματική διατύπωση του προβλήματος είναι:

$$\min \sum_{i,k,t} f_{ik} y_{ikt} + \sum_{ikt} \sum_{j,t'} x_{ikt,jt'} c_{ij} \qquad (1)$$

$$\sum_{ikt} x_{ikt,jt'} \geq r_{jt'} \qquad \forall j,t' \qquad (2)$$

$$x_{ikt,jt'} \leq y_{ikt} * d_{ikt} \qquad \forall\, ikt,jt' \quad (3)$$

$$y_{ikt} \leq 1 \qquad \forall\, ikt \qquad (4)$$

$$x_{ikt,jt'}, y_{ikt} \geq 0 \qquad \forall\, ikt, jt' \quad (5)$$

Όπου

f_{ik} το κόστος εγκατάστασης του εξυπηρετητή i για k ημέρες.

c_{ij} το κόστος εξυπηρέτησης από τον εξυπηρετητή i στον πελάτη j για μία μέρα.

$x_{ikt,jt'}$ ακέραιος αριθμός που περιγράφει αν ο πελάτης j που έχει μια απαίτηση την ημέρα t' εξυπηρετείται από τον εξυπηρετητή i που ενοικιάστηκε την ημέρα t για k ημέρες.

y_{ikt} ακέραιος αριθμός που εκφράζει αν ο εξυπηρετητής i ενοικιάστηκε την ημέρα t για k ημέρες.

d_{ikt} ακέραιος αριθμός που εκφράζει ότι ο εξυπηρετητής i είναι διαθέσιμος για ενοικίαση την ημέρα t για k ημέρες.

r_{jt} ακέραιος αριθμός που καθορίζει αν ο πελάτης j χρειάζεται να εξυπηρετηθεί από κάποιον εξυπηρετητή την ημέρα t.

Ήδη πρέπει να είναι φανερό ότι η πολυπλοκότητα του προβλήματος είναι πολύ μεγαλύτερη από το FL καθώς έχουμε πολλές ημέρες και σε κάθε μία από αυτές μπορεί να εξυπηρετηθεί ένας πελάτης από έναν εξυπηρετητή που ενοικιάστηκε πριν από κάποιες ημέρες και είναι ακόμα διαθέσιμος. Μπορεί να παρατηρήσει κανείς ότι όταν το κόστος κάθε μεμονωμένης ημέρας είναι μικρότερο ή ακόμα και ίσο με το κόστος κάθε ημέρας σε ένα πολλαπλάσιο διάστημα ενοικίασης, τότε δεν υπάρχει λόγος να προτιμούνται τα μεγαλύτερα διαστήματα ενοικίασης. Σε αυτήν την περίπτωση το πρόβλημα εκφυλίζεται σε ένα UFL για κάθε ημέρα και είναι πλέον πολύ ευκολότερη η επίλυσή του.

3. ΑΛΓΟΡΙΘΜΟΙ

Σε αυτό το κεφάλαιο θα αναφέρουμε τους δύο αλγορίθμους των Swamy & Shmoys και των Jain & Vazirani, όπως αυτοί προτάθηκαν στα [2] και [1] αντιστοίχως, καθώς και τους δύο προτεινόμενους αλγορίθμους SSJV και RR, οι οποίοι προέκυψαν από παρατηρήσεις στα πειραματικά αποτελέσματα που λήφθηκαν από τους SS και JV. Εδώ θα περιγράψουμε τον κάθε αλγόριθμο ενώ λεπτομέρειες για το πως υλοποιήθηκε βρίσκονται στο επόμενο κεφάλαιο και ο κώδικας του προγράμματος στο Παράρτημα Α.

3.1 Αλγόριθμος Swamy & Shmoys

Ο πρώτος αλγόριθμος που μελετήθηκε προτείνεται από τους Swamy & Shmoys στο [2] ως ένας απλός αλγόριθμος με προσέγγιση 4 φορές την βέλτιστη τιμή. Ο αλγόριθμος αρχίζει επιλύοντας το χαλαρωμένο πρόβλημα καθώς και το δυϊκό του.

Το χαλαρωμένο ή LP πρόβλημα είναι:

$$\min \sum_i f_i y_i + \sum_i \sum_j x_{ij} c_{ij} \quad (1)$$

$$\sum_i x_{ij} \geq r_j \quad \forall j \quad (2)$$

$$x_{ij} \leq y_i \quad \forall\, i,j \quad (3)$$

$$y_i \leq 1 \quad \forall\, i \quad (4)$$

$$x_{ij}, y_i \geq 0 \quad \forall\, i,j \quad (5)$$

Το δυϊκό του LP είναι το εξής πρόβλημα:

$$\max \sum_j r_j a_j - \sum_i z_i \quad (1)$$

$$\sum_j \beta_{ij} \leq f_i + z_i \quad \forall i \quad (2)$$

$$\alpha_j \leq \beta_{ij} + c_{ij} \quad \forall\, i,j \quad (3)$$

$$\alpha_j, \beta_{ij}, z_i \geq 0 \quad \forall\, i,j \quad (4)$$

Όπου

f_i το κόστος εγκατάστασης του εξυπηρετητή i.

c_{ij} το κόστος εξυπηρέτησης από τον εξυπηρετητή i στον πελάτη j.

r_j η απαίτηση για r διαφορετικούς εξυπηρετητές του πελάτη j.

$\alpha_j, \beta_{ij}, z_i$ οι δυϊκές μεταβλητές του προβλήματος

Ο αλγόριθμος έχει δύο φάσεις με τα βήματα που περιγράφονται παρακάτω:

Αλγόριθμος Swamy & Shmoys [2]

Φάση $1^{η}$:

Επιλύουμε το χαλαρωμένο πρόβλημα. Από τη λύση του χαλαρωμένου προβλήματος ανοίγουμε όσους εξυπηρετητές έχουν y_i =1. Για κάθε πελάτη με x_{ij} >0 και y_i =1 εξυπηρετείται αυτός από τον εξυπηρετητή i.

Φάση $2^{η}$:

Έστω σύνολο S με τους πελάτες που έχουν κάποια υπολειπόμενη απαίτηση. Επαναλαμβάνουμε τα παρακάτω μέχρι S=Ø.

2α. Επιλέγουμε τον πελάτη j με το μικρότερο $α_j$.

2β. Ταξινομούμε τους εξυπηρετητές που εξυπηρετούν μερικά τον πελάτη j, δηλαδή x_{ij} >0 κατά αύξουσα σειρά κόστους εγκατάστασης. Επιλέγουμε υποσύνολο M για το οποίο ισχύει:

$$\sum_{i' \in M} y_{i'} \geq r'_j$$

όπου r'_j η υπολειπόμενη απαίτηση του πελάτη j. Εάν δεν ισχύει η ισότητα χωρίζουμε τον τελευταίο εξυπηρετητή σε δύο τμήματα i_1 και i_2 με το i_1 να παραμένει εντός του M και το i_2 να είναι εκτός, ώστε να ισχύει η ισότητα στην παραπάνω ανίσωση. Επομένως έχουμε για αυτόν τον εξυπηρετητή:

$$y_{i_1} = r'_j - \sum_{i' \in M} y_{i'}, \quad y_{i_2} = y_i - y_{i_1}\ x_{i_1 k} + \ x_{i_2 k} = x_{ik}\ \ x_{i_1 k} \leq \ y_{i_1}\ \ x_{i_2 k} \leq \ y_{i_2}$$

2γ. Ανοίγουμε τους r'_j φθηνότερους εξυπηρετητές από το M. Κάθε πελάτης που εξυπηρετούνταν από κάποιον εξυπηρετητή στο M θα καλύψει την απαίτηση αυτή με τους εξυπηρετητές που ανοίχτηκαν. Οι υπόλοιποι εξυπηρετητές από το M καθώς και ο πελάτης j δεν χρησιμοποιούνται στις επόμενες επαναλήψεις.

3.2 Αλγόριθμος Jain & Vazirani

Ο επόμενος αλγόριθμος προτάθηκε από τους Jain & Vazirani στο [1] και βασίζεται και αυτός στο primal-dual πρόβλημα, λύνοντας το όμως με μία διαφορετική προσέγγιση. Αρχίζει δημιουργώντας ένα UFL με τους πελάτες που έχουν την μέγιστη απαίτηση. Αφού το λύσει αυτό παίρνει τους πελάτες που έχουν απαίτηση ένα λιγότερο και δημιουργεί ένα νέο πρόβλημα. Σε αυτό οι εξυπηρετητές που είναι ήδη ανοιχτοί από την πρώτη επανάληψη έχουν μηδενικό κόστος εγκατάστασης ενώ τα x_{ij} για τους πελάτες που ήδη εξυπηρετούνται από αυτά τίθεται άπειρο, ώστε στην επόμενη φάση να μην κατευθυνθούν στους ίδιους εξυπηρετητές. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να ικανοποιηθούν όλοι οι πελάτες, άρα θα απαιτήσει τόσα βήματα όσο είναι η μέγιστη απαίτηση των πελατών.

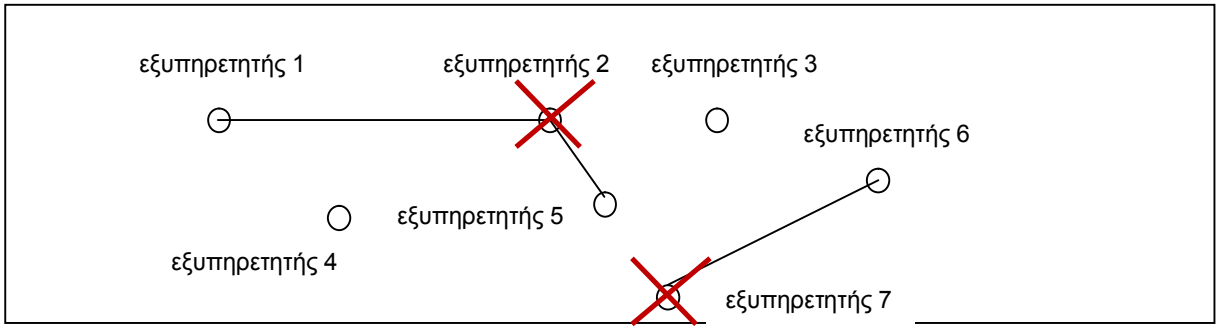
Σε κάθε βήμα του αλγορίθμου αυξάνεται ομοιόμορφα η δυϊκή μεταβλητή $α_j$ για όλους τους πελάτες που συμμετέχουν σε αυτό το βήμα. Σταδιακά ένα από τα δύο παρακάτω θα συμβούν:

α. Ο περιορισμός (3) του δυϊκού που αφορά τις συνδέσεις θα φτάσει στην ισότητα, δηλαδή $α_j - β_{ij} = c_{ij}$ οπότε αν ο εξυπηρετητής i είναι προσωρινά ανοικτός ο πελάτης j θα συνδεθεί σε αυτό, ενώ αν δεν είναι θα αρχίσει να αυξάνεται και το $β_{ij}$ μαζί με το $α_j$ για να διατηρείται η ισότητα.

β. Ο περιορισμός (2) που αφορά τους εξυπηρετητές θα φτάσει στην ισότητα, οπότε ανοίγει προσωρινά ο εξυπηρετητής και οι πελάτες που πληρώνουν για να ανοίξει συνδέονται με αυτόν.

Όταν ολοκληρωθούν όλα τα βήματα της κάθε φάσης, πρέπει να εξασφαλίσουμε ότι κανένας πελάτης δεν πληρώνει για περισσότερους από έναν εξυπηρετητές (υπενθυμίζεται ότι τώρα λύνουμε ένα απλό UFL άρα η απαίτηση είναι να εξυπηρετηθεί ο πελάτης από έναν εξυπηρετητή). Αυτό επιτυγχάνεται φτιάχνοντας ένα γράφο με κόμβους τους εξυπηρετητές και με μία ακμή για κάθε πελάτη που συνεισφέρει στο άνοιγμα δύο εξυπηρετητών. Στη συνέχεια επιλέγουμε ένα maximal independent set των εξυπηρετητών που θα ανοίξουν οριστικά. Για παράδειγμα στο παρακάτω σχήμα κάποιος πελάτης συνεισφέρει στο άνοιγμα των εξυπηρετητών 1 και 2, κάποιος στους εξυπηρετητές 2 και 5 και κάποιος στους 6 και 7. Οριστικά θα ανοίξουν οι εξυπηρετητές 1,3,4,5,6 ενώ οι 2 και 7 θα παραμείνουν κλειστοί.

Γράφος επιλογής εξυπηρετητών μέσω ενός μέγιστου ανεξάρτητου συνόλου.



Ο αλγόριθμος επαναλαμβάνει τις εξής δύο φάσεις μέχρι να λυθεί το πρόβλημα:

Αλγόριθμος Jain & Vazirani [1]

Φάση $1^{η}$:

Δημιουργούμε ένα απλό πρόβλημα UFL με τους πελάτες που έχουν απαίτηση ίση με τη μέγιστη απαίτηση που υπάρχει στο στιγμιότυπο. Επιλύουμε αυτό το πρόβλημα με τον τρόπο που περιγράφεται στη φάση 2.

Φάση $2^{η}$:

α. Αυξάνουμε τη δυϊκή μεταβλητή α_j κατά ένα σταθερό βήμα. Σε κάθε βήμα ελέγχουμε αν συμβαίνει κάτι από τα εξής.

1) Αν $\alpha_j - \beta_{ij} = c_{ij}$ και ο εξυπηρετητής i είναι προσωρινά ανοικτός ο πελάτης j θα συνδεθεί σε αυτό. Αν όχι θα αρχίσει να αυξάνεται και το β_{ij} μαζί με το α_j για να ισχύει ο περιορισμός.

2) Ελέγχουμε αν για κάποιο εξυπηρετητή από αυτούς που αυξήσαμε το β_{ij} ισχύει :

$$\sum_j \beta_{ij} = f_i$$

Αν συμβαίνει αυτό τότε ανοίγουμε προσωρινά αυτόν τον εξυπηρετητή και εξυπηρετούμε όσους πελάτες έχουν συνεισφέρει για το άνοιγμά του.

β. Κατασκευάζουμε ένα γράφο με κόμβους τους εξυπηρετητές και μία ακμή για κάθε πελάτη που συνεισφέρει στο άνοιγμα δύο εξυπηρετητών. Επιλέγουμε ένα maximal independent set των εξυπηρετητών που θα ανοίξουν οριστικά.

Στους εξυπηρετητές που άνοιξαν οριστικά το κόστος εγκατάστασης τίθεται μηδέν ενώ στους πελάτες που εξυπηρετούνται από αυτούς το κόστος εξυπηρέτησης τίθεται άπειρο.

Μειώνουμε την απαίτηση στο αρχικό πρόβλημα των πελατών που εξυπηρετήθηκαν σε αυτή την επανάληψη και συνεχίζουμε από την πρώτη φάση μέχρι η μέγιστη απαίτηση να είναι μηδέν.

3.3 Αλγόριθμος Swamy & Shmoys – Jain & Vazirani (Υβριδικός Αλγόριθμος)

Μετά την παρατήρηση των πρώτων πειραματικών αποτελεσμάτων διαπιστώθηκε η αποτελεσματικότητα της πρώτης φάσης του αλγορίθμου SS. Επίσης ο αλγόριθμος JV έχει αρκετά καλά αποτελέσματα, αλλά δεν εκμεταλλεύεται το μεγάλο ποσοστό προβλημάτων που επιλύονται βέλτιστα από το χαλαρωμένο πρόβλημα. Έτσι επιλέξαμε το συνδυασμό της πρώτης φάσης του SS με τον JV. Ο υβριδικός αλγόριθμος που φτιάχτηκε από τον συνδυασμό τους εκμεταλλεύεται τα πλεονεκτήματα του κάθε ενός και είναι πολύ πιο αποτελεσματικός, όπως παρουσιάζεται αναλυτικά αργότερα.

Η ιδέα είναι αρκετά απλή. Μετά την πρώτη φάση του αλγορίθμου των SS, εκτελείται ο αλγόριθμος JV στο πρόβλημα που έχει απομείνει άλυτο. Σε αυτό οι εξυπηρετητές που είναι ήδη ανοιχτοί από την πρώτη φάση έχουν μηδενικό κόστος

εγκατάστασης ενώ τα x_{ij} για τους πελάτες που ήδη εξυπηρετούνται από αυτούς τίθεται άπειρο, ώστε στην επόμενη φάση να μην κατευθυνθούν στους ίδιους εξυπηρετητές, όπως ακριβώς κάνουμε και στον JV μετά από κάθε φάση.

Αλγόριθμος Swamy & Shmoys – Jain & Vazirani (Υβριδικός Αλγόριθμος)

Φάση $1^η$:
Επιλύουμε το χαλαρωμένο πρόβλημα. Από τη λύση του χαλαρωμένου προβλήματος ανοίγουμε όσους εξυπηρετητές έχουν y_i =1. Για κάθε πελάτη με x_{ij} >0 και y_i =1 εξυπηρετείται αυτός από τον εξυπηρετητή i.
Φάση $2^η$:

Επιλύουμε το υπόλοιπο πρόβλημα με τον αλγόριθμο JV, δηλαδή στους εξυπηρετητές που άνοιξαν το κόστος εγκατάστασης τίθεται μηδέν ενώ στους πελάτες που εξυπηρετούνται από αυτούς το κόστος εξυπηρέτησης τίθεται άπειρο. Στη συνέχεια επαναλαμβάνουμε τα δύο βήματα του JV όπως παρακάτω μέχρι να λυθεί το πρόβλημα.
Βήμα $1^ο$:

Δημιουργούμε ένα απλό πρόβλημα UFL με τους πελάτες που έχουν απαίτηση ίση με τη μέγιστη απαίτηση που υπάρχει στο στιγμιότυπο. Επιλύουμε αυτό το πρόβλημα με τον τρόπο που περιγράφεται στο βήμα 2.

Βήμα $2^ο$:

α. Αυξάνουμε τη δυϊκή μεταβλητή α_j κατά ένα σταθερό βήμα. Σε κάθε βήμα ελέγχουμε αν συμβαίνει κάτι από τα εξής.

1) Αν $\alpha_j - \beta_{ij} = c_{ij}$ και ο εξυπηρετητής i είναι προσωρινά ανοικτός ο πελάτης j θα συνδεθεί σε αυτό. Αν όχι θα αρχίσει να αυξάνεται και το β_{ij} μαζί με το α_j για να ισχύει ο περιορισμός.

2) Ελέγχουμε αν για κάποιο εξυπηρετητή από αυτούς που αυξήσαμε το β_{ij} ισχύει :

$$\sum_j \beta_{ij} = f_i$$

Αν συμβαίνει αυτό τότε ανοίγουμε προσωρινά αυτόν τον εξυπηρετητή και εξυπηρετούμε όσους πελάτες έχουν συνεισφέρει για το άνοιγμά του.

β. Κατασκευάζουμε ένα γράφο με κόμβους τους εξυπηρετητές και μία ακμή για κάθε πελάτη που συνεισφέρει στο άνοιγμα δύο εξυπηρετητών. Επιλέγουμε ένα maximal independent set των εξυπηρετητών που θα ανοίξουν οριστικά.

3.4 Αλγόριθμος Repeated Relaxed

Στην ίδια παρατήρηση της αποτελεσματικότητας της εκμετάλλευσης του χαλαρωμένου προβλήματος στηρίζεται και αυτός ο αλγόριθμος. Το σκεπτικό του αλγορίθμου είναι ότι αφού η πρώτη φάση του αλγορίθμου των SS δουλεύει τόσο καλά γιατί να μην το επαναλαμβάνουμε μέχρι να λυθεί το πρόβλημα. Έτσι μετά την επίλυση του χαλαρωμένου προβλήματος ανοίγουμε τους εξυπηρετητές που έχουν ανοιχθεί πλήρως από το χαλαρωμένο και συνδέουμε τους πελάτες που εξυπηρετούνται έστω και μερικά από αυτά. Οι εξυπηρετητές που ανοίχτηκαν δεν συμμετέχουν πια στο πρόβλημα ενώ οι πελάτες απομακρύνονται όταν καλυφθούν όλες οι απαιτήσεις του καθενός. Εάν δεν έχει ανοιχτεί κάποιος εξυπηρετητής πλήρως, δηλαδή $y_i < 1$ τότε ανοίγουμε τον εξυπηρετητή με το μεγαλύτερο y_i. Εάν όλα τα y_i έχουν την ίδια τιμή τότε τα ταξινομούμε με τον αριθμό των πελατών που εξυπηρετούνται έστω και μερικώς από αυτά και ανοίγουμε εξυπηρετητές με τη σειρά. Κάθε εξυπηρετητής που ανοίγει εξυπηρετεί όσους πελάτες έχουν $x_{ij} >0$ και συνεχίζουμε μέχρι να καλυφθούν όλες οι απαιτήσεις των πελατών.

Με αυτό τον τρόπο σε κάθε επίλυση του χαλαρωμένου ανοίγει τουλάχιστον ένας εξυπηρετητής και άρα το πολύ σε i επαναλήψεις θα λυθεί το πρόβλημα. Στην πράξη βέβαια χρειάζονται πολύ λιγότερες επαναλήψεις από i.

Αλγόριθμος Repeated Relaxed

Επανάληψη $1^{η}$:

Επιλύουμε το χαλαρωμένο πρόβλημα. Από τη λύση του χαλαρωμένου προβλήματος ανοίγουμε όσους εξυπηρετητές έχουν $y_i =1$. Για κάθε πελάτη με $x_{ij} >0$ και $y_i =1$ εξυπηρετείται αυτός από τον εξυπηρετητή i. Αν κανένας εξυπηρετητής δεν έχει $y_i =1$ ανοίγουμε τους εξυπηρετητές που έχουν το μέγιστο y_i . Εάν όλα τα y_i έχουν την ίδια τιμή τότε τα ταξινομούμε με τον αριθμό των πελατών που εξυπηρετούνται έστω και μερικώς από αυτά και ανοίγουμε εξυπηρετητές με τη σειρά. Κάθε εξυπηρετητής που ανοίγει εξυπηρετεί όσους πελάτες έχουν $x_{ij} >0$ και συνεχίζουμε μέχρι να καλυφθούν όλες οι απαιτήσεις των πελατών.

Μειώνουμε την απαίτηση των πελατών που εξυπηρετήθηκαν από την παρούσα επανάληψη. Οι εξυπηρετητές που ανοίξανε στην τρέχουσα επανάληψη και οι πελάτες που δεν έχουν απαίτηση να εξυπηρετηθούν από άλλον εξυπηρετητή απομακρύνονται από το πρόβλημα. Επαναλαμβάνουμε μέχρι να καλυφθούν όλες οι απαιτήσεις των πελατών.

4. ΚΥΡΙΟΤΕΡΑ ΘΕΩΡΗΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Για τους αλγόριθμους SS και JV έχει προσδιοριστεί ένα άνω φράγμα που περιορίζει την μέγιστη απόκλιση της προσεγγιστικής λύσης που δίνουν. Αν και σπάνια η πραγματική λύση του αλγορίθμου πλησιάζει αυτό το άνω φράγμα, όπως έχουν δείξει ο M. Hoefer στο [4] και οι M. Resend, R Werneck στο [5] για το UFL, είναι χρήσιμο να ξέρει όποιος χρησιμοποιεί ένα προσεγγιστικό αλγόριθμο ότι ανεξάρτητα από τη δυσκολία του στιγμιοτύπου ποτέ δεν θα ξεφύγει η λύση του από κάποιο όριο. Παρακάτω αναλύουμε σύντομα πως προκύπτει αυτό το όριο. Για μία καλύτερη ανάλυση της προσέγγισης του αλγορίθμου SS μπορεί να ανατρέξει κανείς στο [2] και για τον αλγόριθμο JV στο [1].

4.1 Αλγόριθμος SS

Ο αλγόριθμος SS αποδεικνύεται ότι έχει μία προσέγγιση έως 4 φορές την βέλτιστη τιμή του προβλήματος. Συνοπτικά αυτή η προσέγγιση προκύπτει ως εξής:

Για την πρώτη φάση, για κάθε x_{ij} ενός πελάτη που συνδέεται σε έναν εξυπηρετητή θα ισχύει (από τους περιορισμούς του δυϊκού) $\alpha_j=c_{ij}+\beta_{ij}$ και επομένως για κάθε i ισχύει:

$$\sum_j \alpha_j = \sum_j (c_{ij} + \beta_{ij}) = \sum_j c_{ij} + \sum_j \beta_{ij} = \sum_j c_{ij} + f_i + z_i$$

Άρα το κόστος της πρώτης φάσης για όλα τα i θα είναι το πολύ:

$$\sum_j n_j \alpha_j - \sum_i z_i$$

όπου n_j το πλήθος των εξυπηρετητών στους οποίους εξυπηρετείται ο πελάτης j.

Στη δεύτερη φάση ανοίγουμε r_j' εξυπηρετητές, δηλαδή όσους χρειάζονται για την κάλυψη της απαίτησης του πελάτη j. Επομένως το κόστος για το άνοιγμα των εξυπηρετητών θα είναι:

$$r_j' * (\text{average cost}) = \sum_{i \in M} y_i * (\text{average cost}) = \sum_{i \in M} f_i y_i$$

Επίσης λόγω της τριγωνικής ανισότητας ισχύει ότι κάθε άλλος πελάτης k που συνδέεται με έναν εξυπηρετητή στον οποίο δεν συνδεόταν στη λύση του relaxed θα έχει ένα κόστος εξυπηρέτησης $c_{ik} \le c_{i'k} + c_{i'j} + c_{ij} \le \alpha_k + 2\alpha_j \le 3\alpha_k$ (υπενθυμίζεται ότι $\alpha_j=c_{ij}+\beta_{ij}$ άρα $\alpha_j \ge c_{ij}$ και $\alpha_j \le \alpha_k$).

