ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τοπική Αναζήτηση Για Προβλήματα Βελτιστοποίησης

Χαράλαμπος Ιωάννης - Ι. - Μητρόπουλος

Επιβλέπων: Ζησιμόπουλος Βασίλειος, Καθηγητής

ΑΘΗΝΑ
ΟΚΤΩΒΡΙΟΣ 2017

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τοπική Αναζήτηση Για Προβλήματα Βελτιστοποίησης

Χαράλαμπος-Ιωάννης Ι. Μητρόπουλος
Α.Μ: 1115201100072

Επιβλέπων: Ζησιμόπουλος Βασίλειος, Καθηγητής

ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία έχει ως σκοπό να παρουσιάσει και να μελετήσει κάποιες γειτονιές που μπορούν να συμβάλουν σε καλύτερα αποτελέσματα πάνω σε προβλήματα, με χρήση της μεθόδου της Τοπικής Αναζήτησης. Έτσι για κάθε ένα πρόβλημα πρώτα γίνεται η παρουσίαση του, ακολουθεί ο ορισμός μιας καλής γειτονιάς για το πρόβλημα αυτό και η κατάληξη είναι η παρουσίαση ενός αλγορίθμου τοπικής αναζήτησης για το εκάστοτε πρόβλημα. Τέλος γίνεται προσπάθεια να βρεθεί τρόπος για να ορίζουμε καλύτερες γειτονιές ανάλογα με το πρόβλημα έτσι ώστε να βρούμε αν γίνεται να υπάρξουν καλύτερα αποτελέσματα από τον αλγόριθμο της τοπικής αναζήτησης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τοπική Αναζήτηση.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Τοπική Αναζήτηση, Πρόβλημα Χωροτοποθέτησης, Κινητή Χωροτοποθέτηση, Πρόβλημα Ικανοποίησημότητας.

ABSTRACT

This dissertation's purpose is to present and study neighborhoods for some problems that concern the Local Search method. Moreover we focus on some neighborhoods that can improve our results in these problems. For each problem first we define a good neighborhood and then a Local Search Algorithm for each problem. Finally we make an attempt to find a way in order to define better neighborhoods to produce better results from the Local Search Algorithm.

SUBJECT AREA: Local Search

KEYWORDS: Local Search, Facility Location Problem, Mobile Facility Location, Boolean satisfiability problem.

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία εκπονήθηκε υπο την επίβλεψη του καθηγητή κ.Βασιλείου Ζησιμόπουλου, τον οποίο θα ήθελα να ευχαριστήσω για τις συμβουλές και τον χρόνο του που διέθεσε απλόχερα κατά την εκπόνηση της εργασίας, αλλά κυρίως για την ενθάρρυνση και υποστήριξη που μου προσέφερε. Τέλος, ευχαριστώ την Μαριάντζελα, τους φίλους μου καθώς και την οικογένεια μου για την ηθική υποστήριξη που μου παρείχαν.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΠΡΟΛΟΓΟΣ

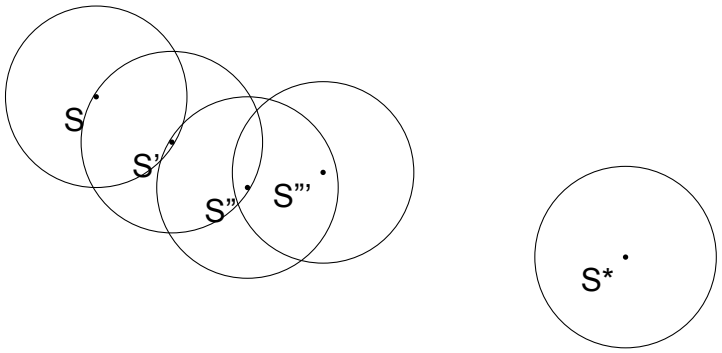
Η πτυχιακή εργασία πραγματοποιήθηκε απο το ενδιαφέρον που είχα πάνω στο μάθημα των Αλγορίθμων και Πολυπλοκότητας και πιο συγκεκριμένα για τη μέθοδο της Τοπικής Αναζήτησης πάνω σε κάποια προβλήματα που αφορούν την επιστήμη των υπολογιστών. Η εργασία αυτή αποτέλεσε έναν τρόπο εμβάθυνσης στο αντικείμενο της Τοπικής Αναζήτησης, καθώς μέσω διαφόρων προβλημάτων δόθηκε η ευκαιρία για περαιτέρω εντρύφηση με το εν λόγω αντικείμενο. Η αποπεράτωση της εργασίας έγινε με διάβασμα των αντίστοιχων προβλημάτων, (όπως το πρόβλημα του Ελάχιστου Καλύμματος Κορυφών, το πρόβλημα του Πλανόδιου Πολιτή και άλλα), την κατανόηση τους και εν τέλει την διεξαγωγή κάποιων συμπερασμάτων για τους αλγορίθμους αυτούς που μπορούν να συμβάλλουν στην βελτίωση τους.

1.ΕΙΣΑΓΩΓΗ

Η Τοπική Αναζήτηση(Local Search) είναι μια γενική μέθοδος που εξευρευνά τον χώρο των δυνατών λύσεων ενός προβλήματος πηγαίνοντας από μια εφικτή λύση S, σε μια γειτονική της S'. Η μέθοδος αυτή είναι εύκολο να σχεδιαστεί για σχεδόν κάθε υπολογιστικό προβλημα, βρίσκοντας καλές λύσεις χωρίς όμως να είναι αναγκαία οι βέλτιστες. Γι'αυτό προσπαθούμε να βρούμε την σχέση της τοπικά βέλτιστης λύσης S με την ολικά βέλτιστη S'. Σε μερικές περιπτώσεις προβλημάτων καταφέρνουμε να βρούμε μια σχέση μεταξύ του S και του S'.

Σε αυτό το σημείο μπορούμε να πούμε με γενικά λόγια πως μπορεί να οριστεί ο αλγόριθμος της Τοπικής Αναζήτησης για προβλήματα Ελαχιστοποίησης και για προβλήματα Μεγιστοποίησης. Έστω P πρόβλημα ελαχιστοποίησης και λυση S του P. Για την λύση S ορίζουμε μια γειτονιά N(S) και σε αυτή τη γειτονιά ψάχνουμε αν υπάρχει λύση S', ώστε $w(S') < w(S)$, με w(S) και w(S') τα βάρη των λύσεων S και S' αντίστοιχα. Αν υπάρχει λύση S' την κρατάμε και ορίζουμε νέα γειτονιά N(S'). Επαναλαμβάνουμε τη διαδικασία μέχρι να μη βρούμε καλύτερη λύση. Έτσι για προβλήματα Ελαχιστοποίησης και Μεγιστοποίησης έχουμε αντίστοιχα:

Πρόβλημα Μεγιστοποίησης	Πρόβλημα Ελαχιστοποίησης
P: Πρόβλημα προς επίλυση S: Μια λύση του προβλήματος N(S): Γειτονιά της λύσης S του προβλήματος w(S): Το βάρος λύσης του προβλήματος	P: Πρόβλημα προς επίλυση S: Μια λύση του προβλήματος N(S): Γειτονιά της λύσης S του προβλήματος w(S): Το βάρος λύσης του προβλήματος
Αλγόριθμος Τοπικής Αναζήτησης: 'Οσο $w(S') > w(S)$, $S' \in N(S)$ $S = S'$ $w(S) = w(S')$ Επανέλαβε	Αλγόριθμος Τοπικής Αναζήτησης: 'Οσο $w(S') < w(S)$, $S' \in N(S)$ $S = S'$ $w(S) = w(S')$ Επανέλαβε



Σχήμα 1.1: Εύρεση καλύτερης λύσης.
Για ένα πρόβλημα Ελαχιστοποίησης: Ξεκινώντας από την λύση S και τη γειτονιά της (κύκλος), συνεχίζουμε στην $S^{'}$ που ειναι καλύτερη, μέχρι και την $S^{'''}$ όπου δεν μπορούμε να πάμε σε καλύτερη λύση. Η S^{*}, είναι η βέλτιστη λύση που δεν μπορεί να επιτευχθεί ποτέ ξεκινώντας από την S.

2.ΠΡΟΒΛΗΜΑΤΑ ΤΟΠΙΚΗΣ ΑΝΑΖΗΤΗΣΗΣ

2.1 Ελάχιστο Κάλυμμα Κορυφών (Vertex Cover)

Πρόβλημα:

Δεδομένου ενός γράφου G, πρέπει να βρούμε ενα υποσύνολο κόμβων έτσι ώστε κάθε ακμή να επικαλύπτεται, δηλαδή ένα εκ των 2 άκρων κάθε ακμής να είναι επιλεγμένο.

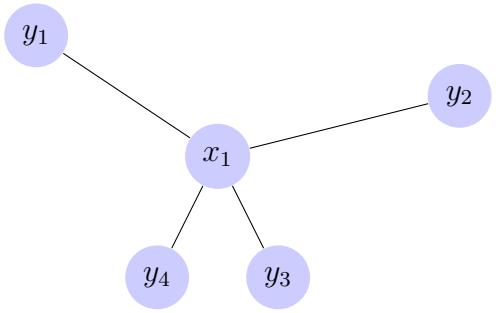
Γειτονιά:

Για να φτιάξουμε μια λύση για το Πρόβλημα του Ελάχιστου Καλύμματος Κορυφών [8], επιλέγουμε αυθαίρετα μια αρχική λύση S, που έχει κάποιους αρχικούς κόμβους. Κάθε λύση έχει κόστος cost(S), που είναι όσο και το μέγεθος του. Θα βρίσκουμε νέα λύση S' που θα προκύπτει από την S, κάνοντας προσθήκη ή διαγραφή ένος κόμβου. Μπορούμε επίσης να παρατηρήσουμε οτι κάθε λύση S έχει το πολύ n γειτονικές λύσεις, καθώς μια νέα γειτονιά προκύπτει από την προσθήκη ή την αφαίρεση ενός και μόνο κόμβου.

Αλγόριθμος Τοπικής Αναζήτησης:

S $\leftarrow$ αρχική λύση
'Οσο $\exists S'$ τέτοια ώστε $cost(S') < cost(S)$
    S $\leftarrow$ S'
Επανέλαβε.

Πολλές φορές θα πρέπει να προσέχουμε στο πρόβλημα του Ελάχιστου Καλύμματος Κορυφών από ποια λύση αρχίζουμε και ποια θα είναι η επόμενη. Μπορεί με μια λάθος επιλογή να μη βρούμε ποτέ το ολικό βέλτιστο. Παράδειγμα αν έχουμε έναν γράφο αστέρι(star graph) $x_1, y_1, y_2, y_3, y_{n-1}$, τότε το ελάχιστο κάλυμμα κορυφών, είναι το x_1. Αν όμως άντι να διαγράψουμε έναν από τους y_1, y_2, y_3, y_{n-1}, διαγράψουμε τον x_1 τότε δεν θα βρούμε ποτέ την ολικά βέλτιστη λύση του προβλήματος γι'αυτόν τον γράφο.



Σχήμα 2.1: Γράφος Αστέρι.
Ένας γράφος αστέρι, όπου αν διαγράψω πρώτα την κορυφή x_1 δεν θα βρεθεί ποτέ το ολικό ελάχιστο του προβλήματος επικάλυψης κορυφών.

2.2 Νευρωνικά Δίκτυα Hopfield

Πρόβλημα:

Τα νευρωνικά δίκτυα Hopfield(Hopfields Networks) [8] είναι ένα μοντέλο επικουρικής μνήμης, στο οποίο λειτουργεί ως μια διευθυνσιοδοτούμενη μνήμη περιεχομένου(content-addressable memory) με δυαδικούς κόμβους. Υπάρχει εγγύηση ότι θα καταλήξουμε σε τοπικό ελάχιστο, ωστόσο μερικές φορές ακολουθώντας λάθος βήματα μπορούμε να οδηγηθούμε σε λάθος τοπικό βέλτιστο και όχι στο προσδοκώμενο. Πιο αναλυτικά έχουμε μια συλλογή από ενότητες(units) που μπορούν να είναι σε μόνο δύο διαφορετικές καταστάσεις(-1 ή 1) και συνδέονται μεταξύ τους και οι γειτονικές ενότητες προσπαθούν να συσχετίσουν τις καταστά-σεις (states) τους. Μπορούμε να θεωρήσουμε ένα νευρωνικό δίκτυο Hopfield ως έναν $G = (V, E)$ γράφο με ακέραια w_e σε κάθε ακμή e. Ονομάζουμε σχηματισμό S(configuration) την ανάθεση τιμής -1 ή +1 σε κόμβο u και την ονομάζουμε s_u. Αν ο u συνδεθεί με τον v με ακμή αρνητικού βάρους, τότε οι u, v θέλουν να έχουν την ίδια κατάσταση, ενώ αν συνδεθούν με ακμή θετικού βάρους θα έχουν διαφορετική κατάσταση. Η τιμή $|w_e|$ δηλώνει την απολύτη τιμή του βάρους της ακμής e.
Έτσι για μια ακμή $e = (u, v)$ θα λέμε ότι:

Είναι καλή (good e) : Αν η απαίτηση της ικανοποιείται από τα 2 άκρα της: Είτε $w_e < 0$ και $s_u = s_v$ είτε $w_e > 0$ και $s_u \neq s_v$.

Είναι κακή (bad e) : Αν η απαίτηση της δεν ικανοποιείται από τα 2 άκρα της: Είτε $w_e < 0$ και $s_u \neq s_v$ είτε $w_e > 0$ και $s_u = s_v$. Συμπεραίνουμε λοιπόν ότι μια ακμή e είναι καλή αν και μόνον αν: $w_e s_u s_v \leq 0$. Τέλος, λέμε ότι ένας κόμβος u είναι ικανοποιημένος αν το συνολικό απολυτο βάρος όλων των καλών ακμών που προσπίπτουν στο u είναι το λιγότερο όσο το συνολικό βάρος όλων των κακών ακμών που προσπίπτουν στο u. Δηλαδή: $\sum_{v:e=(u,v)\in E} w_e s_u s_v$

Αυτό που επιδιώκουμε είναι να κάνουμε τις κατάλληλες αλλαγές στις ακμές του Νευρωνικού Δικτύου έτσι ώστε να μην υπάρχει κανένας κόμβος που να είναι ανικανοποίητος.

Γειτονιά:

Για τα Νευρωνικά Δίκτυα Hopfield ορίζουμε μια κατάσταση ως σταθερή(stable) αν όλοι οι κόμβοι είναι ικανοποιημένοι(satisfied) σε αυτή την κατάσταση.

Για το πρόβλημα των Νευρωνικών Δικτύων Hopfield θα χρησιμοποιήσουμε τις γειτονιές αλλαγής κατάστασης(State-flipping Neighborhood) [8]. Έτσι αυτό που κάνουμε είναι ότι όσο η τωρινή κατάσταση δεν είναι σταθερή($stable$), ψάχνουμε έναν ανικανοποίητο κόμβο u και κάνουμε αλλαγή την κατάσταση του. Αυτή η διαδικασία συνεχίζεται μέχρι να μην υπάρχει τέτοιος ανικανοποίητος κόμβος.

Αλγόριθμος Τοπικής Αναζήτησης:

Στον παρακάτω αλγόριθμο τοπικής αναζήτησης θα παρουσιάσουμε μια σχέση ανάμεσα στο τοπικό και ολικό βέλτιστο σε σχέση με το συνολικό βάρος καλών και κακών ακμών. Έτσι μπορούμε να δούμε τον αλγόριθμο της τοπικής αναζήτησης.

$S \leftarrow$ μια τυχαία κατάσταση
$\Phi(S) = \sum_{goode} |w_e|$. (όπου $\Phi(S)$ το συνολικό απολυτο βάρος όλων των καλών ακμών)
Επίσης για κάθε configuration S έχουμε ότι:
- $\Phi(S) \geq 0$ (αφού το $\Phi(S)$ είναι το άθροισμα των θετικών ακεραίων).
- $\Phi(S) \leq W = \sum_e |w_e|$ (αν το πολύ μία ακμή είναι καλή).

Όσο $\exists u \leftarrow$ ανικανοποίητος
$u \leftarrow$ ικανοποιημένος
$S \leftarrow S'$
Έτσι αν θεωρήσουμε:
- $g_u \leftarrow$ συνολικό απολυτο βάρος καλών ακμών και
- $b_u \leftarrow$ συνολικό απολυτο βάρος κακών ακμών.

$\Rightarrow \Phi(S') = \Phi(S) - g_u + b_u$.
Αφού όμως :
1. ο u είναι ικανοποιημένος στο S και
2. $b_u \geq g_u$ (με g_u, b_u και οι 2 ακέραιοι)

$\Rightarrow b_u = g_u + 1$
Άρα : $\Phi(S') = \Phi(S) + 1$.

Έτσι η τιμή του Φ ξεκινάει από έναν μη αρνητικό ακέραιο και αυξάνεται το πολύ 1 σε κάθε αλλαγή και δεν ξεπερνά το W. Σε κάθε επανάληψη του αλγορίθμου για να εντοπίσουμε έναν ανικανοποίητο κόμβο χρησιμοποιούμε πράξεις n πολυωνυμικού πλήθους.

2.3 Αλγόριθμος Metropolis

Πρόβλημα:

Το πρόβλημα του Αλγορίθμου Metropolis(Metropolis Algorithm) όπως αναφέρεται στο βιβλίο `Algorithm Design` [8] προέκυψε από την προσπάθεια προσομοίωσης της συμπεριφοράς ενός φυσικού συστήματος στατιστικής φυσικής. Η επιδίωξη είναι ότι από μια αρχική κατάσταση, προσπαθούμε με μικρές διαταράξεις ενέργειας, να μεταβούμε σε μια νέα κατάσταση, ενδιαφερόμενοι μόνο για τις καταστάσεις ενέργειας που είναι εφικτό να φτάσουμε.

Γειτονιά:

Ξεκινάμε από μια προκαθορισμένη κατάσταση S, υποθέτουμε ότι υπάρχει πεπερασμένος αριθμός από καταστάσεις ενέργειας. Κάνοντας αλλαγές στην υπάρχουσα κατάσταση S μεταβαίνουμε σε μια νέα κατάσταση S'. Για να μεταβούμε σε μια νέα κατάσταση έχουμε δύο επιλογές :

1. Αν $(S') \leq (S)$. Δηλαδή αν η διαφορά ενεργειών είναι αρνητική τότε η S παραμένει η ίδια.
2. Βρίσκουμε τη διαφορά των 2 ενεργειών $\Delta E = E(S') - E(S)$ και αν $\Delta E > 0$ μεταβαίνουμε στη νέα κατάσταση με πιθανότητα $p = e^{-\Delta/(kT)}$.

Όπου $e^{-/(kT)}$ είναι η συνάρτηση Gibbs-Boltzmann, T είναι η θερμοκρασία και k σταθερά

$k > 0$.

Αλγόριθμος Τοπικής Αναζήτησης:

$S \leftarrow$ τυχαία κατάσταση
$k \leftarrow$ σταθερά
$\leftarrow$ σταθερά
Διαλέγουμε τυχαία κατάσταση S'
γειτονική της τωρινής μας λύσης.
Αν ($E(S') \leq E(S)$) τότε
    $S' \leftarrow S$
Αλλιώς
    Αν ($\Delta E = E(S') - E(S) > 0$) τότε
        $p \leftarrow e^{-\Delta/(kT)}$
        $S' \leftarrow S$ (με πιθανότητα p)
Επανέλαβε.

