ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Pathlet Learning for Compressing Trajectories

Κωνσταντίνος Δ. Μπετχαβάς

Επιβλέπων: **Δημήτριος Γουνόπουλος**, Καθηγητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2019

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Pathlet Learning for Compressing Trajectories

Κωνσταντίνος Δ. Μπετχαβάς
Α.Μ.: 1115201300116

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Δημήτριος Γουνόπουλος**, Καθηγητής

ΠΕΡΙΛΗΨΗ

Η ευρεία ανάπτυξη συσκευών GPS έχει δημιουργήσει γιγαντιαία σύνολα δεδομένων για τις τροχιές πεζών και οχημάτων. Αυτά τα σύνολα δεδομένων παρέχουν μεγάλες ευκαιρίες για την ενίσχυση της κατανόησης των προτύπων μετακίνησης ανθρώπων, ωφελώντας έτσι πολλές εφαρμογές που κυμαίνονται από υπηρεσίες βάσει τοποθεσίας (LBS) έως σχεδιασμός του συστήματος μεταφοράς. Σε αυτό το έργο, εισάγουμε την έννοια του Pathlet με σκοπό τη συμπίεση τροχιών. Λαμβάνοντας μια συλλογή τροχιών σε έναν οδικό χάρτη ως είσοδο, επιδιώκουμε να υπολογίσουμε ένα συμπαγές λεξικό με Pathlets έτσι ώστε ο αριθμός των διαδρομών που χρησιμοποιούνται για να αντιπροσωπεύσουν κάθε τροχιά να ελαχιστοποιείται.

Στην παρακάτω εργασία μελετάμε τους διάφορους αλγόριθμους εύρεσης αυτών των λεξικών και μέσω συμπερασμάτων να καλυτερεύσουμε την λύση μας.

Η πτυχιακή αυτή στηρίζεται στο έργο τών **Chen Chen, Hao Su, Qixing Juang, Lin Zhang, Leonidas Guibas** με την ονομασία **Pathlet Learning for Compressing and Planning Trajectories**.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Συμπίεση Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: συμπίεση, Pathlet, Pathlet Learning, Pathlet Dictionary, τροχιά, integer programming, dynamic programming, GPS trajectories

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

1. ΕΙΣΑΓΩΓΗ

Η ραγδαία διάδοση συσκευών GPS έχει δημιουργήσει τεράστια σύνολα δεδομένων τροχιών.Η συμπίεση αυτών των δεδομένων έχει μεγάλη σημασία. Επιπλέον η σημασιολογία των τροχιών εξηγεί την ύπαρξει κοινών δομών/διαδρομών, επομένως εκεί συνδέεται άμεσα η συμπίεση τους. Αυτή η κατανόηση της τροχιάς υψηλότερου επιπέδου μπορεί να ωφελήσει μια ποικιλία εφαρμογών που κυμαίνονται από τη μελέτη των διαδρομών της μετανάστευσης του πληθυσμού, τα μοντέλα κυκλοφορίας οχημάτων και την κατάσταση οδικών δικτύων πόλεων.

Ο άνθρωπος έχει μεγάλη πιθανότητα να επαναλάβει παρόμοια μοτίβα διαδρομής,επομένως οι τροχιές περιλαμβάνουν υψηλό βαθμό χωρικής και χρονικής τακτικότητας.Αυτό το γεγονός χρησιμοποιούμε για την υλοποίηση του Pathlet Learning. Έτσι προσπαθούμε να εξάγουμε κοινά μονοπάτια που έχουν σημασιολογικό νόημα,το οποίο καλούμε Pathlet Dictionary.

Το Pathlet Learning μπορεί να υλοποιηθεί ώς πρόβλημα γραμμικού προγραμματισμού,με αντικειμενική λειτουργία την ελαχιστοποίησει του μεγέθους του Pathlet Dictionary καθώς και του αριθμού των διαδρομών που χρησιμοποιούνται για την ανακατασκευή κάθε τροχιάς. Όμως επειδή η λύση δεν είναι κλιμακωτή για μεγάλα πλήθος δεδομένων, εισάγουμε μια αποσυνδεδεμένη προσέγγιση, η οποία βελτιστοποιεί ένα χαμηλότερο όριο της αντικειμενικής λειτουργίας. Εν τέλει υλοποιούμε το πρόβλημα μας ως πρόβλημα δυναμικού προγραμματισμού, ώστε να είναι κλιμακωτό και ταχύτερο.