Αν αθροίσουμε όλα τα παραπάνω, δηλαδή (κόστος 1ης φάσης + κόστος εγκατάστασης 2ης φάσης + κόστος εξυπηρέτησης 2ης φάσης) έχουμε:

$$\left(\sum_j n_j \alpha_j - \sum_i z_i\right) + \sum_i f_i y_i + 3\sum_j r'_j \alpha_j \le$$

$$2\left(\sum_j n_j \alpha_j - \sum_i z_i\right) + 4\sum_j r'_j \alpha_j \le 4\left(\sum_j n_j \alpha_j - \sum_i z_i\right) = 4\,OPT$$

4.2 Αλγόριθμος JV

Ο αλγόριθμος JV έχει προσέγγιση που στην χειρότερη περίπτωση είναι ίση με 3 logk*OPT, όπου k είναι η μέγιστη τιμή της απαίτησης των πελατών. Αυτό προκύπτει ως εξής: Ο αλγόριθμος ξεκινάει από μία κενή λύση, την I_k, C_k όπου I_k είναι

το υποσύνολο των εξυπηρετητών που ανοίγει στο k βήμα του αλγορίθμου και C_k είναι το υποσύνολο των πελατών που συνδέονται σε αυτές στο ίδιο βήμα. Κάθε βήμα επεκτείνει τη λύση από την I_p, C_p στην I_{p-1}, C_{p-1} μέχρι να φτάσουμε στην I_0, C_0. Σκοπός είναι να αποδείξουμε ότι:

Το κόστος της I_{p-1}, C_{p-1} μείον το κόστος της I_p, C_p είναι το πολύ 3*OPT/p.

Από αυτό προκύπτει ότι το κόστος της τελικής λύσης I_0, C_0 είναι το πολύ 3 H_k OPT.

Σε αυτό θα καταλήξουμε με τον εξής συλλογισμό. Λόγω του τρόπου λειτουργίας κάθε βήματος ξέρουμε ότι ο περιορισμός του δυϊκού είναι “tight” και άρα ισχύει:

$$\sum_{i,j} c_{ij} x_{ij} + \sum_{i} f_i = \sum_{j} \alpha_j$$

Επίσης από την τριγωνική ανισότητα όπως στον προηγούμενο αλγόριθμο έχουμε ότι $c_{ik} \leq c_{i'k} + c_{i'j} + c_{ij} \leq \alpha_k + 2\alpha_j \leq 3\alpha_k$. Για όλα τα i,j του p βήματος θα ισχύει:

$$\sum_{i,j} c_{ij} \leq 3 * \sum_{j} \alpha_j$$

Από τα δύο τελευταία προκύπτει ότι το κόστος της I_{p-1}, C_{p-1} μείον το κόστος της I_p, C_p είναι το πολύ 3*OPTp και άρα καταλήγουμε στο ζητούμενο.

5. ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ – ΚΑΤΑΣΚΕΥΗ ΔΕΔΟΜΕΝΩΝ

Οι αλγόριθμοι που εξετάζονται υλοποιήθηκαν με τη χρήση της γλώσσας προγραμματισμού C# στο περιβάλλον ASP της MICROSOFT. Η επιλογή αυτή προτιμήθηκε γιατί συνδυάζει τις δυνατότητες μίας αντικειμενοστραφούς γλώσσας προγραμματισμού με την ευκολία παρουσίασης της WEB εφαρμογής. Επίσης χρησιμοποιήθηκε μία απλή βάση δεδομένων με δύο πίνακες, για την αποθήκευση των αποτελεσμάτων στον SQL SERVER 2008. Με αυτόν τον τρόπο τα αποτελέσματα είναι διαθέσιμα για στατιστική επεξεργασία άμεσα χωρίς να χρειάζεται ο υπολογισμός τους εκ' νέου.

Για τον υπολογισμό της βέλτιστης λύσης χρησιμοποιήθηκε ο ανεξάρτητος επιλύτης (stand alone solver) GLPSOL ο οποίος βασίζεται στη βιβλιοθήκη συναρτήσεων GLPK. Τα στιγμιότυπα που δημιουργήθηκαν λύθηκαν με τη μέθοδο B&B.

Η επίλυση των προβλημάτων έγινε σε υπολογιστή Intel 2 quad στα 2,4 Ghz με 2 Gb μνήμη. Το λειτουργικό σύστημα είναι windows 7 32-bit.

Η εφαρμογή που αναπτύχθηκε παρέχει τις εξής δυνατότητες:

1. Δημιουργία τυχαίων στιγμιοτύπων του προβλήματος FTFL με τις επιθυμητές παραμέτρους και επίλυση της βέλτιστης λύσης, του χαλαρωμένου προβλήματος και του δυϊκού του.
2. Παρουσίαση των παραπάνω αποτελεσμάτων και υπολογισμός σε αυτά τα στιγμιότυπα των αλγορίθμων των SS και JV.
3. Δημιουργία τυχαίων στιγμιοτύπων του προβλήματος OFL με τις επιθυμητές παραμέτρους και επίλυση της βέλτιστης λύσης και του χαλαρωμένου προβλήματος.
4. Παρουσίαση των αποτελεσμάτων των στιγμιοτύπων OFL.
5. Μετατροπή των προβλημάτων του ιστοχώρου ufLib του ινστιτούτου Max-Planck (http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/) [8] σε κατάλληλη μορφή και προσθήκη τυχαίας απαίτησης για κάθε πελάτη, καθώς αυτά τα προβλήματα αφορούν το UFL. Στη συνέχεια επίλυση της βέλτιστης λύσης και του χαλαρωμένου προβλήματος.
6. Καταχώρηση στη βάση δεδομένων μίας ομάδας στιγμιοτύπων που κατασκευάστηκαν με τις ίδιες παραμέτρους. Για κάθε στιγμιότυπο αποθηκεύεται η βέλτιστη τιμή, η τιμή του χαλαρωμένου προβλήματος και η τιμή που υπολόγισε κάθε ένας από τους εξεταζόμενους αλγόριθμους.
7. Στατιστικά στοιχεία για τα δεδομένα που έχουν αποθηκευτεί στη ΒΔ όπως:
   α. Διασπορά του σφάλματος για κάθε αλγόριθμο. Ως σφάλμα ενός στιγμιοτύπου ορίζουμε το πηλίκο (λύση – βέλτιστη λύση / βέλτιστη λύση)
   β. Συμπεριφορά του integrality gap (IG) σε σχέση με το σφάλμα. Ως IG χρησιμοποιούμε το λόγο (βέλτιστη ακέραια λύση προς λύση του χαλαρωμένου προβλήματος). Επειδή η τιμή του IG είναι στις περισσότερες περιπτώσεις πολύ μικρή, στα διαγράμματα την απεικονίζουμε πολλαπλασιασμένη επί 10000, ώστε να μην έχουμε αμελητέα νούμερα.
   γ. Συμπεριφορά του IG σε σχέση με το κόστος εγκατάστασης και το κόστος εξυπηρέτησης.
   δ. Συμπεριφορά του σφάλματος του αλγορίθμου JV σε σχέση με την απαίτηση των πελατών.
8. Μελέτη του αριθμού πελατών που εξυπηρετούνται από κάθε εξυπηρετητή και του πλήθους των διαφορετικών τιμών x_{ij} σε κάθε στιγμιότυπο.

5.1 Σχήμα της Βάσης Δεδομένων

Η βάση δεδομένων που σχεδιάστηκε αποτελείται από τις δύο σχέσεις που απεικονίζονται παρακάτω. Αυτές χρησιμοποιούνται ώς εξής: Στη σχέση Folders αποθηκεύονται δεδομένα που περιγράφουν την κατασκευή ενός τύπου στιγμιοτύπων, πχ ο αριθμός πελατών, ο αριθμός εξυπηρετητών κλπ. Επίσης αποθηκεύονται συγκεντρωτικά αποτελέσματα της επίλυσης κάθε αλγορίθμου όπως το μέσο σφάλμα της προσεγγιστικής λύσης. Στη σχέση Instances αποθηκεύονται δεδομένα του κάθε στιγμιοτύπου όπως η τιμή της λύσης που έδωσε κάθε αλγόριθμος καθώς και το μέσο κόστος εγκατάστασης και εξυπηρέτησης. Κάθε στιγμιότυπο ανήκει σε ένα τύπο στιγμιοτύπων από αυτούς που περιγράφονται στη σχέση Folders και αυτό απεικονίζεται με το εξωτερικό κλειδί FOLDER_ID.

5.2 Υλοποίηση SS αλγόριθμου

Ο αλγόριθμος SS υλοποιήθηκε σε μία μέθοδο η οποία διαβάζει την λύση του χαλαρωμένου και του δυϊκού καθώς και τα δεδομένα του στιγμιοτύπου από το dat αρχείο. Στη συνέχεια τα αποθηκεύει σε δομές στη μνήμη και εκτελεί την πρώτη φάση του αλγορίθμου, η οποία είναι αρκετά απλή στην υλοποίηση. Ακολουθεί η δεύτερη φάση με τα τρία βήματα που την αποτελούν και τέλος υπολογίζεται το κόστος της λύσης το οποίο επιστρέφεται στην καλούσα διαδικασία.

Πριν επιστρέψει το κόστος η μέθοδος ελέγχει την ορθότητα της λύσης που υπολογίστηκε και επιστρέφει -1 εάν είναι λανθασμένη η λύση και -2 εάν δεν έχει υπολογιστεί η λύση του δυϊκού.

5.3 Υλοποίηση JV αλγόριθμου

Ο αλγόριθμος JV υλοποιήθηκε σε μία μέθοδο η οποία διαβάζει τα δεδομένα του στιγμιοτύπου από το dat αρχείο. Η υλοποίηση της σταδιακής αύξησης του α_{ij} έγινε με βήμα που μπορεί να επιλεγεί από το χρήστη μεταξύ των τιμών (0,001 0,01 0,1 1 10 100). Στην πράξη η επιλογή του βήματος σχετίζεται με την τάξη μεγέθους του κόστους εξυπηρέτησης και του κόστους εγκατάστασης, γιατί αν επιλεγεί πολύ μικρό βήμα ο αλγόριθμος καθυστερεί χωρίς κέρδος στην ακρίβεια, ενώ αν επιλεγεί πολύ μεγάλο δεν προσεγγίζει την βέλτιστη λύση.

Για την εύρεση του maximal independent set χρησιμοποιήθηκε ο αλγόριθμος greedy ελαχίστου βαθμού η υλοποίηση του οποίου περιλαμβάνεται στον κώδικα της μεθόδου στο παράρτημα Α.

5.4 Υλοποίηση SSJV αλγόριθμου

Η υλοποίηση αυτού του αλγορίθμου αποτελείται από το συγκερασμό των δύο προηγούμενων και δεν περιλαμβάνει κάτι νέο. Όπως και πριν διαβάζουμε τη λύση του χαλαρωμένου και κρατάμε τους εξυπηρετητές που έχουν $y_i=1$ και σε αυτούς συνδέουμε όσα $x_{ij}>0$. Στη συνέχεια θέτουμε μηδενικό κόστος εγκατάστασης σε αυτούς τους εξυπηρετητές και πολύ μεγάλο c_{ij} που αναπαριστά το άπειρο για τους πελάτες που εξυπηρετούνται από αυτούς και συνεχίζουμε με τον αλγόριθμο JV για να λύσουμε το πρόβλημα όπως ακριβώς λειτουργεί αρχικά ο αλγόριθμος. Τέλος ελέγχεται η λύση για την ορθότητα της, δηλαδή αν καλύπτονται όλες οι απαιτήσεις κάθε πελάτη και επιστρέφεται το κόστος της λύσης που βρέθηκε.

5.5 Υλοποίηση RR αλγόριθμου

Η υλοποίηση αυτού του αλγορίθμου, πρώτα διαβάζει την λύση του χαλαρωμένου προβλήματος και κρατάει τους εξυπηρετητές που ανοίγουν και τους πελάτες που συνδέονται σε αυτούς. Στη συνέχεια διαβάζει τη λύση του επόμενου χαλαρωμένου προβλήματος που δεν περιλαμβάνει τους προηγούμενους εξυπηρετητές καθώς και τους πελάτες που έχουν ικανοποιήσει τις απαιτήσεις τους. Αυτό επαναλαμβάνεται μέχρι να μην υπάρχουν πελάτες που δεν έχουν καλύψει όλες τις απαιτήσεις τους.

Σε κάθε βήμα ανοίγουν όσοι εξυπηρετητές έχουν τιμή y_i ίση με το μέγιστο y_i που παρουσιάζεται στη λύση. Ένα πρόβλημα που παρουσιάστηκε σε κάποιο στιγμιότυπο κατά τις δοκιμές του αλγορίθμου, ήταν ότι άνοιξαν πολλοί εξυπηρετητές και είχαν όλοι $y_i=0,5$. Σε αυτή την περίπτωση δεν ανοίγει όλους τους εξυπηρετητές γιατί κάτι τέτοιο θα ανέβαζε πολύ το κόστος της λύσης. Αντίθετα, ταξινομεί τους εξυπηρετητές με τον αριθμό των πελατών που εξυπηρετούνται από τον καθένα. Ανοίγουμε τον εξυπηρετητή που εξυπηρετεί τους περισσότερους πελάτες και τους συνδέουμε σε αυτόν. Επαναλαμβάνεται το ίδιο μέχρι να καλυφθούν όλοι οι πελάτες. Τέλος ελέγχεται η ορθότητα της λύσης και επιστρέφεται το κόστος της.

5.6 Κατασκευή Στιγμιοτύπων FTFL

Στην παρούσα μελέτη χρησιμοποιήθηκαν τρία είδη στιγμιοτύπων.

1. Το πρώτο είδος αφορά στιγμιότυπα στα οποία το κόστος εξυπηρέτησης προσδιορίστηκε υπολογίζοντας την ευκλείδεια απόσταση των πελατών από κάθε εξυπηρετητή έτσι ώστε να τηρείται η τριγωνική ανισότητα. Πιο αναλυτικά, επιλέγεται

σαν παράμετρος το μέγεθος ενός τετραγώνου. Μέσα σε αυτό το τετράγωνο τοποθετούνται τυχαία όλοι οι πελάτες και οι εξυπηρετητές. Η απόσταση ενός πελάτη από έναν εξυπηρετητή είναι το κόστος εξυπηρέτησης αυτού του πελάτη από το συγκεκριμένο εξυπηρετητή. Σε ότι αφορά το κόστος εγκατάστασης επιλέγεται μία μέγιστη τιμή και το κόστος κάθε εξυπηρετητή ορίζεται τυχαία και ομοιόμορφα μεταξύ του 0 και του μέγιστου. Επίσης το σύστημα δέχεται σαν παράμετρο τη μέγιστη απαίτηση των πελατών και ορίζει τυχαία με κανονική κατανομή για τον καθένα μία τιμή μεταξύ του 1 και αυτής της μέγιστης τιμής.

Με αυτή τη μέθοδο κατασκευάστηκαν και επιλύθηκαν 5900 στιγμιότυπα από 40 διαφορετικούς τύπους. Οι τύποι των στιγμιοτύπων που κατασκευάστηκαν φαίνονται στο Παράρτημα Β. Κάθε στιγμιότυπο επιλύθηκε με τη μέθοδο B&B για τον προσδιορισμό της βέλτιστης λύσης καθώς και με τους 4 αλγορίθμους που υλοποιήθηκαν και περιγράφονται παραπάνω.

Καθώς μεγαλώνει η διάσταση του προβλήματος περιορίσαμε τη μελέτη σε μικρότερο αριθμό στιγμιοτύπων καθώς απαιτείται περισσότερος χρόνος για την επίλυση των προβλημάτων και ιδιαίτερα της βέλτιστης λύσης, αφού ο αλγόριθμος για την μέθοδο B&B είναι εκθετικής πολυπλοκότητας.

2. Από την επίλυση αυτών των στιγμιοτύπων έγινε φανερό ότι μεγάλος αριθμός προβλημάτων έχουν ακέραιες λύσεις που υπολογίζονται από τη μέθοδο simplex χωρίς να απαιτείται η εφαρμογή B&B, όπως περιγράφεται αναλυτικά στα επόμενα κεφάλαια. Αυτή η παρατήρηση οδήγησε στην περαιτέρω μελέτη της δυσκολίας του προβλήματος και την ανάλυση του IG σε σχέση με το κόστος εγκατάστασης, το κόστος εξυπηρέτησης και την αναλογία μεταξύ τους. Για αυτό το λόγο κατασκευάστηκαν στιγμιότυπα με τον εξής τρόπο:

Το κόστος εγκατάστασης και το κόστος εξυπηρέτησης υπολογίζονται όπως προηγουμένως, με μέγιστη τιμή 100 για το κόστος εγκατάστασης και διάσταση του τετραγώνου 100 για το κόστος εξυπηρέτησης. Στη συνέχεια ορίζεται μία μεγάλη τιμή από τον χρήστη και το σύστημα επιλέγει, ομοιόμορφα σε αυτό το εύρος, μία τυχαία τιμή για κάθε στιγμιότυπο την οποία προσθέτει σε όλα τα κόστη εγκατάστασης. Ομοίως επιλέγει μία τυχαία τιμή και την προσθέτει σε όλα τα κόστη εξυπηρέτησης του συγκεκριμένου στιγμιοτύπου. Ο στόχος είναι να δημιουργήσουμε ένα πλήθος στιγμιότυπα στα οποία ο λόγος του μέσου κόστους εγκατάστασης προς το μέσο κόστος εξυπηρέτησης θα έχει μεγάλη διακύμανση για να δούμε εάν και πώς αυτό επηρεάζει τη δυσκολία του προβλήματος.

Με αυτόν τον τρόπο κατασκευάστηκαν τα παρακάτω στιγμιότυπα:

100 πελάτες, 100 εξυπηρετητές, r_j από 1 έως 4, f_i από 0 έως 100, c_{ij} από 0 έως 100, εκτροπή από 0 έως 10000, 100 στιγμιότυπα
100 πελάτες, 100 εξυπηρετητές, r_j από 1 έως 4, f_i από 0 έως 100, c_{ij} από 0 έως 100, εκτροπή από 0 έως 100000, 100 στιγμιότυπα
100 πελάτες, 100 εξυπηρετητές, r_j από 1 έως 4, f_i από 0 έως 100, c_{ij} από 0 έως 100, εκτροπή από 0 έως 1000000, 100 στιγμιότυπα

Επίσης κατασκευάστηκαν 600 στιγμιότυπα με 100 πελάτες, 100 εξυπηρετητές, r_j από 1 έως 8, στα οποία το μέγιστο κόστος εγκατάστασης κυμαινόταν τυχαία για κάθε στιγμιότυπο από 0 έως 1000 και ομοίως το τετράγωνο του κόστους εξυπηρέτησης επιλεγόταν τυχαία στο διάστημα από 0 έως 1000.

3. Τέλος χρησιμοποιήθηκαν κάποια στιγμιότυπα από την βιβλιοθήκη UfLib στη διεύθυνση http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/ του ινστιτούτου Max-Planck, τα οποία τροποποιήθηκαν ώστε να είναι κατάλληλα για

το πρόβλημα FTFL. Πιο συγκεκριμένα, προστέθηκε για κάθε πελάτη μία τυχαία απαίτηση από 1 έως 5 και διαμορφώθηκε το αρχείο στη δομή που αναγνωρίζει το GLPSOL. Λόγω της δυσκολίας των συγκεκριμένων στιγμιοτύπων λύθηκε περιορισμένος αριθμός. Ενδεικτικά, στιγμιότυπα του τύπου Koerkel Gosh assymetrik με διάσταση 250 x 250 χρειάστηκαν περί τις 50 ώρες για την επίλυση της βέλτιστης λύσης ενώ μεγαλύτερα στιγμιότυπα τύπου k-median με διάσταση 500 x 500 δεν κατέστη δυνατό να λυθούν. Άλλωστε και στην ιστοσελίδα από την οποία λήφθηκαν τα στιγμιότυπα [8] δεν υπάρχουν λύσεις για όλα τα προβλήματα, παρόλο ότι είναι ευκολότερη η επίλυση στην μορφή του UFL.

Οι λύσεις που παρέχονται στην ιστοσελίδα [8] χρησιμοποιήθηκαν και σαν επαλήθευση της ορθότητας των λύσεων του GLPSOL. Θέτοντας σαν απαίτηση σε όλους τους πελάτες 1 εκφυλίζεται το πρόβλημα στο UFL και πλέον η λύση του solver είναι ίδια με τη λύση του στιγμιοτύπου που παρέχεται από την UfLib σε όλες τις περιπτώσεις που ελέγχθηκαν. Αναλυτικά τα στιγμιότυπα που εξετάστηκαν είναι:

α. 100 στιγμιότυπα τύπου Bilde-Krarup [15] με διάσταση 30 x 80 και 120 με διάσταση 50 x 100. Πρόκειται για μη μετρικά στιγμιότυπα σχετικά μικρών διαστάσεων. Τα κόστη εγκατάστασης τίθενται ίδια για όλα τους εξυπηρετητές ενώ τα κόστη εξυπηρέτησης επιλέγονται τυχαία με κανονική κατανομή στο διάστημα 0-1000.

β. 50 στιγμιότυπα του τύπου Galvao-Raggi [16] σε διαστάσεις 50 x 50, 70 x 70, 100 x 100, 150 x 150, 200 x 200. Είναι μετρικά στιγμιότυπα που κατασκευάζονται με τη χρήση ενός γράφου με m*n ακμές. Κάθε ακμή έχει ένα κόστος το οποίο επιλέγεται τυχαία στο [0,n] εκτός από την n=150 η οποία επιλέγεται στο [0,500]. Το c_{ij} υπολογίζεται σαν το ελάχιστο μονοπάτι από το i στο j. Για τον υπολογισμό του κόστους εγκατάστασης χρησιμοποιείται κανονική κατανομή σε δοθέν διάστημα.

γ. 30 στιγμιότυπα του τύπου chessboard [8] τα οποία έχουν διάσταση 144x144 και σταθερό κόστος εγκατάστασης 3000 ενώ τα κόστη εξυπηρέτησης υπολογίζονται σε τόρο, και επομένως δεν είναι μετρικά.

δ. 30 στιγμιότυπα 100x100 με κόστος εξυπηρέτησης υπολογισμένο από την ευκλείδεια απόσταση [8] σε τετράγωνο μεγέθους 7000 και σταθερό κόστος εγκατάστασης 3000.

ε. 30 στιγμιότυπα τύπου uniform [8] 100x100 με σταθερό κόστος εγκατάστασης 3000 και τα κόστη εξυπηρέτησης επιλέγονται τυχαία με κανονική κατανομή στο διάστημα 0-10000.

στ. 15 στιγμιότυπα τύπου Large Duality Gap [18] 100x100 όπου έχουμε σταθερό κόστος εγκατάστασης 3000 και τα κόστη εξυπηρέτησης κάθε πελάτη έχουν τιμή στο (1,2,3,4) για 10 εξυπηρετητές ενώ όλα τα άλλα έχουν κόστος 3000.

ζ. 15 στιγμιότυπα τύπου Large Duality Gap [18] 100x100 όπου έχουμε σταθερό κόστος εγκατάστασης 3000 και τα κόστη εξυπηρέτησης κάθε εξυπηρετητή έχουν τιμή στο (1,2,3,4) για 10 πελάτες ενώ όλοι οι άλλοι έχουν κόστος 3000.

Επίσης επιλύθηκαν ενδεικτικά από 1 στιγμιότυπο των τύπων Koerkel Gosh assymetrik [17], Koerkel Gosh symetrik [17], Perfect Codes, Finite projective planes [18] αλλά δεν ήταν δυνατή η λύση περισσότερων, λόγω της δυσκολίας που παρουσιάζουν στον υπολογισμό της βέλτιστης λύσης, ώστε να εξαχθούν χρήσιμα αποτελέσματα. Εξάλλου οι εξεταζόμενοι αλγόριθμοι αφορούν μετρικά στιγμιότυπα και μόνο τα Galvao-Raggi και τα ευκλείδεια ανήκουν σε αυτή την κατηγορία.

5.7 Κατασκευή Στιγμιοτύπων OFL

Για την κατασκευή στιγμιοτύπων για το πρόβλημα OFL ακολουθήθηκε η ίδια διαδικασία σε ότι αφορά τα κόστη εξυπηρέτησης. Για τα κόστη εγκατάστασης, όπως προαναφέρθηκε έχει νόημα το πρόβλημα μόνο εάν είναι πιο συμφέρουσα η επιλογή μεγαλύτερων διαστημάτων ενοικίασης, αλλιώς το πρόβλημα εκφυλίζεται σε ένα facility location ανά ημέρα. Έτσι ορίζεται ένα τυχαίο κόστος για την πρώτη ημέρα και σε κάθε επόμενη προστίθεται ένα ποσοστό αυτού που κυμαίνεται τυχαία μεταξύ 0,2 και 0,5 του κόστους της πρώτης μέρας. Επίσης ένας πελάτης έχει πιθανότητα 30% να έχει απαίτηση σε μία ημέρα και κάθε διάστημα ενοικίασης είναι διαθέσιμο μία συγκεκριμένη μέρα με πιθανότητα 30%. Αυτό ενέχει τον κίνδυνο να μην είναι δυνατή η εξυπηρέτηση των πελατών από κανένα εξυπηρετητή για κάποια μέρα, και επομένως κάποιο στιγμιότυπο να μην έχει εφικτή λύση. Στην πράξη η πιθανότητα να συμβεί κάτι τέτοιο είναι πολύ μικρή και δεν εμφανίστηκε σε κανένα στιγμιότυπο από όσα δοκιμάστηκαν. Ακόμα και αν δημιουργηθεί κάποιο στιγμιότυπο που να μην έχει εφικτή λύση, αυτό δεν δημιουργεί πρόβλημα αφού μπορούμε να το αντικαταστήσουμε με άλλο που να μην έχει αυτό το ελάττωμα.