Υπάρχουν φορές όπου ο αλγόριθμος Metropolis δεν λειτουργεί όπως θα θέλαμε, όπως π.χ. αν έχουμε έναν γράφο G χωρίς ακμές. Με $Gradientdescent$ μέθοδο το πρόβλημα θα λυνόταν γρήγορα σβήνοντας έναν έναν κόμβο μέχρι να μην έμενε κανένας. Ωστόσο, με τον αλγόριθμο Metropolis αν έχουμε μία περίπτωση που η τωρινή μας λύση έχει n κόμβους, οπου $c << n$ και n είναι όλοι οι κόμβοι του γράφου, τότε σε αυτή την περίπτωση έχουμε μια μεγάλη πιθανότητα υπάρχει το ενδεχόμενο η λύση που θα προκύψει να έχει $C + 1$ αντί για $C - 1$ κόμβους.

2.4 Προσέγγιση Προβλήματος Μέγιστης Τομής Γράφου μέσω της Τοπικής Αναζήτησης

Πρόβλημα:

Δεδομένου ενός γράφου $G = (V, E)$, έχοντας κάθε ακμή ένα βάρος w_e, τον χωρίζουμε σε 2 μέρη (A, B). Αυτό που επιδιώκουμε είναι να γίνει μέγιστο το βάρος των ακμών που φεύγουν από το μέρος A και πάνε στο μέρος B και το αντίστροφο.

Γειτονιά:

Έστω ότι εχουμε διαχωρίσει τον G σε (A, B), αν έχουμε κόμβο u του οποίου το συνολικό βάρος των ακμών που φεύγουν από τον u στους κόμβους που είναι στην ίδια πλευρά του γράφου με αυτόν, υπερβαίνει το συνολικό βάρος των ακμών που φεύγουν από τον u προς τους κόμβους στη δεύτερη πλεύρα, τότε ο u πρέπει να μεταφερθεί στη δεύτερη πλεύρα. Λέμε τότε ότι έχουμε μια $singleflip$ γειτονιά και πάμε από το (A, B) στο (A', B'), αν μετακινήσουμε μόνο έναν κόμβο u.

Αλγόριθμος Τοπικής Αναζήτησης (χωρίς βάρη):

Σε αυτή την εκδοχή της τοπικής αναζήτησης [8], χρησιμοποιώντας αυθαίρετο σύνολο $S \subset V$,είτε κάνουμε προσθήκη ενός κόμβου στο S είτε κάνουμε αφαίρεση ενός κόμβου από το S, αν και μόνον αν βελτιώνει την χωρητικότητα που περικόπτεται. Οι κόμβοι παίρνουν

τιμή 1 αν και μόνον αν ανήκουν στο μέρος Α και -1 αν και μόνον αν ανήκουν στο μέρος Β.

Επίσης ορίζουμε 2 χωρίσματα (A, B) και $(', ')$ αν μπορούμε να μεταβούμε από το ένα στο άλλο με τη μεταφορά μόνο ενός κόμβου από το A στο B. Τώρα μπορούμε να παρουσιάσουμε τον αλγόριθμο τοπικής αναζήτησης.

Όσο $\exists u \in A$ με $\sum_{v \in A} w_{uv} > \sum_{v \in B} w_{uv}$
$u \leftarrow$ μετακινείται στον B
Όσο $\exists u \in B$ με $\sum_{v \in A} w_{uv} < \sum_{v \in B} w_{uv}$
$u \leftarrow$ μετακινείται στον A

Απο τις παραπάνω ανισότητες και τον ορισμό του τοπικού ελάχιστου προκύπτει:
$2 \sum_{\{u,v\} \subset A} w_{uv} < \sum_{u \in A, v \in B} w_{uv}$
$2 \sum_{\{u,v\} \subset B} w_{uv} < \sum_{u \in A, v \in B} w_{uv}$
Μετά από πρόσθεση και διαίρεση με το 2 αυτών των ανισώσεων έχουμε:
$\sum_{\{u,v\} \subset A} w_{uv} + \sum_{\{u,v\} \subset B} w_{uv} < \sum_{u \in A, v \in B} w_{uv}$ (*)

1. Όμως γνωρίζουμε ότι το άθροισμα των βαρών για όλες τις ακμές που πηγαίνουν από τον A στον B και το αντίθετο είναι: $\sum_{u \in A, v \in B} w_{uv} = w(A, B)$
2. Το αριστερό μέλος της ανίσωσης είναι όλα τα βάρη ακμών που δεν πηγαίνουν από το A στο B.
3. Ξέρουμε ότι W είναι το σύνολο όλων των ακμών Δηλαδή: $W \leftarrow w(A, B) + \sum_{\{u,v\} \subset A} w_{uv} + \sum_{\{u,v\} \subset B} w_{uv}$

Αν προσθέσουμε το $w(A, B)$ και στα 2 μέρη της ανίσωσης (*) έχουμε:
$W < 2 * w(A, B) \Rightarrow w(A, B) \geq 1/2W.$
$\Rightarrow w(A, B) \geq 1/2w(A^*, B^*).$

Άρα αυτό που δείξαμε είναι ότι στη βέλτιστη λύση με τη single flip γειτονιά, οι ακμές που πηγαίνουν από το A στο B είναι τουλάχιστον οι μισές των συνολικών ακμών του γράφου. Αυτό αποδεικνύει ότι το τοπικό βέλτιστο είναι $2-approximation$ του Προβλήματος Μέγιστης Τομής Γράφου.

Αλγόριθμος Τοπικής Αναζήτησης: (με βάρη):

Στην περίπτωση όμως όπου ο γράφος μας έχει βάρη, ο χρόνος που χρειάζεται για να τελείωσει είναι εκθετικός ως προς το V. Όπως αναφέρεται και στο `Approximation Algorithms` [7] πολλοί αλγόριθμοι μπορούν να τροποποιηθούν ώστε να τερματίζουν με ένα προσεγγιστικό τοπικό βέλτιστο έτσι ώστε: (1) ο χρόνος που χρειάζεται για να τερματιστεί να είναι πολυωνυμικός και (2) η ποιότητα της λύσης είναι παρόμοια με αυτή του αυθεντικού αλγορίθμου τοπικής αναζήτησης. Θα παρουσιάσουμε έναν αλγόριθμο τοπικής αναζήτησης για το Πρόβλημα Μέγιστης Τομής Γράφου με ε> 0 και n να είναι ο αριθμός των κόμβων στον G.

$d(v) \leftarrow$ ο βαθμός του κόμβου v
$v^* = argmax_{v \in V} w(d(v))$
$S \leftarrow \{v^*\}$
$d(S) \leftarrow$ αριθμός των ακμών που διασχίζουν το S
$w(d(S)) \leftarrow \sum_{e \in d(S)} w(e)$
Επανέλαβε
    Αν $(\exists v \in V$ ώστε $w(d(S+v)) > (1+$
    $\varepsilon/n) w(d(S)))$ τότε:
        $S \leftarrow S + v$
    Αλλιώς Αν $(\exists v \in S$ ώστε $w(d(S-v)) > (1+$
    $\varepsilon/n) w(d(S)))$ τότε:
        $S \leftarrow S - v$
    Αλλιώς
        Επέστρεψε S
    Τέλος Αν
Μέχρι να είναι Αληθές.

Ο παραπάνω αλγόριθμος τερματίζει όταν η βελτίωση είναι κοντά σε ένα συντελεστή $1+\varepsilon/n$ σε σχέση με την τωρινή λύση. Έτσι το τελικό αποτέλεσμα S είναι ένα προσεγγιστικό τοπικό βέλτιστο. Θα αποδείξουμε ότι:
ΠΡΟΤΑΣΗ 1: Αν S είναι το αποτέλεσμα του τροποποιημένου αλγορίθμου τοπικής αναζήτησης του Προβλήματος Μέγιστης Τομής Γράφου, τότε: $w(d(S)) \geq 1/(2(1+\varepsilon/4)) * w()$.

Απόδειξη: [7]

$S_0 \leftarrow$ αρχική γετονιά
$d(S) \leftarrow$ αριθμός των ακμών που διασχίζουν το S
$w(d(S)) \leftarrow \sum_{e \in d(S)} w(e)$
$a_v \leftarrow w(d(S) \cap d(v))$ (βάρη των ακμών του $v(d(v))$) (που διασχίζουν το S)(*)
$b_v \leftarrow w(d(v)) - a_v$(**)
1. Έστω ότι $v \in V \setminus S$ και $a_v < b_v$
    τότε αν μετακινήσουμε το v στο S θα αυξηθεί το $w(d(S))$ και το S δεν θα μπορεί να είναι το τοπικό βέλτιστο.
2. Αν $v \in S$ και $a_v < b_v$ τότε
    $w(d(S-v)) > w(d(S))$ το S δεν μπορεί να είναι τοπικό βέλτιστο.
Απο $(1), (2) \Rightarrow w(d(S) \cap d(v)) \geq w(d(v))/2$ (3)

4. Αφού κάθε ακμή ενώνεται με ακριβώς 2 κόμβους ισχύει ότι:

$$w(d(S)) = 1/2 \sum_{v \in V} w(d(S) \cap d(v)) \geq 1/2 \sum_{v \in V} w(d(v))/2 \geq 1/2 w(E) \geq 1/2 OPT \text{ (5)}$$

Ξέροντας ότι: $OPT \leq w(E)$.

Αν S είναι το τοπικό βέλτιστο τότε:

$b_v - a_v \leq \varepsilon/n * w(d(S))$.

Αλλιώς:

μια τοπική κίνηση που χρησιμοποιεί το v θα βελτίωνε το S περισσότερο από $1+\varepsilon/n$.

Απο τις σχέσεις $(3), (5)$ προκύπτει ότι:

$w(d(S)) = 1/2 \sum_{v \in V} b_v$

Αφού $a_v = 2 * a_v/2$ και προσθέτοντας και αφαιρώντας $(+b_v, -b_v)$:

$w(d(S)) = 1/2 \sum_{v \in V} ((a_v + b_v) - (b_v - a_v))/2$

Απο τις σχέσεις: (*) και (**) προκύπτει:

$$w(d(S)) \geq 1/4 \sum_{v \in V} (w(d(S)) - \varepsilon/n * w((d(S)))$$

Όμως $w() = 1/2 w(d(S)))$, άρα:

$$w(d(S)) \geq 1/2 w(E) - 1/4 \sum_{v \in V} \varepsilon/n * w((d(S)))$$

Άρα:

$w(d(S)) \geq 1/2 w(E) - (1/4) \varepsilon w((d(S)))$

Έτσι από την παραπάνω ανισότητα βγάζουμε το συμπέρασμα ότι:

$w((d(S)))(1+\varepsilon/4) \geq w(E)/2$ (6)

Χρησιμοποιώντας το αποτέλεσμα της ΠΡΟΤΑΣΗΣ 1 θα υπολογίσουμε τα βήματα τερματισμού του αλγορίθμου της τοπικής αναζήτησης.

ΠΡΟΤΑΣΗ 2: Ο τροποποιημένος αλγόριθμος τοπικής αναζήτησης τερματίζει σε: $O((1/\varepsilon) n log n)$ βήματα.

Απόδειξη:

Παρατηρούμε από την σχέση (6) ότι κάθε τοπική βελτίωση αλλάζει το $w(d(S))$ κατα $(1+\varepsilon/n)$. Έτσι αν k είναι ο αριθμός από αλληλοεπιδράσεις που τρέχει ο αλγόριθμος τότε: $(1+\varepsilon/n)^k w(S_0) < w(d(S))$ (7) όπου S_0 η αρχική λύση και S το τελικό αποτέλεσμα. Όμως ξέρουμε ότι ο συνολικός αρθμός ακμών $w(E)$ είναι μεγαλύτερος από τον αριθμό ακμών που διασχίζουν το S $(w(d(S)))$, δηλαδή $w(d(S)) < w(E)$ (8). Άρα από τις σχέσεις $(6), (7), (8) \Rightarrow (1+\varepsilon/n)^k w(E)/2n < w(E)$.

Από τα παραπάνω προκύπτει ότι ο αλγόριθμος τοπικής αναζήτησης τερματίζει σε χρόνο: $k = ((1/\varepsilon) n log n)$.

Υπάρχει περίπτωση ο αλγόριθμος τοπικής αναζήτησης να πάει καλύτερα από 1/2?

Έστω ότι έχουμε έναν διμερή γράφο $K_{2n,2n}$ με $2n$ κόμβους σε κάθε μεριά. Αν L, R είναι μέρη σε σύνολο S όπου $|S \cap L| = n = |S \cap R|$ είναι τοπικό βέλτιστο με $d(S) = |E|/2$. Η βέλτιστη λύση είναι $||$.

Έτσι δείχνουμε ότι το τοπικό βέλτιστο δεν είναι καλύτερο από $1/2 - approximation$.

3.ΕΥΡΕΣΗ ΑΠΟΔΟΤΙΚΩΝ ΓΕΙΤΟΝΙΩΝ

3.1 Τοπική Αναζήτηση για Μεγιστοποιήση Submodular Συναρτήσεων

Σε αυτή την ενότητα θα προσπαθήσουμε να μελετήσουμε την τοπική αναζήτηση για μεγιστοποίηση μη-αρνητικών submodular συναρτήσεων [7]. Έστω $f : 2V \rightarrow R^+$ είναι μία μη-αρνητικήsubmodular συνάρτηση στο V. Μία f λέγεται $submodular$ αν: $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ για όλα τα $A, B \subset V$. Αντίστοιχα η f είναι $submodular$ αν $f(A + v)-f(A) \geq f(B + v)-f(B)$ για όλα τα $A \subset B$ και $v \in B$. Λέμε ότι η f είναι μονότονη αν $f(A) \leq f(B)$ για όλα τα $A \subset B$. Η f είναι συμμετρική αν $f(A) = f(V \setminus A)$ για όλα τα $A \in V$. Παρουσιάζουμε δύο παραδείγματα τέτοιων προβλημάτων:

ΠΑΡΑΔΕΙΓΜΑ 1 [7]: Έστω $S_1, S_2, ..., S_n$ είναι υποσύνολα ενός συνόλου U και επίσης $A \subset V$. Έστω $V = \{1, 2, ..., n\}$, ορίζουμε μία $f : 2V \rightarrow R^+$ όπου $f(A) = |\bigcup_{i \in A} S_i|$. Η f είναι μονότονη $submodular$. Μπορούμε να κάνουμε αντιστοίχιση των βαρών στα στοιχεία του U μέσω συνάρτησης $w : U \rightarrow R^+$. Η συνάρτηση που ορίζεται ως: $f(A) = w(U_{i \in A} S_i)$ είναι επίσης μονότονη $submodular$.

ΠΑΡΑΔΕΙΓΜΑ 2[7]: Οι συναρτήσεις περικοπής σε γράφους. Έστω $G = (V, E)$ ένας μη κατευθυνόμενος γράφος με μη-αρνητικά βάρη ακμών $w : E \rightarrow R^+$ και $d_G(S)$ ο αριθμός ακμών του G που διασχίζουν το S, ενώ $d^+_G(S)$ είναι ο αριθμός των ακμών που φεύγουν από το S. Η συνάρτηση περικοπής $f : 2V \rightarrow R^+$ ορισμένη ως: $f(S) = \sum_{e \in d_G(S)} w(e)$ είναι μια συμμετρική submodular συνάρτηση, δεν είναι μονότονη παρα μόνο αν ο γράφος είναι τετριμμένος($trivial$). Αν ο G είναι κατευθυνόμενος τότε ορίζουμε την $f(S) = \sum_{e \in d_{G^+}} w(e)$ τότε η f είναι submodular αλλά όχι αναγκαία συμμετρική.

Πρόβλημα:

Το πρόβλημα μας γενικεύει το πρόβλημα της μέγιστης τομής κατευ-θυνόμενου και μη κατευθυ-νόμενου γράφου. Στο πρόβλημα μεγιστοποίησης submodular συναρτήσεων έχουμε μία μη αρνητική $submodular$ συνάρτηση f που είναι ορισμένη σε ένα σύνολο V. Αυτό που πρέπει κάνουμε είναι να μεγιστοποιήσουμε τη συνάρτηση f σε υποσύνολο $S \subset V$. Δηλαδή πρέπει: $max_{S \subset V} f(S)$.

Αν η f είναι μονότονη τότε το πρόβλημα είναι τετριμμένο αφού η V είναι η βέλτιστη λύση. Το πρόβλημα γίνεται NP-hard όταν η f δεν είναι απαραίτητα μονότονη.

Γειτονιά:

Αρχικά ορίζουμε ένα σύνολο S που είναι κενό. Αν υπάρχει κόμβος $v \in (V \cup S)$ ώστε η συνάρτηση f με την προσθήκη αυτού του κόμβου να είναι καλύτερη τότε $S \leftarrow S + v$. Ενώ αν υπάρχει $v \in S$ ώστε με την αφαίρεση του να γίνεται καλύτερη η f, τότε $S \leftarrow S-v$. Αλλιώς θα έχουμε βρεί το τοπικό βέλτιστο. Επομένως, οι γειτονιές αλλάζουν με την προσθήκη η αφαίρεση κόμβου v ανάλογα με το αν κάνει καλύτερη την συνάρτηση f.

Αλγόριθμος Τοπικής Αναζήτησης: Θεωρούμε έναν αλγόριθμο τοπικής αναζήτησης για το πρόβλημα μεγίστοποίησης submodular συναρτήσεων που δίνει 1/3 − approximation και 1/2 − approximation, όταν η f είναι συμμετρική. Ο αλγόριθμος είναι ο παρακάτω:

$S \leftarrow \emptyset$
Επανέλαβε:
Αν ($\exists v \in (V \setminus S)$ έτσι ώστε $f(S+v) \geq f(S)$)
$S \leftarrow S + v$
αλλιώς αν ($\exists v \in S$ έτσι ώστε $f(S-v) \geq f(S)$)
$S \leftarrow S - v$
αλλιώς
Το S είναι το τοπικό βέλτιστο
Επέστρεψε το καλύτερο μεταξύ των S και $V \setminus S$
Τέλος Αν
Μέχρι να είναι Αληθής.

Θα ξεκινήσουμε την ανάλυση του αλγορίθμου με ένα βασικό λήμμα της submodularity:

Λήμμα 1: Έστω $f : 2V \rightarrow R_+$ submodular συνάρτηση στο V. Έστω $A \subset B \subset V$. Τότε:

1. Αν $f(B) \geq f(A)$ τότε υπάρχει ένα στοιχείο $v \in B \setminus A$ έτσι ώστε $f(A+v) - f(A) \geq 0$. Πιο γενικά υπάρχει στοιχείο $v \in B \setminus A$ έτσι ώστε $f(A+v) - f(A) \geq (1/|B \setminus A|)(f(B) - f(A)$.
2. Αν $f(A) \geq f(B)$ τότε υπάρχει ένα στοιχείο $v \in B \setminus A$ έτσι ώστε $f(B-v) - f(B) \geq 0$. Πιο γενικά υπάρχει στοιχείο $v \in B \setminus A$ έτσι ώστε $f(B-v) - f(B) \geq (1/|B \setminus A|)((f(A) - f(B))$.

Απο αύτα λοιπόν προκύπτει η παρακάτω συνέπεια.

Συνέπεια 1: Έστω S είναι ένα τοπικό βέλτιστο για έναν αλγόριθμο τοπικής αναζήτησης και έστω S^* η βέλτιστη λύση.