Τα δεδομένα με τα οποία εξετάζουμε τους αλγορίθμους έχουν δωθεί από ένα csv αρχείο που περιλαμβάνει map-matched τροχιές και διαδρομές με κοινά μονοπάτια ώστε να έχει λογική η εφαρμογή του Pathlet Learning. Αλλιώς χρησιμοποιούμε δεδομένα τυχαία ή μη που γνωρίζουμε ότι περιλαμβάνουν και κοινές διαδρομές. Βέβαια οι αλγόριθμοι έχουν δημιουργηθεί έτσι ώστε αν δεν γίνεται συμπίεση των δεδομένων, έστω και λίγη, τότε να προειδοποιεί και να μην κρατάει το αποτέλεσμα, πράγμα το οποίο είναι δυνατόν αν δεν έχουμε κοινές υποδιαδρομές.

2. ΔΗΛΩΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

Παρακάτω θα ορίσουμε το πρόβημα του Pathlet Learning απο μια συλλογή τροχιών(trajectories).

Αναφερόμαστε στο σύνολο των τροχιών ως κεφαλαίο Τ.Αυτό το σύνολο μπορούμε να το πάρουμε με την χαρτογράφησει(map-matching) GPS συντεταγμένων στον οδικό χάρτη.Καλούμε μιά διαδρομή/μονοπάτι μικρό p αν είναι είναι υποδιαδρομή ενός ή περισσοτέρων τροχιών του Τ.Δηλώνουμε το σύνολο όλων των δυνατών υπομονοπατιών μιάς τροχιάς t ως P(t).Γνωρίζουμε ότι αν μια ακμή έχει |t| ακμές, τότε $|P(t)|=|t|\cdot(|t|+1)/2$.Ένα Pathlet Dictionary είναι ένα σύνολο απο Pathlets (P), το οποίο με την κατάλληλη αλληλουχία απο Pathlets μπορεί να ανακατασκευάσει τροχίες.Το πλήθος των Pathlets σε ένα Pathlet Dictionary αναφέρεται ώς |P|.Προφανώς το μεγαλύτερο Pathlet Dictionary ωρίζεται ώς $\bar{P}$, και εισούται με την ένωση όλων των πιθανών υποδιαδρομών για όλες τις τροχίες.

Μιά τροχία μπορεί να ανακατασκευαστεί απο ένα Pathlet Dictionary,εφόσον μια αλληλουχία απο Pathlets το επιτρέπουν.Ανάλογα με τα Pathlets μπορεί να υπάρχουν παραπάνω απο μία αλληλουχία ανακατασκευείς της τροχιάς.Εμάς μας ενδιαφέρει η αλληλουχία με το μικρότερο μέγεθος και πλήθος απο Pathlets. Θέλουμε να κάνουμε εξαγωγή Pathlets με περιορισμένη βελτιστοποίηση.Δηλαδή για την εφαρμογή του Pathlet Learning, θέλουμε να ελαχιστοποιήσουμε το μέγεθος του Pathlet Dictionary, καθώς και το πλήθος των Pathlets που χρησιμοποιούνται για την ανακατασκευεί των τροχιών.Λόγω τις ανάγκης περιορισμών(constraints) λύνουμε πρόβλημα βελτιστοποίησης, τα οποία συνήθως λύνονται ως προβλήματα γραμμικού προγραμματισμου.

Η σχέση μεταξύ μεγέθους του Pathlet Dictionary και του πλήθους απο Pathlets για ανακατασκευεί τροχιών εξαρτάται απο μια μεταβλητή λ δικής μας επιλογής, το οποίο ευνοεί μία απο τις δύο επιλογές περισσότερο, γι’αυτό η επιλογή του μας δίνει διαφορετικό αποτέλεσμα.

3. ΑΛΓΟΡΙΘΜΟΙ ΥΛΟΠΟΙΗΣΗΣ

Αρχικά ορίζουμε και λύνουμε το πρόβλημα μας ως πρόβλημα γραμμικού/ακέραιου προγραμματισμού με περιορισμούς (integer programming with constraints).Αργότερα το τροποποιούμε ώστε να βρούμε μια αποτελεσματική επίλυση με χρήση δυναμικού προγραμματισμού.

3.1 ΥΛΟΠΟΙΗΣΗ ΜΕ ΓΡΑΜΜΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

3.1.1 ΜΕΡΟΣ Α

Κάθε Pathlet $p \in \bar{P}$ σχετίζεται με έναν δυαδικό δείκτη $x_p \in \{1,2\}$, όπου $x_p = 1$ αν $p \in P$, και $x_p = 0$ αλλιώς. Επομένως το τελικό Pathlet Dictionary θα περιλαμβάνει τα Pathlets $p \in \bar{P}$ με $x_p = 1$.