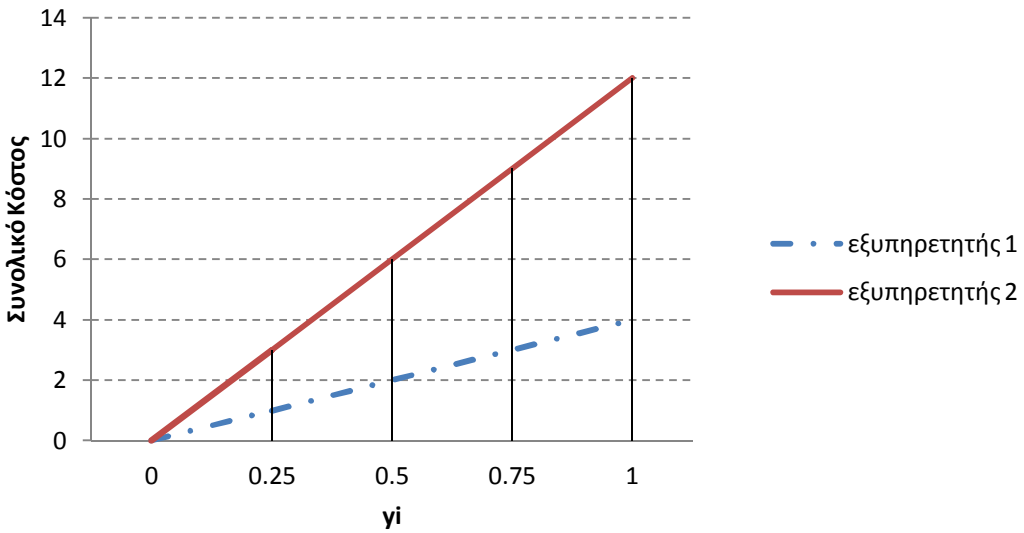
6. ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

6.1 Η δυσκολία του προβλήματος FTFL

Η πρώτη εντύπωση που αποκομίζει κάποιος διαβάζοντας την διατύπωση του προβλήματος είναι ότι πρόκειται για ένα δύσκολο πρόβλημα που θα έχει πραγματικές τιμές ακόμα και σε μικρά στιγμιότυπα. Με βάση αυτή την εντύπωση, ήταν πραγματική έκπληξη, η διαπίστωση ότι είναι δύσκολο να κατασκευάσει κανείς μετρικά στιγμιότυπα σε μικρή διάσταση που να μην έχουν ακέραιες τιμές ήδη από τη λύση του χαλαρωμένου προβλήματος.

Αν το σκεφτούμε λίγο περισσότερο όμως, καταλαβαίνουμε γιατί συμβαίνει αυτό. Ας υποθέσουμε ένα στιγμιότυπο με ένα πελάτη και δύο εξυπηρετητές. Η απαίτηση του πελάτη είναι 1 και μπορεί να καλυφθεί από οποιονδήποτε από τους δύο. Ισχυριζόμαστε ότι αυτό το στιγμιότυπο δεν μπορεί να έχει πραγματική λύση.

Αυτός ο ισχυρισμός προκύπτει από την διαπίστωση ότι αυξάνοντας το y_i του $1^{ου}$ εξυπηρετητή και το αντίστοιχο x_{ij}, αυξάνεται γραμμικά το συνολικό κόστος. Το ίδιο ισχύει και για τον δεύτερο εξυπηρετητή. Επομένως δεν υπάρχει συνδυασμός του y_1 και του y_2 που να δίνει μικρότερο κόστος από το συνολικό κόστος του y_1 (4 στο συγκεκριμένο παράδειγμα). Έτσι γίνεται κατανοητό ότι δεν μπορούμε να κατασκευάσουμε για οποιεσδήποτε τιμές ένα τέτοιο στιγμιότυπο που να έχει πραγματικές τιμές στη λύση του χαλαρωμένου, όπως φαίνεται και από το επόμενο σχήμα.



Ας το δούμε και από την άλλη πλευρά. Σε ποια περίπτωση θα έχει ένα στιγμιότυπο πραγματικές τιμές; Ένα παράδειγμα (σίγουρα όχι το μόνο) του πως μπορεί να συμβεί αυτό είναι το εξής: ένας πελάτης μπορεί να εξυπηρετηθεί από ένα ανοιχτό εξυπηρετητή με κόστος k. Επίσης μπορεί να εξυπηρετηθεί από ένα άλλο κλειστό εξυπηρετητή με κόστος k-ε. Αν υπάρχουν και άλλοι πελάτες διαθέσιμοι να δώσουν από ε/2 ο καθένας, μπορούν να ανοίξουν εν μέρει, πχ 50% τον δεύτερο εξυπηρετητή και έτσι να εξυπηρετηθούν από εκεί φθηνότερα.

Για να συμβεί μία τέτοια συγκυρία θα πρέπει να έχουμε αρκετά μεγάλη διάσταση του προβλήματος και κάποιους περιορισμούς στα κόστη εγκατάστασης και εξυπηρέτησης. Στον πίνακα 1 που ακολουθεί φαίνονται τα ποσοστά πραγματικών

στιγμιοτύπων από 6020 στιγμιότυπα. Στην στήλη «πελάτες» αναγράφεται ο αριθμός πελατών στα στιγμιότυπα αυτού του τύπου και στη στήλη «εξυπηρετητές» ο αριθμός των εξυπηρετητών. Η στήλη «Μέγιστη απαίτηση» δείχνει το διάστημα στο οποίο επιλέγεται με κανονική κατανομή η απαίτηση κάθε πελάτη να εξυπηρετηθεί από r διαφορετικούς εξυπηρετητές. Το κόστος εγκατάστασης σε όλα τα στιγμιότυπα επιλέγεται τυχαία από 0 έως 100 και ομοίως το κόστος εξυπηρέτησης επιλέγεται τυχαία μέχρι τον αριθμό που φαίνεται στην στήλη «Κόστος εξυπηρέτησης». Στην τελευταία στήλη φαίνεται το ποσοστό των στιγμιοτύπων για κάθε τύπο, που έχουν πραγματικές λύσεις, δηλαδή που η λύση του χαλαρωμένου προβλήματος δεν συμπίπτει με τη βέλτιστη.

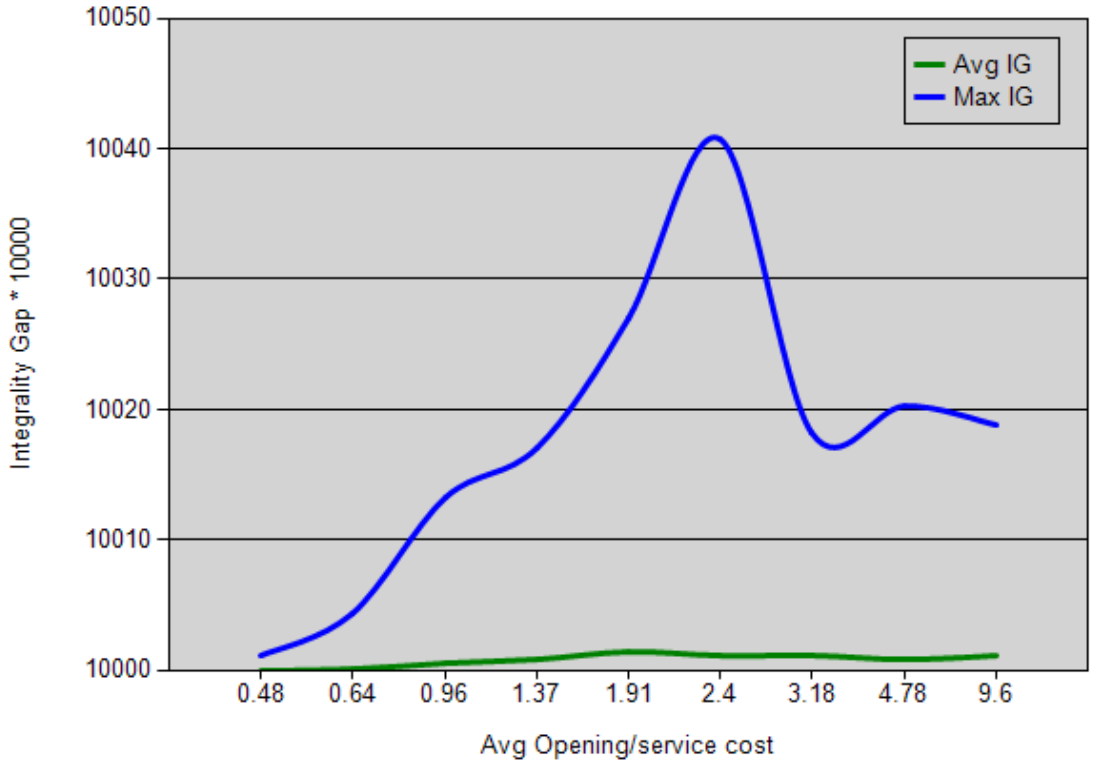
Είναι προφανές ότι η διάσταση του προβλήματος παίζει καθοριστικό ρόλο στην πιθανότητα να έχουμε πραγματικά προβλήματα, αφού μόνο στη μεγαλύτερη διάσταση 400x200 έχουμε το 50% των προβλημάτων να έχουν πραγματικές τιμές ενώ κάθε άλλο είδος στιγμιοτύπων έχει χαμηλότερο ποσοστό από αυτό. Επίσης η σχέση κόστους εγκατάστασης και εξυπηρέτησης φαίνεται να παίζει κάποιο ρόλο, αν παρατηρήσουμε προβλήματα ίδιων διαστάσεων.

Πιο αναλυτικά, για να μελετηθεί η σχέση κόστους εγκατάστασης και εξυπηρέτησης έγιναν τα διαγράμματα 1 και 2 που δείχνουν πως συμπεριφέρεται το μέσο και το μέγιστο IG καθώς μεταβάλλεται η σχέση κόστους εγκατάστασης και εξυπηρέτησης. Κάθε σημείο του διαγράμματος αποτελείται από μία ομάδα 300 στιγμιοτύπων κατασκευασμένων με τον ίσιο τρόπο. Υπενθυμίζεται ότι σαν IG ορίζεται ο λόγος (βέλτιστη ακέραια λύση προς λύση του χαλαρωμένου προβλήματος).

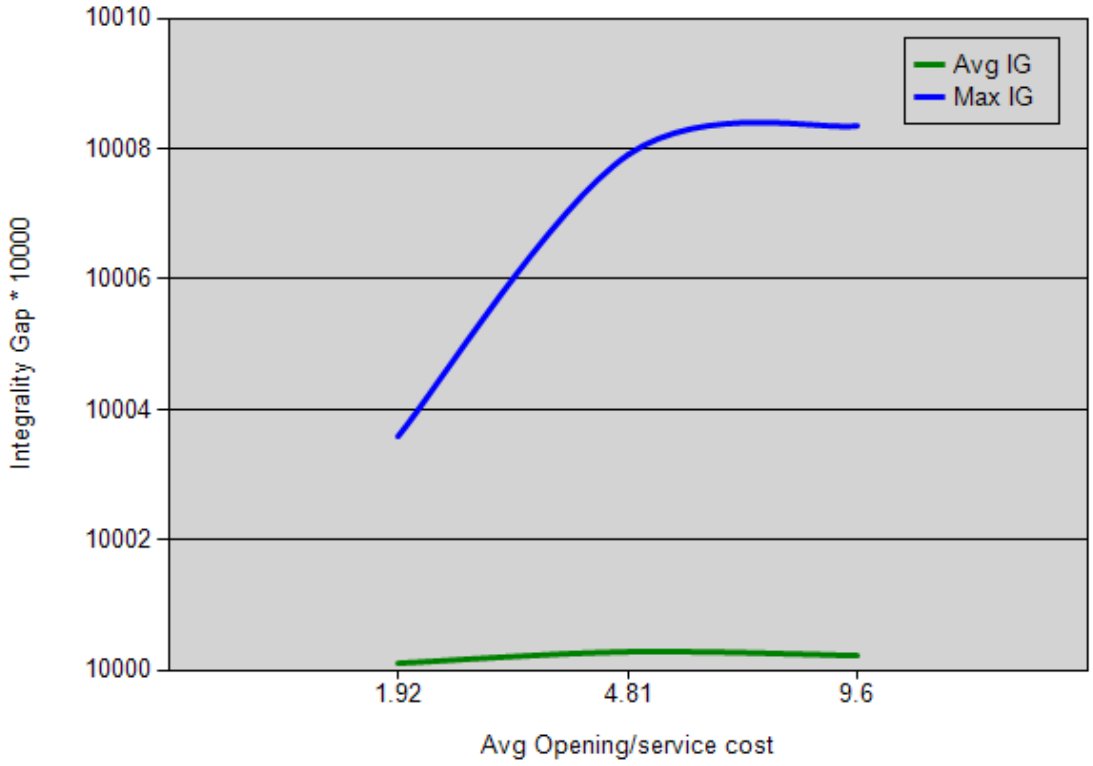
Πίνακας 1. Ποσοστό πραγματικών προβλημάτων σε τυχαία στιγμιότυπα

Πελάτες	Εξυπηρετητές	Μέγιστη απαίτηση	Κόστος εξυπηρέτησης	Ποσοστό % χαλαρωμένων προβλημάτων με x_{ij} στο R
50	50	5	20	14,3333333
50	50	5	50	15,3333333
50	50	5	70	15,6666667
100	200	10	50	23
100	200	10	20	23
100	200	30	20	8
100	200	30	50	22
100	200	5	20	32
100	200	5	50	32
100	300	10	20	24
100	300	10	50	26
100	300	30	20	20
100	300	30	50	26
100	300	5	20	40
100	300	5	50	28
100	50	10	10	10,3333333
100	50	10	20	14
100	50	10	50	8
100	50	5	10	14
100	50	5	100	20
100	50	5	150	8,33333333
100	50	5	20	15,3333333
100	50	5	200	2
100	50	5	30	24
100	50	5	40	19,6666667
100	50	5	50	26,3333333
100	50	5	70	23,3333333
150	50	5	20	29
200	50	5	20	24,5
250	50	5	20	20
300	100	5	50	44
400	200	5	50	50
Σύνολο				18,14

Διάγραμμα 1. Σχέση μέσου και μέγιστου IG με το λόγο κόστους εγκατάστασης προς κόστος εξυπηρέτησης για μέγιστη απαίτηση = 5 σε στιγμιότυπα 100x50.



Διάγραμμα 2. Σχέση μέσου και μέγιστου IG με το λόγο κόστους εγκατάστασης προς κόστος εξυπηρέτησης για μέγιστη απαίτηση = 10 σε στιγμιότυπα 100x50.



Τα συμπεράσματα που εξάγονται από τα διαγράμματα 1 και 2 είναι ότι επηρεάζει ο λόγος κόστους εγκατάστασης προς κόστος εξυπηρέτησης το μέγιστο και το μέσο IG και κατά συνέπεια το ποσοστό των προβλημάτων που έχουν πραγματικές τιμές, και παρουσιάζει ένα μέγιστο, διαφορετικό για κάθε νούμερο απαίτησης των πελατών. Όταν η απαίτηση έχει κανονική κατανομή από 1 έως 5, δηλαδή μία μέση τιμή 3, το μέγιστο αυτό παρουσιάζεται στο 1,91. Αντίστοιχα για απαίτηση από 1 έως 10 το μέγιστο μετακινείται στο 4,81. Αυτό εξηγείται ως εξής: τα ποιο «δύσκολα» προβλήματα παρουσιάζονται όταν το συνολικό κόστος εγκατάστασης και το συνολικό κόστος εξυπηρέτησης έχουν συγκρίσιμες τιμές. Αν αυξήσουμε τη μέση απαίτηση τότε για να παραμείνουν κοντινές οι τιμές θα πρέπει να μειωθεί το μέσο κόστος εξυπηρέτησης.

Παρ' όλο το γεγονός ότι υποχωρούν σημαντικά τα ποσοστά των δύσκολων προβλημάτων όταν ξεφεύγουμε από αυτή την ιδανική αναλογία, αυτό δεν σημαίνει ότι δεν έχουμε πραγματικά προβλήματα σε άλλες περιπτώσεις. Για να μελετηθεί η δυσκολία του προβλήματος σε ακραίες τιμές του λόγου κόστους εγκατάστασης προς κόστος εξυπηρέτησης, κατασκευάστηκαν στιγμιότυπα στα οποία υπάρχει μεγάλη διακύμανση στο λόγο αυτό, με τον τρόπο που περιγράφεται στο τμήμα: Κατασκευή Στιγμιοτύπων FTFL. Τα αποτελέσματα παρουσιάζονται παρακάτω.

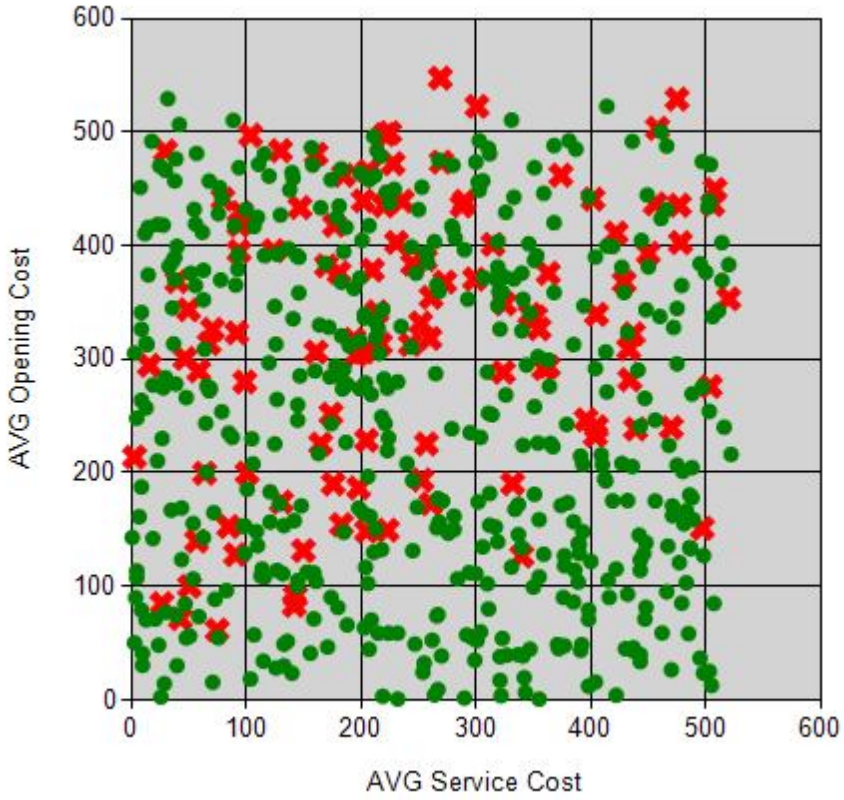
Στα διαγράμματα 3 και 4 έχουμε 600 στιγμιότυπα 100x100 με μέγιστη απαίτηση 8 και τα κόστη εγκατάστασης και εξυπηρέτησης να κυμαίνονται από 0 έως 1000. Στο διάγραμμα 3 φαίνονται με (κόκκινους) σταυρούς τα προβλήματα που έχουν πραγματικές τιμές ενώ στο διάγραμμα 4 μόνο αυτά για τα οποία το IG, δηλαδή η διαφορά ακεραίας μείον χαλαρωμένης λύσης προς την ακέραια είναι μεγαλύτερο από 0,0001. Τα διαγράμματα 5 και 6 είναι τα αντίστοιχα για στιγμιότυπα 100x100 με απαίτηση 4 και διακύμανση του κόστους εγκατάστασης και εξυπηρέτησης σε ένα εύρος 100 στο οποίο προστίθεται ένας αριθμός 0-10000.

Για τα διαγράμματα 3 και 4 παρατηρούμε ότι τα πραγματικά προβλήματα έχουν μία διασπορά που καλύπτει σχεδόν όλους τους συνδυασμούς μέσης τιμής κόστους εγκατάστασης και εξυπηρέτησης. Η μόνη περιοχή που δεν εμφανίζονται πολλά πραγματικά προβλήματα είναι όταν μεγαλώνει πολύ το κόστος εξυπηρέτησης, οπότε η λύση του προβλήματος είναι προφανής (ανοίγει για κάθε πελάτη τους πιο κοντινούς εξυπηρετητές αδιαφορώντας για το κόστος εγκατάστασης που είναι τάξεις μεγέθους μικρότερο). Εδώ η έκπληξη είναι ότι θα αναμέναμε να είναι εύκολη η λύση και των προβλημάτων στο άλλο άκρο, δηλαδή με μεγάλο κόστος εγκατάστασης. Η απάντηση είναι ότι δε συμβαίνει κάτι τέτοιο γιατί προφανώς θα ανοίξουμε λίγους μόνο εξυπηρετητές αλλά το ποιοί θα είναι αυτοί δεν είναι εύκολο να καθοριστεί. Μάλιστα στο διάγραμμα 6 σε αυτή την περιοχή βρίσκονται όλα τα προβλήματα με το μεγαλύτερο IG.

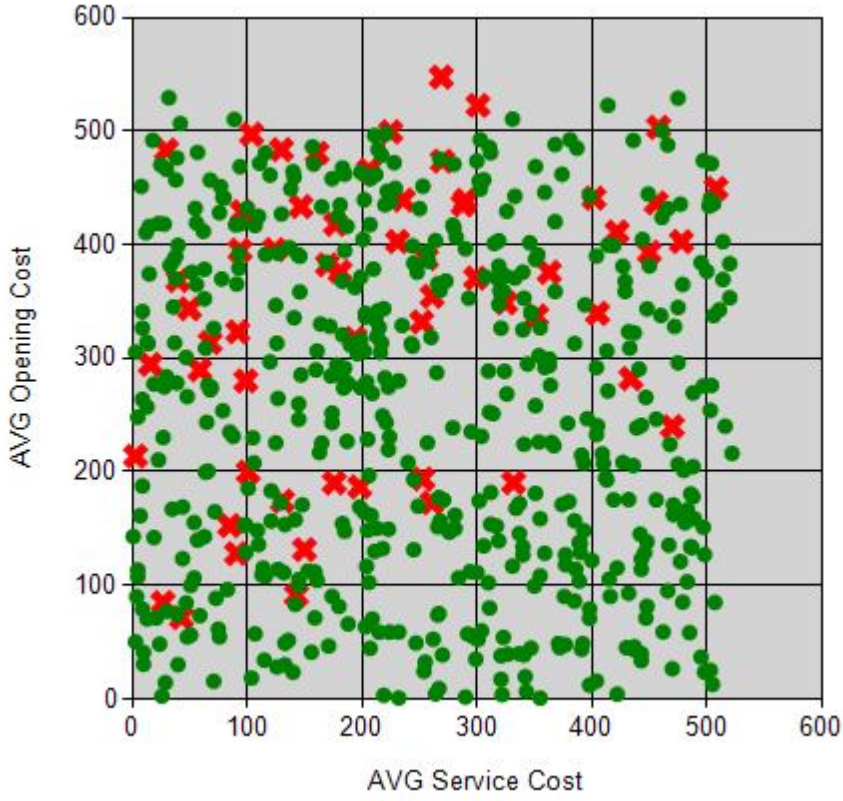
Τα ίδια συμπεράσματα εξάγονται και από τον πίνακα 2 που δείχνει ότι έχουμε πραγματικά προβλήματα ακόμα και για ακραίες τιμές του λόγου κόστος εγκατάστασης προς κόστος εξυπηρέτησης. Τα δεδομένα αυτά αφορούν στιγμιότυπα 100x100 με απαίτηση 4 και διακύμανση του κόστους εγκατάστασης και εξυπηρέτησης σε ένα εύρος 100 στο οποίο προστίθεται ένας αριθμός 0-1000000.

Τα παραπάνω μπορεί να σκεφτεί κανείς ότι αντιτίθενται στα συμπεράσματα που εξήχθησαν από τα διαγράμματα 1 και 2 (ο αριθμός πραγματικών προβλημάτων μειώνεται όταν ξεφεύγουμε από μία ιδανική αναλογία κόστους εγκατάστασης προς κόστος εξυπηρέτησης). Υπενθυμίζουμε ότι εδώ μελετώνται προβλήματα με διαφορετικό τρόπο κατασκευής και επομένως δεν υπάρχει πεδίο σύγκρισης.

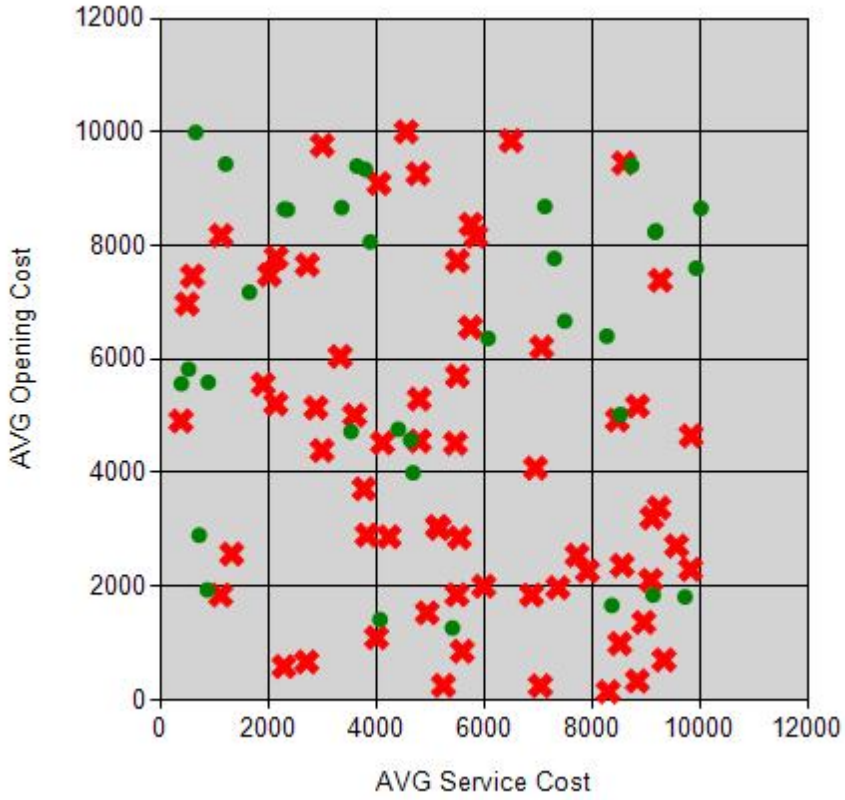
Διάγραμμα 3. Διασπορά πραγματικών στιγμιοτύπων με μέγιστο κόστος εγκατάστασης και κόστος εξυπηρέτησης 1000 και μέγιστη απαίτηση 8.