- Τότε από το 1. προκύπτει ότι: $f(S) \geq f(S \cap S^*)$ (1), αφού $\exists v \in S \setminus (S \cap S^*)$ ώστε
$$f((S \cap S^*) + v) - f((S \cap S^*)) > 0.$$
- Απο το 2. ισχύει ότι: $f(S) \geq f(S \cup S^*)$ (2), διότι $\exists v \in (S \cup S^*) \setminus S$ ώστε
$$f((S \cup S^*) - v) - f((S \cup S^*)) > 0.$$

Θεώρημα: Θα αποδείξουμε ότι ο αλγόριθμος τοπικής αναζήτησης είναι 1/3 − approximation και γίνεται 1/2 − approximation εάν η f είναι συμμετρική.

Απόδειξη: ΄Εστω S το τοπικό βέλτιστο και το S^* το ολικό βέλτιστο για μια συγκεκριμένη περίπτωση. Απο την προηγούμενη συνέπεια έχουμε $f(S) \geq f(S \cap S^*)$ και $f(S) \geq f(S \cup S^*)$. Το αποτέλεσμα του αλγορίθμου είναι το καλύτερο μεταξύ των S και $V \setminus S$. Από την submodularity έχουμε: $f(V \setminus S) + f(S \cup S^*) \geq f((V \setminus S) \cap (S \cup S^*) + f((V \setminus S) \cup (S \cup S^*)$

Θα εφαρμόσουμε τις πράξεις συνόλων και έτσι προκύπτει ότι:

$f((V \setminus S) \cap (S \cup S^*) + f((V \setminus S) \cup (S \cup S^*) =$
$f(((V \setminus S) \cap S) \cup ((V \setminus S) \cap S^*)) + f(((V \setminus S) \cup S) \cup S^*)$

Όμως βλέπουμε ότι: $f((V \setminus S) \cap (S \cup S^*) + f((V \setminus S) \cup (S \cup S^*) = f(S^* \setminus S)$ και

$f(((V \setminus S) \cup S) \cup S^*) = f(V)$.
Άρα: $f(V \setminus S) + f(S \cup S^*) \geq f(S^* \setminus S) + f(V) \Rightarrow f(V \setminus S) + f(S \cup S^*) \geq f(S^* \setminus S)$ (3).

Χρησιμοποιώντας τις σχέσεις (1), (2), (3) έχουμε:

$$\begin{aligned} 2f(S) + f(V \setminus S) &= f(S) + f(S) + f(V \setminus S) \text{ (από σχέσεις (1), (2), (3))} \\ &\Rightarrow f(S) + f(S) + f(V \setminus S) \geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = OPT \text{ (4).} \end{aligned}$$

Ετσι $2f(S) + f(V \setminus S) \geq OPT$ άρα και $max\{f(S), f(V \setminus S)\} \geq OPT/3$.

Αν η f είναι συμμετρική χρησιμοποιούμε το Λήμμα 1 και ισχυριζόμαστε ότι $f(S) \geq f(S \cap S^*)$ όπως πριν, αλλά και επίσης $f(S) \geq f(S \cup S^*)$ όπου το A είναι μια συντομογραφία για το $V \setminus A$, εφόσον η f είναι συμμετρική $f(S \cup \bar{S^*}) = f(V \setminus (S \cup \bar{S^*})) = f(\bar{S} \cap S^*) = f(S^* \setminus S)$ (5). Έτσι από όλες τις παραπάνω σχέσεις έχουμε:

$$\begin{aligned} 2f(S) &\geq f(S \cap S^*) + f(S \cup \bar{S^*}) \\ &\geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = OPT. \end{aligned}$$

Άρα έχουμε: $f(S) \geq OPT/2$.

Ο τρέχον χρόνος του αλγορίθμου της τοπικής αναζήτησης μπορεί να μην είναι πολυωνυμικός, αλλά μπορούμε να κάνουμε μια τροποποίηση του αλγορίθμου έτσι ώστε να γίνει ισχυρά πολυωνυμικός με $(1/3\text{–}o(1))\text{–}approximation$ και $(1/2\text{–}o(1))\text{–}approximation$ για συμμετρικό.

3.2 Σχέσεις μεταξύ γειτονιών

Σε αυτό το σημείο πρέπει να δούμε τη σημαίνει να έχουμε μια καλώς ορισμένη γειτονιά, όπως αναφέρεται στο βιβλίο `Algorithm Design` [8]. Στα μέχρι στιγμής προβλήματα έχουμε επικεντρώθει στην εύρεση γειτονικής λύσης κατα τη διάρκεια κάθε βήματος και όχι στον ορισμό μιας καλής γειτονιάς εξ αρχής. Μια καλή γειτονιά πρέπει λοιπόν να έχει τα εξής χαρακτηριστικά:

- Η γειτονιά πρέπει να είναι αρκετά μεγάλη έτσι ώστε να μην κολλάμε σε ένα κακό τοπικό βέλτιστο.
- Και ταυτόχρονα δεν πρέπει να είναι τόσο μεγάλη, ώστε αποτελεσματικά να μπορούμε να ψάχνουμε ένα σύνολο από γείτονες για πιθανές τοπικές κινήσεις.

Στην ενότητα 1.2.3 δείξαμε για το πρόβλημα της μέγιστης τομής γράφων έναν 2-approximation αλγόριθμο για το τοπικό βέλτιστο. Ας φανταστούμε τώρα ότι έχουμε μεγαλύτερες γειτονιές από τις single flip. Λέμε οτι 2 μέρη (A, B) και (A', B') είναι k-flip γείτονες, για $k \geq 1$, αν το (A', B') προκύπτει από το (A, B) μετακινώντας το πολύ k κόμβους από το ένα μέρος στο άλλο. Έτσι αν υποθέσουμε οτι ένα (A, B) είναι τοπικό βέλτιστο σε μια k-flip γειτονιά, τότε είναι και σε μια $k^{'} - flip$ γειτονιά, για $\forall k' \leq k$.

Οι Kernighan and Lin (1970) [1], πρότειναν έναν άλλο τρόπο παραγωγής γειτονικών λύσεων,

υπολογιστικά πιο αποτελεσματικό, ονομάζοντας τον K-L heuristic. Οι γείτονες ενός (A, B) μέρους του γράφου σύμφωνα με τον παρακάτω αλγόριθμο με $n-$στάδια:

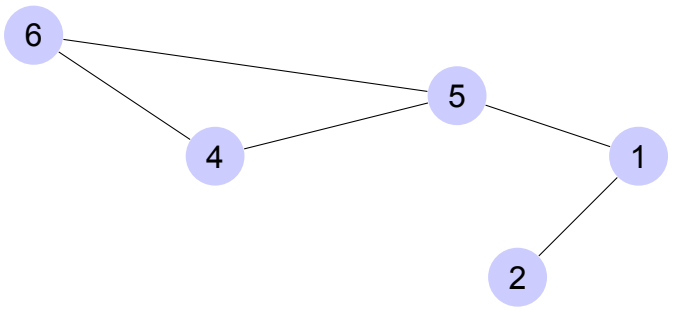
$G(V, E) \leftarrow$ αρχικός γράφος
$\leftarrow$ πρώτο χώρισμα του G
$B \leftarrow$ δεύτερο χώρισμα του G
$A \cap B \leftarrow \emptyset$
$D \leftarrow$ σύνολο τιμών
$\forall a \in A \leftarrow$ έχει μια τιμή από το D
$\forall b \in B \leftarrow$ έχει μια τιμή από το D
$G_V \leftarrow$ κενή λίστα
$_V \leftarrow$ κενή λίστα
$_V \leftarrow$ κενή λίστα
$c(a, b) \leftarrow$ κόστος πιθανής ακμής στα a, b
$g^{'} \leftarrow 0$
Επανέλαβε
    Για ($n = 1$ μέχρι $|V/2|$)
        Βρές $a \in A$
        Βρές $b \in B$
        $g \leftarrow D[a] + D[b] - 2 * c(a, b)$
        Αν ($g > g^{'}$)
            $g^{'} \leftarrow g$

            -Πρώτα αφαιρούμε τους κόμβους a, b
            από τα χωρίσματα A, B αντίστοιχα
            $A \leftarrow A \setminus a$ (αφαιρούμε τον κόμβο a από το A)
            $B \leftarrow B \setminus b$ (αφαιρούμε τον κόμβο b από το B)

            -Προσθέτουμε στις λίστες G_V, A_V, B_V αντίστοιχα
            τα g, a, b που βρήκαμε
            $G_V \leftarrow G_V + g$ (προσθέτουμε στην λίστα)
                (G_V το g που βρήκαμε)
            $A_V \leftarrow A_V + a$ (προσθέτουμε στην λίστα)
                ($_V$ το a που βρήκαμε)

            $B_V \leftarrow B_V + b$ (προσθέτουμε στην λίστα)
                (B_V το b που βρήκαμε)

            $\forall a \in A \setminus a \leftarrow$ καινούρια τιμή από το D
            $\forall b \in B \setminus b \leftarrow$ καινούρια τιμή από το D
        Τέλος Αν
    Τέλος Για



Σχήμα 3.1: Single-step κινήσεις

Αν έχουμε το χώρισμα $A(4,6)$ και το χώρισμα $B(1,2,5)$, σε μία single step κίνηση θα μεταβούμε σε μία νέα γειτονιά S', όπου το 5 θα έχει παει στο A χώρισμα, γιατί οι ακμές που από το 5 προς τους κόμβους του A, είναι περισσότερες απ'ότι προς τους κόμβους του B.

Βρές k ώστε
  Το $G_V[1], G_V[2], ..., G_V[k]$ να πάρει τη μέγιστη
  δυνατή τιμή δηλαδή:
  $G_{max} \leftarrow G_V[1], G_V[2], ..., G_V[k]$
Αν ($G_{max} > 0$) τότε
  Θα κάνουμε αλλαγή των k πρώτων στοιχείων
  της λίστας A με αυτών της λίστας B, δηλαδή:
  $A_V[1], A_V[2], ..., A_V[k] \leftarrow B_V[1], B_V[2], ..., B_V[k]$
  $B_V[1], B_V[2], ..., B_V[k] \leftarrow A_V[1], A_V[2], ..., A_V[k]$
Μέχρι ($G_{max} \leq 0$)
Επέστρεψε $G(V,E)$

Βλέπουμε οτι η K-L heuristic δοκιμάζει μια αλληλουχία από $flips$, ακόμα και αν φαίνεται ότι κάνει χειρότερα τα πράγματα , με την ελπίδα κάποιο χώρισμα (A_i, B_i) που προκύψει, να είναι καλύτερο από το (A,B). Πρέπει να σημειωθεί ότι εκτελεί μόνο $n - flips$ συνολικά και καθένα από αυτά κάνει $O(n)$ χρόνο για να εκτελεστεί. Για μεγάλες τιμές του k η K-L heuristic μοιάζει πιο αποδοτική από το k-flip. Επιπλέον η K-L heuristic έχει αποδειχθεί πολύ ισχυρή στην πράξη.

3.3 Κατηγοριοποίηση (Classification) μέσω της Τοπικής Αναζήτησης

Ωστόσο όσο πιο περίπλοκο το πρόβλημα, τόσο μεγαλύτερη και η ανάγκη εύρεσης πιο περίπλοκων γειτονιών, ώστε να πάρουμε καλά αποτελέσματα από την μέθοδο της τοπικής αναζήτησης.

Στο πρόβλημα που θα παραθέσουμε παρακάτω θα διαπιστώσουμε ότι μία state-flipping γειτονιά [8] θα καταλήξει σε ένα κακό τοπικό βέλτιστο. Για καλύτερη λειτουργικότητα θα κάνουμε χρήση μιας εκθετικά μεγάλης γειτονιάς. Το πρόβλημα όμως είναι ότι δεν έχουμε πια τη δυνατότητα να ψάχνουμε μία-μία όλες τις γειτονιές για την εύρεση καλύτερης λύσης. Αντ' αυτού θα χρησιμοποιήσουμε έναν πιο έξυπνο αλγόριθμο για να δούμε αν υπάρχει μια καλύτερη λύση.

Έστω ότι έχουμε μια εικόνα, το πρόβλημα μας είναι να ταξινομήσουμε κάθε $pixel$ αν ανήκει στο προσκήνιο ή στο παρασκήνιο της εικόνας αυτής. Κάθε $pixel\ i$ έχει μια πιθανότητα a_i να βρίσκεται στο προσκήνιο μιας εικόνας, και μια πιθανότητα b_i να βρίσκεται στο παρασκήνιο. Αν ισχύει ότι: $a_i > b_i$ τότε το $pixel\ i$ θα βρίσκεται στο προσκήνιο της εικόνας, αλλιώς θα ανήκει στο παρασκήνιο. Θα υποθέσουμε επίσης ότι αυτές οι πιθανότητες a_i, b_i είναι αυθαίρετοι μη αρνητικοί αριθμοί. Επίσης αν όλοι οι γείτονες ενός $pixel\ i$ ανήκουν στο παρασκήνιο, τότε και το i πρέπει να ανήκει και αυτό στο παρασκήνιο. Τέλος για κάθε ζευγάρι γειτονικών $pixel(i, j)$ έχουμε ποινή διαχωρισμού (separation penalty) $p_{ij} > 0$. Έτσι το πρόβλημα μπορεί να διατυπωθεί με την παρακάτω μαθηματική εξίσωση: $q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E, |A \cap \{i,j\}|=1} p_{ij}$

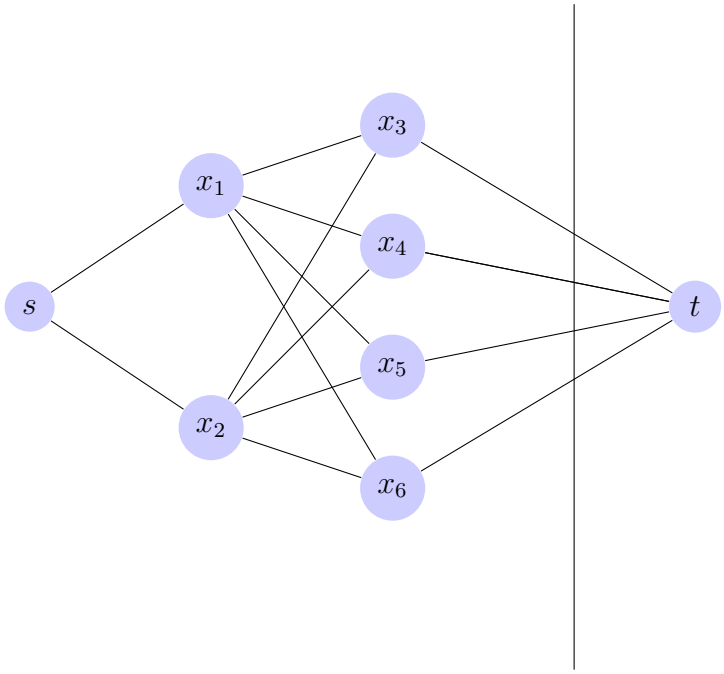
Έτσι το πρόβλημα ανάγεται σε πρόβλημα βέλτιστης απόδοσης ετικετών, βρίσκοντας μια διαχώριση (partition) $(,)$ του γράφου που να μεγιστοποιεί το $q(A, B)$.

Ας φανταστούμε τώρα ότι εκτός από την ταξινόμηση για το αν ανήκει στο προσκήνιο ή στο παρασκήνιο έχουμε και άλλου είδους ταξινόμηση, παραδείγματος χάρη την απόσταση των $pixel$ από την κάμερα. Έτσι δημιουργείται ένα πιο περίπλοκο πρόβλημα με παραπάνω από δύο τιτλοφορήσεις που δεν μπορεί να αντιμετωπιστεί με μια single flip γειτονιά. Ένα τέτοιο πρόβλημα παρουσιάζεται παρακάτω.

Πρόβλημα:

Δεδομένου ενός γράφου $G = (V, E)$, όπου V είναι τα $pixel$ μιας εικόνας. Στόχος είναι να κάνουμε ταξινόμηση για κάθε κόμβο στο V αν ανήκει στο προσκήνιο ή στο παρασκήνιο της εικόνας. Όμως αυτή τη φορά εκτός από το περιθώριο λάθους της ποινής διαχωρισμού (separation penalty), έχουμε και την ποινή εκχώρησης (assignment penalty), δηλαδή να κάνουμε λάθος εναπόθεση μιας ετικέτας σε έναν κόμβο.

Ας φανταστούμε λοιπόν ότι έχουμε το αντίστοιχο πρόβλημα με παραπάνω από 2 κλάσεις για να τους δώσουμε ετικέτες(labeling) σε γράφο $G = (V, E)$ και ένα σύνολο L από k διαφορετικές ετικέτες. Στόχος μας είναι να κανουμε τιτλοφόρηση κάθε κόμβου στο V με κάποιο είδος τιτλοφόρησης του L έτσι ώστε να γίνει ελαχιστοποίηση κάποιας ποινής. Έτσι έχουμε δύο ειδών ποινές (penalty) που θα παίξουν ρόλο στο πρόβλημα μας. Για κάθε ακμή $i, j \in E$ έχουμε ποινή διαχωρισμού (separation penalty): $p_{i,j} \geq 0$ για τιτλοφόρηση 2 διαφορετικών κομβων i, j με διαφορετικές ταμπέλες. Για κάθε κόμβο $i \in V$ και κάθε τιτλοφόρηση $a \in L$ έχουμε ποινή εκχώρησης (assignment penalty): $c_i(a) \geq 0$ για την ανάθεση της τιτλοφόρησης a στον κόμβο i. Το *Πρόβλημα Τιτλοφόρησης(Labeling Problem)* είναι να βρούμε μια τιτλοφόρηση $f : V \to L$ που κάνει ελάχιστοποίηση στο συνολικό περιθώριο λάθους [8]. Έτσι $\Phi(f) = \sum_{i \in V} c_i f(i) + \sum_{(i,j) \in E: f(i) \neq f(j)} p_{ij}$



Σχήμα 3.2: Ένα κακό Τοπικό Βέλτιστο.
Γίνεται αποκοπή των ακμών του t, ενώ αν γινόταν αποκοπή των ακμών του s θα οδηγούμασταν σε καλύτερο Τοπικό Βέλτιστο.

Γειτονιά:

Two Labels κάθε φορά: Η βασική ιδέα είναι η τοπική αναζήτηση να χρησιμοποιήσει τον πολυωνυμικό αλγόριθμο για να βρεί καλύτερα βήματα. Παρακάτω θα δώσουμε τον ορισμό μιας γειτονιάς που όμως δεν λειτουργεί σε όλες τις περιπτώσεις.

Για μια τιτλοφόρηση f διαλέγουμε 2 ταμπέλες $a, b \in L$ και προσέχουμε τους κόμβους που έχουν μόνο ταμπέλες a, b για να βάλουμε ετικέτες στην f. Σε ένα βήμα επιτρέπουμε σε κάθε υποσύνολο αυτών των κόμβων να κάνει flip από το a στο b ή το αντίθετο. Πιο γενικά δύο τιτλοφορήσεις f, f' είναι γειτονικές αν υπάρχουν 2 ταμπέλες $a, b \in L$ ώστε για όλες τις υπόλοιπες ταμπέλες $c \notin \{a, b\}$ και όλους τους υπόλοιπους κόμβους $i \in V$ να ισχύει ότι: $f(i) = c$ αν και μόνο αν $f'(i) = c$.

Μία τέτοια γειτονιά είναι πολύ καλύτερη από μία single-flip,υπάρχουν περιπτώσεις όπως αυτή στο Σχήμα 2.2 όπου τα αποτελέσματα μας δεν είναι καλά διότι γίνεται αποκοπή των λάθος ακμών. Έτσι επιζητούμε μια ακόμα καλύτερη γειτονιά.