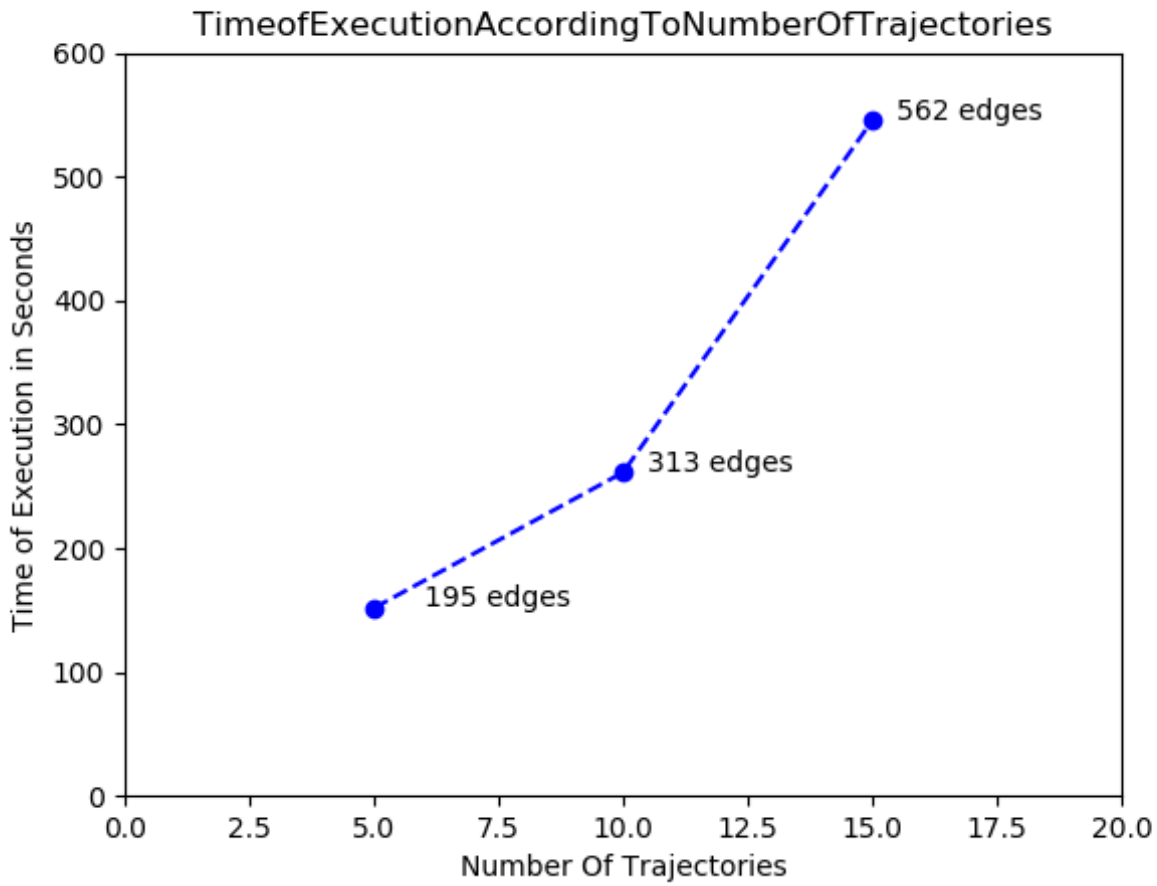
Για να διατυπώσουμε τον περιορισμό επανακατασκευής, μια τροχιά μπορεί να επανακατασκευεί απο μία αλληλουχία από Pathlets μεσα στο P(t), συσχετίζουμε κάθε Pathlet $p \in P(t)$ με έναν δυαδικό δείκτη $x_{t,p} \in \{1,2\}$,όπου $x_{t,p} = 1$ αν το Pathlet p χρησιμοποιείται στην επανακατασκευεί της τροχιάς και $x_p = 0$ αλλιώς.