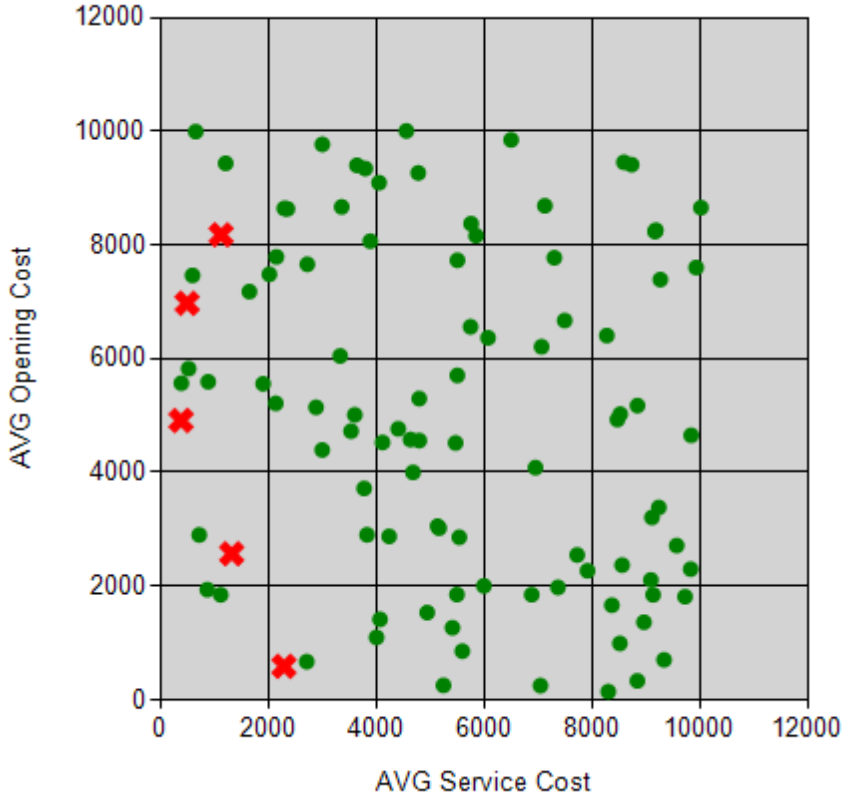
Διάγραμμα 4. Διασπορά των στιγμιοτύπων του διαγράμματος 3 με IG>1,0001



Διάγραμμα 5. Διασπορά πραγματικών στιγμιοτύπων με μέγιστο κόστος εγκατάστασης και κόστος εξυπηρέτησης 10000 και μέγιστη απαίτηση 4.



Διάγραμμα 6. Διασπορά των στιγμιοτύπων του διαγράμματος 5 με IG>1,0001



Πίνακας 2. IG σε διάφορες τιμές του λόγου κόστος εγκατάστασης προς κόστος εξυπηρέτησης.

Gap	Λόγος
0,0000000814	0,01
0,0000000000	0,01
0,0000000424	0,02
0,0000005332	0,04
0,0000002388	0,05
0,0000000000	0,05
0,0000000000	0,07
0,0000011152	0,08
0,0000000000	0,08
0,0000000169	0,11
0,0000000000	0,11
0,0000000000	0,12
0,0000000000	0,16
0,0000000000	0,16
0,0000002674	0,18
0,0000000000	0,22
0,0000000000	0,24
0,0000000000	0,25
0,0000000167	0,3
0,0000000000	0,33
0,0000001584	0,36
0,0000000586	0,38
0,0000000368	0,42
0,0000000952	0,48
0,0000000000	0,53
0,0000000000	0,61
0,0000000332	0,64
0,0000004137	0,65
0,0000000608	0,66
0,0000000000	0,66
0,0000001169	0,67
0,0000002600	0,69
0,0000000000	0,69

Gap	Λόγος
0,0000000000	0,69
0,0000000000	0,7
0,0000006015	0,71
0,0000002423	0,72
0,0000002403	0,72
0,0000003034	0,74
0,0000001520	0,75
0,0000000000	0,77
0,0000000410	0,77
0,0000001179	0,79
0,0000000000	0,85
0,0000000000	0,86
0,0000001521	0,88
0,0000001162	0,91
0,0000000438	0,93
0,0000000000	0,95
0,0000004010	0,99
0,0000003310	1
0,0000000000	1,02
0,0000015156	1,07
0,0000000074	1,09
0,0000005012	1,19
0,0000002765	1,21
0,0000002563	1,34
0,0000004158	1,34
0,0000008844	1,35
0,0000000753	1,37
0,0000000000	1,39
0,0000003232	1,46
0,0000005375	1,5
0,0000000000	1,53
0,0000000000	1,57
0,0000000369	1,57
0,0000000000	1,61

Gap	Λόγος
0,0000000908	1,65
0,0000010423	1,67
0,0000000000	1,77
0,0000000000	1,8
0,0000000000	1,83
0,0000004542	1,93
0,0000008669	1,98
0,0000000000	2
0,0000002755	2,05
0,0000007233	2,14
0,0000000464	2,24
0,0000007229	2,54
0,0000004760	2,92
0,0000001527	2,92
0,0000003719	2,99
0,0000007993	3,04
0,0000002247	3,88
0,0000004264	4,37
0,0000000000	4,64
0,0000006217	5,67
0,0000007450	5,87
0,0000000647	6,19
0,0000000000	6,54
0,0000011534	6,9
0,0000012042	7
0,0000008381	7,18
0,0000007618	7,84
0,0000023815	9,16
0,0000000000	9,3
0,0000026919	12,43
0,0000013449	12,43
0,0000025672	24,5
0,0000028732	350,99

Μετά από αυτά τα αποτελέσματα καταλήγουμε στο συμπέρασμα ότι σε πολλές περιπτώσεις δεν χρειάζεται να καταφύγει κανείς σε προσεγγιστικούς αλγόριθμους, αφού και μόνο η λύση του χαλαρωμένου προβλήματος θα δώσει με μεγάλη πιθανότητα μία ακέραια λύση. Βέβαια αν το πρόβλημα που αντιμετωπίζουμε έχει πολύ μεγάλη διάσταση (πάνω από 400x400) ή αν έχει κάποια ειδική δομή πχ πολύ κοντινές τιμές στα κόστη εξυπηρέτησης μεταξύ τους, τότε είναι αρκετά πιθανό να χρειαζόμαστε μία διαφορετική προσέγγιση από τη λύση του χαλαρωμένου.

Τα παραπάνω ισχύουν για την περίπτωση των μετρικών προβλημάτων γιατί αλλιώς μπορεί να έχουμε πολύ δύσκολα στιγμιότυπα και σε μικρότερες διαστάσεις. Για παράδειγμα τα στιγμιότυπα Large duality gap σε διάσταση 100x100 είναι πολύ δύσκολα αν και κάθε κλήση του Simplex χρειάζεται λιγότερο από 3 δευτερόλεπτα.

Αυτή η απροσδόκητη ευκολία του προβλήματος επιβεβαιώνεται για το UFL και από τα πειραματικά αποτελέσματα του Martin Hoefer στο [4], όπου αναφέρει ότι η λύση των προβλημάτων κατά βέλτιστο τρόπο με το λογισμικό CPLEX απαιτούσε περίπου 2,5 φορές το χρόνο που απαιτούσαν οι προσεγγιστικοί αλγόριθμοι που εξετάζονταν και βασίζονται σε local search και greedy μεθόδους. Επίσης στην ιστοσελίδα http://www.math.nsc.ru/AP/benchmarks/UFLP/Engl/ben_eng.html αναφέρεται ότι συχνά στιγμιότυπα που είναι κατασκευασμένα για να παρουσιάζουν δυσκολία σε αλγόριθμους local search λύνονται εύκολα με B&B. Άρα επιβεβαιώνεται ότι το πρόβλημα UFL δεν είναι τόσο δύσκολο στην συνηθισμένη περίπτωση και μπορούμε να ισχυριστούμε το ίδιο και για το FTFL, όπως φαίνεται από την πειραματική μας μελέτη σε αυτή την εργασία.

6.2 Το πρόβλημα Facility Leasing

Σε αυτό το πρόβλημα μελετήθηκε μικρός αριθμός από στιγμιότυπα σε πολύ μικρές διαστάσεις και για 100 χρονικές περιόδους. Για ευκολία θα αναφερόμαστε στις χρονικές περιόδους ως ημέρες, αν και μπορεί να μην είναι αναγκαστικά ημέρες τα διαστήματα ενοικίασης.

Σε ότι αφορά την δυσκολία του προβλήματος μπορούμε να πούμε ότι είναι συγκρίσιμη με το FTFL, καθώς σε 20 στιγμιότυπα μόνο 2 είχαν πραγματικές τιμές. Αυτό που κάνει εντύπωση είναι η πολύ μεγαλύτερη πολυπλοκότητα του προβλήματος OFL, η οποία εμπόδισε και τη μελέτη μεγαλύτερων στιγμιοτύπων.

Ενώ στο FTFL έχουμε i*j μεταβλητές x, εδώ έχουμε t_1*i*k*j*t_2 όπου t_1 οι ημέρες που αρχίζοντας ένα διάστημα ενοικίασης μπορεί να εξυπηρετήσει την απαίτηση που εξετάζουμε, k τα διαθέσιμα διαστήματα ενοικίασης και t_2 η ημέρα που εμφανίζεται η απαίτηση του πελάτη. Αυτό συμβαίνει γιατί για κάθε απαίτηση σε μία συγκεκριμένη ημέρα πρέπει να εξετάσουμε όλες τις προηγούμενες κατά τις οποίες αν ξεκινήσει ένα διάστημα ενοικίασης μπορεί να καλύψει αυτήν την απαίτηση. Έτσι αν έχουμε ένα στιγμιότυπο με πολλές ημέρες είναι πολύ μεγάλη η διάσταση του προβλήματος ακόμα και για λίγους εξυπηρετητές και πελάτες, γιατί ο χρόνος υπάρχει στο τετράγωνο, όπως αναφέρεται παραπάνω.

Αυτή η πολυπλοκότητα κάνει πιο μεγάλη την ανάγκη για προσεγγιστικούς αλγορίθμους σε αυτό το πρόβλημα, και μάλιστα τέτοιους ώστε να το λύνουν σε σύντομο χρόνο. Ακόμα και το χαλαρωμένο πρόβλημα στα στιγμιότυπα που λύθηκαν χρειαζόταν περίπου τον ίδιο χρόνο με τη βέλτιστη λύση, άρα δεν είναι πρακτικοί αλγόριθμοι που να βασίζονται στη λύση του χαλαρωμένου προβλήματος.

6.3 Πειραματικά αποτελέσματα αλγορίθμων

Σχετικά με τα αποτελέσματα της μελέτης, θα αναφερθούν πρώτα οι χρονικές απαιτήσεις κάθε αλγορίθμου και μετά θα μελετηθεί η σχέση του σφάλματος για τους δύο πρώτους αλγορίθμους με το IG. Στη συνέχεια θα δούμε τη συσχέτιση της αποτελεσματικότητας του αλγορίθμου των Jain & Vazirani με την απαίτηση των πελατών και πόσο επαληθεύεται η θεωρητική προσέγγιση 3 H_k OPT. Τέλος στο επόμενο κεφάλαιο θα δούμε τις επιδόσεις κάθε αλγορίθμου στο σημαντικότερο τομέα, δηλαδή πόσο κοντά στη βέλτιστη ακέραια λύση είναι η προσέγγιση που παρέχει.

Από πλευράς απαιτούμενου χρόνου μπορούμε να πούμε ότι και οι 4 αλγόριθμοι έχουν πολύ καλές επιδόσεις αλλά παρουσιάζουν και σημαντικές διαφορές μεταξύ τους. Όλοι εκτός του JV απαιτούν τη λύση του χαλαρωμένου προβλήματος, το οποίο σε μεγάλη διάσταση χρειάζεται αρκετό χρόνο. Ενδεικτικά σε διάσταση 100χ100 χρειάζεται περίπου 3 δευτερόλεπτα αλλά αυτό αυξάνεται γρήγορα και για ένα πρόβλημα 400χ400 απαιτεί πάνω από 10 λεπτά. Επιπλέον ο αλγόριθμος SS χρειάζεται και τη λύση του δυϊκού προβλήματος το οποίο κάνει περίπου άλλο τόσο χρόνο για να λυθεί. Αντίστοιχα ο αλγόριθμος RR επαναλαμβάνει πολλές φορές το χαλαρωμένο πρόβλημα. Για μετρικά προβλήματα συνήθως χρειάζεται 1-2 επαναλήψεις και το πολύ 7. Για μη μετρικά στιγμιότυπα χρειάστηκαν μέχρι και 56 επαναλήψεις, όμως οι άλλοι αλγόριθμοι δεν μπορούν να λύσουν μη μετρικά στιγμιότυπα ούτε σε τόσο χρόνο.

Το συμπέρασμα σε ότι αφορά το χρόνο που χρειάζεται κάθε αλγόριθμος είναι ότι ο αλγόριθμος JV υπερτερεί αλλά με μία προϋπόθεση, την κατάλληλη επιλογή του βήματος. Ποιο είναι όμως το κατάλληλο βήμα; Στον πίνακα 3 βλέπουμε ότι όταν τα κόστη εγκατάστασης και εξυπηρέτησης κυμαίνονται μέχρι 100000 ένα βήμα 100 δίνει σημαντικά υψηλότερο σφάλμα από ένα βήμα 10. Ακόμα μικρότερη τιμή θα προκαλούσε μεγάλη καθυστέρηση (αρκετές ώρες) για τα 100 στιγμιότυπα της ομάδας. Επίσης δοκιμάστηκαν σε στιγμιότυπα με πιο μικρές τιμές στα κόστη εγκατάστασης και εξυπηρέτησης τιμές στο βήμα μικρότερες του 1 αλλά δεν προσέφεραν σε ακρίβεια ενώ προκάλεσαν και μεγαλύτερη καθυστέρηση στον υπολογισμό της λύσης. Άρα η κατάλληλη επιλογή του βήματος είναι συνδυασμός του μεγέθους στα κόστη εγκατάστασης και εξυπηρέτησης, της επιθυμητής ακρίβειας και του διατιθέμενου χρόνου.

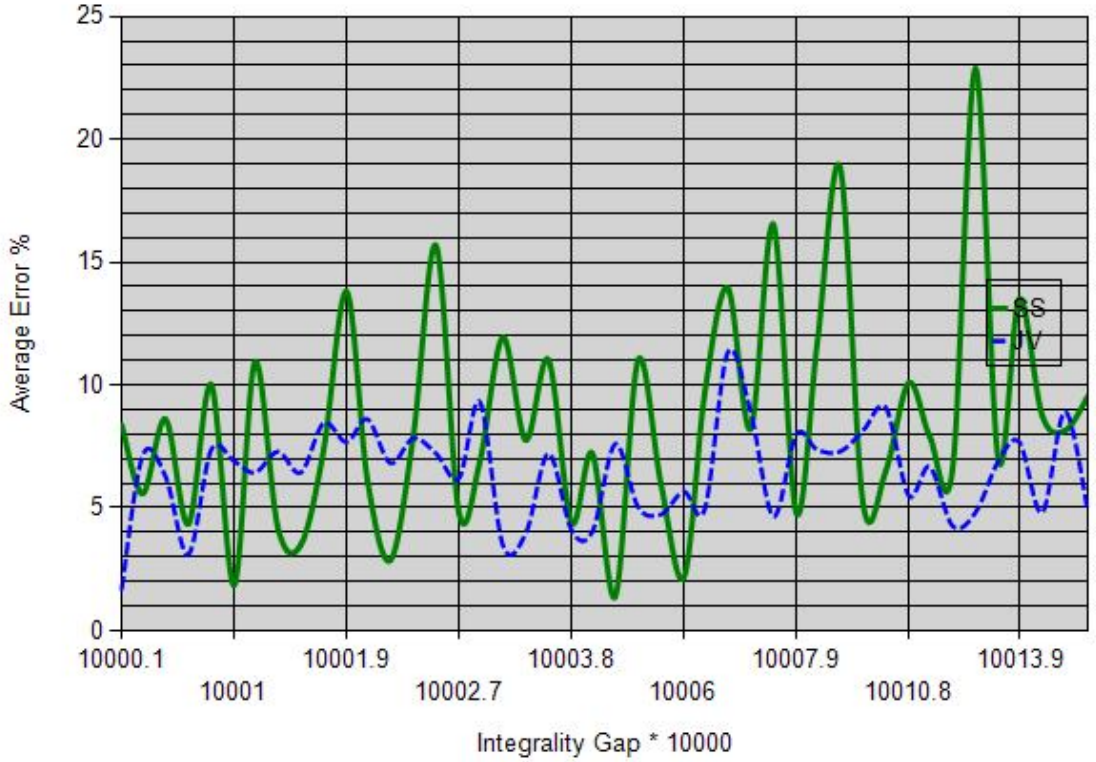
Πίνακας 3. Αποτελέσματα Αλγορίθμου JV συναρτήσει του βήματος για στιγμιότυπα 100x100 με μέγιστη απαίτηση 4.

Μέγιστο Σφάλμα	Μέσο Σφάλμα	Βήμα
0,10627	0,06988	0,01
0,10781	0,06839	0,1
0,10011	0,05874	1
0,45438	0,3331	10
4,12463	3,1823	100

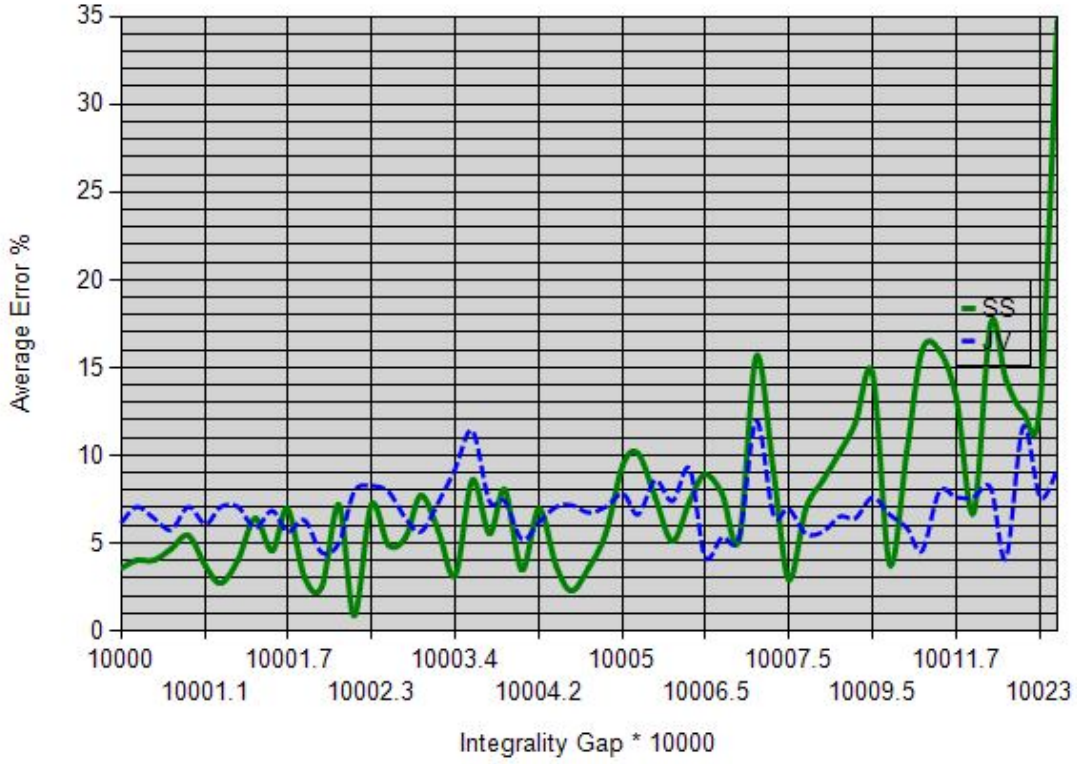
Σχετικά με το ποία είναι η εξάρτηση του σφάλματος από το IG έγιναν μερικά διαγράμματα που δείχνουν με συνεχή (πράσινη) γραμμή το σφάλμα του αλγορίθμου SS, με διακεκομμένη (μπλέ) το σφάλμα του αλγορίθμου JV, με συνεχή (κίτρινη) γραμμή το σφάλμα του αλγορίθμου SSJV και με διακεκομμένη (κόκκινη) το σφάλμα του αλγορίθμου RR. Είναι προφανές στα διαγράμματα 7 έως 10 ότι υπάρχει μία εξάρτηση για τον αλγόριθμο SS η οποία όμως δεν είναι απόλυτη. Με αυτό εννοούμε ότι υπάρχουν προβλήματα με μεγάλο σφάλμα και μικρό gap και το αντίστροφο, όμως ο μέσος όρος έχει μία αυξητική τάση καθώς αυξάνει το gap. Για τον αλγόριθμο JV δεν μπορούμε να πούμε το ίδιο γιατί φαίνεται να είναι σταθερός και ανεξάρτητος από το gap.

Αντίστοιχα, στα διαγράμματα 11 και 12 υπάρχει μία συσχέτιση του σφάλματος των αλγορίθμων SSJV και RR με το IG η οποία και πάλι δεν είναι σαφής. Αυτά τα αποτελέσματα μπορούμε να πούμε ότι είναι αναμενόμενα καθώς οι τρείς αλγόριθμοι SS, SSJV και RR χρησιμοποιούν τη λύση του χαλαρωμένου προβλήματος. Αντίθετα ο αλγόριθμος JV δεν χρησιμοποιεί το χαλαρωμένο πρόβλημα αλλά λειτουργεί με άλλο τρόπο, όπως έχει περιγραφεί προηγουμένως. Το κάθε ένα από αυτά τα διαγράμματα αφορά 300 στιγμιότυπα ώστε να έχουμε ένα αρκετά μεγάλο δείγμα.

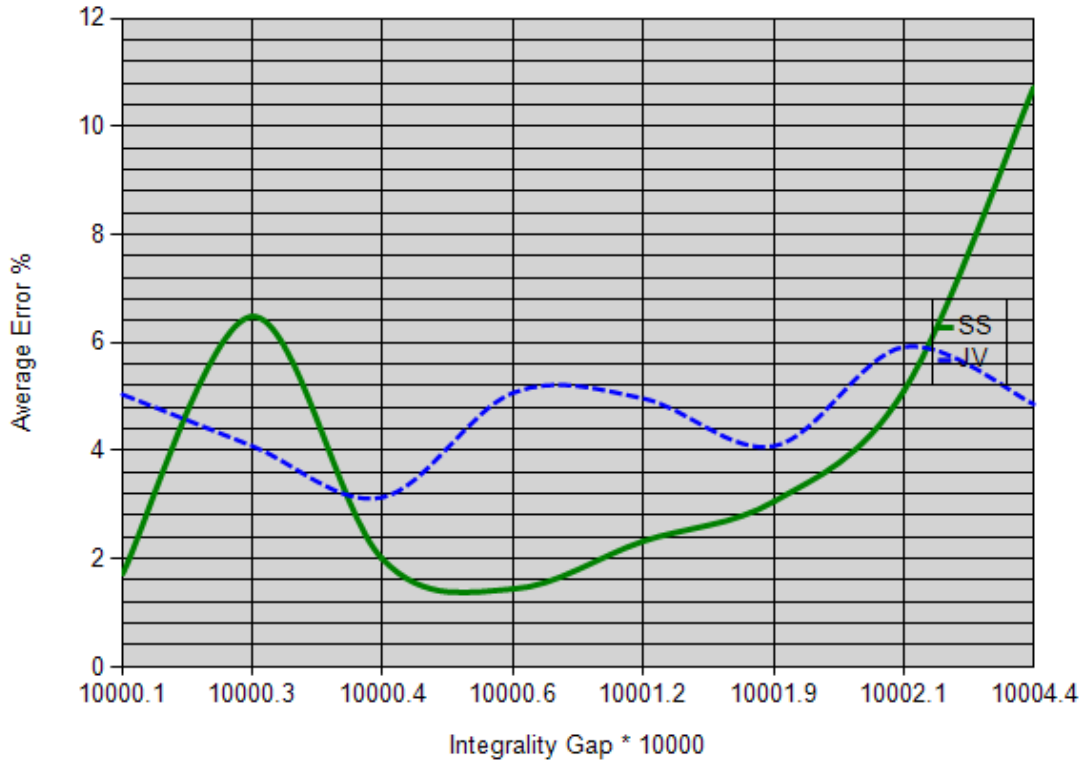
Διάγραμμα 7. Το σφάλμα συναρτήσει του IG για τους αλγόριθμους SS, JV σε στιγμιότυπα με 50 πελάτες, 50 εξυπηρετητές και μέγιστη απαίτηση 5.