Μια Καλή Local Search Neighborhood [8]: Μία καλή γειτονιά θα είναι αυτή που έχουμε δυνατότητα επανατιτλοφόρησης κόμβων με διαφορετικές ταμπέλες σε ένα μόνο βήμα. Λόγω αυτού θα πρέπει να βρούμε μια γειτονιά μεγάλη αρκετά ώστε να έχει αυτή την ιδιότητα και συνάμα να μπορούμε να βρούμε ένα καλύτερο βήμα στην τοπική αναζήτηση σε πολυωνυμικό χρόνο.

Διαλέγουμε λοιπόν μια ετικέτα $\in L$ και επικεντρωνόμαστε στους κόμβους που δεν έχουν ετικέτα για να την τιτλοφόρηση της f. Σε κάθε βήμα της τοπικής αναζήτησης θα μπορεί

ένα υποσύνολο κόμβων να αλλάξει την ετικέτα του σε .

Γειτονικές Τιτλοφορήσεις: Δύο τιτλοφορήσεις f και f' , θα λέμε ότι είναι γειτονικές αν υπάρχει μια $\in L$ ώστε όλοι οι κόμβοι $i \in V$ είναι είτε $f'(i) = f(i)$ είτε $f'(i) =$.

Θα δείξουμε ότι για κάθε τιτλοφόρηση f μπορούμε να βρούμε τον καλύτερο γείτονα μέσω k υπολογισμών ελάχιστης περικοπής.

Έστω ότι έχουμε γράφο $G = (V, E)$, δημιουργούμε νέο γράφο, που αρχικά είναι κενός, τον $G' = (V', E')$ που θα του προσθέτουμε κόμβους και ακμές. Σε αυτό το σημείο θα παρουσιάσουμε μια ταμπέλα-πηγή (source s) καθώς και μια βοηθητική-ταμπέλα (sink t). Η ταμπέλα s αντιπροσωπεύει μια ταμπέλα a, ενώ η t αντιπροσωπεύει την επιλογή κάθε κόμβου να κρατήσει την αρχική του ταμπέλα.

Η βασική ιδέα είναι να βρούμε μια ελάχιστη τομή σε έναν νέο γράφο G' σε ένα $s - side$ και ένα $t - side$, ώστε να γίνει επανατιτλοφόρηση όλων των κόμβων στο $s - side$ για να έχουν ταμπέλα a και όλοι οι κόμβοι του $t - side$ να έχουν τις αρχικές τους ταμπέλες.

Για κάθε κόμβο του G θα έχουμε έναν αντίστοιχο κόμβο σε ένα νέο σύνολο V' και θα προσθέτουμε ακμές (i, t) και (s, i) στο E'. Η ακμή (i, t) θα έχει χωρητικότητα $c_i()$, εάν καθώς γίνεται η περικοπή της ο κόμβος i τοποθετηθεί στο source side και άρα το i παίρνει την ετικέτα a, έαν όμως ο i πάει στο sink side τότε θα διατηρήσει την αρχική του τιτλοφόρηση $f(i) \geq a$. Απο την άλλη η (i, s) έχει χωρητικότητα $c_i(f(i))$ αν $f(i) \neq$ ή έναν μεγάλο αριθμό M αν $f(i) =$. Η μεγάλη χωρητικότητα του αριθμού M αποτρέπει τους κόμβους i που έχουν $f(i) = a$ να τοποθετηθούν στο $t - side$.

Τέλος, αν έχουμε i και j που είναι στην $t - side$ του γράφου, τότε η ακμή που τους χωρίζει δεν θα περικοπεί, ο i και ο j χωρίζονται αν $f(i) \neq f(j)$. Άρα για μια ακμή (i, j), αν $f(i) = f(j)$ ή αν ένα εκ των i, j έχουν ετικέτα a τότε προσθέτουμε ακμή (i, j) στο E' με χωρητικότητα p_{ij}. Για τις ακμές $e = (i, j)$ όπου $f(i) \neq f(j)$ και ούτε έχουν ετικέτα a, πρέπει να κάνουμε κάτι ώστε οι i, j να παραμείνουν χωριστοί ακόμα και αν βρίσκονται και οι δύο στο $t - side$. Για κάθε τέτοια ακμή προσθέτουμε έναν κόμβο e στο V' και προσθέτουμε τις ακμές $(i, e), (e, j), (e, s)$ όλες με χωρητικότητα p_{ij}.

Αλγόριθμος Τοπικής Αναζήτησης:

$f^* \leftarrow$ βέλτιστη τιτλοφόρηση
Για μια τιτλοφόρηση ορίζουμε:
$V_a^* = \{i : f^*(i) = a\} \leftarrow$ σύνολο κόμβων που είναι τιτλοφορημένοι με a στο f^*
$f \leftarrow$ τοπικό βέλτιστο
Δημιουργούμε μια γειτονιά f_a με τον παρακάτω τρόπο:
Ξεκινάμε από τη γειτονιά f
Για $\forall i \in V_a^*$ επανέλαβε
επανατιτλοφόρηση i στον a
$f \leftarrow f \cup \{i\}$
Τέλος Επανάληψης.
Το f είναι τοπικό βέλτιστο και άρα η γειτονιά f_a δεν έχει μικρότερη ποινή απο:
$\Phi(f_a) \geq \Phi(f)$.
Η πιθανή αλλαγή στις ποινές επανατοποθέτησης προέρχεται από τους κόμβους στο V_a^*:
Για κάθε $i \in V_a^*$ η αλλαγή είναι:
$c_i(f^*(i)) - c_i(f(i))$
Οι ποινές διαχωρισμού μεταξύ των 2 ταμπελών διαφέρουν μόνο στις ακμές (i, j) που έχουν τουλάχιστον ένα άκρο στο V_a^*.
Για την τιτλοφόρηση f και τον γείτονα f_a έχουμε:
$\Phi(f_a)$-$\Phi(f) \leq \sum_{i \in V_a^*}[c_i(f^*(i)) - c_i(f(i))]$
$+ \sum_{(i,jleavingVa^*)} pij$
$- \sum_{(i,j)inorleavingVa^*, f(i) \neq f(j)} p_{ij}$
Αυτό το αποτέλεσμα προκύπτει αφού:
1. Η μόνη δυνατή αλλαγή στις ποινές εκχώρισης μπορεί να προέλθει από τους κόμβους στο V_a^*, γιατί για $\forall i \in V_a^*$ η διαφορά είναι ακριβώς: $c_i(f^*(i)) - c_i(f(i))$, άρα οι ποινές διαχωρισμού είναι ακριβώς: $\sum_{i \in V_a^*}[c_i(f^*(i)) - c_i(f(i))]$
2. Η ποινή επανατοποθέτησης μιας ακμής (i, j) μπορεί να διαφέρει ανάμεσα σε 2 τιτλοφορήσεις μόνο αν η ακμή (i, j) έχει τουλάχιστον ένα τελείωμα στο V_a^*. Άρα για την f ισχύει ότι η ποινή επανατοποθέτησης είναι ακριβώς:
$\sum_{(i,j)inorleavingV_a^*, f(i) \neq f(j)} p_{ij}$
3. Ενώ αντίστοιχα για την f η ποινή επανατοποθέτησης είναι το πολύ:
$\sum_{(i,j)leavingV_a^*} p_{ij}$
Ξέρουμε ότι $\Phi(f) - \Phi(f) \geq 0$ για κάθε $\in L$. Αν προσθέσουμε τη παραπάνω ανισότητα για όλες τις τιτλοφορήσεις:
$0 \leq \sum_{\in L}(\Phi(f) - \Phi(f))$
$\leq \sum_{\in L}[\sum_{i \in V_a^*}(c_i(f^*(i)) - c_i(f(i))) + \sum_{(i,j)leavingV_a^*} p_{ij}$
$- \sum_{(i,j)inorleavingV_a^*, f(i) \neq f(j)} p_{ij}]$

Συγκεντρώνουμε όλους τους θετικούς όρους στο αριστερό μέρος και όλους τους αρνητικούς όρους στο δεξί.
Αριστερά: $\forall i : c_i(f^*(i)) \rightarrow$ ποινή επανατοποθέτησης του f^*.
- Έτσι έχουμε 2 φορές τον όρο p_{ij} για κάθε μια από τις ακμές που χωρίζονται από το f^*.(μια για κάθε μια από τις τιτλοφορήσεις $f^*(i)$ και $f^*(j)$)

Δεξιά: $\forall i : c_i(f(i)) \rightarrow$ ποινή επανατιτλοφόρησης του f.
- Έχουμε αυτή τη ποινή διαχώρισης το λιγότερο 1 φορά και πιθανώς 2 φορές αν υπάρχει διαχώριση και από τη f^*.

Συνολικά παίρνουμε το ακόλουθο αποτέλεσμα:
$2\Phi(f^*) \geq \sum_{\in L}[\sum_{i \in V_a^*} c_i(f^*(i)) + \sum_{(i,j) leaving V_a^*} p_{ij}]$
$\geq \sum_{\in L}[\sum_{i \in V_a^*} c_i(f(i)) + \sum_{(i,j) in or leaving V_a^*, f(i) \neq f(j)} p_{ij}] \geq \Phi(f)$
Αποδεικνύοντας έτσι ότι:
Για κάθε τοπικά βέλτιστη τιτλοφόρηση f ή ολικά βέλτιστη τιτλοφόρηση f^*, έχουμε $\Phi(f) \leq 2\Phi(f^*)$.

Με αυτόν τον τρόπο βρήκαμε έναν 2-approximation algorithm για την πιθανή ελάχιστη ποινή. Πρέπει να δούμε αν ο αλγόριθμος τερματίζει σε πολυωνυμικό χρόνο. Για τον λόγο αυτό θα έχουμε μια σταθερά $\varepsilon \geq 0$. Έτσι, δεδομένου μιας τιτλοφόρησης f και μιας γειτονικής τιτλοφόρησης f' αν: $\Phi(f') \leq (1-\varepsilon/3k)\Phi(f)$, λέμε ότι έχουμε σημαντική βελτίωση. Για να είμαστε σίγουροι οτι ο αλγόριθμος θα τρέξει σε πολυωνυμικό χρόνο θα δεχόμαστε μόνο τις σημαντικές βελτιώσεις και θα τερματίζουμε αν δεν βρούμε μια τέτοια.

4.ΤΟΠΙΚΗ ΑΝΑΖΗΤΗΣΗ ΓΙΑ ΠΡΟΒΛΗΜΑΤΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ

4.1 Το Πρόβλημα του Πλανόδιου Πωλητή (TSP)

Πρόβλημα:

Στο Πρόβλημα του Πλανόδιου Πωλητή(TSP) [8, 1, 12] , έχουμε ένα σύνολο από πόλεις $\{1, 2, 3, .., n\}$ και σαν δεδομένο έχουμε έναν nxn συμμετρικό πίνακα $C = (c_{ij})$ με τα κόστη για τη μεταφορά από την πόλη i στην πόλη j. Το κόστος για μεταφορά από την πόλη i στον εαυτό της ειναι ίσο με 0 και τα κόστη είναι μη αρνητικά. Το κόστος για μετάβαση από την $i \to j$ είναι ίσο με το κόστος μετάβασης $j \to i$ λόγω του ότι έχουμε συμμετρικό πίνακα C. Αν θεωρήσουμε ότι έχουμε έναν μη κατευθυνόμενο γράφο με όλες τις πόλεις και τα κόστη είναι οι ακμές που τις συνδέουν, τότε μια εφικτή λύση περιέχει έναν Χαμιλτόνιο κύκλο πάνω στον γράφο. Έτσι έχουμε μια διάσχιση των πόλεων , ώστε $k(1), k(2), ..., k(n)$, όπου η πόλη i έχει μια μοναδική εικόνα $k(i)$. Το κόστος αυτής της διάσχισης είναι: $c_{k(n)k(1)} + \sum_{i=1}^{n-1} c_{k(i)k(i+1)}$

Παρατηρούμε ότι κάθε διαδρομή έχει n ξεχωριστές παραστάσεις, αφού δεν μας νοιάζει ποια πόλη θα επισκεφτούμε πρώτη. Επίσης είναι NP-complete να αποφασίσουμε αν ένας μη κατευθυνόμενος γράφος $G = (V, E)$ περιέχει Χαμιλτόνιο κύκλο ή όχι. Γι'αυτόν τον λόγο θα δούμε την επιλογή γειτονιάς.

Γειτονιά:

Οι καλές γειτονιές για το Πρόβλημα του Πλανόδιου Πωλητή είναι η 2-opt και η k-opt [8, 9, 12] με $(k \geq 3)$. Δεδομένης μιας διαδρομής T, μια διαδρομή $T^{'}$ ανήκει σε μια k-opt γειτονιά της T, αν μπορεί να προέλθει από την T αφαιρώντας το πολύ $k^{'} \leq k$ ακμές από την T και προσθέτοντας $k^{'}$ νέες. Έτσι σε μια 2-opt γειτονιά μπορούμε να κάνουμε το πολύ 2 αλλαγές πόλεων τη φορά μέχρι την εύρεση μιας καλύτερης λύσης.

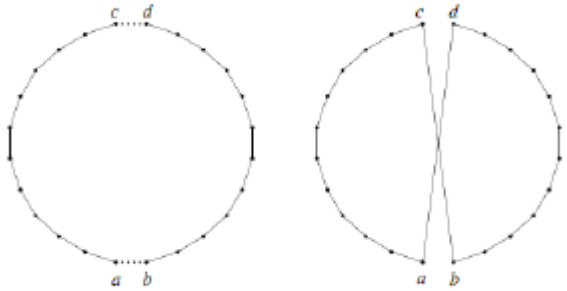
Το Πρόβλημα Του Πλανόδιου Πωλητή είναι PLS complete κάτω από κάθε k-opt κίνηση για μια σταθερά k. Έχει αποδειχθεί από τους Johnson, Papadimitriou and Yannakakis and Krentel πως για καθένα αρκετά μεγάλο k, υπάρχουν περιπτώσεις και αρχικές διαδρομές , έτσι ώστε κάθε τοπική αναζήτηση που χρησιμοποιεί k-opt και αρχίζει από τις συγκεκριμένες αρχικές διαδρομές, χρειάζεται εκθετικό αριθμό βημάτων για να καταλήξει σε μια τοπική βέλτιστη λύση του προβλήματος.

Αλγόριθμος Τοπικής Αναζήτησης:

Οι Chandra, Karloff, Tovey [1], έχουν αποδείξει αρκετά σημαντικά προσεγγιστικά αποτελέσματα για διαφορετικές εκδοχές του Προβλήματος του Πλανόδιου Πωλητή.

Για τυχαία προβλήματα του Πλανόδιου Πωλητή δείχνουμε ότι ο προσδοκώμενος αριθμός αλληλεπιδράσεων πριν φτάσουμε σε τοπικό βέλτιστο είναι πολυωνυμικός και ο αναμενόμενος προσεγγιστικός χειρότερος λόγος είναι φραγμένος από μια σταθερά.

Θα δείξουμε αρχικά σύμφωνα με τους Chandra, Karloff, Tovey, ότι για ένα μετρικό Πρόβλημα Του Πλανόδιου Πωλητή μια τοπική αναζήτηση που χρησιμοποιεί 2-opt γειτονιά επιτυγχάνει, κατα προσεγγιση, λόγο το πολύ $4\sqrt{n}$ για κάθε n. Έπισης δείχνουν ότι με την χρήση μεγαλύτερης γειτονιάς δεν υπάρχει θεμελιώδης και αισθητή βελτίωση, αφού η τοπική



Σχήμα 4.1: 2-opt.

αναζήτηση με k-opt γειτονιά έχει λόγο επίδοσης το λιγότερο: $1/4n^{1/2k}$ για άπειρα n.

$wt(e) \leftarrow$ το βάρος της ακμής e
$C_{opt} \leftarrow$ το κόστος της βέλτιστης διαδρομής
$T \leftarrow$ τοπικά βέλτιστη διαδρομή με κόστος C σε μια 2-opt γειτονιά E_k
$k \in \{1, 2, .., n\} \leftarrow$ σύνολο από μεγάλες ακμές του T.

Για την γειτονιά E_k ισχύει ότι: $E_k = \{e \in T \mid wt(e) \geq 2C_{opt}/\sqrt{k}\}$. Πρέπει να αποδείξουμε ότι $|E_k| < k$. Έστω ότι V είναι ένα σύνολο από κόμβους. Για κάθε υποσύνολο $V' \subset V$ έχουμε το $C_{opt}(V')$ ως το μήκος του μικρότερου μονοπατιού στο V'. Έστω ότι έχουμε το μονοπάτι T και $t_1, t_2, ..., t_r$, με $r = |E_k| \geq k$, είναι οι ουρές από το E_k στο T. Τότε υπάρχουν το λιγότερο $\sqrt{k}$ ουρές που βρίσκονται σε αποσταση το λιγότερο $C_{opt}/\sqrt{k}$ η μια από την άλλη.

Αν θεωρήσουμε το Πρόβλημα του Πλανόδιου Πωλητή στο σύνολο V' από ουρές, τότε η μικρότερη διαδρομή έχει μήκος $C_{opt}(V')$, που είναι μεγαλύτερο από $\sqrt{k} * C_{opt}/\sqrt{k} = C_{opt}$. Όμως έτσι ερχόμαστε σε αντιπαράθεση με το γεγονός ότι αφού οι αποστάσεις ικανοποιούν την τριγωνική ανισότητα, τότε για κάθε υποσύνολο $V' \subset V$ ισχύει ότι: $C_{opt}(V') \leq C_{opt}(V)$. Άρα αποδεικνύουμε ότι τελικά: $|E_k| < k$.

Αφού ισχύει ότι $|E_k| < k$, τότε το βάρος της k^{th} μεγαλύτερης ακμής στο T είναι το πολύ $2C_{opt}/\sqrt{k}$. Άρα έχουμε:

$C = \sum_{k=1}^{n} wt(k^{th}$ largest edge)
$\leq 2C_{opt} \sum_{k=1}^{n} 1/\sqrt{k}$
$\leq 2C_{opt} \int_{x=0}^{n} (1/\sqrt{x})dx$
$= 4\sqrt{n}C_{opt}$

Αποδείξαμε με αυτόν τον τρόπο ότι ο αλγόριθμος της τοπικής αναζήτησης με μια $2 - opt$ γειτονιά μπορεί κατα προσέγγιση να δώσει ένα $4\sqrt{n}$ approximation ratio για το Πρόβλημα του Πλανόδιου Πωλητή.

4.2 Boolean Προβλήματα Ικανοποίησης και SAT Προβλή-ματα

Πρόβλημα:

Το Boolean Πρόβλημα Ικανοποίησης [11] είναι το πρόβλημα να καθορίσεις αν υπάρχει λύση που να ικανοποιεί μια συγκεκριμένη Boolean φόρμουλα. Δηλαδή θέλουμε να καταλά-βουμε εαν οι μεταβλητές μιας δεδομένης Boolean φόρμουλας μπορούν να αντικατασταθούν με τιμές Αληθής και Ψευδής, με τέτοιον τρόπο ώστε η συνολική αποτίμηση του συστήματος να είναι Αληθής. Σε τέτοια περίπτωση η φόρμουλα αυτή λέγεται ικανοποιημένη(satisfiable). Σε αντίθετη περίπτωση αν δεν υπάρχουν τέτοιες αναθέσεις τιμών, τότε το σύστημα λέγεται μη ικανοποιμένη(unsatisfiable).