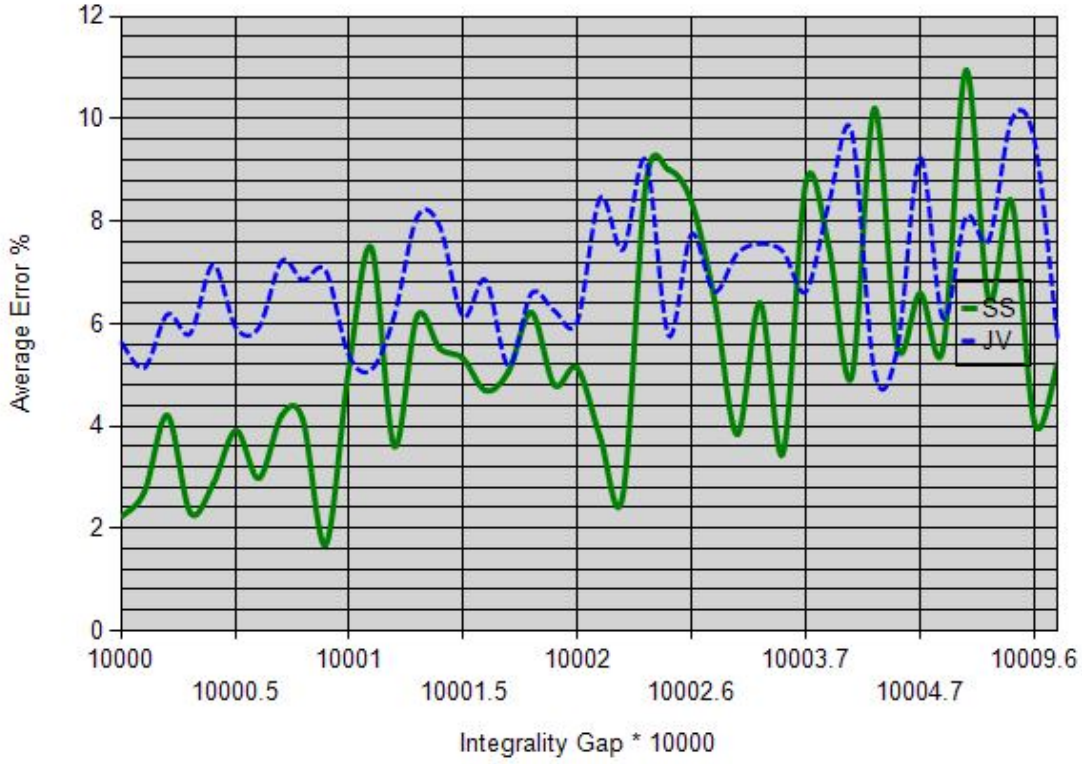
Διάγραμμα 8. Το σφάλμα συναρτήσει του IG για τους αλγόριθμους SS, JV σε στιγμιότυπα με 100 πελάτες, 50 εξυπηρετητές και μέγιστη απαίτηση 5.



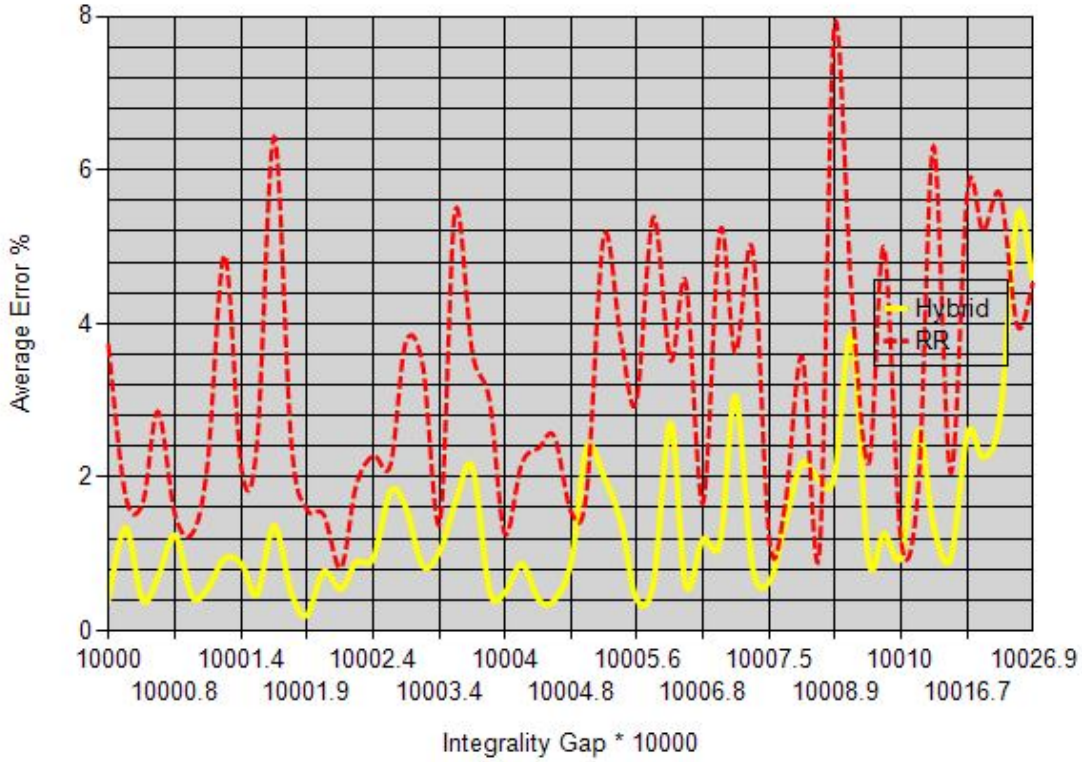
Διάγραμμα 9. Το σφάλμα συναρτήσει του IG για τους αλγόριθμους SS, JV σε στιγμιότυπα με 100 πελάτες, 300 εξυπηρετητές και μέγιστη απαίτηση 10.



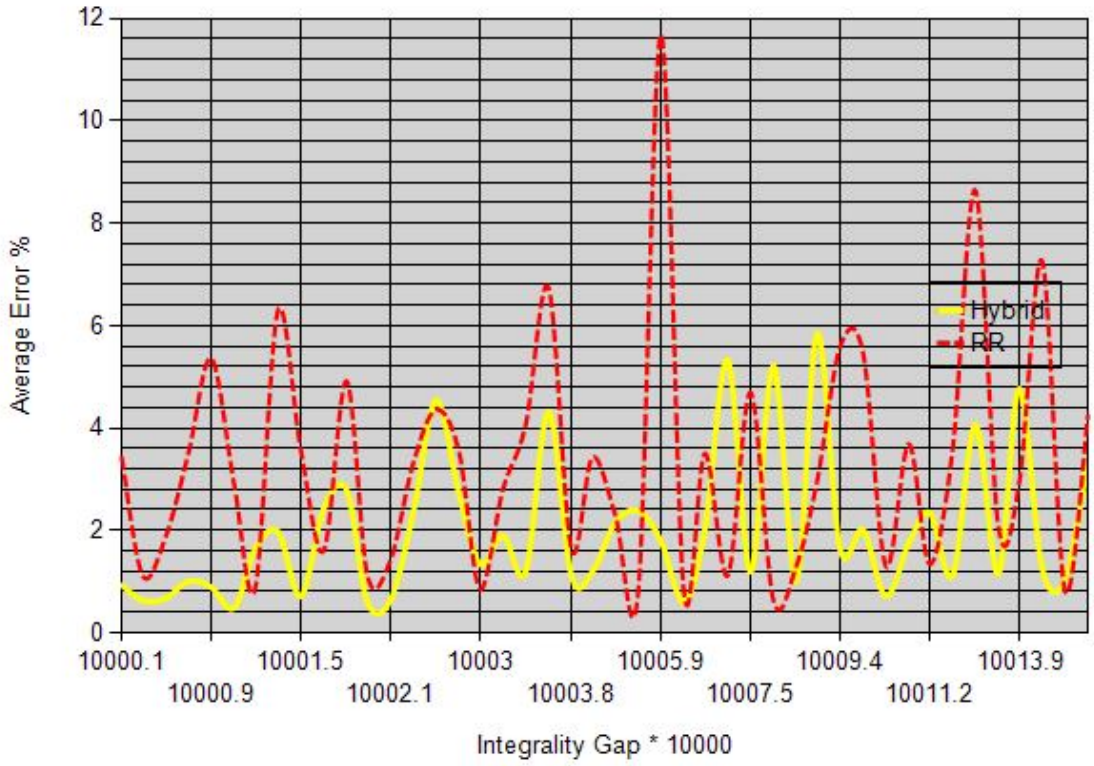
Διάγραμμα 10. Το σφάλμα συναρτήσει του IG για τους αλγόριθμους SS, JV σε στιγμιότυπα με 100 πελάτες, 100 εξυπηρετητές και μέγιστη απαίτηση 8.



Διάγραμμα 11. Το σφάλμα συναρτήσει του IG για τους αλγόριθμους SSJV, RR σε στιγμιότυπα με 100 πελάτες, 50 εξυπηρετητές και μέγιστη απαίτηση 5.



Διάγραμμα 12. Το σφάλμα συναρτήσει του IG για τους αλγόριθμους SSJV, RR σε στιγμιότυπα με 50 πελάτες, 50 εξυπηρετητές και μέγιστη απαίτηση 5.

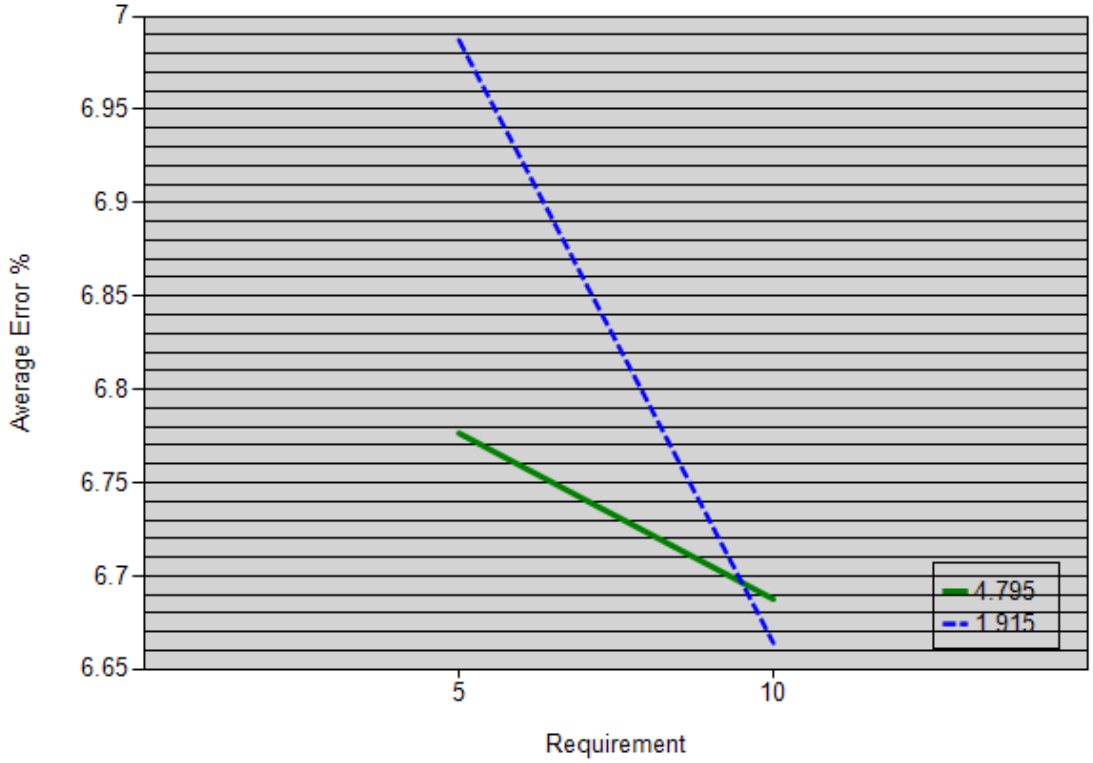


Στα επόμενα διαγράμματα θα μελετήσουμε την απόδοση του αλγορίθμου JV και πώς αυτή εξαρτάται από την απαίτηση των πελατών. Στα διαγράμματα 13 έως 16 βλέπουμε πως κυμαίνεται το μέσο σφάλμα του αλγορίθμου για ένα πλήθος στιγμιοτύπων σε δύο αναλογίες κόστους εγκατάστασης και εξυπηρέτησης, την μία με διακεκομμένη (μπλέ) γραμμή και την άλλη με συνεχή (πράσινη) γραμμή. Στο υπόμνημα κάθε διαγράμματος φαίνεται ο λόγος κόστους εγκατάστασης προς κόστος εξυπηρέτησης.

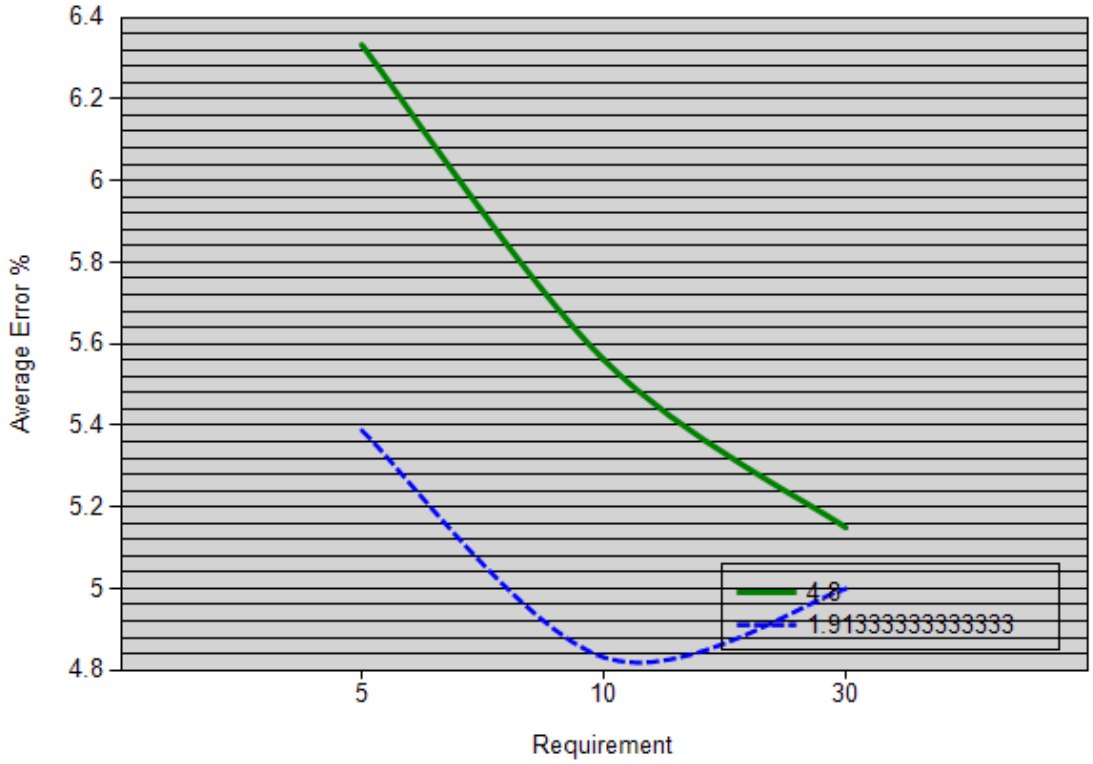
Καθώς αυξάνεται η μέγιστη απαίτηση των πελατών η θεωρητική προσέγγιση του αλγορίθμου μας λέει ότι πρέπει να αυξηθεί λογαριθμικά και το σφάλμα. Ωστόσο κάτι τέτοιο δεν επαληθεύεται από τα πειραματικά δεδομένα. Βλέπουμε και στα τέσσερα διαγράμματα μία διακύμανση με πολύ μικρό εύρος, σε τρίτο-τέταρτο δεκαδικό, που άλλοτε είναι αύξουσα και άλλοτε φθίνουσα. Αυτό δείχνει ότι πρόκειται για τυχαία διακύμανση που δεν μπορεί να αποδοθεί στην αύξηση της απαίτησης και δεν σχετίζεται με αυτήν.

Να σημειώσουμε ότι στη διάσταση 100χ50 (διάγραμμα 13) φτάσαμε μόνο μέχρι 10 την μέγιστη απαίτηση γιατί όσο μεγαλώνει η απαίτηση και πλησιάζει τον αριθμό των εξυπηρετητών γίνεται πιο εύκολο το πρόβλημα και άρα παρεμβαίνουμε έμμεσα στο σφάλμα. Για αυτό το λόγο μελετήθηκε και μία ομάδα στιγμιοτύπων με διάσταση 100χ1000 (διάγραμμα 16) ώστε να μην επηρεάζει τη δυσκολία του προβλήματος η μεταβολή της απαίτησης των πελατών. Σε αυτά τα στιγμιότυπα η απαίτηση μεταβάλλεται από 5-30 και πάλι χωρίς να ακολουθεί η υποδεικνυόμενη θεωρητική αύξηση του σφάλματος.

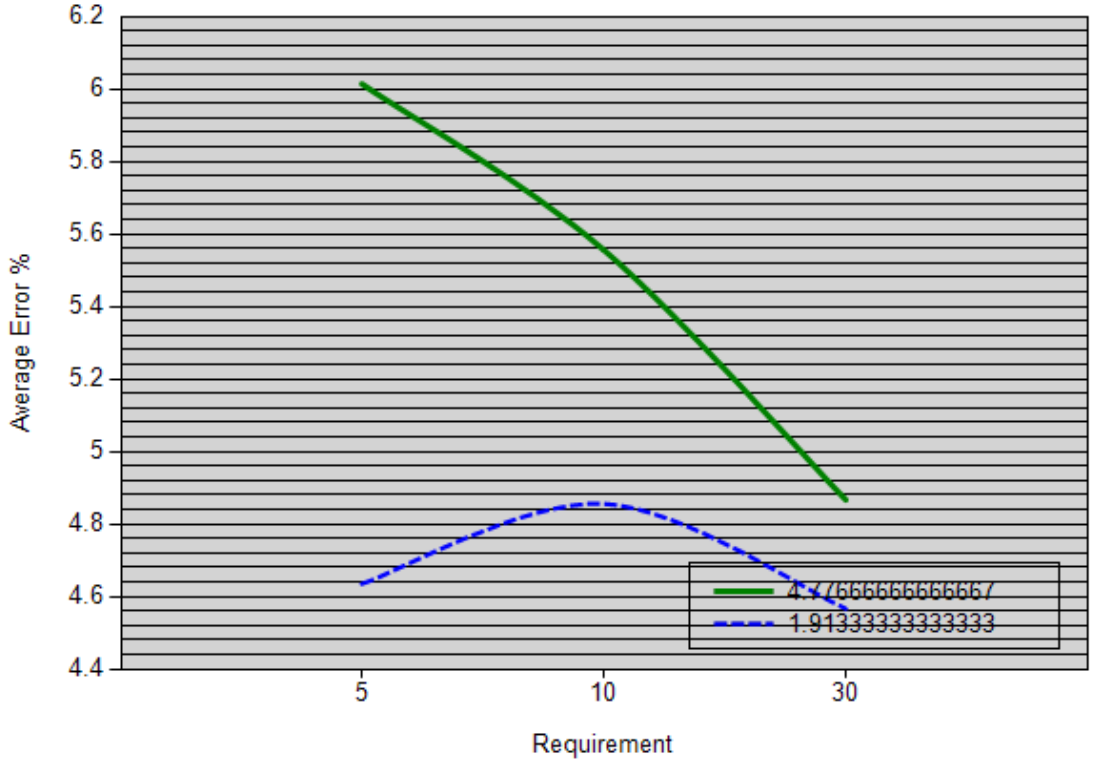
Διάγραμμα 13. Διακύμανση σφάλματος αλγ. Jain&Vazirani σε στιγμιότυπα 100χ50. Στο υπόμνημα αναγράφεται ο λόγος μέσου κόστους εγκατάστασης προς κόστος εξυπηρέτησης.



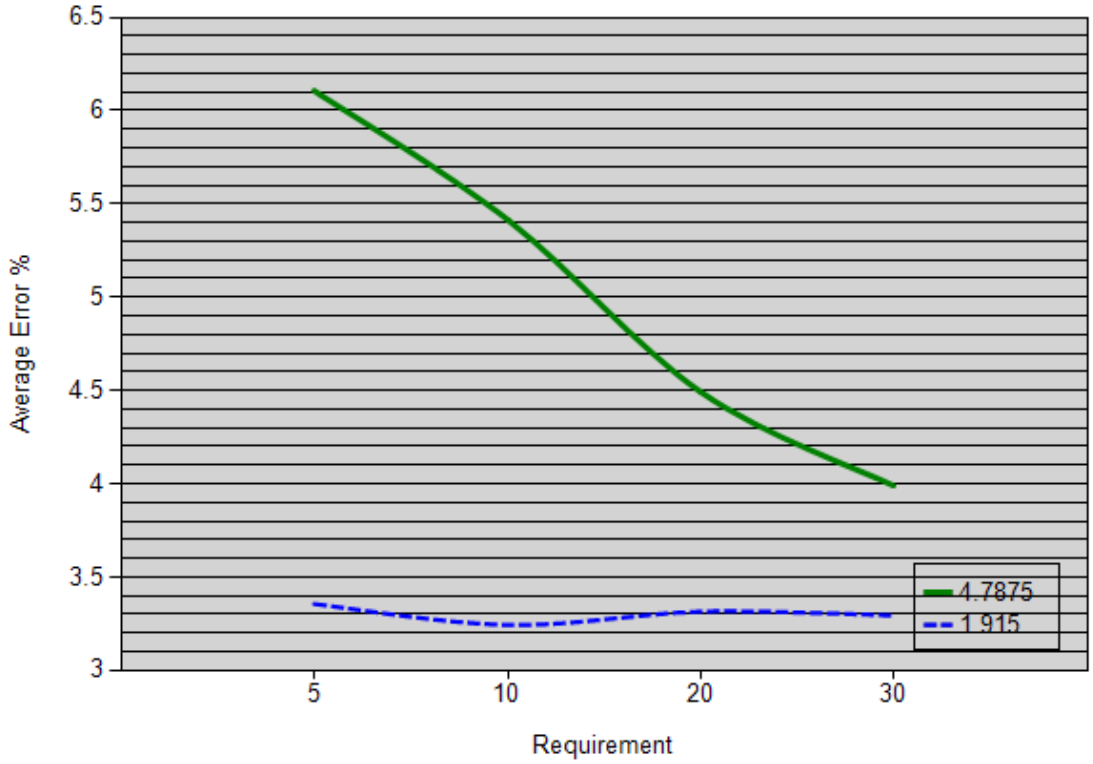
Διάγραμμα 14. Διακύμανση σφάλματος αλγ. Jain&Vazirani σε στιγμιότυπα 100χ200. Στο υπόμνημα αναγράφεται ο λόγος μέσου κόστους εγκατάστασης προς κόστος εξυπηρέτησης.



Διάγραμμα 15. Διακύμανση σφάλματος αλγ. Jain&Vazirani σε στιγμιότυπα 100χ300. Στο υπόμνημα αναγράφεται ο λόγος μέσου κόστους εγκατάστασης προς κόστος εξυπηρέτησης.



Διάγραμμα 16. Διακύμανση σφάλματος αλγ. Jain&Vazirani σε στιγμιότυπα 100χ1000. Στο υπόμνημα αναγράφεται ο λόγος μέσου κόστους εγκατάστασης προς κόστος εξυπηρέτησης.



7. ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ

Ο σημαντικότερος στόχος αυτής της πειραματικής μελέτης είναι να προσδιοριστεί η απόδοση των αλγορίθμων στην πράξη, δηλαδή πόσο κοντά στην βέλτιστη ακέραια λύση είναι η προσέγγιση που δίνουν. Για να προσδιοριστεί αυτό λύθηκαν χιλιάδες στιγμιότυπα (ο όγκος των αρχείων των στιγμιοτύπων και των λύσεών τους πλησιάζει τα 40 Gbyte) και φτιάχτηκαν δεκάδες γραφικές παραστάσεις και διαγράμματα.

Τα διαγράμματα 17-22 είναι από τα πιο αντιπροσωπευτικά και παρουσιάζουν για διάφορες ομάδες στιγμιοτύπων, πόσα στιγμιότυπα έλυσε ο κάθε αλγόριθμος με μία συγκεκριμένη προσέγγιση. Έτσι ο οριζόντιος άξονας έχει τις προσεγγίσεις που επιτεύχθηκαν, με το μέγιστο να επιλέγεται κάθε φορά αυτόματα ώστε να είναι το μεγαλύτερο σφάλμα και των τεσσάρων αλγορίθμων. Στον κάθετο άξονα αναπαριστάται ο αριθμός στιγμιοτύπων που αντιστοιχούν σε κάθε προσέγγιση. Με τελείες (πράσινο) παρουσιάζεται ο αλγόριθμος των SS και με συνεχή γραμμή (μπλέ) των JV. Σε γραμμή με παύλες (κίτρινο) είναι η επίδοση του αλγορίθμου SSJV που συνδυάζει τους δύο πρώτους και σε πιο έντονες παύλες (κόκκινο) του RR. Είναι προφανές ότι όσο πιο πολλά στιγμιότυπα λύνονται κοντά στο 1, δηλαδή στη βέλτιστη λύση, τόσο καλύτερα συμπεριφέρεται ο αλγόριθμος.

Η πρώτη παρατήρηση που εξάγεται είναι ότι όλοι οι αλγόριθμοι αποδίδουν πολύ καλύτερα από την θεωρητική τους προσέγγιση. Αυτό δεν είναι έκπληξη γιατί η θεωρία δίνει ένα άνω όριο και γνωρίζουμε ότι συνήθως η απόδοση είναι καλύτερη από αυτό το όριο σε πραγματικά στιγμιότυπα του προβλήματος. Παρομοίως στο [5], για το UFL αναφέρεται ότι το μέγιστο σφάλμα που εντοπίστηκε ήταν στο 2% της βέλτιστης λύσης.

Πιο αναλυτικά, ο αλγόριθμος SS εκμεταλλεύεται την λύση του χαλαρωμένου και έτσι για ένα μεγάλο ποσοστό των εξεταζόμενων στιγμιοτύπων δίνει την βέλτιστη λύση χωρίς απώλειες και πολύ γρήγορα (αν δεν συνυπολογίσουμε το χρόνο λύσης του χαλαρωμένου και του δυϊκού). Ωστόσο για τα υπόλοιπα στιγμιότυπα δεν συμπεριφέρεται πολύ καλά και συχνά το μέγιστο σφάλμα από τους 4 αλγορίθμους οφείλεται σε αυτόν. Επειδή αυτό το μέγιστο σφάλμα αφορά ένα μόνο στιγμιότυπο δεν είναι ορατό στα διαγράμματα, μπορεί όμως να το δει κανείς στον πίνακα 4.

Η αιτία αυτού του σφάλματος μπορεί να αναγνωριστεί στον τρόπο λειτουργίας του αλγορίθμου. Όταν ανοίγουμε r’ εξυπηρετητές από την ομάδα M τους πελάτες που ήταν συνδεδεμένοι σε εξυπηρετητές που δεν ανοίξαμε τους εξυπηρετούμε από τους εξυπηρετητές που ανοίγουν. Αυτό μέσω της τριγωνικής ανισότητας σημαίνει ότι απαιτείται το πολύ 3 φορές το κόστος που χρειάζονταν αρχικά. Αν αυτό επαναληφθεί για αρκετούς πελάτες τότε έχουμε ένα σημαντικό σφάλμα στο τέλος του αλγορίθμου.

Αναφορικά με τον αλγόριθμο JV το μεγάλο του μειονέκτημα είναι ότι δεν βασίζεται στη λύση του χαλαρωμένου και έτσι χάνει ένα μεγάλο ποσοστό τέλειων λύσεων που παρέχεται από αυτό. Παρ’ όλα αυτά επιτυγχάνει καλές επιδόσεις και τα περισσότερα στιγμιότυπα λύνονται περίπου στο 1,06 για όλους τους τύπους των προβλημάτων. Επίσης το μέγιστο σφάλμα είναι μικρότερο από τον SS και αυτό είναι πολύ σημαντικό για κάποιον που θέλει να υλοποιήσει έναν αλγόριθμο σε μία εμπορική εφαρμογή.

Μία άλλη παράμετρος είναι το γεγονός ότι ο αλγόριθμος JV δεν χρειάζεται τη λύση του χαλαρωμένου και αυτό τον κάνει πιο γρήγορο στο συνολικό χρόνο υπολογισμού της λύσης, με την προϋπόθεση της κατάλληλης επιλογής του βήματος, όπως εξηγήθηκε στο προηγούμενο κεφάλαιο.

Αν και οι δύο αλγόριθμοι SS και JV εφαρμόζονται σε μετρικά προβλήματα, έγιναν μερικές δοκιμές και σε μη μετρικά προβλήματα. Για τον αλγόριθμο SS μερικά στιγμιότυπα έδωσαν προσέγγιση 24 φορές την βέλτιστη λύση. Αντίθετα ο αλγόριθμος των JV απαιτεί την τριγωνική ανισότητα περισσότερο για την απόδειξη της προσέγγισης που έχει και όχι τόσο στον τρόπο λειτουργίας του αλγορίθμου. Έτσι και σε μη μετρικά στιγμιότυπα δίνει μία καλή προσέγγιση, αν και χειρότερη από τα μετρικά όπως φαίνεται και στο διάγραμμα 23.

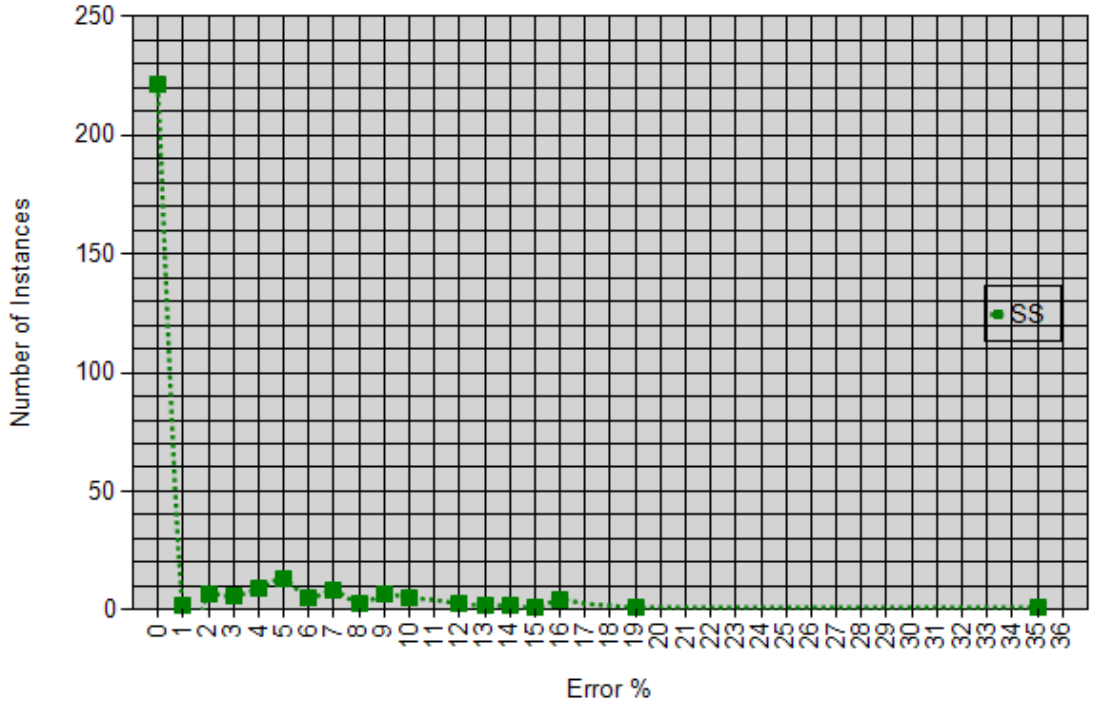
Ο πιο αποτελεσματικός από τους τέσσερις φαίνεται να είναι ο αλγόριθμος SSJV, που καταφέρνει να λύνει σχεδόν όλα τα μετρικά προβλήματα στο 1,01 της βέλτιστης λύσης και έχει αρκετά καλή προσέγγιση ακόμα και σε μη μετρικά προβλήματα. Το κλειδί αυτής της επίδοσης είναι ότι συνδυάζει τα καλύτερα στοιχεία από τους δύο πρώτους αλγορίθμους και εκμεταλλεύεται την λύση του χαλαρωμένου ενώ στη συνέχεια δουλεύει πολύ καλά με τον JV για το πρόβλημα που απομένει. Σε όλα τα διαγράμματα που αφορούν μετρικά στιγμιότυπα ο αλγόριθμος αυτός υπερέχει ξεκάθαρα από τους άλλους αλγορίθμους.

Ο αλγόριθμος RR δίνει επίσης πολύ καλά αποτελέσματα, σαφώς καλύτερα από τους δύο πρώτους αν και όχι τόσο καλά όσο του SSJV. Σημαντικό είναι ότι δίνει πολύ καλές προσεγγίσεις και σε μη μετρικά προβλήματα, στα οποία όμως χρειάζεται πολλές επαναλήψεις και άρα υπολογίσιμο χρόνο.

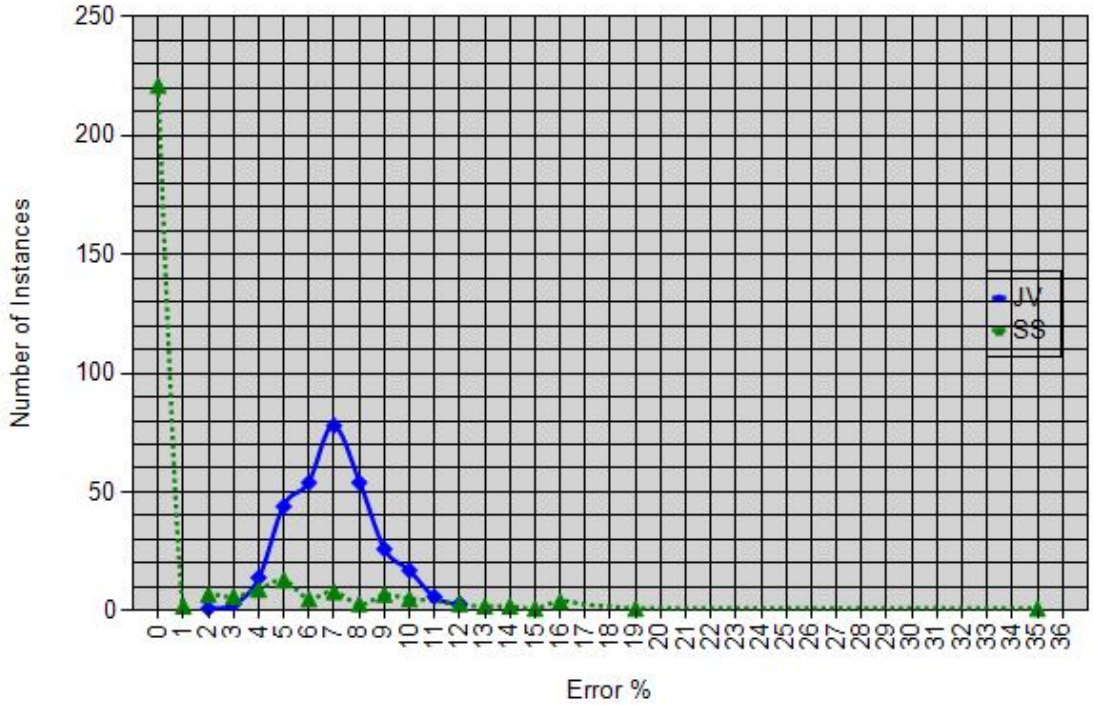
Πίνακας 4. Μέση και μέγιστη προσέγγιση των αλγορίθμων JV και SS σε διάφορους τύπους στιγμιοτύπων

Πελάτες	Εξυπ/τές	Απαίτ.	Κόστος εξυπηρ.	Επαν.	Max JV	Avg JV	Max SS	Avg SS
100	200	10	50	100	0,06946	0,04833	0,14391	0,01358
100	200	10	20	100	0,08644	0,05567	0,13838	0,00852
100	200	30	20	100	0,07578	0,05161	0,02502	0,00117
100	200	30	50	100	0,07158	0,05001	0,03923	0,00288
100	200	5	20	100	0,11175	0,06335	0,17455	0,02619
100	200	5	50	100	0,08671	0,05395	0,20684	0,02208
100	300	10	20	50	0,07264	0,05557	0,14828	0,01053
100	300	10	50	50	0,06975	0,04858	0,10729	0,00957
100	300	30	20	50	0,06965	0,04864	0,04314	0,00448
100	300	30	50	50	0,06016	0,04568	0,0506	0,00427
100	300	5	20	50	0,08065	0,06038	0,20053	0,02548
100	300	5	50	50	0,08346	0,04635	0,1954	0,02243
100	50	10	10	300	0,14595	0,06471	0,08399	0,0032
100	50	10	20	300	0,11758	0,06685	0,10083	0,00502
100	50	10	50	300	0,10606	0,06664	0,0846	0,00275
100	50	5	10	300	0,13626	0,07559	0,16881	0,00971
100	50	5	100	300	0,11887	0,06119	0,1903	0,01276
100	50	5	150	300	0,0938	0,05472	0,10693	0,00373
100	50	5	20	300	0,11223	0,06778	0,23157	0,01224
100	50	5	200	300	0,08078	0,04558	0,04945	0,00052
100	50	5	30	300	0,12109	0,06981	0,17747	0,01926
100	50	5	40	300	0,12121	0,0693	0,2906	0,0178
100	50	5	50	300	0,1232	0,06988	0,34787	0,01924
100	50	5	70	300	0,12482	0,06582	0,26995	0,01611
150	50	5	20	300	0,13412	0,07646	0,22842	0,02255
200	50	5	20	200	0,13179	0,07991	0,21246	0,01795
250	50	5	20	50	0,13044	0,08785	0,16631	0,01632
300	100	5	50	50	0,10575	0,07958	0,27673	0,0425
400	200	5	50	20	0,09618	0,0754	0,10273	0,02416
50	50	5	20	300	0,12548	0,05471	0,15532	0,00832
50	50	5	50	300	0,11352	0,0546	0,229	0,0132
50	50	5	70	300	0,09862	0,05238	0,22979	0,01107
				6020	0,14595	0,061465	0,34787	0,01342

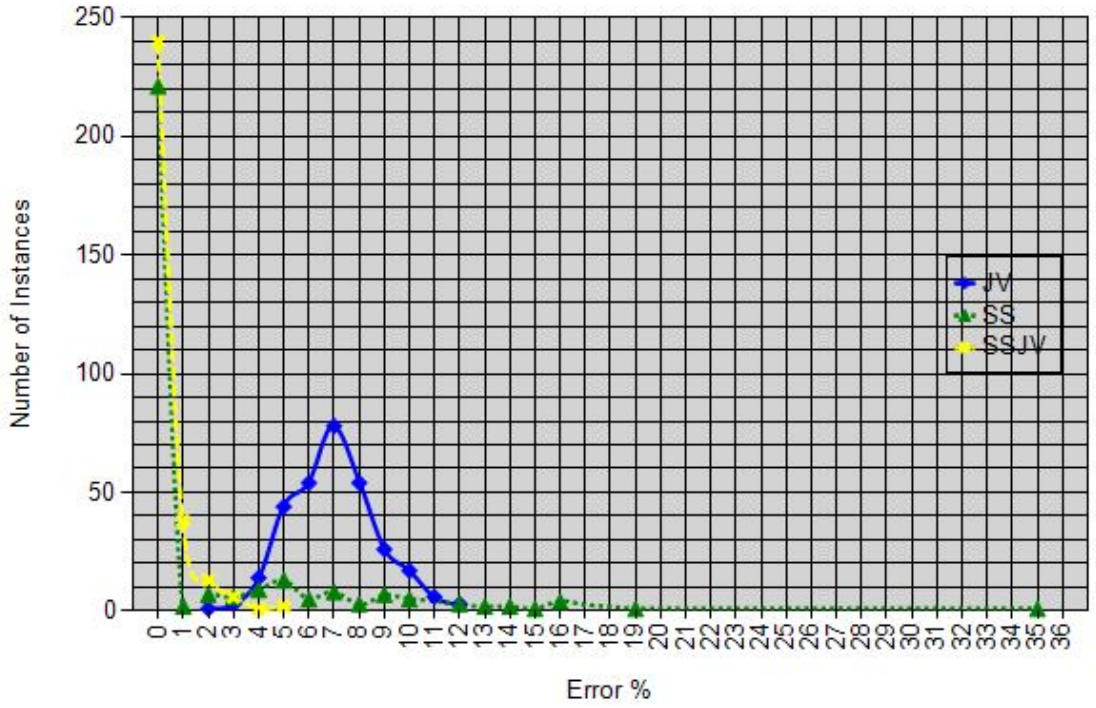
Διάγραμμα 17. Διασπορά σφάλματος του αλγορίθμου SS σε 300 στιγμιότυπα 100χ50



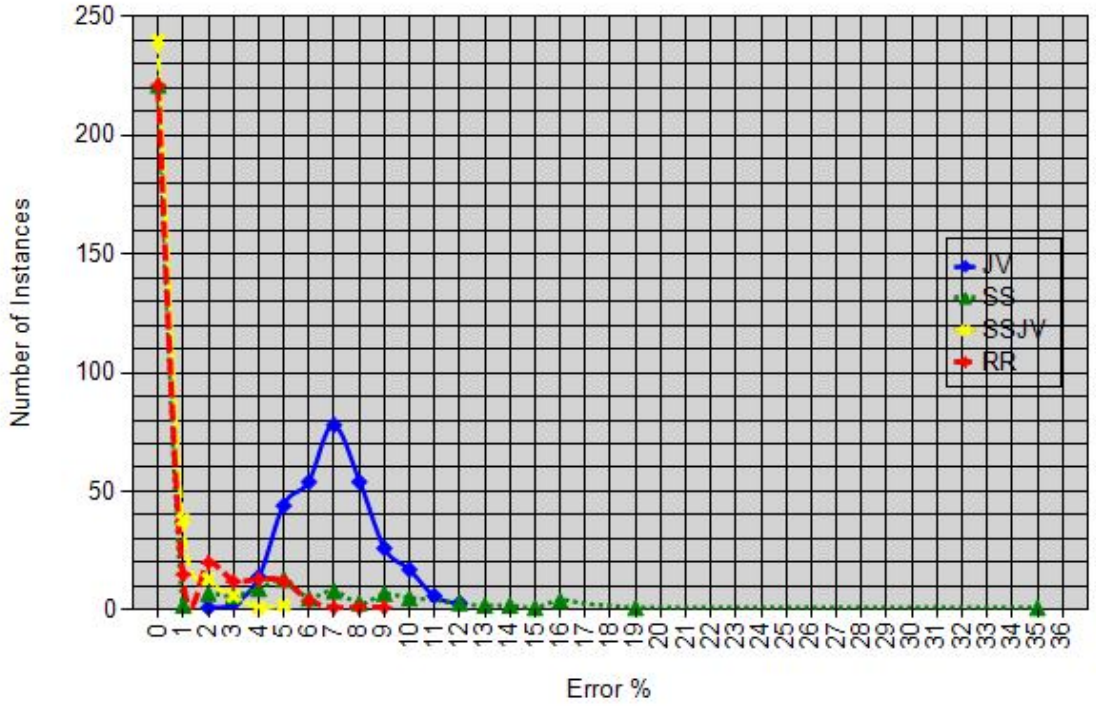
Διάγραμμα 18. Διασπορά σφάλματος των αλγορίθμων SS και JV στα ίδια στιγμιότυπα με το διάγραμμα 17



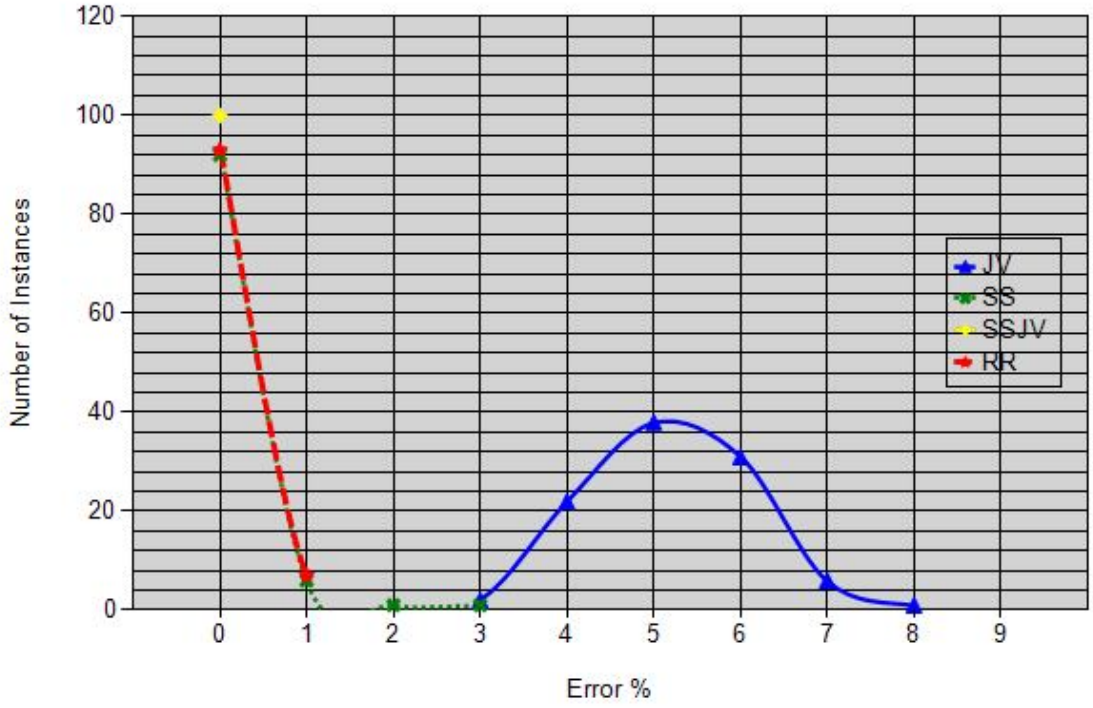
Διάγραμμα 19. Διασπορά σφάλματος των αλγορίθμων SS, JV και SSJV στα ίδια στιγμιότυπα με το διάγραμμα 17



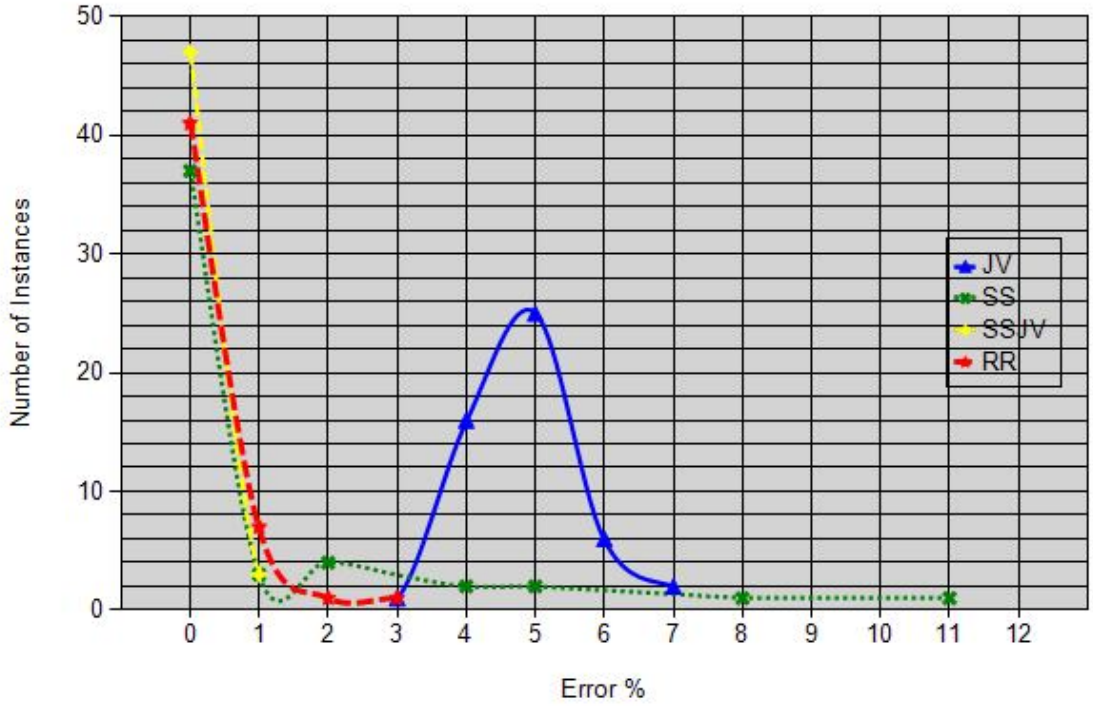
Διάγραμμα 20. Διασπορά σφάλματος των 4 αλγορίθμων στα στιγμιότυπα του διαγράμματος 17.