Παράδειγμα: Η λογική εξίσωση a AND NOT b είναι ικανοποιημένη, διότι μπορούμε να βρούμε τιμές ώστε $a = TRUE$ και $b = FALSE$ ώστε a AND NOT b $= TRUE$.

Το Boolean Πρόβλημα Ικανοποίησης είναι ένα πρόβλημα SAT και είναι πρόβλημα απόφασης για το αν ένα συγκεκριμένο σύστημα είναι ικανοποιημένο ή όχι. Έχει αποδειχθεί ότι είναι NP-complete πρόβλημα όπως και τα άλλα SAT προβλήματα.

Θα δώσουμε τους ορισμούς τι είναι ένα $3 - SAT$ και ένα $k - SAT$ πρόβλημα.

3-SAT πρόβλημα: Ένα πρόβλημα όπου αποτελείται μόνο από μεταβλητές και κάθε μία από αυτές έχει αυστηρά μόνο τρείς Boolean literals.
Παράδειγμα: $((a \vee b \vee c) \wedge ... \wedge (a \vee b \vee c))$.

k-SAT πρόβλημα: Ένα πρόβλημα όπου αποτελείται από μεταβλητές και κάθε μία από αυτές έχει αυστηρά k Boolean literals.
Παράδειγμα: $((a_1 \vee a_2 ... \vee a_k) \wedge ... \wedge (a_1 \vee a_2 ... \vee a_k))$.

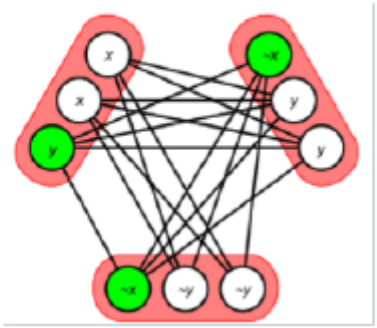
Ένας τρόπος για να λυθεί ένα SAT πρόβλημα είναι ένας στοχαστικός αλγόριθμος τοπικής αναζήτησης η WalkSAT.

Γειτονιά:

Έστω ότι έχουμε ένα σύστημα F και στόχος μας είναι να ελαχιστοποιήσουμε την αντικειμε-νική συνάρτηση(objective function) $E = E_F(s) =$ με τον αριθμό των μη ικανοποιημένων στοιχείων(unsatisfied clauses) στη φόρμουλα F κάτω από τις αναθέσεις s που είναι αληθείς. Αρχικά ο Selman πρότεινε έναν άπληστο (greedy GSAT) αλγόριθμο, όπου σε κάθε βήμα η μεταβλη-τή που θα γίνεται flip θα καθορίζεται από το πιο flip οδηγεί στη μεγαλύτερη μείωση της E. Η μέθοδος αυτή βελτιώθηκε αυξάνοντας τα άπληστα βήματα με ένα ρυθμιζόμενο κλάσμα p τυχαίων βημάτων, δημιουργώντας έτσι τον NoisyGSAT αλγόριθμο.

Στη συνέχεια ο Papadimitriou πρότεινε σε κάθε βήμα να κοιτάμε μόνο τις μεταβλητές που παραμένουν ανικανοποίητες. Η εφαρμογή αυτή οδηγεί στη δημιουργία του WalkSAT. Άρα η γειτονιά αποτελείται κάθε φορά από τις μεταβλητές όπου παραμένουν ανικανοποίητες.

Αρχικά θα μιλήσουμε για την Μεταβλητή Αναζήτηση Γειτονιάς(Variable neighborhood search - VNS) όπου στόχος της είναι να βρεί μια αλληλεπίδραση μεταξύ διαφοροποίησης και εντατικοποίησης. Η διαφοροποίηση αναφέρεται στο γεγονός της εξερεύνησης πολλών διαφορετικών περιοχών κάθε χώρου.



Σχήμα 4.2: 3-SAT
Ένα 3-SAT πρόβλημα γνωστό και ως πρόβλημα 3-κλίκας(3-clique problem). Οι πράσινοι κόμβοι σχηματίζουν μια 3-κλίκα και ικανοποιούν x =Αληθές , y =Ψευδές.

Αντίστοιχα η εντατικοποίηση είναι η δυνατότητα ύπαρξης υψηλής ποιότητας λύσεων σε αυτές τις περιοχές. Αρχικά στην Μεταβλητή Αναζήτηση Γειτονιάς διαλέγουμε ένα αρχικό σύνολο γειτονιών. Έστω N_k, $(k = 1, 2, ..., k_{max})$ το προκαθορισμένο σύνολο και $N_k(x)$ το σύνολο λύσεων της k^{th} γειτονιάς του x. Επίσης S_{start} είναι η αρχική λύση. Η Μεταβλητή Αναζήτηση Γειτονιάς ξεκινά κάνοντας παραγωγή μια τυχαίας λύσης S_{rand} από τη γειτονιά: $N_1(S_{rand} \in N_1(S_{start})$. Έστω ότι η S_{new} δηλώνει το τοπικό βέλτιστο όταν η τοπική αναζήτηση χρησιμοποιηθεί με δεδομένο εισαγωγής S_{rand}. Αν S_{new} είναι καλύτερο σε σχέση με το S_{rand}, η λύση ανανεώνεται και μια νέα τοπική αναζήτηση με τυχαία λύση από $N_1(S_{new})$ εκτελείται. Αν αποτύχει τότε η Μεταβλητή Αναζήτηση Γειτονιάς προχωράει στην επόμενη γειτονιά. Η αποτελεσματικότητα της Μεταβλητής Αναζήτησης Γειτονιάς είναι ανάλογη με την διάταξη που έχει γίνει στις γειτονιές.
Τώρα θα αναφερθούμε στον Walksat-based Algorithm Μεταβλητής Γειτονιάς [6, 5] που έχει 2 στάδια:

- Βήμα 1: Έστω P το σύνολο μεταβλητών του προβλήματος που πρέπει να επιλυθεί. Αρχικά έχουμε ενα σύνολο από γειτονιές που ικανοποιούν την παραπάνω ιδιότητα: $N_1(x) \subset N_2(x) \subset ... \subset N_{k_{max}}(x)$. Η αρχική γειτονιά με $k = 0$ αποτελείται από μια κίνηση βασισμένη στο single flip μίας μεταβλητής. Το flip σημαίνει ότι αλλάζω την κατάσταση μίας μεταβλητής στο αντίθετο από αυτό που είναι (π.χ από Αληθές σε Ψευδές και από Ψευδές σε Αληθές). Η πρώτη γειτονιά N_1 κατασκευάζεται από το P με συγχώνευση μεταβλητών. Πηγαίνουμε στις μεταβλητές με τυχαία σειρά, αν μια μεταβλητή l_i δεν έχει γίνει αντιστοίχιση ακόμα, τότε μια τυχαία μη αντιστοιχίσιμη μεταβλητή l_j επιλέγεται και μια νέα μεταβλητή l_k (κλάση), που αποτελείται από τις l_i, l_j, δημιουργείται. Το σύνολο N_1 περιέχει μια κίνηση βασισμένη στο flipping προκαθορισμένων κλάσεων που η κάθε μια περιέχει 2^1 μεταβλητές. Οι νεές κλάσεις χρησιμοποιούνται για την δημιουργία νέας, μεγαλύτερης γειτονιάς N_2 και συνεχίζει τη διαδικασία μέχρι να φτάσουμε στον επιθυμητό αριθμό γειτονιών (k_{max}). Έπειτα μια τυχαία λύση δημιουργείται από την N_{max}.
- Βήμα 2: Στη δεύτερη φάση στοχεύουμε στο να διαλέξουμε διαφορετικές γειτονιές

σύμφωνα με μια στρατηγική για την αποτελεσματικότητα της διαδικασίας αναζήτησης. Έτσι ξεκινάμε από τη μεγαλύτερη γειτονιά $N_{k_{max}}$ συνεχίζοντας σε μικρότερες γειτονιές. Αυτό γίνεται για να έχουμε καλύτερη διαφοροποίηση και εντατικοποίηση. Η μεγαλύτερη γειτονιά μας δίνει τη δυνατότητα να δούμε κάθε κλάση μεταβλητών ως μια ακέραια οντότητα και να περιοριστεί μόνο στο χώρο λύσεων όπου οι μεταβλητές που είναι μέσα σε μια κλάση έχουν την ίδια αξία. Με αυτόν τον τρόπο κάθε αλλαγή από μια γειτονιά σε μια άλλη δηλώνει μια μείωση στο μέγεθος της γειτονιάς. Όταν πια η αναζήτηση έχει φτάσει στο σημείο που επιθυμούμε στη γειτονιά N_i, πρέπει να κάνουμε προβολή αυτής της γειτονιάς στην N_{i-1} που είναι ο γονέας της. Η προβολή γίνεται με τον εξής τρόπο: Αν μια κλάση $c_i \in N_m$ έχει ανάθεση σε Αληθές, τότε το ζεύγος κλάσεων $c_j, c_k \in N_{m-1}$, γίνεται επίσης Αληθές. Τελικά εφαρμόζουμε τον WalkSAT αλγόριθμο σε αυτή τη γειτονιά που προκύπτει.

Αλγόριθμος Τοπικής Αναζήτησης:

Αρχικά θα παρουσιάσουμε τον WalkSAT αλγόριθμο [6, 5] σε αλγοριθμική μορφή, που είναι η πιο διαδεδομένη και αποτελεσματική μέθοδος για Προβλήματα Ικανοποίησης. Έπειτα θα αναλύσουμε μερικά αποτελέσματα γι'αυτόν. Ο αλγόριθμος του WalkSAT είναι ο παρακάτω:

WalkSat(F, p):
$s =$ τυχαίο σύνολο από αρχικές αληθείς αναθέσεις.
Όσο τα $flips < max_flips$ επανέλαβε:
Αν το s ικανοποιεί το F τότε το
$Output \leftarrow s$
Τέλος Αλγορίθμου
Αλλιώς:
-Διαλέγουμε έναν τυχαίο μη ικανοποιημένο όρο C στο F.
-Αν κάποιες μεταβλητές στο C μπορούν να γίνουν flip χωρίς να αλλάξει κάποιος ήδη ικανοποιημένος όρος, τότε διαλέγουμε τέτοια μεταβλητή x με τυχαίο τρόπο,
αλλιώς:
-Με πιθανότητα p, διαλέγουμε x στο C με τυχαίο τρόπο.
-Με πιθανότητα $(1-p)$, διαλέγουμε x στο C που σπάει έναν ελάχιστο αριθμό ήδη υπάρχουσων ικανοποιημένων όρων.
-Κάνουμε flip τον x.
Τέλος Αν.
Τέλος Επανάληψης.

Σε κάθε βήμα διαλέγουμε μια μεταβλητή για να κάνουμε flip με τον εξής τρόπο. (Θα λέμε ότι σταματάμε αν διαλέγουμε μια μεταβλητή και την κάνουμε flip από μια λανθασμένη κατάσταση(falsified clause)). Αρχικά, μια λανθασμένη κατάσταση C διαλέγεται τυχαία. Αν υπάρχει μεταβλητή που θα σπάσουμε με τιμή 0 σε ένα C, τότε θα γίνει flip και οι μέχρι

τώρα ενώσεις θα σπάσουν με τυχαίο τρόπο. Αν δεν υπάρχει τέτοια μεταβλητή, με μία συγκεκριμένη πιθανότητα p, μια από της μεταβλητές στο C θα επιλεχθεί τυχαία. Στις υπόλοιπες περιπτώσεις, μία από τις μεταβλητές με ελάχιστη τιμή σπασίματος(minimum break value) στο C θα επιλεχθεί. Ορίζουμε 2 σημαντικές έννοιες:

$break_{min} = min_{x \in Vars(C)} break(x)$ (1)
$minbVars = \{x \mid x \in Vars(C), break(x) = break_{min}\}$ (2)

Έτσι το $break_{min}$ είναι το ελάχιστο σπάσιμο από όλες τις μεταβλητές που είναι σε λανθάνουσα μορφή και το $minbVars$ είναι μια στοίβα που αποθηκεύει όλες τις μεταβλητές με ελάχιστο σπάσιμο σε λανθάνουσα μορφή. Παρακάτω παρουσιάζεται ο Αλγόριθμος 1:

$C \leftarrow$ με τυχαία επιλεγμένη λανθάνουσα μορφή
$break_{min} \leftarrow MAXINT$
$minbVars \leftarrow \emptyset$
Για $i = 1$ στο $|C|$ κάνε:
    $v \leftarrow i^{th}$ μεταβλητή στο C
    Αν $break(v) < break_{min}$ τότε
        $break_{min} \leftarrow break(v)$
        με μοναδικό v, επαναφορά minbVars
    Αλλιώς Αν $break(v) = break_{min}$ τότε
        βάλε v στο $minbVars$
Αν $break_{min} = 0$ τότε
    $v \leftarrow$ τυχαία μεταβλητή στο $minbVars$
Αλλιώς
    Με πιθανότητα p
        $v \leftarrow$ τυχαία μεταβλητή στο C
    Με πιθανότητα 1-p
        $v \leftarrow$ τυχαία μεταβλητή στο $minbVars$
        flip v

Η αποδοτικότητα των υλοποιήσεων του αλγορίθμου εξαρτάται από την αποδοτικότητα των τιμών σπασίματος των μεταβλητών. Χρειαζόμαστε λοιπόν τις παρακάτω δομές για να υπολογίσουμε τις τιμές σπασιμάτων των μεταβλητών:

- TrueLitCount: ’Ενας πίνακας που καταγράφει τον αριθμό των true literals για όλα τα clauses. (Παραδείγματος χάρη: $TrueLitCount(0) = 2$ σημαίνει ότι το πρώτο clause έχει 2 literals).
- PosLitClause(x) για κάθε μεταβλητή x: Ένας πίνακας που αποθηκεύει τα index numbers των clauses όπου τα θετικά literal x εμφανίζονται. (Παραδείγματος χάρη: $PosLitClause(x) = \{0, 3, 6, 8, 23, 90\}$ σημαίνει ότι το literal x εμφανίζεται μόνο σε clauses των οποίων το index numbers είναι μόνο: $0, 3, 6, 8, 23, 90$)
- NegLitClause(x) για κάθε μεταβλητή x: Ένας πίνακας που αποθηκεύει το index number των clauses όπου ο negative literal $\neg x$ εμφανιζεται.

Παρατήρηση 1: Για μια μεταβλητή x, ένα clause συνεισφέρει ένα στο $break(x)$ μόνο όταν το clause έχει ένα μόνο true literal, που είναι το true literal of x.

Παρακάτω παρουσιάζεται ο αλγόριθμος 2:

$break(x) \leftarrow 0$
Αν ο x έχει αληθή τιμή
    Για κάθε clause στο $c \in PosLitClause(x)$
κάνε
        Αν $TrueLitCount(c) = 1$ τότε
            $break(x)++$
        Αν $break(x) > break_{min}$ τότε
επέστρεψε
Αλλιώς
    Για κάθε $clausec \in NegLitClause(x)$ κάνε
        Αν $TrueLitCount(c) = 1$ τότε
            $break(x)++$
        Αν $break(x) > break_{min}$
Τότε επέστρεψε
Τέλος Αν

Όπως παρατηρούμε και στον αλγόριθμο 2, για να υπολογίσεις το $break(x)$ χρειάζεται να κάνεις σκανάρισμα σε έναν από τους 2 πίνακες από indexes του x. (π.χ $PosLitClause(x)$, $NegLitClause(x)$). Για μία μεταβλητή x που θέλουμε να υπολογίσουμε την τιμή σπασίματος της, αρχικά αρχικοποιούμε το $break(x) = 0$. Μετά αν η τιμή αλήθειας του x είναι Αληθής σε ορισμένες αναθέσεις τότε κάνουμε σκανάρισμα στον $PosLitClause(x)$ πίνακα και υπολογίζουμε $break(x)$ ως εξής: Για κάθε $c \in PosLitClause(x)$, κοιτάμε αν το clause περιέχει ακριβώς έναν true literal. Αν ναί, τότε ο literal x είναι ο μόνος true literal στο clause c. Έτσι κάνουμε flip την τιμή του x θα έκανε το clause c να γίνει e falsified from satisfied και έτσι το $break(x)$ θα αυξανόταν κατά 1. Αν η αληθής τιμή του x είναι Ψευδής σε ορισμένες αναθέσεις, θα κάνουμε σκανάρισμα το $NegLitClause(x)$ και θα υπολογίσουμε το $break(x)$ με τον ίδιο τρόπο.

Είναι φανερό ότι ο αλγόριθμος 2 υπολογίζει σωστά το $break(x)$ εάν το x έχει το ελάχιστο σπάσιμο από όλα τα άλλα. Για όλες τις υπόλοιπες μεταβλητές όπου το σπάσιμο είναι μεγαλύτερο από το $break_{min}$, δεν είναι αναγκαίο να υπολογίσουμε τα σπάσιμο τους.

Η διαδικασία υπολογισμού είναι πολύ αποτελεσματική. Δεδομένου ενός $k - SAT$, για μια μεταβλητή x, το μέγεθος των $PosLitClause(x)$ ή $NegLitClause(x)$ είναι $1/2 * km/m * r = k/2 * r$. Συγκεκριμένα για $3 - SAT$ για να υπολογίσουμε μια μεταβλητή σπασίματος χρειαζόμαστε μόνο $3/2 * r$, που είναι περίπου 6 όταν $r = 4.2$. Επιπλέον όπως είπαμε όταν η τιμή του σπασίματος είναι μεγαλύτερη από $break_{min}$, μπορούμε αμέσως να τερματίσουμε την διαδικασία.

4.3 Καλύτερη Δυναμική Απόκρισης και Συστήματα Ισορροπίας Nash

Μέχρι στιγμής έχουμε δεί προβλήματα όπου η τοπική αναζήτηση χρησιμοποιείται για επίλυση προβλημάτων που έχουν μονάχα έναν σκοπό. Δηλαδή να εκτελούμε πράξεις τοπικής

αναζήτησης για να προκύψει μια λύση που θα κάνει ελαχιστοποίηση ή μεγιστοποίηση το συνολικό κόστος. Υπάρχουν περιπτώσεις ωστόσο όπου υπάρχουν παραπάνω από ένας συντελεστές που πρέπει να ικανοποιηθούν, ο καθένας με τους δικούς του σκοπούς και απαιτήσεις προσδοκώντας στην βέλτιστη γι αυτόν λύση. Θα δούμε πως η ύπαρξη παραπάνω του ενός συντελεστή οδηγεί σε ένα νέο είδος τοπικής αναζήτησης.

Πρόβλημα:

Θα περιγράψουμε τη Καλύτερη Δυναμική Απόκρισης και τα Συστήματα Ισορροπίας Nash (best response dynamics and nash equilibrium) όπως ορίζεται στο βιβλίο `Algorithm Design` [8]. Σε ένα δίκτυο όπως το Nash internet υπάρχουν φορές όπου ένας αριθμός κόμβων θέλουν ταυτόχρονα να συνδεθούν με μια πηγή που αναπαριστάται με κόμβο s. Για παράδειγμα μπορεί ο s να παράγει δεδομένα τα οποία είναι χρήσιμα για πολλούς κόμβους. Θα μοντελο-ποιήσουμε την κατάσταση ενός συστήματος ισσοροπίας με έναν γράφο $G = (V, E)$ με κόστη $c_e \geq 0$ για κάθε ακμή. Υπάρχει ένας κόμβος πηγή $s \in V$ και k βοηθητικοί κόμβοι $t_1, t_2, ..., t_k \in V$. Κάθε βοηθητικός κόμβος t_j επιθυμεί ένα μονοπάτι P_j από το s στο t_j έχοντας όσο λιγότερο κόστος γίνεται. Αν δεν υπάρχουν αλληλεπιδράσεις μεταξύ των βοηθητικών κόμβων, μπορούμε να θεωρήσουμε ότι έχουμε: k διαφορετικά προβλήματα μικρότερων μονοπατιών. Κάθε βοηθητικός κόμβος επιθυμεί ένα μονοπάτι $s - t_j$ όπου το συνολικό κόστος ακμών θα γίνεται ελάχιστο.