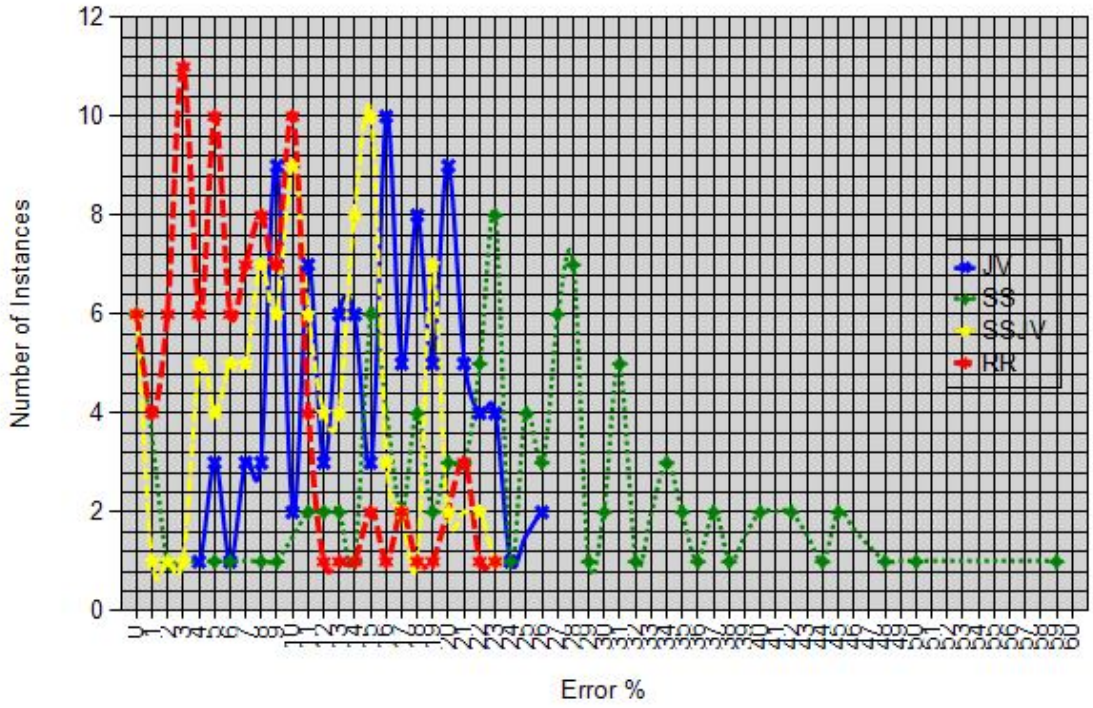
Διάγραμμα 21. Διασπορά σφάλματος των 4 αλγορίθμων σε στιγμιότυπα 100χ200



Διάγραμμα 22. Διασπορά σφάλματος των 4 αλγορίθμων σε στιγμιότυπα 100χ300



Διάγραμμα 23. Διασπορά σφάλματος των 4 αλγορίθμων σε μη μετρικά στιγμιότυπα 30χ80



8. ΣΥΜΠΕΡΑΣΜΑΤΑ

Όλες οι προηγούμενες παρατηρήσεις μας οδηγούν στα παρακάτω συμπεράσματα:

1. Το πρόβλημα FTFL δεν είναι δύσκολο σε μικρές και μεσαίες διαστάσεις και μόνο μεγάλα στιγμιότυπα ή ειδικές κατασκευές θα μας δημιουργήσουν πρόβλημα σε μία B&B αντιμετώπιση.
2. Τα περισσότερα από τα τυχαία στιγμιότυπα που εξετάστηκαν είχαν ακέραιες λύσεις ήδη από τη λύση του χαλαρωμένου. Αυτό οφείλεται όπως αναφέρθηκε παραπάνω στο ότι το πρόβλημα δεν είναι δύσκολο στη γενική περίπτωση.
3. Το μέσο IG είναι πολύ μικρό σε τυχαία μετρικά στιγμιότυπα και μόνο με ειδικούς τρόπους κατασκευής μπορούμε να πετύχουμε μεγαλύτερο IG.
4. Η δυσκολία του προβλήματος εξαρτάται από την αναλογία κόστους εγκατάστασης – κόστους εξυπηρέτησης και όταν το κόστος εξυπηρέτησης υπερέχει πολύ τότε σπάνια εμφανίζονται δύσκολα προβλήματα, ενώ το αντίθετο, δηλαδή δύσκολα προβλήματα με κόστος εγκατάστασης πολύ μεγαλύτερο του κόστος εξυπηρέτησης είναι συνηθισμένο.
5. Η προσέγγιση του αλγορίθμου SS χειροτερεύει όσο αυξάνει το IG αλλά η σχέση μεταξύ τους δεν είναι απόλυτη και εμφανίζονται συχνά εξαιρέσεις αυτού του κανόνα.
6. Η προσέγγιση του αλγορίθμου JV δεν εξαρτάται καθόλου από την μέγιστη απαίτηση των πελατών, όπως υποδεικνύει η θεωρητική προσέγγιση του αλγορίθμου.
7. Ο αλγόριθμος SSJV που προτείνεται με τον συνδυασμό των δύο πρώτων δουλεύει πολύ καλύτερα και από τους δύο και δίνει μία καλή προσέγγιση πολύ κοντινή στη βέλτιστη ακέραια λύση.
8. Ο αλγόριθμος RR απαιτεί περισσότερο χρόνο αλλά δίνει επίσης μία καλή προσέγγιση ακόμα και για μη μετρικά στιγμιότυπα.
9. Η προσέγγιση που επιτυγχάνεται στην πράξη είναι πολύ μικρότερη από την θεωρητική υποδεικνυόμενη. Αυτό αποδίδεται στο ότι το θεωρητικό άνω όριο οφείλει να καλύπτει όλες τις περιπτώσεις, ακόμα και τις πιο ακραίες.
10. Το πρόβλημα OFL παρουσιάζει πολύ μεγάλη πολυπλοκότητα και η διάστασή του εξαρτάται από το χρονικό διάστημα που εξετάζουμε περισσότερο από τον αριθμό των εξυπηρετητών και πελατών.

9. ΕΡΩΤΗΜΑΤΑ ΓΙΑ ΠΕΡΑΙΤΕΡΩ ΔΙΕΡΕΥΝΗΣΗ

Μετά από αυτά τα συμπεράσματα προκύπτουν εύλογες προοπτικές ως εξής: Αφού οι δύο νέοι αλγόριθμοι που προτάθηκαν λειτουργούν τόσο καλά μήπως θα ήταν δυνατόν να προσδιοριστεί και ένα θεωρητικό άνω φράγμα στο σφάλμα της προσέγγισης που δίνουν, ώστε να μπορεί κανείς να είναι βέβαιος ότι δεν θα ξεφύγει πάνω από κάποιο όριο η λύση που παίρνει σε κάθε περίπτωση;

Για τον αλγόριθμο SSJV μπορούμε να συνδυάσουμε την θεωρητική προσέγγιση των δύο μητρικών αλγορίθμων και να πούμε ότι δεν μπορεί να επιτευχθεί κάτι καλύτερο από το 3logk του JV. Ωστόσο είναι ανοιχτή η πρόκληση μίας καλύτερης θεωρητικής προσέγγισης από αυτήν.

Σε ότι αφορά τον αλγόριθμο RR, ξέρουμε ότι σε κάθε επανάληψη ανοίγει τουλάχιστον ένα εξυπηρετητής. Έτσι αν προσδιορίσουμε το σφάλμα που μπορεί να προκύψει από κάθε επανάληψη τότε i φορές αυτό θα είναι το μέγιστο σφάλμα που θα δώσει ο αλγόριθμος.

Πέρα από αυτά θα ήταν πολύ ενδιαφέρον να μελετηθεί το πρόβλημα και σε ακόμα μεγαλύτερες διαστάσεις, ώστε να προσδιοριστεί η συμπεριφορά του και να επαληθευτούν τα συμπεράσματα της παρούσας εργασίας σε μεγάλα στιγμιότυπα που δεν εξετάστηκαν λόγω περιορισμένων πόρων.

Επίσης περαιτέρω μελέτης χρήζει το πρόβλημα OFL το οποίο δεν ήταν ο κύριος σκοπός αυτής της μελέτης, όμως παρουσιάζει ιδιαιτερότητες και ενδεχομένως είναι πιο πρόσφορο για προσεγγιστικούς αλγορίθμους, λόγω της πολυπλοκότητας που παρουσιάζει.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Αγγλικός όρος	Ελληνικός όρος
Uncapacitated Facility Location	Χωροθέτηση πόρων
Fault Tolerant Facility Location	Χωροθέτηση πόρων με ανοχή σε σφάλματα
Offline Facility Leasing	Ενοικίαση πόρων
Integrality Gap	Ακέραιο εύρος
Branch and Bound	Μέθοδος διακλάδωσης και φράγματος

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

Σύντμηση	Επεξήγηση
UFL	Uncapacitated Facility Location
FTFL	Fault Tolerant Facility Location
OFL	Offline Facility Leasing
SS	Swamy & Shmoys
JV	Jain & Vazirani
IG	Integrality Gap
B&B	Branch and Bound

ΑΝΑΦΟΡΕΣ

[1] K. Jain, V .V. Vazirani. An approximation algorithm for the fault tolerant metric facility location problem. Algorithmica 2004 pages 433-439.
[2] C. Swamy, D. Shmoys. Fault tolerant facility location. ACM 2008 pages 51:1-51:26.
[3] C. Nagarajan, D Williamson. Offline and online facility leasing. 13th International Conference, IPCO 2008 pages 303-315.
[4] M. Hoefer. Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location In *Experimental and efficient algorithms: Second International Workshop, WEA 2003* (pp. 165-178). Berlin, Germany: Springer.
[5] M. Resend, R Werneck. A hybrid multistart heuristic for the uncapacitated facility location problem. European journal of operational research 2001 pages 54-68.
[6] S. Guha , A. Meyerson , K. Munagala. Improved Algorithms for Fault Tolerant Facility Location (2001). Proceeding SODA '01 Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms pages 636-641.
[7] J. Byrka , A. Srinivasan. Fault-tolerant facility location: a randomized dependent LP-rounding algorithm. IPCO 2010
[8] M. Hoefer. UfLib.UFL benchmarks, benchmark generators http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/
[9] http://www.math.nsc.ru/AP/benchmarks/UFLP/Engl/ben_eng.html
[10] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/uncapinfo.html
[11] S. Xu and H. Shen. The fault-tolerant facility allocation problem. ISAAC, pages 689–698, Berlin, Heidelberg,2009.
[12] K. Jain, V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. J. ACM, 48(2):274–296, 2001
[13] K. Jain, M. Mahdian, E. Markakis, A. Saberi, V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J.ACM, 50(6):795–824, November 2003.
[14] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. ICALP, volume 6756, pages 77–88. Springer, 2011
[15] O. Blide, J. Krarup. Sharp lower bounds and efficient algorithms for the simple plant location problem. Annuals of discrete Mathematics I (1997) 79-97.
[16] R.D. Galvao, L.A. Raggi. A method for solving to optimality uncapacitated facility location problems. Annuals of Operational Research 18 (1989) 225-244.
[17] D. Ghosh. Neighborhood search heuristics for the uncapacitated facility location problem. European journal of Operational Research 150 (2003) 150-162.
[18] Y. Kochetov, D. Ivanenko. Computationally difficult instances for the uncapacitated facility location problem. Proceedings of the 5th Metaheuristics International Conference (MIC) 2003, 41.1-41.6.

ΠΑΡΑΡΤΗΜΑ Α

Κώδικας αλγορίθμου Swamy & Shmoys

```
protected double cmdAlgorithm(int countΕξυπηρετητές, int countΠελάτες, int
count)
    {
        string path = @"E:\bin\";
        if (ddlFolder.SelectedIndex != 0)
        {
            path += ddlFolder.SelectedItem.ToString() + @"\";
        }
        if (File.Exists(@" " + path + "ftfl_dual_" + count.ToString() +
".sol"))
        {
            lblError.Visible = false;
            var text = File.ReadAllText(@" " + path + "ftfl_dual_" +
count.ToString() + ".sol");
            string txt = text.ToString();
            string[] Split = txt.Split(new Char[] { '\r' });
            int rows = Split.GetLength(0);

            int countX = countΕξυπηρετητές * countΠελάτες;

            List<double> alpha = new List<double>();
            for (int i = countΠελάτες; i > 0; i--)
            {
                string a = Split[rows - 21 - countX - countΕξυπηρετητές -
i].Substring(25).Remove(23);
                double aj = Convert.ToDouble(a.Trim());
                alpha.Add(aj);
            }
            var text1 = File.ReadAllText(@" " + path + "ftfl_" +
count.ToString() + ".sol");
            string txt1 = text1.ToString();
            string[] Split1 = txt1.Split(new Char[] { '\r' });
            int rows1 = Split1.GetLength(0);
            //get y
            List<double> psi = new List<double>();
            for (int i = countΕξυπηρετητές; i > 0; i--)
            {
                string y = Split1[rows1 - 21 - i].Substring(25).Remove(23);
                double yi = Convert.ToDouble(y.Trim());
                psi.Add(yi);
            }
            //get x
            List<double> xij = new List<double>();
            for (int i = countX; i > 0; i--)
            {
                string x = Split1[rows1 - 21 - countΕξυπηρετητές -
i].Substring(25).Remove(23);
                double xi = Convert.ToDouble(x.Trim());
                xij.Add(xi);
            }
            //read f,c,r
            var text2 = File.ReadAllText(@" " + path + "ftfl_" +
count.ToString() + ".dat");
            string txt2 = text2.ToString();
            string[] Split2 = txt2.Split(new Char[] { '\r' });
            int rows2 = Split2.GetLength(0);
            //get r
```

```
            List<int> ar = new List<int>();
            for (int i = countΠελάτες; i > 0; i--)
            {
                int index = Split2[rows2 - 3 - i].IndexOf(' ');
                string r = Split2[rows2 - 3 - i].Substring(index);
                int rj = Convert.ToInt16(r.Trim());
                ar.Add(rj);
            }
            //get c
            double[,] cij = new double[countΕξυπηρετητές, countΠελάτες];
            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                string rowc = Split2[rows2 - 5 - countΠελάτες -
countΕξυπηρετητές + i];
                string[] Split3 = rowc.Split(new Char[] { ' ' });
                for (int j = 0; j < countΠελάτες; j++)
                {
                    cij[i, j] = Convert.ToDouble(Split3[j + 1].Trim());
                }
            }
            //get f
            List<double> fi = new List<double>();
            for (int i = countΕξυπηρετητές; i > 0; i--)
            {
                int index = Split2[rows2 - 7 - countΕξυπηρετητές -
countΠελάτες - i].IndexOf(' ');
                string f = Split2[rows2 - 7 - countΕξυπηρετητές -
countΠελάτες - i].Substring(index);
                double fy = Convert.ToDouble(f.Trim());
                fi.Add(fy);
            }
            //phase 1
            int[] r1 = new int[countΠελάτες];//r1=r' residual requirement
            int[] r2 = new int[countΠελάτες];//r2=r-r' number of
εξυπηρετητές serving j client
            List<int> synoloS = new List<int>();//set S with πελάτες having
residual requirement>0

            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                if (psi[i] == 1)
                {
                    for (int j = 0; j < countΠελάτες; j++)
                    {
                        if (xij[i + (countΕξυπηρετητές * j)] > 0) //&&
r2[j]<ar[j])
                        {
                            xij[i + (countΕξυπηρετητές * j)] = 1;
                            r2[j]++;
                        }
                    }
                }
            }
            //phase 2

            //find πελάτες with residual requirement
            for (int j = 0; j < countΠελάτες; j++)
            {
                r1[j] = ar[j] - r2[j];
                if (r1[j] > 0)
                {
```

```
                    synoloS.Add(j); //πελάτες with residual requirement >0
are inserted in S
                }
            }
            //repeat until end
            while (synoloS.Count > 0)
            {
                List<int> synoloF = new List<int>();
                List<int> synoloM = new List<int>();
                double sumYi = 0;
                //s1 choose πελάτης j
                double minA = 1000000;
                int index = 0;
                for (int i = 0; i < synoloS.Count; i++)
                {
                    if (alpha[synoloS[i]] < minA)
                    {
                        minA = alpha[synoloS[i]];
                        index = synoloS[i];
                    }
                }
                //s2
                //search εξυπηρετητές that fractionaly serve πελάτης j
                for (int i = 0; i < countΕξυπηρετητές; i++)
                {
                    if (psi[i] < 1 && psi[i] > 0)
                    {
                        if (xij[i + (countΕξυπηρετητές * index)] > 0)
                        {
                            synoloF.Add(i);
                        }
                    }
                }
                //fill set M
                while (r1[index] > sumYi && synoloF.Count > 0)
                {
                    //search cheaper facility in set Fj
                    double cheaper = 1000000;
                    int indexOfCheaper = 0;
                    for (int i = 0; i < synoloF.Count; i++)
                    {
                        if (fi[synoloF[i]] < cheaper)
                        {
                            cheaper = fi[synoloF[i]];
                            indexOfCheaper = synoloF[i];
                        }
                    }
                    //add cheaper facility in set M
                    synoloM.Add(indexOfCheaper);
                    synoloF.Remove(indexOfCheaper);
                    sumYi += psi[indexOfCheaper];
                }

                if (r1[index] < sumYi)
                {
                    //last facility in M is
                    int lastFac = synoloM[synoloM.Count - 1];
                    //divide last facility in M
                    psi[lastFac] = sumYi - r1[index];
                    for (int j = 0; j < countΠελάτες; j++)
                    {
```

```
                        if (xij[lastFac + (countΕξυπηρετητές * j)] >
psi[lastFac])
                        {
                            xij[lastFac + (countΕξυπηρετητές * j)] =
psi[lastFac];
                        }
                    }
                }
                //s3 open r' least expensive εξυπηρετητές
                int rIndex = r1[index];
                for (int i = 0; i < rIndex; i++)
                {
                    //open facility from M
                    psi[synoloM[i]] = 1;

                    //serve connected πελάτες to opened facility
                    for (int j = 0; j < countΠελάτες; j++)
                    {
                        if (xij[synoloM[i] + (countΕξυπηρετητές * j)] > 0
&& r1[j] > 0)
                        {
                            xij[synoloM[i] + (countΕξυπηρετητές * j)] = 1;
                            r1[j]--;
                            if (r1[j] == 0)
                            {
                                //exclude πελάτης j from set S
                                synoloS.Remove(j);
                            }
                        }
                    }
                }
                //search πελάτες served from M
                for (int j = 0; j < countΠελάτες; j++)
                {
                    for (int i = rIndex; i < synoloM.Count; i++)
                    {
                        for (int k = 0; k < rIndex; k++)
                        {
                            if (xij[synoloM[i] + (countΕξυπηρετητές * j)] >
0 && r1[j] > 0 && xij[synoloM[k] + (countΕξυπηρετητές * j)] < 1)
                            {
                                xij[synoloM[k] + (countΕξυπηρετητές * j)] =
1;

                                r1[j]--;
                                if (r1[j] == 0)
                                {
                                    //exclude πελάτης j from set S
                                    synoloS.Remove(j);
                                }
                            }
                        }
                    }
                }
                //dispose εξυπηρετητές from M that were not opened
                int lastInM = synoloM.Count;
                if (rIndex < sumYi)
                {
                    lastInM = synoloM.Count - 1;
                }
                for (int i = rIndex; i < lastInM; i++)
                {
```

```
                    psi[synoloM[i]] = 0;
                }
            }
            //count cost of solution
            double cost = 0;
            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                if (psi[i] == 1)
                {
                    cost += fi[i];
                    for (int j = 0; j < countΠελάτες; j++)
                    {
                        if (xij[i + (countΕξυπηρετητές * j)] == 1)
                        {
                            cost += cij[i, j];
                        }
                    }
                }
            }
            //check feasibility
            for (int j = 0; j < countΠελάτες; j++)
            {
                int arj = 0;
                for (int i = 0; i < countΕξυπηρετητές; i++)
                {
                    if (xij[i + (countΕξυπηρετητές * j)] == 1 && psi[i] ==
1)
                    {
                        arj++;
                    }
                }
                if (arj < ar[j])
                {
                    return -1;
                }
            }
            return cost;
        }
        else
        {
            return -2;
        }
    }
```

Κώδικας αλγορίθμου Jain & Vazirani

```
protected double cmdAlgorithmVar(int countΕξυπηρετητές, int countΠελάτες,
int count)
    {
        string path = @"E:\bin\";
        if (ddlFolder.SelectedIndex != 0)
        {
            path += ddlFolder.SelectedItem.ToString() + @"\";
        }

        int countX = countΕξυπηρετητές * countΠελάτες;
        List<int> psi = new List<int>();
        for (int i = 0; i < countΕξυπηρετητές; i++)
        {
            psi.Add(0);
        }
```

```
        double[,] xij = new double[countΕξυπηρετητές, countΠελάτες];

        //read f,c,r
        var text2 = File.ReadAllText(@"  " + path + "ftfl_" +
count.ToString() + ".dat");
        string txt2 = text2.ToString();
        string[] Split2 = txt2.Split(new Char[] { '\r' });
        int rows2 = Split2.GetLength(0);
        //get r
        List<int> ar = new List<int>();
        for (int i = countΠελάτες; i > 0; i--)
        {
            int index = Split2[rows2 - 3 - i].IndexOf(' ');
            string r = Split2[rows2 - 3 - i].Substring(index);
            int rj = Convert.ToInt16(r.Trim());
            ar.Add(rj);
        }
        //get c
        double[,] cij = new double[countΕξυπηρετητές, countΠελάτες];
        double[,] cij_initial = new double[countΕξυπηρετητές,
countΠελάτες];
        double sum = 0;
        for (int i = 0; i < countΕξυπηρετητές; i++)
        {
            string rowc = Split2[rows2 - 5 - countΠελάτες -
countΕξυπηρετητές + i];
            string[] Split3 = rowc.Split(new Char[] { ' ' });
            for (int j = 0; j < countΠελάτες; j++)
            {
                cij[i, j] = Convert.ToDouble(Split3[j + 1].Trim());
                cij_initial[i, j] = Convert.ToDouble(Split3[j + 1].Trim());
                sum +=cij[i,j];
            }
        }
        avg_sc = sum / (countΕξυπηρετητές*countΠελάτες);
        //get f
        List<double> fi = new List<double>();
        List<double> fi_initial = new List<double>();
        for (int i = countΕξυπηρετητές; i > 0; i--)
        {
            int index = Split2[rows2 - 7 - countΕξυπηρετητές - countΠελάτες
- i].IndexOf(' ');
            string f = Split2[rows2 - 7 - countΕξυπηρετητές - countΠελάτες
- i].Substring(index);
            double fy = Convert.ToDouble(f.Trim());
            fi.Add(fy);
            fi_initial.Add(fy);
        }
        avg_oc = fi.Average();
        bool finished = false;
        while (!finished)
        {
            double[] alpha = new double[countΠελάτες];
            double[,] beta = new double[countΕξυπηρετητές, countΠελάτες];
            double[] psi_tentative = new double[countΕξυπηρετητές];
            double[,] xij_tentative = new double[countΕξυπηρετητές,
countΠελάτες];
            //put in synoloS πελάτες with remaining requirements
            List<int> synoloS = new List<int>();
            for (int j = 0; j < countΠελάτες; j++)
            {
```

```
                if (ar[j] == ar.Max() && ar.Max() > 0)
                {
                    synoloS.Add(j);
                }
            }
            if (synoloS.Count == 0)
            {
                finished = true;
                break;
            }
            //begin phase 1
            while (synoloS.Count > 0)
            {
                //increase all aj by 1
                for (int j = 0; j < synoloS.Count; j++)
                {
                    alpha[synoloS[j]]                                    +=
Convert.ToDouble(ddlvima.SelectedItem.ToString());
                }
                //search constraints
                for (int i = 0; i < countΕξυπηρετητές; i++)
                {
                    double sumBij = 0;
                    for (int j = 0; j < synoloS.Count; j++)
                    {
                        if  (alpha[synoloS[j]]  -  beta[i,  synoloS[j]]  >=
cij[i, synoloS[j]])
                        {
                            if (psi_tentative[i] == 1)
                            {
                                xij_tentative[i, synoloS[j]] = 1;
                                synoloS.RemoveAt(j);
                                j--;
                            }
                            else if (psi[i] == 1)
                            {
                                xij[i, synoloS[j]] = 1;
                                cij[i, synoloS[j]] += 100000000000;
                                ar[synoloS[j]]--;
                                synoloS.RemoveAt(j);
                                j--;
                            }
                            else
                            {
                                beta[i,synoloS[j]]                       +=
Convert.ToDouble(ddlvima.SelectedItem.ToString());
                                sumBij += beta[i, synoloS[j]];
                            }
                        }
                    }
                    if (sumBij >= fi[i] &&psi[i]<1)
                    {
                        psi_tentative[i] = 1;
                        for (int j = 0; j < synoloS.Count; j++)
                        {
                            if (alpha[synoloS[j]] - beta[i, synoloS[j]] + 1
>= cij[i, synoloS[j]])
                            {
                                xij_tentative[i, synoloS[j]] = 1;
                                synoloS.RemoveAt(j);
                                j--;
```

```
                            }
                        }
                    }
                }
            }
            /////phase 2
            //Create graph
            List<int> vertices = new List<int>();
            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                if (psi_tentative[i] == 1)
                {
                    vertices.Add(i);
                }
            }
            if (vertices.Count > 0)
            {
                int[,] edjes = new int[vertices.Count, vertices.Count];
                //initialize table
                for (int j = 0; j < vertices.Count; j++)
                {
                    for (int i = 0; i < vertices.Count; i++)
                    {
                        edjes[i, j] = 0;
                    }
                }
                for (int j = 0; j < countΠελάτες; j++)
                {
                    List<int> paidΕξυπηρετητές = new List<int>();
                    for (int i = 0; i < vertices.Count; i++)
                    {
                        if (beta[vertices[i], j] > 0)
                        {
                            paidΕξυπηρετητές.Add(vertices[i]);
                        }
                    }
                    while (paidΕξυπηρετητές.Count > 1)
                    {
                        for (int i = 1; i < paidΕξυπηρετητές.Count; i++)
                        {
                            edjes[vertices.IndexOf(paidΕξυπηρετητές[0]),
vertices.IndexOf(paidΕξυπηρετητές[i])] = 1;
                            edjes[vertices.IndexOf(paidΕξυπηρετητές[i]),
vertices.IndexOf(paidΕξυπηρετητές[0])] = 1;
                            paidΕξυπηρετητές.RemoveAt(0);
                        }
                    }
                }
                int[] degree = new int[vertices.Count];

                //find maximun independent set
                List<int> openFacility = new List<int>();
                while (true)
                {
                    for (int i = 0; i < vertices.Count; i++)
                    {
                        degree[i] = 100000;
                        for (int j = 0; j < vertices.Count; j++)
                        {
                            if (edjes[i, j] > -1 && degree[i] == 100000)
                            {
```

```
                            degree[i] = 0;
                        }
                        if (edjes[i, j] == 1)
                        {
                            degree[i]++;
                        }
                    }
                }
                int minDegree = 100000;
                int index = 0;
                for (int i = 0; i < vertices.Count; i++)
                {
                    if (degree[i] < minDegree)
                    {
                        minDegree = degree[i];
                        index = i;
                    }
                }
                if (minDegree < 100000)
                {
                    openFacility.Add(vertices[index]);
                    //delete vertice from original vertices
                    for (int i = 0; i < vertices.Count; i++)
                    {
                        if (edjes[index, i] == 1)
                        {
                            for (int j = 0; j < vertices.Count; j++)
                            {
                                edjes[i, j] = -1;
                                edjes[j, i] = -1;
                            }
                        }
                        edjes[index, i] = -1;
                        edjes[i, index] = -1;
                    }
                }
                else
                {
                    break;
                }
            }
            //open εξυπηρετητές
            for (int i = 0; i < openFacility.Count; i++)
            {
                psi[openFacility[i]] = 1;
                fi[openFacility[i]] = 0;
                //connect πελάτες to opened εξυπηρετητές
                for (int j = 0; j < countΠελάτες; j++)
                {
                    if (xij_tentative[openFacility[i], j] == 1)
                    {
                        xij[openFacility[i], j] = 1;
                        cij[openFacility[i], j] += 100000000000;
                        ar[j]--;
                    }
                }
            }
        }
    }
    //count cost of solution
    double cost = 0;
```

```
        for (int i = 0; i < countΕξυπηρετητές; i++)
        {
            if (psi[i] == 1)
            {
                cost += fi_initial[i];
                for (int j = 0; j < countΠελάτες; j++)
                {
                    if (xij[i, j] == 1)
                    {
                        cost += cij_initial[i, j];
                    }
                }
            }
        }
        return cost;
    }
```

Κώδικας Hybrid αλγορίθμου

```
protected double NewAlgorithm(int countΕξυπηρετητές, int countΠελάτες, int
count)
    {
        string path = @"E:\bin\";
        if (ddlFolder.SelectedIndex != 0)
        {
            path += ddlFolder.SelectedItem.ToString() + @"\";
        }
            int countX = countΕξυπηρετητές * countΠελάτες;
            var  text1  =  File.ReadAllText(@"  "  +  path  +  "ftfl_"  +
count.ToString() + ".sol");
            string txt1 = text1.ToString();
            string[] Split1 = txt1.Split(new Char[] { '\r' });
            int rows1 = Split1.GetLength(0);
            //get y
            List<double> psi = new List<double>();
            for (int i = countΕξυπηρετητές; i > 0; i--)
            {
                string y = Split1[rows1 - 21 - i].Substring(25).Remove(23);
                double yi = Convert.ToDouble(y.Trim());
                psi.Add(yi);
            }
            //get x
            double[,] xij = new double[countΕξυπηρετητές, countΠελάτες];
            for (int i = 0; i <countX; i++)
            {
                string x = Split1[rows1 - 21 - countΕξυπηρετητές - countX+
i].Substring(25).Remove(23);
                xij[(i%countΕξυπηρετητές),(i/countΕξυπηρετητές)]=
Convert.ToDouble(x.Trim());
            }
            //read f,c,r
            var  text2  =  File.ReadAllText(@"  "  +  path  +  "ftfl_"  +
count.ToString() + ".dat");
            string txt2 = text2.ToString();
            string[] Split2 = txt2.Split(new Char[] { '\r' });
            int rows2 = Split2.GetLength(0);
            //get r
            List<int> ar = new List<int>();
            List<int> ar_initial = new List<int>();
            for (int i = countΠελάτες; i > 0; i--)
            {
```

```
                int index = Split2[rows2 - 3 - i].IndexOf(' ');
                string r = Split2[rows2 - 3 - i].Substring(index);
                int rj = Convert.ToInt16(r.Trim());
                ar.Add(rj);
                ar_initial.Add(rj);
            }
            //get c
            double[,] cij = new double[countΕξυπηρετητές, countΠελάτες];
            double[,] cij_initial = new double[countΕξυπηρετητές,
countΠελάτες];
            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                string rowc = Split2[rows2 - 5 - countΠελάτες -
countΕξυπηρετητές + i];
                string[] Split3 = rowc.Split(new Char[] { ' ' });
                for (int j = 0; j < countΠελάτες; j++)
                {
                    cij[i, j] = Convert.ToDouble(Split3[j + 1].Trim());
                    cij_initial[i, j] = Convert.ToDouble(Split3[j +
1].Trim());
                }
            }
            //get f
            List<double> fi = new List<double>();
            List<double> fi_initial = new List<double>();
            for (int i = countΕξυπηρετητές; i > 0; i--)
            {
                int index = Split2[rows2 - 7 - countΕξυπηρετητές -
countΠελάτες - i].IndexOf(' ');
                string f = Split2[rows2 - 7 - countΕξυπηρετητές -
countΠελάτες - i].Substring(index);
                double fy = Convert.ToDouble(f.Trim());
                fi.Add(fy);
                fi_initial.Add(fy);
            }
            //phase 1
            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                if (psi[i] == 1)
                {
                    fi[i] = 0;
                    for (int j = 0; j < countΠελάτες; j++)
                    {
                        if (xij[i,j] > 0)
                        {
                            xij[i,j] = 1;

                            cij[i, j] += 100000000000;
                            ar[j]--;
                        }
                    }
                }
            }
            //phase 2
            bool finished = false;
            while (!finished)
            {
                double[] alpha = new double[countΠελάτες];
                double[,] beta = new double[countΕξυπηρετητές,
countΠελάτες];
                double[] psi_tentative = new double[countΕξυπηρετητές];
```

```
                double[,] xij_tentative = new double[countΕξυπηρετητές,
countΠελάτες];
                //put in synoloS πελάτες with remaining requirements
                List<int> synoloS = new List<int>();
                for (int j = 0; j < countΠελάτες; j++)
                {
                    if (ar[j] == ar.Max() && ar.Max() > 0)
                    {
                        synoloS.Add(j);
                    }
                }
                if (synoloS.Count == 0)
                {
                    finished = true;
                    break;
                }
                //begin subphase 1
                while (synoloS.Count > 0)
                {
                    //increase all aj by 1
                    for (int j = 0; j < synoloS.Count; j++)
                    {
                        alpha[synoloS[j]]                                   +=
Convert.ToDouble(ddlvima.SelectedItem.ToString());
                    }
                    //search constraints
                    for (int i = 0; i < countΕξυπηρετητές; i++)
                    {
                        double sumBij = 0;
                        for (int j = 0; j < synoloS.Count; j++)
                        {
                            if (alpha[synoloS[j]] - beta[i, synoloS[j]] >=
cij[i, synoloS[j]])
                            {
                                if (psi_tentative[i] == 1)
                                {
                                    xij_tentative[i, synoloS[j]] = 1;
                                    synoloS.RemoveAt(j);
                                    j--;
                                }
                                else if (psi[i] == 1)
                                {
                                    xij[i, synoloS[j]] = 1;
                                    cij[i, synoloS[j]] += 100000000000;
                                    ar[synoloS[j]]--;
                                    synoloS.RemoveAt(j);
                                    j--;
                                }
                                else
                                {
                                    beta[i,          synoloS[j]]          +=
Convert.ToDouble(ddlvima.SelectedItem.ToString());
                                    sumBij += beta[i, synoloS[j]];
                                }
                            }
                        }
                        if (sumBij >= fi[i] && psi[i]<1)
                        {
                            psi_tentative[i] = 1;
                            for (int j = 0; j < synoloS.Count; j++)
                            {
```

```
                                if (alpha[synoloS[j]] - beta[i, synoloS[j]]
+ 1 >= cij[i, synoloS[j]])
                                {
                                    xij_tentative[i, synoloS[j]] = 1;
                                    synoloS.RemoveAt(j);
                                    j--;
                                }
                            }
                        }
                    }
                    /////subphase 2
                    //Create graph
                    List<int> vertices = new List<int>();
                    for (int i = 0; i < countΕξυπηρετητές; i++)
                    {
                        if (psi_tentative[i] == 1)
                        {
                            vertices.Add(i);
                        }
                    }
                    if (vertices.Count > 0)
                    {
                        int[,] edjes = new int[vertices.Count, vertices.Count];
                        //initialize table
                        for (int j = 0; j < vertices.Count; j++)
                        {
                            for (int i = 0; i < vertices.Count; i++)
                            {
                                edjes[i, j] = 0;
                            }
                        }
                        for (int j = 0; j < countΠελάτες; j++)
                        {
                            List<int> paidΕξυπηρετητές = new List<int>();
                            for (int i = 0; i < vertices.Count; i++)
                            {
                                if (beta[vertices[i], j] > 0)
                                {
                                    paidΕξυπηρετητές.Add(vertices[i]);
                                }
                            }
                            while (paidΕξυπηρετητές.Count > 1)
                            {
                                for (int i = 1; i < paidΕξυπηρετητές.Count;
i++)
                                {

edjes[vertices.IndexOf(paidΕξυπηρετητές[0]),
vertices.IndexOf(paidΕξυπηρετητές[i])] = 1;

edjes[vertices.IndexOf(paidΕξυπηρετητές[i]),
vertices.IndexOf(paidΕξυπηρετητές[0])] = 1;
                                    paidΕξυπηρετητές.RemoveAt(0);
                                }
                            }
                        }
                        int[] degree = new int[vertices.Count];

                        //find maximun independent set
                        List<int> openFacility = new List<int>();
```

```
                    while (true)
                    {
                        for (int i = 0; i < vertices.Count; i++)
                        {
                            degree[i] = 100000;
                            for (int j = 0; j < vertices.Count; j++)
                            {
                                if (edjes[i, j] > -1 && degree[i] ==
100000)
                                {
                                    degree[i] = 0;
                                }
                                if (edjes[i, j] == 1)
                                {
                                    degree[i]++;
                                }
                            }
                        }
                        int minDegree = 100000;
                        int index = 0;
                        for (int i = 0; i < vertices.Count; i++)
                        {
                            if (degree[i] < minDegree)
                            {
                                minDegree = degree[i];
                                index = i;
                            }
                        }
                        if (minDegree < 100000)
                        {
                            openFacility.Add(vertices[index]);
                            //delete vertice from original vertices
                            for (int i = 0; i < vertices.Count; i++)
                            {
                                if (edjes[index, i] == 1)
                                {
                                    for (int j = 0; j < vertices.Count;
j++)
                                    {
                                        edjes[i, j] = -1;
                                        edjes[j, i] = -1;
                                    }
                                }
                                edjes[index, i] = -1;
                                edjes[i, index] = -1;
                            }
                        }
                        else
                        {
                            break;
                        }
                    }
                    //open εξυπηρετητές
                    for (int i = 0; i < openFacility.Count; i++)
                    {
                        psi[openFacility[i]] = 1;
                        fi[i] = 0;
                        //connect πελάτες to opened εξυπηρετητές
                        for (int j = 0; j < countΠελάτες; j++)
                        {
                            if (xij_tentative[openFacility[i], j] == 1)
```

```
                            {
                                xij[openFacility[i], j] = 1;
                                cij[openFacility[i], j] += 100000000000;
                                ar[j]--;
                            }
                        }
                    }
                }
            }
            //count cost of solution

            double cost = 0;
            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                if (psi[i] == 1)
                {
                    cost += fi_initial[i];
                    for (int j = 0; j < countΠελάτες; j++)
                    {
                        if (xij[i,j] == 1)
                        {
                            cost += cij_initial[i, j];
                        }
                    }
                }
            }
            //check feasibility
            for (int j = 0; j < countΠελάτες; j++)
            {
                int arj = 0;
                for (int i = 0; i < countΕξυπηρετητές; i++)
                {
                    if (xij[i,j] == 1 && psi[i] == 1)
                    {
                        arj++;
                    }
                }
                if (arj < ar_initial[j])
                {
                    return -1;
                }
            }
            return cost;
    }
```

Κώδικας repeated χαλαρωμένου αλγορίθμου

```
protected double phaseΧαλαρωμένου(int countΕξυπηρετητές, int countΠελάτες,
int count)
    {
        string path = @"E:\bin\";
        if (ddlFolder.SelectedIndex != 0)
        {
            path += ddlFolder.SelectedItem.ToString() + @"\";
        }
        int countX = countΕξυπηρετητές * countΠελάτες;
        string phase = count.ToString();
        List<double> psi = new List<double>();
        double[,] xij = new double[countΕξυπηρετητές, countΠελάτες];
        //read f,c,r
        var   text2   =   File.ReadAllText(@"   "   +   path   +   "ftfl_"   +
count.ToString() + ".dat");
```

```
        string txt2 = text2.ToString();
        string[] Split2 = txt2.Split(new Char[] { '\r' });
        int rows2 = Split2.GetLength(0);
        //get r
        List<int> ar_initial = new List<int>();
        List<int> ar = new List<int>();
        for (int i = countΠελάτες; i > 0; i--)
        {
            int index = Split2[rows2 - 3 - i].IndexOf(' ');
            string r = Split2[rows2 - 3 - i].Substring(index);
            int rj = Convert.ToInt16(r.Trim());
            ar_initial.Add(rj);
            ar.Add(rj);
        }
        //get c
        double[,] cij_initial = new double[countΕξυπηρετητές,
countΠελάτες];
        for (int i = 0; i < countΕξυπηρετητές; i++)
        {
            string rowc = Split2[rows2 - 5 - countΠελάτες -
countΕξυπηρετητές + i];
            string[] Split3 = rowc.Split(new Char[] { ' ' });
            for (int j = 0; j < countΠελάτες; j++)
            {
                cij_initial[i, j] = Convert.ToDouble(Split3[j + 1].Trim());
            }
        }
        //get f
        List<double> fi_initial = new List<double>();
        double cost = 0;
        for (int i = countΕξυπηρετητές; i > 0; i--)
        {
            int index = Split2[rows2 - 7 - countΕξυπηρετητές - countΠελάτες
- i].IndexOf(' ');
            string f = Split2[rows2 - 7 - countΕξυπηρετητές - countΠελάτες
- i].Substring(index);
            double fy = Convert.ToDouble(f.Trim());
            fi_initial.Add(fy);
        }
        List<double> psiPhase = new List<double>();
        for (int i = countΕξυπηρετητές; i > 0; i--)
        {
            psi.Add(0);
        }
        double[,] xijPhase = new double[countΕξυπηρετητές, countΠελάτες];
        var text = File.ReadAllText(@" " + path + "ftfl_" + phase +
".sol");
        string txt = text.ToString();
        string[] Split = txt.Split(new Char[] { '\r' });
        int rows = Split.GetLength(0);
        //get y
        for (int i = countΕξυπηρετητές; i > 0; i--)
        {
            string y = Split[rows - 21 - i].Substring(25).Remove(23);
            double yi = Convert.ToDouble(y.Trim());
            psiPhase.Add(yi);
        }
        //get x
        for (int i = 0; i < countX; i++)
        {
```

```
            string x = Split[rows - 21 - countΕξυπηρετητές - countX +
i].Substring(25).Remove(23);
            xijPhase[(i % countΕξυπηρετητές), (i / countΕξυπηρετητές)] =
Convert.ToDouble(x.Trim());
        }
        for (int i = 0; i < countΕξυπηρετητές; i++)
        {
            if (psiPhase[i] == psiPhase.Max())
            {
                psi[i] = 1;
                for (int j = 0; j < countΠελάτες; j++)
                {
                    if (xijPhase[i, j] > 0 && ar[j] > 0)
                    {
                        xij[i, j] = 1;
                        ar[j]--;
                    }
                }
            }
        }
        phase += "_2";
        int phases = 2;
        while(File.Exists(@" " + path + "ftfl_" + phase + ".sol"))
        {
            string line1 = "";
            string line2 = "";
            using (StreamReader reader = new StreamReader(@" " + path +
"ftfl_" + phase + ".dat"))
            {

                line1 = reader.ReadLine();
                line2 = reader.ReadLine();
            }

            int πελάτες = Convert.ToInt16(line2.Substring(11).Replace(";",
" "));
            int                          εξυπηρετητές                          =
Convert.ToInt16(line1.Substring(11).Replace(";", " "));
            int countXphase = εξυπηρετητές * πελάτες;
            List<int> ar_phase = new List<int>();
            List<int> ar_equal = new List<int>();
            for (int j = 0; j < countΠελάτες; j++)
            {
                ar_phase.Add(ar[j]);
                ar_equal.Add(ar[j]);
            }
            List<double> psiPhase1 = new List<double>();
            List<double> psi_equal = new List<double>();
            double[,] xijPhase1 = new double[εξυπηρετητές, πελάτες];

            var text1 = File.ReadAllText(@" " + path + "ftfl_" + phase +
".sol");
            string txt1 = text1.ToString();
            string[] Split1 = txt1.Split(new Char[] { '\r' });
            int rows1 = Split1.GetLength(0);
            //get y
            for (int i = εξυπηρετητές; i > 0; i--)
            {
                string y = Split1[rows1 - 21 - i].Substring(25).Remove(23);
                double yi = Convert.ToDouble(y.Trim());
                psiPhase1.Add(yi);
```

```
                psi_equal.Add(yi);
            }
            //get x
            for (int i = 0; i < countXphase; i++)
            {
                string x = Split1[rows1 - 21 - εξυπηρετητές - countXphase +
i].Substring(25).Remove(23);
                xijPhase1[(i  %  εξυπηρετητές),  (i  /  εξυπηρετητές)]  =
Convert.ToDouble(x.Trim());
            }
            if (psiPhase1.Max() == 0.5)
            {
                List<double> connections = new List<double>();
                for (int i = 0; i < εξυπηρετητές; i++)
                {
                    connections.Add(0);
                    for (int j = 0; j < πελάτες; j++)
                    {
                        if (xijPhase1[i, j] > 0)
                        {
                            connections[i]++;
                        }
                    }
                }
                while (ar_equal.Max() > 0)
                {
                    int open = connections.IndexOf(connections.Max());
                    connections[open] = 0;
                    psi_equal[open] = 0;
                    int countC = 0;
                    for (int j = 0; j < countΠελάτες; j++)
                    {
                        if (ar_phase[j] > 0)
                        {
                            if (xijPhase1[open, countC] > 0)
                            {
                                ar_equal[j]--;
                            }
                            countC++;
                        }
                    }
                }
                for (int i = 0; i < εξυπηρετητές; i++)
                {
                    if (psi_equal[i] > 0)
                    {
                        psiPhase1[i] = 0;
                    }
                }
            }
            int countF = 0;
            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                if (psi[i] != 1 && countF<εξυπηρετητές)
                {
                    if (psiPhase1[countF] == psiPhase1.Max())
                    {
                        psi[i] = 1;
                        int countC = 0;
                         for (int j = 0; j < countΠελάτες; j++)
                         {
```

```
                                if (ar_phase[j] > 0 && countC < πελάτες)
                                {
                                    if  (xijPhase1[countF,  countC]  >  0  &&
ar[j]>0)
                                    {
                                        xij[i, j] = 1;
                                        ar[j]--;
                                    }
                                    countC++;
                                }
                            }
                        }
                        countF++;
                    }
                }
                phases++;
                phase = count.ToString()+"_"+phases.ToString();
            }
            cost = 0;
            //count cost of solution

            for (int i = 0; i < countΕξυπηρετητές; i++)
            {
                if (psi[i] == 1)
                {
                    cost += fi_initial[i];
                    for (int j = 0; j < countΠελάτες; j++)
                    {
                        if (xij[i, j] == 1)
                        {
                            cost += cij_initial[i, j];
                        }
                    }
                }
            }
            //check feasibility
            for (int j = 0; j < countΠελάτες; j++)
            {
                int arj = 0;
                for (int i = 0; i < countΕξυπηρετητές; i++)
                {
                    if (xij[i, j] == 1 && psi[i] == 1)
                    {
                        arj++;
                    }
                }
                if (arj < ar_initial[j])
                {
                    return -1;
                }
            }
            return cost;
        }
```

Κώδικας κατασκευής στιγμιοτύπων

```
protected void serial(object sender, EventArgs e)
    {
        if (checkValid() == 1)
        {
            try
            {
```

```
                int countΕξυπηρετητές = Convert.ToInt16(txtΕξυπηρετητές.Text);
                int countΠελάτες = Convert.ToInt16(txtΠελάτες.Text);
                int decim = Convert.ToInt16(txtDec.Text);
                int εγκατάστασης = Convert.ToInt16(txtΕγκατάστασης.Text);
                int start = Convert.ToInt16(txtNumber.Text.ToString());
                int dimensions = Convert.ToInt16(txtDimensions.Text.ToString());
                int sc = Convert.ToInt16(txtcost.Text);
                Random rnd = new Random();
                // create a writer and open the file
                for (int k = 0; k < Convert.ToInt16(txtRepeats.Text.ToString()); k++)
                {
                    int offsetΕγκατάστασης = 0;
                    int offsetService = 0;

                    File.Delete(@"C:\Program Files (x86)\GnuWin32\bin\ftfl_" + (start + k).ToString() + ".dat");
                    File.Delete(@"C:\Program Files (x86)\GnuWin32\bin\ftfl_" + (start + k).ToString() + ".sol");
                    File.Delete(@"C:\Program Files (x86)\GnuWin32\bin\ftfl_int_" + (start + k).ToString() + ".sol");
                    msg.Visible = false;
                    TextWriter tw = new StreamWriter(@"C:\Program Files (x86)\GnuWin32\bin\ftfl_" + (start + k).ToString() + ".dat");
                    string data = "param i := " + countΕξυπηρετητές + ";\r\n";
                    data += "param j := " + countΠελάτες + ";\r\n";
                    data += "param : f :=\r\n";
                    double ran;
                    for (int i = 1; i <= countΕξυπηρετητές; i++)
                    {
                        ran = Math.Round(rnd.NextDouble() * εγκατάστασης, decim) + offsetΕγκατάστασης;
                        data += i.ToString() + " " + ran.ToString() + "\r\n";
                    }
                    data += ";\r\n param c :";

                    double[,] εξυπηρετητές = new double[countΕξυπηρετητές, dimensions];
                    double[,] πελάτες = new double[countΠελάτες, dimensions];

                    for (int i = 1; i <= countΠελάτες; i++)
                    {
                        data += i.ToString() + " ";
                    }
                    data += ":=\r\n";
                    for (int i = 1; i <= countΕξυπηρετητές; i++)
                    {
                        for (int d = 0; d < dimensions; d++)
                        {
                            εξυπηρετητές[i - 1, d] = Math.Round(rnd.NextDouble() * sc, 2);
                        }
                        data += i.ToString() + " ";
                        for (int j = 1; j <= countΠελάτες; j++)
                        {
                            for (int d = 0; d < dimensions; d++)
```

```
                                {
                                    πελάτες[j - 1, d] =
Math.Round(rnd.NextDouble() * sc, 2);
                                }
                                double sum = 0;
                                for (int d = 0; d < dimensions; d++)
                                {
                                    sum += Math.Pow(εξυπηρετητές[i - 1, d] -
πελάτες[j - 1, d], 2);
                                }
                                data += (Math.Round(Math.Sqrt(sum), decim) +
offsetService).ToString() + " ";
                            }
                            data += "\r\n";
                        }
                        data += ";\r\n param : r :=\r\n";
                        for (int i = 1; i <= countΠελάτες; i++)
                        {
                            ran = rnd.Next(1,
Convert.ToInt16(txtRedundancy.Text.ToString()) + 1);
                            data += i.ToString() + " " + ran.ToString() +
"\r\n";
                        }
                        data += ";\r\n end;";

                        // write a line of text to the file
                        tw.WriteLine(data.ToString());

                        // close the stream
                        tw.Close();

                        count = start + k;

                        ftfl_nomip("_" + count.ToString());
                        ftfl("_" + count.ToString());
                    }
                }
                catch { }
            }
        }
```

ΠΑΡΑΡΤΗΜΑ Β

Τύποι τυχαίων στιγμιοτύπων που κατασκευάστηκαν.

50 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 20, 300 στιγμιότυπα
50 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 50, 300 στιγμιότυπα
50 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 100, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 10, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 20, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 30, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 40, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 50, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 70, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 100, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 150, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 200, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 10, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 20, 300 στιγμιότυπα
100 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 50, 300 στιγμιότυπα
100 πελάτες, 200 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 20, 100 στιγμιότυπα
100 πελάτες, 200 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 50, 100 στιγμιότυπα
100 πελάτες, 200 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 20, 100 στιγμιότυπα
100 πελάτες, 200 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 50, 100 στιγμιότυπα
100 πελάτες, 200 εξυπηρετητές, r_j από 1 έως 30, f_i από 0 έως 100, c_{ij} από 0 έως 20, 100 στιγμιότυπα
100 πελάτες, 200 εξυπηρετητές, r_j από 1 έως 30, f_i από 0 έως 100, c_{ij} από 0 έως 50, 100 στιγμιότυπα
100 πελάτες, 300 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 20, 50 στιγμιότυπα
100 πελάτες, 300 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 50, 50 στιγμιότυπα

100 πελάτες, 300 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 20, 50 στιγμιότυπα
100 πελάτες, 300 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 50, 50 στιγμιότυπα
100 πελάτες, 300 εξυπηρετητές, r_j από 1 έως 30, f_i από 0 έως 100, c_{ij} από 0 έως 20, 50 στιγμιότυπα
100 πελάτες, 300 εξυπηρετητές, r_j από 1 έως 30, f_i από 0 έως 100, c_{ij} από 0 έως 50, 50 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 20, 20 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 50, 20 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 20, 20 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 10, f_i από 0 έως 100, c_{ij} από 0 έως 50, 20 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 20, f_i από 0 έως 100, c_{ij} από 0 έως 20, 20 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 20, f_i από 0 έως 100, c_{ij} από 0 έως 50, 20 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 30, f_i από 0 έως 100, c_{ij} από 0 έως 20, 20 στιγμιότυπα
100 πελάτες, 1000 εξυπηρετητές, r_j από 1 έως 30, f_i από 0 έως 100, c_{ij} από 0 έως 50, 20 στιγμιότυπα
200 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 20, 200 στιγμιότυπα
200 πελάτες, 100 εξυπηρετητές, r_j από 1 έως 50, f_i από 0 έως 100, c_{ij} από 0 έως 50, 20 στιγμιότυπα
250 πελάτες, 50 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 20, 50 στιγμιότυπα
300 πελάτες, 100 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 50, 50 στιγμιότυπα
400 πελάτες, 200 εξυπηρετητές, r_j από 1 έως 5, f_i από 0 έως 100, c_{ij} από 0 έως 50, 20 στιγμιότυπα