Το πρόβλημα γίνεται ενδιαφέρον με την προοπτική οι βοηθητικοί κόμβοι να είναι σε θέση να κάνουν κοινή χρήση τα κόστη των ακμών με τους υπόλοιπους κόμβους. Ας υποθέσουμε ότι μόλις όλοι οι βοηθητικοί κόμβοι έχουν διαλέξει το μονοπάτι τους, ο καθένας να θέλει να πληρώσει μόνο το κόστος που του αναλογεί από κάθε ακμή στο μονοπάτι του. Άρα για κάθε κόστος c_e διαιρείται ανάλογα με το πόσοι βοηθητικοί κόμβοι συμμετέχουν στην ακμή e.

Σε αυτό το σημείο θα δώσουμε 2 ορισμούς που είναι σχετικοί με κάθε σύνολο όπου πολλοί βοηθητικοί κόμβοι αλληλεπιδρούν για τους δικούς τους σκοπούς και στόχους για να παράγουν μια κοινή λύση:

- Αρχικά κάθε βοηθητικός κόμβος είναι κάθε στιγμή προετοιμασμένος να βελτιώσει τη λύση του αναλόγως με τις αλλαγές που έγιναν από τους άλλους κόμβους. Θα αναφερόμαστε σε αυτή τη διαδικασία ως καλύτερη δυναμική αποκρισης. Δηλαδή μας ενδιαφέρει κάθε βοηθητικός κόμβος να κάνει ανανέωση της κατάστασης του, ώστε να έχει την καλύτερη δυνατή αλληλεπίδραση με τους άλλους κόμβους στην τωρινή κατάσταση.
- Δεύτερον, μας ενδιαφέρουν πιο πολύ οι σταθερές λύσεις. Δηλαδή μας ενδιαφέρει να φτάσουμε σε μία κατάσταση στην οποία κάθε βοηθητικός κόμβος δεν χρειάζεται να αλλάξει γιατί είναι στην καλύτερη δυνατή κατάσταση. Όταν πλέον δεν θα χρειάζεται να αλλάξει κανένας κόμβος την κατάσταση του θα λέμε ότι έχουμε ένα Σύστημα Ισσοροπίας Nash.

Σε αυτό το σημείο πρέπει να δούμε τη σύνδεση του προβλήματος με την τοπική αναζήτηση. Κάθε βοηθητικός κόμβος έχει τη δική του συνάρτηση να κάνει ελαχιστοποίηση και δεν είναι ξεκάθαρο αν συνολικά υπάρχει βελτίωση. Για παράδειγμα αν υπάρχει βελτίωση στον t_i

που μειώνεται το κόστος του μπορεί να υπάρχει αύξηση σε έναν t_j. Άρα υπάρχουν 2 βασικά ερωτήμα:

- Αν υπάρχει ένα Σύστημα Ισορροπίας Nash. Δηλαδή το γεγονός ότι υπάρχει βελτίωση σε έναν μόνο βοηθητικό κόμβο δεν μας εγγυάται ότι υπάρχει και συνολική. Δεν είναι επίσης σαφές αν τα συστήματα καλύτερης δυναμικής απόκρισης τερματίζουν πάντα, καθώς όποτε βελτιώνεται ενα t_1, ένα t_2 γίνεται χειρότερο και το ανάποδο.
- Την ύπαρξη σταθερότητας. Δοσμένου ενός συνόλου από βοηθητικούς κόμβους, με κόμβους $t_1, t_2, ..., t_k$, μπορούμε να προτείνουμε μία συλλογή από μονοπάτια , με τον κάθε βοηθητικό κόμβο να έχει 2 ιδίοτητες:
  - Το σύνολο των μονοπατιών σχηματίζει ένα Σύστημα Ισορροπίας Nash.
  - Το συνολικό κόστος όλων των βοηθητικών κόμβων είναι όσο μικρότερο γίνεται.

Γειτονιά:

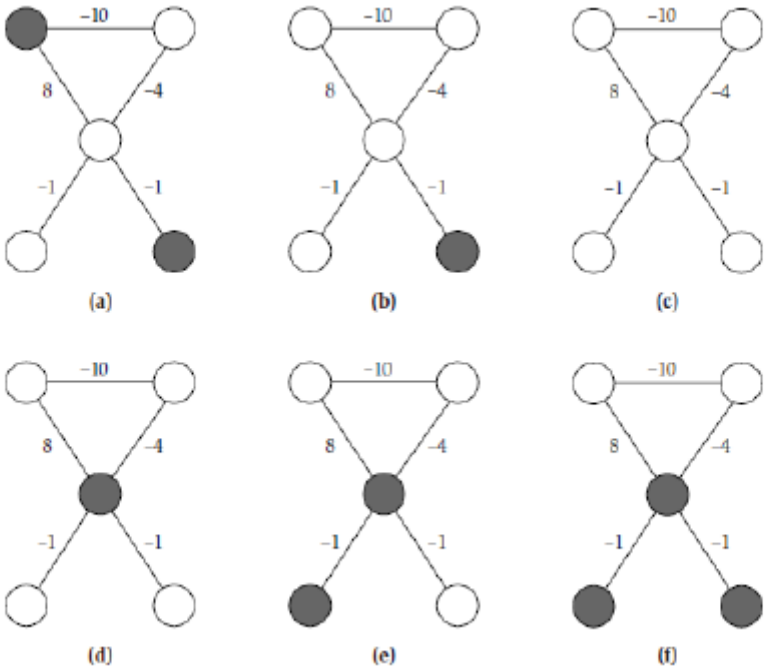
Η ιδέα είναι ότι δεν χρειαζόμαστε το συνολικό κόστος όλων των βοηθητικών κόμβων. Κάθε ποσότητα με την οποία μειώνεται το μονοπάτι κάθε βοηθητικός κόμβος και που μπορεί να κάνει μείωση του κόστους σε πεπερασμένο αριθμό φορών, είναι μία καλή αλλαγή. Γνωρίζοντας αυτό θα δοκιμάσουμε να φτιάξουμε μια διαδικασία που θα έχει αυτά τα 2 χαρακτηριστικά, μη έχοντας σημασία για το συνολικό κόστος, αλλά είναι αρκετά καλή εφόσον πληροί αυτές τις 2 ιδιότητες.

Έτσι διαισθητικά, αν έχουμε x βοηθητικούς κόμβους σε μια ακμή e, δυνητικά αν ο πρώτος σταματήσει να χρησιμοποιεί την e, θα έχουμε μείωση c_e/x, $c_e/(x-1)$ όταν σταματήσει ο επόμενος και ούτε ο κάθε εξής. Άρα συνολικά: $c_e(1/x+1/(x-1)+...+1/2+1) = c_e * H(x)$. Ορίζοντας το σύνολο μονοπατιών $P_1, P_2, ..., P_k$ ως $\Phi(P_1, P_2, ..., P_k)$. Για κάθε ακμή έστω x_e δηλώνει τον αριθμό των βοηθητικών κόμβων οι οποίοι χρησιμοποιούν την ακμή e στο μονοπάτι τους. Έχουμε: $\Phi(P_1, P_2, ..., P_k) = \sum_{e \in E} c_e * H(x_e)$, $H(0) = 0$.

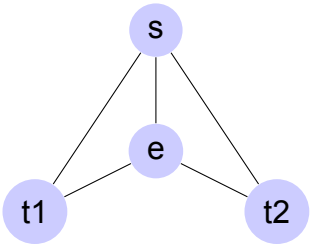
Ένα παράδειγμα του προβλήματος που οδηγεί σε σταθερή λύση φαίνεται στο Σχήμα 3.3. Απο τα (a)-(f) φαίνονται τα βήματα του state-flipping αλγορίθμου για ένα γράφο με 5 κόμβους σε ένα Σύστημα Ισορροπίας Nash.

Αλγόριθμος Τοπικής Αναζήτησης:

Η συνάρτηση Φ είναι πολύ χρήσιμη για να μας δώσει ένα φράγμα στην σταθερότητα. Παρόλο που η Φ δεν είναι ίση με το συνολικό κόστος όλων των κόμβων, είναι πολύ κοντά σε αυτό.



Σχήμα 4.3: State-flipping.



Σχήμα 4.4: Ένα σύστημα ισορροπίας Nash διαφέρει από το ολικό βέλτιστο.
Με μήκη ακμών (s, t1) = 3, (s, t2) = 5, (s, e) = 5 , (e, t1) = 1, (e, t2) = 1 ένα δίκτυο όπου η μοναδική ισορροπία Nash διαφέρει από το ολικό βέλτιστο.

$C(P_1, P_2, ..., P_k) \leftarrow$ το συνολικό κόστος όλων των κόμβων με μονοπάτια $(P_1, P_2, ..., P_k)$.
Η σχέση μεταξύ της συνάρτησης κόστους C και της δυνητικής συνάρτησης κόστους Φ είναι:

$$C(P_1, P_2, ..., P_k) \leq \Phi\ (P_1, P_2, ..., P_k) \leq H(k) * C(P_1, P_2, ..., P_k).$$

Αυτό ισχύει διότι, αν ορίσουμε E^+ ως σύνολο όλων των ακμών που ανήκουν τουλάχιστον μια φορά στα μονοπάτια $(P_1, P_2, ..., P_k)$, έχουμε:

$$C(P_1, P_2, ..., P_k) = \sum e \in E_+ c_e.$$

Επίσης ισχύει ότι: $x_e \leq k$ για όλες τις e. Έτσι:

$$C(P_1, P_2, ..., P_k) = \sum e \in E_+ c_e \leq \sum e \in E_+ c_e * H(x_e) = \Phi(P_1, P_2, ..., P_k)$$

και

$$\Phi(P_1, P_2, ..., P_k) = \sum e \in E_+ c_e * H(x_e) \leq \sum e \in E_+ c_e * H(k) = H(k) * C(P_1, ..., P_k)$$

Έτσι υπάρχει Σύστημα Ισορροπίας Nash στην οποία το συνολικό κόστος όλων των κόμβων υπερβαίνει το κόστος του ολικού βέλτιστου κατά ένα συντελεστή $H(k)$. Διότι αν ξεκινήσουμε από ένα ολικό βέλτιστο των μονοπατιών $P_1^*, ..., P_k^*$, κατά την εκτέλεση μπορεί το συνολικό κόστος των κόμβων να βελτιώνεται, άλλα η δυνητική συνάρτηση να μειώνεται. Άρα έχουμε: $\Phi(P_1, P_2, ..., P_k) \leq \Phi\ (P_1^*, ..., P_k^*)$.

Επίσης οι συναρτήσεις C και Φ διαφέρουν κατά $H(k)$. Οπότε: $C(P_1, P_2, ..., P_k) \leq \Phi\ (P_1, P_2, ..., P_k) \leq \Phi\ (P_1^*, P_2^*, ..., P_k^*) \leq H(k) * C(P_1^*, P_2^*, ..., P_k^*)$

Έτσι δείξαμε ότι πάντα υπάρχει ένα Σύστημα Ισορροπίας Nash, όπου το συνολικό του κόστος είναι $H(k)$ κατά του ολικού βέλτιστου.

4.4 Πρόβλημα Χωροτοποθέτησης

Πρόβλημα:

Στο Πρόβλημα Χωροτοποθέτησης(Uncapacitated Facility Location Problem) [3, 4, 10] έχουμε ένα σύνολο από εταιρίες F(εταιριών) με κόστη για να ανοίξει μια εταιρία και ένα σύνολο από πελάτες C, όπου πρέπει να ικανοποιηθούν οι απαιτήσεις τους. Στόχος είναι να βρεθεί ένα σύνολο από εταιρίες έτσι ώστε να ικανοποιηθούν όλοι οι πελάτες στο C από τις εταιρίες του S, ετσι ώστε να επιτύχουμε ελαχιστοποίηση του συνολικού κόστους ανοικτών εταιριών(facility costs) και ελαχιστοποίηση της συνολικής απόστασης μεταξύ οποιουδήποτε πελάτη $j \in C$ που εξυπηρετείται από μια οποιαδήποτε εταιρία $s_{(j)} \in S$. Δηλαδή: $cost(S) = \sum_{i \in S} f_i + \sum_{(j) \in C} c_{s_{(j)j}}$. με:

$S \subset F$
C: σύνολο από πελάτες.
S: σύνολο από εταιρίες.
f_i: κόστος ανοίγματος της εταιρίας i.
$c_{s_{(j)j}}$: απόσταση μεταξύ ενός πελάτη j και μίας εταιρίας i.
$s_{(j)j} \in S$: Ο πελάτης j εξυπηρετείται από την $s_{(j)} \in S$

Γειτονιά:

Για να δημιουργήσουμε μια γειτονιά στο πρόβλημα της Ενοικίασης Εταιριών με απεριόριστη χωρητικότητα, ορίζουμε αυθαίρετα μια τυχαία αρχική λύση, που περιέχει k εταιρίες. Αυτό που κάνουμε είναι είτε να ανοίξουμε μία νέα εταιρία αν $|F| \leq k$, είτε να επιλέξουμε κάνοντας αλλαγή μίας νέας εταιρίας για να μπεί στην λύση και μία s απ'την ήδη υπάρχουσα λύση για να βγεί από αυτή. Αν η νέα λύση που βρούμε μετά από αύτο το flip είναι καλύτερη τότε ανανεώνουμε την γειτονιά έτσι μέχρι τώρα έχουμε τα παρακάτω:

$N(s_0) \leftarrow$ μία αρχική λύση
$N^T(s) \leftarrow$ Ανοιχτές εταιρίες που δεν ανήκουν στην τωρινή λύση
- Ανοίγουμε μία εταιρία $s^{'}$ που ανήκει στο $N^T(s)$ και μας δίνει το καλύτερο δυνατό αποτέλεσμα.
Δηλαδή:
$s' = N^T(s)|open(s') = best(s)$
- Η νέα γειτονιά προκύπτει:
1. είτε από προσθήκη της $s^{'}$ στην γειτονιά,
2. είτε από αφαίρεση μιας s από τη γειτονιά,
3. είτε και από τα 2 μαζί.

Δηλαδή:
$N(s) \leftarrow \{S + s'\}$
$\cup\{S-s|s \in S\} \cup \{S-s+s'|s \in S\}$

Αλγόριθμος Τοπικής Αναζήτησης:

Στον αλγόριθμο της τοπικής αναζήτησης , θα αρχικοποιήσουμε μια γειτονιά που θα περιέχει έναν αριθμό από εταιρίες και θα χρησιμοποιήσουμε μια "διαδικασία" op η οποία περιέχει (1) - είτε την προσθήκη μιας εταιρίας , (2) - είτε την αφαίρεση της, (3) - είτε την αντικατάσταση της (swap) με μια άλλη που είναι εκτός λύσης.

Λήμμα 1: Για το Κόστος Εξυπηρέτησης ισχύει ότι: $cost_S(S') \leq cost_f(S) + cost_S(S)$

Απόδειξη: Έστω ότι έχουμε την βέλτιστη λύση (optimum solution) O. Ας θεωρήσουμε μια πράξη κατά την οποία γίνεται προσθήκη μιας εταιρίας $o \in O$. Κάνουμε ανάθεση όλων των πελατών $()$ sto . Απο το τοπικό βέλτιστο του S έχουμε: $f_o + \sum_{j \in N_O(o)}(O_j - S_j) \geq 0$. Ακόμα

και αν $o \in S$ η ανισότητα εξακολουθεί να ισχύει. Αν προσθέσουμε για $\forall o \in O$ αυτές τις ανισότητες προκύπτει ότι: $cost_S(S') \leq cost_f(S) + cost_S(S)$.

Λήμμα 2: Για το Κόστος Εταιρίας ισχύει ότι: $cost_f(S') \leq cost_f(S) + 2cost_S(S)$

Απόδειξη: Όπως και στην προηγούμενη απόδειξη θεωρούμε ένα συγκεκριμένο $o \in O$. Έστω μια χαρτογράφηση (mapping) $\pi : N_O(o) \rightarrow N_O(o)$ που είναι 1-1. Θεωρούμε ότι αν $|N_s^o| > 1/2 * |N_O(o)|$, τότε για όλα τα $j \in N_s^o$, για τα οποία ισχύει ότι: $\pi(j) \in N_s^o$, έχουμε $\pi(j) = j$.

Για να βρούμε μια τέτοια συνάρτηση π ακολουθούμε την παρακάτω διαδικασία. Έστω $|N_s^o| > 1/2 * |N_O(o)|$, διαλέγουμε οποιουσδήποτε $|N_s^o| - |() \setminus |N_s^o||$ πελάτες j από το $|N_s^o|$ και κάνουμε $\pi(j) = j$. Στους πελάτες που απέμειναν στο $N_O(o)$ και ισχύει ότι: $|N_s^o| \leq 1/2 * |N_O(o)|$, θα θεωρήσουμε ότι: $\pi(N_s^o) \cap N_s^o = \emptyset$.

Θεωρούμε ότι μια εταιρία $s \in S$ είναι καλή αν δεν πιάνει (capture) κανένα $o \in O$ που έχει: $|N_s^o| \leq 1/2 * |N_O(o)|$. Θεωρούμε την πράξη όπου αφαιρούμε μια εταιρία $s \in S$. Έστω $j \in N_S(s)$ και $\pi(j) \in N_S(s^{'})$. Αφού η s δεν πιάνει καμία εταιρία $o \in O$, προκύπτει ότι $s^{'} \neq s$. Έαν εναποθέσουμε το j στο $s^{'}$, προκύπτει ότι:
$-f_s + \sum_{j \in N_S(s)}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) \geq 0$
Για όλα τα $j \in N_S(s), \pi(j) \neq j \Rightarrow \sum_{j \in N_S(s), \pi(j)=j} O_j = 0$. Άρα μπορούμε να ξαναγράψουμε την παραπάνω ανισότητα ως εξής:
$-f_s + \sum_{j \in N_S(s)}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) + 2\sum_{j \in N_S(s), \pi(j)=j} O_j \geq 0$ (1)

Για μία κακή εταιρία $s \in S$ το κόστος εταιρίας το βρίσκουμε με τον εξής τρόπο. Ας υποθέσουμε ότι η s πιάνει ένα υποσύνολο από εταιρίες $P \subset O$. Έστω $o \in P$ είναι η εταιρία πιο κοντά στην s. Θεωρούμε $swap\langle s, o\rangle$. Οι πελάτες $j \in N_S(s)$ τοποθετούνται στις εταιρίες στο $S - s + o$ με τον εξής τρόπο:

1. Αν $\pi(j) \in N_S(s^{'})$ για $s^{'} \neq s$. Τότε ο j πάει στον $s^{'}$. Αν $j \in N_O(o^{'})$. Έχουμε $c_{js^{'}} \leq c_{jo^{'}} + c_{\pi(j)o^{'}} + c_{\pi(j)s^{'}} = O_j + O_{\pi(j)} + S_{\pi(j)}$
2. Αν $\pi(j) = j \in N_S(s)$ και $j \in N_O(o)$. Τότε ο j πάει στο o.
3. Αν $\pi(j) = j \in N_S(s)$ και $j \in N_O(o^{'})$ για $o^{'} \neq o$. Ο πελάτης j πάει στην εταιρία o. Απο την τριγωνική ανισότητα έχουμε: $c_{jo} \leq c_{js} + c_{so}$. Αφού το $o \in P$ είναι η πιο κοντινή εταιρία στο s, ισχύει ότι $c_{so} \leq c_{so^{'}} \leq c_{js} + c_{jo^{'}}$. Άρα: $c_{jo} \leq c_{js} + c_{js} + c_{jo^{'}} = S_j + S_j + O_j$.

Έτσι για το $swap\langle s, o\rangle$, έχουμε την παρακάτω ανισότητα:

$$f_o + f_s + \sum_{j \in N_S(s), \pi(j) \neq j}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j)$$
$$+ \sum_{j \in N_O(o), \pi(j)=j \in N_S(s)}(O_j - S_j) + \sum_{j \notin N_O(o), \pi(j)=j \in N_S(s)}(S_j + S_j + O_j + S_j) \geq 0 \text{ (2)}$$

Ας θεωρήσουμε μια πράξη όπου μια εταιρία $o^{'} \in P - o$ προστίθεται. Οι πελάτες $j \in N_O(o)$ που ισχύει ότι $\pi(j) = j \in N_S(s)$, τοποθετούνται στην εταιρία $o^{'}$ και προκύπτει η παρακάτω ανισότητα.

$$f_{o^{'}} + \sum_{j \in N_O(o^{'}), \pi(j)=j \in N_S(s)}(O_j - S_j) \geq 0 \text{ (για κάθε } o^{'} \in P - o) \text{ (3)}$$

Προσθέτοντας την ανισότητα (2) στην ανισότητα (3), για μία κακή εταιρία $s \in S$

$$\sum_{o^{'} \in P} f_{o^{'}} - f_s + \sum_{j \in N_S(s), \pi(j) \neq j}(O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) + 2\sum_{j \in N_S(s), \pi(j)=j} O_j \geq 0 \text{ (4)}$$

Ο τελευταίος όρος στα αριστερά είναι ένα άνω φράγμα στο άθροισμα των δύο τελευταίων όρων στην ανισότητα (2) και του τελευταίου όρου της ανισότητας (3) για όλα τα $o^{'} \in P - o$. Τώρα κάνουμε πρόσθεση της ανισότητας (1) για όλες τις καλές εταιρίες $s \in S$ και την ανισότητα (4) για όλες τις κακές εταιρίες $s \in S$ και τέλος τις ανισότητες $f_o \geq 0$ για όλα τα $o \in O$, που δεν έχουν πιαστεί από καμία $s \in S$. Το συμπέρασμα που βγαίνει είναι:

$$\sum_{o \in O} f_o + \sum_{s \in S} f_s + \sum_{\pi(j) \neq j} (O_j + O_{\pi(j)} + S_{\pi(j)} - S_j) + 2 \sum_{\pi(j)=j} O_j \geq 0$$

Σημειώνουμε ότι $\sum_{\pi(j) \neq j} O_j = \sum_{\pi(j) \neq j} O_{\pi(j)}$ και $\sum_{\pi(j) \neq j} S_j = \sum_{\pi(j) \neq j} S_{\pi(j)}$.

Άρα από τα παραπάνω έχουμε: $\sum_{\pi(j) \neq j} (O_j + O_{\pi(j)} + S_{\pi(j)}) = 2 \sum_{\pi(j) \neq j} O_j \Rightarrow cost_f(O) + cost_f(S) + 2 * cost_s(O) \geq 0$.

Αφού αποδείξαμε το Λήμμα 1 και το Λήμμα 2 μπορούμε να διατυπώσουμε το παρακάτω θεώρημα.

Θεώρημα: Ο αλγόριθμος της τοπικής αναζήτησης στο Πρόβλημα Χωροτοποθέτησης μπορεί να προσεγγιστεί από έναν παράγοντα: $1 + \sqrt{2} + \epsilon$

Απόδειξη: Γνωρίζουμε ότι τα Κόστη εξυπηρέτησης και το Κόστος Εταιρίας είναι $cost_S(s)$ και $cost_f(S)$ αντίστοιχα. Έστω $cost^{'}_f(A)$ και $cost^{'}_S(A)$ είναι τα αντίστοιχα κόστη μιας λύσης A και το S είναι το τοπικό βέλτιστο. Αυτές οι δύο λύσεις είναι ίδιες κατά έναν παράγοντα $a > 0$. Έτσι: $cost_f(S) + cost_S(s) = cost^{'}_f(A)/a + cost^{'}_S(A)$
$\leq (cost^{'}_f(O) + 2cost^{'}_S(O))/a + cost^{'}_f(O) + cost^{'}_S(O)$
$= (1 + a)cost_f(O) + (1 + 2/a)cost_S(O)$
Αν υποθέσουμε ότι $a = \sqrt{2}$, προκύπτει ότι: $cost(S) \leq (1 + \sqrt{2})cost(O)$. Άρα ο αλγόριθμος της τοπικής αναζήτησης έχει μια προσέγγιση $1 + \sqrt{2} + \epsilon$.

4.5 Κινητή Χωροτοποθέτηση

Μελετήσαμε στην προηγούμενη ενότητα το Πρόβλημα Χωροτοποθέτησης με απεριόριστη χωρητικότητα, όπου έχουμε ένα σύνολο από εταιρίες με κόστη ανοίγματος και ένα σύνολο από πελάτες που επιθυμούν να εξυπηρετηθούν, ανοίγοντας κάποιες εταιρίες ώστε να ελαχιστοποιηθεί το άθροισμα του κόστους των ανοιγμένων εταιριών και των κοστών ανάθεσης πελατών(client assigments costs).

Πρόβλημα:

Σε αυτή την ενότητα θεωρούμε το πρόβλημα Χωροτοποθέτησης όπου οι εταιρίες είναι κινητές(mobile) και μπορούν να επανατοποθετηθούν(relocated) σε προορισμούς κοντά σε πελάτες για να τους εξυπηρετήσουν περισσότερο αποτελεσματικά, με το να μειώνουν τα κόστη των πελατών [2]. Πιο συγκεκριμένα θεωρούμε ένα πρόβλημα Κινητής Χωροτοποθέτησης (mobile facility location problem) ως εξής: Έχουμε έναν πλήρη γράφο $G = (V, E_G)$, με κόστη $c(u, v)$ στις ακμές, ένα σύνολο $D \subset V$ από πελάτες, όπου ο κάθε πελάτης j έχει d_j μονάδες από απαιτήσεις και ένα σύνολο $F \subset V$ από k αρχικές εταιρίες ενοικίασης. Χρησιμοποιούμε τον όρο εταιρία i για να δηλώσουμε την εταιρία της οποίας η αρχική τοποθεσία είναι $i \in F$. Μια λύση S στη Κινητή Χωροτοποθέτηση μετακινεί κάθε εταιρία

i σε μια τελική τοποθεσία $s_i \in V$ (που μπορεί να είναι ίδιο με το i), έχοντας ένα κόστος μετακίνησης $c(i, s_i)$, κάνοντας ανάθεση κάθε πελάτη j σε μια τελική τοποθεσία $s \in S$, με κόστος ανάθεσης $d_j c(j, s)$. Το συνολικό κόστος του S είναι το άθροισμα όλων των κοστών μετακίνησης και των κοστών ανάθεσης. Άρα στον κάθε πελάτη θα γίνει ανάθεση στην πιο κοντινή τοποθεσία μέσα στο S, το κόστος του S είναι: $MFL(S) = \sum_{i \in F} c(i, s_i) + \sum_{j \in D} d_j c(j, s(j))$

Όπου $s(v)$ (για οποιονδήποτε κόμβο v) δίνει την τοποθεσία στο S κοντινότερα στο v (με αυθαίρετο σπάσιμο των ισοτήτων). Υποθέτουμε ότι τα κόστη των ακμών είναι μετρικά.

Η Κινητή Χωροτοποθέτηση γενικεύεται σε μικρότερα προβλήματα μετακίνησης [2] που έχουν αναλυθεί από τον De-maine. Σε αυτά τα προβλήματα, έχουμε έναν αρχικό σχηματισμό σε έναν γράφο, όπου τοποθετούμε σημάδια(”pebbles”) στους κόμβους και/ή ακμές. Ο στόχος είναι να μετακινήσουμε τα σημάδια αυτά έτσι ώστε να εξασφαλίσουμε το επιθυμητό τελικό σχηματισμό, ένω παράλληλα να ελαχιστοποιήσουμε τον αριθμό μετακινήσεων των σημαδιών για να φτάσουμε στο επιθυμητό αποτέλεσμα. Η Κινητή Χωροτοποθέτηση αναλύθηκε από τον Demaine ως ένα πρόβλημα μετακίνησης όπου τα σημάδια από εταιρίες και πελάτες(facility-and-client-pebbles) τοποθετούνται σε αρχικές τοποθεσίες από εταιρίες και πελάτες, και στον τελικό σχηματισμό κάθε σημάδι ενός πελάτη πρέπει να είναι στην ίδια τοποθεσία με κάποιο σημάδι εταιρίας.

Γειτονιά:

Παρατηρούμε ότι δεδομένων των τελικών τοποθετήσεων των εταιριών, μπορούμε να βρούμε το ελάχιστο κόστος με τη μετακίνηση των αρχικών τοποθεσιών στις τελικές τοποθεσίες, λύνοντας ένα πρόβλημα τέλειου ταιριάσματος ελάχιστου κόστους(minimum-cost perfect-matching problem). Έτσι, δίνουμε έμφαση στο να καθορίσουμε ένα καλό σύνολο από τελικές τοποθεσίες. Σε κάθε βήμα λοιπόν, έχουμε το δικαίωμα να κάνουμε προσθήκη και αφαίρεση ένος προκαθορισμένου αριθμού, έστω p, από τοποθεσίες. Για κάθε p μπορούμε να βρούμε την καλύτερη τοπική κίνηση αποτελεσματικά (αφού το κόστος ενός συνόλου από τελικές τοποθετήσεις μπορεί να υπολογιστούν σε πολυωνυμικό χρόνο). Ας σημειωθεί ωστόσο ότι δεν έχουμε περιορισμούς για το πώς θα γίνει το ταίριασμα μεταξύ των αρχικών και των τελικών προορισμών, που μπορεί να αλλάξει σε μια τοπική κίνηση, καθώς μια τέτοια κίνηση μπορεί να περιέχει όλες τις εταιρίες. Επίσης είναι σημαντικό να έχουμε μια ευκαμψία(flexibility), ώστε η τοπική αναζήτηση σε κάθε βήμα να κάνει μετακίνηση ένος σταθερού αριθμού από εταιρίες σε επιλεγμένους προορισμούς, έχοντας ένα μη φραγμένο προσεγγιστικό φράγμα.

Μια από τις δυσκολίες στο να εφαρμόσουμε την τοπική αναζήτηση στη Κινητή Χωροτοποθέτηση είναι η εξής: Το κόστος του ανοίγματος ενός συνόλου S από τοποθεσίες είναι το κόστος του ελάχιστου κόστους τέλειου ταιριάσματος του F στο S, όπου σε αντίθεση με τα υπόλοιπα προβλήματα ενοικιάσεων εταιριών, είναι μια συνάρτηση του S. Στα περισσότερα προβλήματα αυτού του είδους με κόστη ανοίγματος για τα οποία η τοπική αναζήτηση γνωρίζουμε ότι δουλευε, θα κάνουμε αλλαγή σε μία εταιρία που θα χρησιμοποιείται για το συνολικό βέλτιστο και θα έχουμε ένα φράγμα στην αλλαγή των κοστών των εταιριών, ώστε μετά να επιλέξουμε πώς να κάνουμε νέα ανάθεση των πελατών ώστε να είναι πιο αποτελεσματικά τα κόστη($cost-effective$). Στη Κινητή Χωροτοποθέτηση, δεν έχουμε αυτή

την ευκαμψία και προσεκτικά πρέπει να επιλέξουμε πώς να κάνουμε αλλαγή των εταιριών ώστε να είμαστε σίγουροι ότι έχουμε καλό ταίριασμα στους νέους προορισμούς μετά από μία αλλαγή και με όσο λιγότερο κόστος επανατοποθέτησης των πελατών.

Έτσι οδηγούμαστε να σκεφτούμε τελικά μονοπάτια επανατοποθετήσεων για να κάνουμε νέα ταιρίασματα των εταιριών σε νέους προορισμούς μετά από μια αλλαγή. Αυτά τα μονοπάτια είναι της μορφής $(..., s_i, o_i, s_{it}, ...)$, όπου s_i και o_i είναι οι νέες τοποθεσίες στις οποίες η εταιρία i μετακινείται στα τοπικά και ολικά βέλτιστα, S και O αντίστοιχα, και s_{it} είναι η $S-$τοποθεσία που είναι πιο κοντά στο o_i. Δεδομένης μίας κίνησης αλλαγής που περιέχει το αρχική και την τελική τοποθεσία ενός τέτοιου μονοπατιού Z, μπορούμε να πάρουμε ένα φράγμα, για την μετακίνηση όλων των εταιριών $i \in Z$ όπου s_i είναι η αρχή του μονοπατιού ή το o_i εξυπηρετεί έναν μεγάλο αριθμό από πελάτες. Για τις εταιρίες που απομένουν, σπάμε το μονοπάτι Z σε κατάλληλα χρονικά διαστήματα, που το καθένα περιέχει έναν σταθερό αριθμό από μη μετρήσιμες τοποθεσίες που συμμετέχουν με τη σειρά τους σε μια αλλαγή πολλαπλών προορισμών(multi-location swap). Αυτή η νέα αλλαγή(interval-swap) δεν φαίνεται αρχικά να είναι χρήσιμη, αφού μπορούμε να φράξουμε μόνο το κόστος αλλαγής, λόγω του ότι είναι ένα σημαντικό πολλαπλάσιο του κόστους του τοπικού βέλτιστου.

Αλγόριθμος Τοπικής Αναζήτησης:

Για να καθορίσουμε την τελική λύση του προβλήματος θα πρέπει να ορίσουμε ένα σύνολο από τελικούς προορισμούς των εταιριών, αφού τότε μπορούμε να υπολογίσουμε αποτελεσματικά την καλύτερη κίνηση των εταιριών από την αρχική στην τελική τοποθεσία, καθώς και ποιες εταιρίες θα εξυπηρετούν τον κάθε πελάτη. Έτσι έχουμε τον ακόλουθο αλγόριθμο τοπικής αναζήτησης [2].

Δεδομένου ενός συνόλου S από $k = |F|$ τοποθεσίες, μπορούμε να μετακινηθούμε σε οποιοδήποτε άλλο σύνολο S_t από k άλλες τοποθεσίες έτσι ώστε $|S\ S_t| = |S_t\ S| \leq p$, όπου p είναι μία προκαθορισμένη τιμή. Δηλώνουμε αυτή την κίνηση ως $swap(S\ S_t, S_t\ S)$. Ο αλγόριθμος τοπικής αναζήτησης αρχίζει με ένα αυθαίρετο σύνολο από k τελικές τοποθεσίες. Σε κάθε επανάληψη, επιλέγουμε μια κίνηση τοπικής αναζήτησης που αποφέρει την μεγαλύτερη μείωση στο συνολικό κόστος και ανανεώνει το σύνολο των τελικών προορισμών με τον εξής τρόπο: Αν δεν υπάρχουν κινήσεις που βελτιώνουν τα κόστη, τότε τερμάτιζουμε. Για να επιτύχουμε πολυωνυμικό χρόνο εκτέλεσης, κάνουμε μια τροποποίηση, ώστε να επιλέγουμε μία κίνηση τοπικής αναζήτησης αν και μόνον αν η μείωση του κόστους είναι το λιγότερο $e(currentcost)$.

Ανάλυση που οδηγεί σε 5-approximation:Θα αναλύσουμε τον αλγόριθμο της τοπική αναζήτηση και θα δείξουμε ότι είναι $(5 + o(1)) - approximation$. Θεωρούμε ότι ο αλγόριθμος τερματίζει όταν βρούμε τοπικό βέλτιστο, για να βεβαιωθούμε ότι τρέχει σε πολυωνυμικό χρόνο, η εκτέλεση μείωνεται ακόμα κατά έναν παράγοντα $(1 + e)$.

Θεώρημα 1 [2]: Έστω F^* και C^* είναι τα κόστη μετακίνησης και επανατοποθέτησης αντίστοιχα, για την βέλτιστη λύση. Το συνολικό κόστος κάθε τοπικού βέλτιστου με το πολύ $p-$αλλαγές είναι το πολύ: $(3 + O(1/p^{1/3}))F^* + (5 + O(1/p^{1/3}))C^*$.

Έχουμε τους παρακάτω ορισμούς για να συνεχίσουμε την αποδειξη:

- $S = \{s_1, s_2, ..., s_l\}$ για να δηλώσουμε το τοπικό βέλτιστο, όπου η εταιρία i μετακινείται στον τελικό προορισμό $s_i \in S$.
- $O = \{o_1, ..., o_k\}$ για να δηλώσουμε την ολικά βέλτιστη λύση, όπου η εταιρία i μετακινείται στο o_i.
- Για κόμβο $v, s(v)$ είναι η τοποθέτηση του S δίπλα στο v. Ορίζουμε $s^*(v)$ ως την τοποθέτηση στο Ο δίπλα στο v.
- Ορίζουμε $c(i, s_i)\ by\ f_i$ και $c(i, o_i)\ by\ f^*$. Έτσι f_i και f^* είναι τα κόστη μετακίνησης των i στα S και O αντίστοιχα. Επίσης $c(j, s(j))$ στο c_j και $c(j, s^*(j))$ στο $c^*(j)$. Έτσι $c(j)$ και $c^*(j)$ είναι τα κόστη επανατοποθέτησης για τα τοπικά και ολικά βέλτιστα αντίστοιχα.
- Έτσι στο $MFL(S) = \sum_{i \in F} f_i + \sum_{j \in D} c_j$. Έχουμε $D(s) = \{j \in D : s(j) = s\}$ είναι το σύνολο όλων των πελάτων που έχουν ανατεθεί στις τοποθεσίες $s \in S$ και $D^*(o) = \{j \in D : s^*(j) = o\}$.

- Για ένα σύνολο $D(A) = \bigcup_{s \in A} D(s)$, ορίζουμε $D^*(A)$ για $A \subset O$.
- Τέλος ορίζουμε $cap(s)\{o \in O : s(o) = s\}$. Λέμε ότι το s καταλαμβάνει(captures) όλες τις τοποθεσίες στο $cap(s)$.

Λήμμα 1: Για κάθε πελάτη j, έχουμε $c(j, s(s^*(j)))$−$c(j, s(j)) \leq 2c^*$.

Απόδειξη: Έστω $s = s(j), o = s^*(j), s^{'} = s(o)$. Το λήμμα ισχύει εάν $s_r = s = o$. Αλλιώς , $c(j, s_r)$−$c(j, s) \leq c(j, o) + c(o, s_r)$−$c(j, s) \leq c^* + c(o, s)$−$c(j, s) \leq c^* + c(o, j) = 2c^*$, όπου η δεύτερη ανισότητα ισχύει αφού η $s^{'}$ είναι η πιο κοντινή τοποθεσία του ο στο S.

Για να αποδείξουμε το approximation λόγο θα καθορίσουμε ένα σύνολο από κινήσεις τοπικής αναζήτησης για το τοπικό βέλτιστο, γνωρίζοντας ότι καμία από αυτές τις κινήσεις δεν βελτιώνει το κόστος, θα βρούμε ένα όριο για το τοπικό βέλτιστο. Ας θεωρήσουμε τον διμερή γράφο: $\hat{G} = (F \cup S \cup O, \{(s_i, i), (i, o_i), (o_i, s(o_i))\}_{i \in F})$. Ο γράφος $\hat{G}$ αποτελείται από απλά μονοπάτια P και από κύκλους C. Όσο υπάρχει κύκλος στον διμερή γράφο, κάνουμε προσθήκη αυτού του κύκλου στον C, αφαιρούμε όλους τους κόμβους από τον C και κάνουμε αναδρομή στο κομμάτι του $\hat{G}$ που δεν έχουμε ελέγχει ακόμα. Μετά από αυτό το βήμα ενάς κόμβος v που είναι άκυκλος έχει: Ακριβώς μια ακμή που φεύγει από αυτόν αν $v \in S^{'}$. Ακριβώς μία ακμή που έρχεται σε αυτόν και μία που φεύγει από αυτόν αν $v \in F^{'}$ και ακριβώς μια που έρχεται σε αυτόν και περισσότερες από μία που φεύγουν αν $v \in O$. Επαναλαμβανόμενα επιλέγουμε κόμβο $v \in S$ που δεν έχει αμκές που έρχονται σε αυτόν, συμπεριλαμβανομένου του μονοπατιού P ξεκινώντας από το v, κάνοντας διαγραφή όλων των κόμβων του P και αναδρομικά σε όλο τον διμερή γράφο. Έτσι κάθε τριάδα (s_i, i, o_i) είναι ένα μοναδικό μονοπάτι ή κύκλος στο $P \cup C$. Ορίζουμε $center(s)$, $o \in O$ έτσι ώστε (o, s) είναι ακμή στο $P \cup C$ αν το s δεν έχει ακμές που έρχονται προς αυτόν στο $P \cup C$ τότε $center(s) = nil$.

Θα χρησιμοποιήσουμε τα P, C για να ορίσουμε τις εξής αλλαγές:

Για ένα $P = \{s_{i1}, i_1, o_{i1}, ..., s_{ir}, i_r, o_{ir}\}$, ορίζουμε $start(P) = s_{i1}$ και $end(P) = o_{ir}$. Για κάθε $s \in S$, έχουμε $P_c(s) = \{P : end(P) \in cap(s)\}, T(s) = \{start(P) : P \in P_c(s)\} = cap(s)\ center(s)$. Επίσης ισχύει ότι $|P_c(s)| = |T(s)| = |H(s)| = |cap(s)$−$1|, \forall s \in S$ με $cap(s) \geq 1$. Ορίζουμε: $T(A) = \bigcup_{s \in A} T(S), H(A) = \bigcup_{s \in A} H(s), P_c(A) = \bigcup_{s \in A} P_c(s)$. Μία αλλαγή(shift) είναι μία προσθήκη κόμβου s και αφαίρεση κόμβου ο. Το κόστος μεταξύ του F και $S \cup \{o\}\ \{s\}$ με την μετακίνηση κάθε αρχικής τοποθεσίας $i \in Z, i \neq i^{'}$ στο $s(o_i) \in Z$ μετακινώντας το i_r στο $o_{i'}$. Έτσι έχουμε φράγμα στο κόστος μετακίνησης λόγω της παρακάτω εργασίας: $shift(s, o) = \sum_{i \in Z} f_i^* - f_i + \sum_{i \in Z: o_i \neq o} c(o_i, s(o_i)) \leq 2 \sum_{i \in Z} f_i^* - c(o, s(o))$.

Στην τελευταία ανισότητα χρησιμοποιούμε το γεγονός ότι: $c(o_i, s(o_i)) \leq c(o_i, s_i) \leq f^* + f_i, \forall i$. Για ένα $pathP \in P$, λέμε $shift(P)$ ως συντομία για $shift(start(P), end(P))$.

Ανάλυση των αλλαγών μεταξύ των κόμβων: [2] Θα περιγράψουμε τις κινήσεις τοπικής αναζήτησης που χρησιμοποιούνται για την ανάλυση. Ορίζουμε ένα σύνολο από αλλαγές (swaps), έτσι ώστε κάθε $o \in O$ να αλλάζει με το λιγότερο μία, ή το πολύ δύο. Ορίζουμε κάθε μία από τις τοποθεσίες στο S ως ένα από τους παρακάτω τρείς τύπους:

- S_0 : Τοποθεσία $s \in S$ με $cap(s) = 0$.

- S_1 : Τοποθεσίες $s \in S$ S_0 με $D^*(center(s)) \leq t$ ή $|cap(s)| \geq t$.
- S_2 : Τοποθεσίες $s \in S$ με $D^*(center(s)) \geq t$ και $0 \leq |cap(s)| \leq t$.

Επίσης ορίζουμε το $S_3 = S_0 \cup \{s \in S_1 : |cap(s)| \leq t\}$ ώστε: $s \in S_3$ ανν $|cap(s) \leq t|$ και $|D^*(center(s))| \leq t$.

Παρατηρούμε ότι είναι εύκολο να παράγουμε μια ανισότητα για μία τοποθεσία $s \in S_0$: αν κάνουμε διαγραφή s και κάνουμε νέα ανάθεση για κάθε $j \in D(s)$ στο σ(σ$^*(j)$) (το location στο S είναι το πιο κοντινό για να εξυπηρετήσει την $j \in O$). Μπορούμε επίσης να κάνουμε παραγωγή μιας κατάλληλης ανισότητας για την τοποθεσία $s \in S_2$ αφού μπορούμε να κάνουμε αλλαγή το $cap(s)$ και αφαίρεση του $\{s\} \cup T(s)$. Το κόστος που αυξάνεται από αυτή την κίνηση είναι φραγμένο απο: $\sum_{P \in P_c(s)} shift(P)$ και $c(s, center(s))$.

Λήμμα 3.3 [2]: Για κάθε κύκλο $Z \in C$, έχουμε $0 \leq \sum_{i \in Z}(-f_i + f_i^* + c(o_i, s(o_i)))$.

Απόδειξη: Δεδομένου του ταιριάσματος $F \cap Z$ στο $S \cap Z$: κάνουμε νέο ταίριασμα i στο σ(o_i). Το κόστος του καινούριου νέου αυτού ταιριάσματος είναι: $\sum_{i \notin Z} f_i + \sum_{i \in Z} c(i, s(o_i))$ το οποίο πρέπει το λιγότερο να είναι $\sum_i f_i$ αφού το τελευταίο είναι το ελάχιστο κόστος ταιριάσματος του F στο S. Έτσι παίρνουμε: $0 \leq \sum_{i \in Z}(-f_i + c(i, s(o_i))) \leq \sum_{i \in Z}(-f_i + f_i^* + c(o_i, s(o_i)))$.

Λήμμα 3.4 [2]: Έστω $s \in S_2$ και $o = center(s)$, θεωρούμε $swap(X = \{s\} \cup T(s), Y = cap(s))$. Έχουμε:

$$0 \leq MFL((S\ X) \cup Y) - MFL(S) \leq$$
$$\sum_{P \in P_c(s), i \in P} 2f_i^* + \sum_{j \in D^*(o)}(t + 1/t * c_j^* - t - 1/t * c_j) + \sum_{j \in D(\{s\} \cup T(s)), j \notin D_o^*} 2c_j^*.$$

Απόδειξη: Μπορούμε να δούμε αυτή την αλλαγή πολλών τοποθεσιών κάνοντας $swap$ μεταξύ του σημείου έναρξης και του σημείου τερματισμού με την πράξη:

$swap(start(P), end(P))$ για κάθε $P \in P_c(s)$ και $swap(s, o)$ ταυτόχρονα.

Για κάθε $swap(start(P), end(P))$ η αύξηση του κόστους μετακίνησης φράσεται από $shift(P) \leq \sum_{i \in P} 2f_i^*$. Για το $swap(s, o)$ μετακινούμε την εταιρία i, όπου $s = s_i$ στο o, έτσι ώστε η αύξηση του κόστους να είναι το πολύ: $c(s, o) = c(s(o), o) \leq c(s(j), o) \leq c_j + c_j^*$ για κάθε $j \in D^*(o)$. Έτσι αφού $|D^*(o)| \geq t$, έχουμε $c(s, o) \leq \sum_{j \in D^*(o)}(c_j + c_j^*)/t$.

Βάζουμε ένα άνω φράγμα στην αλλαγή του κόστους επανατοποθέτησης, επανατοποθετώντας τους πελάτες στο $D^*(o) \cup D(X)$ με τον παρακάτω τρόπο. Επανατοποθετούμε κάθε $j \in D^*(o)$ στο . Κάθε $j \in D(X)$ $D^*(o)$ είναι τοποθετημένο στο σ$^*(j)$, αν σ$^*(j) \in Y$, αλλιώς στο $s^{'}$ = σ(σ$^{*(j)}$). Ισχύει ότι: $s^{'} \notin X : s^{'} \neq s$ αφού σ$^*(j) \notin cap(s)$ και $s^{'} \notin T(s)$ αφού $\bigcup_{s^{''} \in T(s)} cap(s^{''} = \emptyset$. Έτσι για κάθε πελάτη j η αλλαγή στο κόστος είναι το πολύ $2c_j^*$. Άρα το συνολικό κόστος επανατοποθέτησης αλλάζει κατά: $\sum_{j \in D^*(o)}(c_j^* - c_j) + \sum_{j \in D(X)\ D^*(o)} 2c_j^*$.

Έτσι βγάλαμε ένα άνω φράγμα για την μέγιστη τιμή που μπορεί να πάρει το: $MFL((S \backslash X) \cup Y) - MFL(S)$. Τέλος πρέπει να τονιστεί ότι για την παραπάνω ανάλυση: (1) θεωρούμε μόνο πολυωνυμικό αριθμό από αλλαγές μεταξύ των εταιριών αφού υπάρχουν το πολύ $O(n^p)$ κινήσεις και (2) έχουμε συγκεκριμένο βάρος (το πολύ 1) από τις ανισότητες που προκύπτουν από κάθε τέτοια κίνηση.

5.ΣΥΜΠΕΡΑΣΜΑΤΑ

Στο κεφάλαιο αυτο παρατίθενται συμπεράσματα που απορρέουν από την ανάλυση των προαναφερθέντων προβλημάτων πάνω στην Τοπική Αναζήτηση, που μας βοηθάνε όχι μόνο να κατανοήσουμε την χρήση της μεθόδου της Τοπικής Αναζήτησης, αλλά και να βρούμε ενδεχόμενες καλύτερες λύσεις σε μερικά από τα παραπάνω προβλήματα.

Για κάθε ένα από τα προβλήματα ξεχωριστά διατυπώνουμε τα συμπεράσματα:

- Για το πρόβλημα του Ελάχιστου Καλύμματος Κορυφών πρέπει σε κάθε νέα γειτονιά να μειώνεται συνεχώς το κόστος. Όμως πρέπει πάμτα να προσέχουμε από ποιον κόμβο ξεκινάμε καθώς μπορεί από μία λάθος επιλογή κόμβου να μην βρούμε ποτέ το ολικό βέλτιστο, όπως στον γράφο αστέρι.
- Στα Νευρωνικά Δίκτυα Hopfield παρουσιάζουμε μία γειτονιά αλλαγής κατάστασης, καθώς προσπαθούμε να αλλάξουμε την κατάσταση κάθε κόμβου μέχρι να υπεισέλθουμε σε μία κατάσταση σταθερότητας.
- Εν συνεχεία στον Αλγόριθμο Metropolis παρουσιάζουμε την μετάβαση από μία γειτονιά σε μία άλλη αν και μόνον αν η διαφορά των ενεργειών που έχουν οι δύο καταστάσεις δεν υπερβαίνει ένα όριο. Παρουσιάζεται γενικότερα η ιδέα της μετάβασης από μία γειτονιά σε μία αλλή μόνο αν υπάρχει βελτίωση ενός συγκεκριμένου όρου, που σε αυτή την περίπτωση είναι η διαφορά των ενεργειών.
- Στο πρόβλημα της Μέγιστης Τομής Γράφου Μέσω Της Τοπικής Αναζήτησης, παρουσιάζεται τόσο η περίπτωση όπου οι κόμβοι δεν έχουν βάρη, όσο και η περίπτωση που έχουν βάρη. Για την περίπτωση που ο κάθε κόμβος έχει βάρος παρουσιάζουμε πότε τερματίζει ο αλγόριθμος της τοπικής αναζήτησης, καθώς και την απόδειξη γιατί δεν γίνεται να πάρουμε καλύτερο αποτέλεσμα από $1/2 - approximation$.
- Στη Τοπική Αναζήτηση Για Μεγιστοποιήση Submodular Συναρτήσεων, εξηγούμε τι είναι μία submodular συνάρτηση και πώς μπορούμε να την βελτιώσουμε προσθέτοντας ή αφαιρώντας έναν κόμβο. Παρουσιάζονται επίσης και οι αποδείξεις όπου η λύση του αλγορίθμου της τοπικής αναζήτησης είναι $1/3 - approximation$ που γίνεται $1/2 - approximation$ εάν η f γίνει συμμετρική.
- Ωστόσο επείδη σε μερικά πιο πολύπλοκα προβλήματα πρέπει να οριστούν και πιο πολύπλοκες γειτονιές, παρουσιάζουμε μεθόδους που πιο αποδοτικές γειτονιές μας δίνουν καλύτερα αποτελέσματα στον αλγόριθμο της τοπικής αναζήτησης, όπως η $K - Lheuristics$.
- Στο πρόβλημα της Ταξινόμησης Μέσω Της Τοπικής Αναζήτησης για πρώτη φορά παρουσιάζεται η περίπτωση που ένας κόμβος πρέπει να επιλέξει ανάμεσα k διαφορετικές τιλοφορήσεις. Έτσι γίνεται προσπάθεια για μία καλώς ορισμένη γειτονιά, καθώς και για έναν αλγόριθμο τοπικής αναζήτησης που μπορεί να μας δώσει μια καλή λύση.
- Στο πρόβλημα του Πλανόδιου Πωλητή παρουσιάζεται η $2 - opt$ και η $k - opt$ γειτονιά και ένας αλγόριθμος τοπικής αναζήτησης για το πρόβλημα αυτό.

- Στα Boolean Προβλήματα Ικανοποίησης SAT Προβλήματα, εξηγείται τι είναι ένα SAT πρόβλημα. Επειδή πολλές φορές είναι δύσκολο να λυθεί ένα τέτοιο πρόβλημα παρουσιάζεται ο αλγόριθμος της $WalkSat$ που αναλύεται.
- Στην Καλύτερη Δυναμική Απόκρισης Συστήματα Ισορροπίας Nash, έχουμε γειτονιές αλλαγής κατάστασης(state-flipping neighborhood) και παρου-σιάζεται ο αλγόριθμος τοπικής αναζήτησης για το βέλτιστο αποτέλεσμα.
- Στο Πρόβλημα Χωροτοποθέτησης παρουσίαζεται η συνάρτηση η οποία πρέπει να ελαχιστοποιηθεί για να έχουμε τη βέλτιστη λύση. Ωστόσο αυτή η σύναρτηση αποτελείται από δύο διαφορετικά κόστη, το κόστος εταιρίας και το κόστος ανάθεσης. Έτσι πρέπει για τη μετάβαση σε μια νέα γειτονιά να έχουμε μείωση και στα δύο. Σίγουρα αποτελεί αντικείμενο μελέτης το πώς θα μπορούσαμε να ορίσουμε μια καλύτερη γειτονιά για το πρόβλημα αυτό, που θα οδηγήσει σε καλύτερα αποτελέσματα.
- Τέλος στη Κινητή Χωροτοποθέτηση για πρώτη φορά στη δημιουργία γειτονιάς χρησιμοποιούμε τα μονοπάτια επανατοποθετήσεων, όπου είναι μια σειρά από προορισμούς για να γίνουν νέα ταιριάσματα με τις εταιρίες. Τελικά κάνουμε μια ανάλυση του αλγορίθμου τοπικής αναζήτησης που μας οδηγεί σε μί0α $5 - approximation$ λύση. Ωστόσο αξίζει να σημειωθεί ότι η λύση μπορεί να βελτιωθεί ακόμα περισσότερο.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος Όρος	**Ελληνικός Όρος**
Neighborhood	Γειτονιά
Approximation	Προσέγγιση
Local Search	Τοπική Αναζήτηση
Vertex Cover	Ελάχιστο Κάλυμμα Κορυφών
Classification	Κατηγοριοποίηση
Traveling Salesman Problem	Το Πρόβλημα του Πλανόδιου Πωλητή
Facility Location Problem	Πρόβλημα Χωροτοποθέτησης
Mobile Facility Location Problem	Κινητή Χωροτοποθέτηση

ΑΝΑΦΟΡΕΣ

[1] Afrati F.N. (2006). *On Approximation Algorithms for Data Mining Applications*. In: Bampis E., Jansen K., Kenyon C. (eds) Efficient Approximation and Online Algorithms. Lecture Notes in Computer Science, vol 3484. Springer, Berlin, Heidelberg.

[2] Ahmadian S. , Friggstad Z. , Swamy C. *Local-Search based Approximation Algorithms for Mobile Facility Location Problems*. SODA '13 Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms, Pages 1607-1621. (2013).

[3] Anupam Gupta A. , Tangwongsan K. *Simpler Analyses of Local Search Algorithms for Facility Location*. Computer Science Department Data Structures and Algorithms, (2008).

[4] Arya V. , Garg N. , Khandekar R. , Meyerson A. , Munagala K. , Pandit V. *Local Search Heuristics for k-Median And Facility Location Problems*. SIAM Journal on Computing archive Volume 33 Issue 3, (2004). pp 544 - 562.

[5] Bouhmala N. *A Variable Neighborhood Walksat-Based Algorithm for MAX-SAT Problems*. The Scientific World Journal Volume 2014 (2014), Article ID 798323, 11 pages.

[6] Cai S. *Faster Implementation for WalkSAT*. Elsevier Journals (2013).

[7] Chekuri C. , Ene A. *Approximation Algorithms for Submodular Multiway Partition*. Foundations of Computer Science (FOCS) 2011 IEEE 52nd Annual Symposium on. pp. 807-816, (2011).

[8] Kleinberg J., Tardos E. *Algorithm Design*. Cornell University, 1^{st} Edition, (2006), pp 661-690.

[9] Lu H. , Ravi R. *The Power of Local Optimization: Approximation Algorithms for Maximum-Leaf Spanning Tree*. In Proceedings, Thirtieth Annual Allerton Conference on Communication, Control and Computing (1996).

[10] Nagarajan C. , Williamson D. *Offline and Online Facility Leasing. Discrete Optimization*. Volume 10, Issue 4, November 2013, Pages 361-370.

[11] Seitz S., Alava M., Orponen P. (2005) *Threshold Behaviour of WalkSAT and Focused Metropolis Search on Random 3-Satisfiability*. In: Bacchus F., Walsh T. (eds) Theory and Applications of Satisfiability Testing. SAT 2005. Lecture Notes in Computer Science, vol 3569. Springer, Berlin, Heidelberg

[12] Wiliamson D., Shmoys D. *The Design of Approximation Algorithms*. Cambridge University Press (2011). pp 35-54, 105-107.