Αφού θέσαμε τους δυαδικούς δείκτες και ορίσαμε όλες τις μεταβλητές μας το πρόβλημα βελτιστοποίησης Pathlet Learning μπορεί να λυθεί με το παρακάτω ακέραιο πρόγραμμα:

$$
\begin{aligned}
\min \quad & \sum_{p \in \overline{\mathcal{P}}} x_p + \lambda \sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}(t)} x_{t,p} \\
s.t. \quad & x_{t,p} \leq x_p, \quad \forall p \in \mathcal{P}(t), t \in \mathcal{T} \\
& \sum_{p \in t, e \in p} x_{t,p} = 1, \quad \forall e \in t, t \in \mathcal{T} \\
& x_p \in \{0,1\}, \quad \forall p \in \overline{\mathcal{P}} \\
& x_{t,p} \in \{0,1\}, \quad \forall p \in \mathcal{P}(t), t \in \mathcal{T}.
\end{aligned}
$$

Όμως η επίλυση με τον παραπάνω τρόπο δεν είναι αποτελεσματικός γιατί η λύση αυτή δεν είναι κλιμακωτή και περιλαμβάνει μεγάλο πλήθος μεταβλητών για επίλυση με γραμμικό προγραμματισμό.Στη συνέχεια θα δωθούν αποτελέσματα πάνω στη χρήση του αλγορίθμου αυτου ώστε να αιτιολογηθεί η αναποτελεσματικότητα του.

Τρέξαμε τον αλγόριθμο για 5,10 και 15 τροχιές(trajectories),με 195,313 και 562 άκρες αντίστοιχα. Χρησιμοποιήσαμε πολλοί λίγες τροχιές επειδή ο αλγόριθμος δεν είναι καθόλου πρακτικός.Ο χρόνος εκτέλεσης βρίσκεται στο παρακάτω διάγραμμα.



Σχήμα 1: Χρόνος Εκτέλεσης Ανάλογα με Πλήθος Τροχιών

Βλέποντας το διάγραμμα είναι προφανές ότι ο αλγόριθμος αυτός χωρίς μετατροπές δεν είναι πρακτικός για μεγάλα σύνολα δεδομένων, αφού δεν είναι κλιμακωτός και η πολυπλοκότητα του είναι εκθετική. Πρέπει να μειώσουμε το πλήθος των δεδομένων του γραμμικού προβλήματος και τις σχέσεις μεταξύ των μεταβλητών για διαφορετικές τροχιές για να μειώσουμε τον χρόνο εκτέλεσης τις λύσης.Αυτές οι μετατροπές εξηγούνται στο επόμενο κεφάλαιο.

3.1.2 ΜΕΡΟΣ Β

Για να αντιμετωπιστεί το πρόβλημα τις κλιμάκωσης και τις πολυπλοκότητας, αναπτύσουμε ένα περισσότερο κλιμακωτό πρόβλημα όπου η βασική ιδέα είναι η επίλυση μιάς τροποποιημένης αντικειμενικής λειτουργίας(objective function) η οποία μπορεί να αποσυνδεθεί σε ανεξάρτητους αντικειμενικούς όρους για κάθε τροχιά. Συγκεκριμένα εξετάζουμε ένα κατώτερο όριο στην αρχική αντικειμενική λειτουργία, το οποίο προκύπτει παρακάτω:

$$
\begin{aligned}
&\sum_{p\in\overline{\mathcal{P}}} x_p + \lambda \sum_{t\in\mathcal{T}} \sum_{p\in\mathcal{P}(t)} x_{t,p} \\
=&\sum_{p\in\overline{\mathcal{P}}} \sum_{t\in\mathcal{T}(p)} \frac{x_p}{|\mathcal{T}(p)|} + \lambda \sum_{t\in\mathcal{T}} \sum_{p\in\mathcal{P}(t)} x_{t,p} \\
\geq&\sum_{p\in\overline{\mathcal{P}}} \sum_{t\in\mathcal{T}(p)} \frac{x_{t,p}}{|\mathcal{T}(p)|} + \lambda \sum_{t\in\mathcal{T}} \sum_{p\in\mathcal{P}(t)} x_{t,p} \\
=&\sum_{t\in\mathcal{T}} \sum_{p\in\mathcal{P}(t)} (\lambda + \frac{1}{|\mathcal{T}(p)|}) x_{t,p} := f,
\end{aligned}
$$

όπου $\mathcal{T}(p) = \{t | t \in \mathcal{T}, p \in \mathcal{P}(t)\}$.

Επομένως χρησιμοποιώντας το παραπάπω στον πρώτο μας αλγόριθμο, έχουμε ένα αποσυνδεμένο πρόβλημα βελτιστοποίσης για κάθε τροχιά $t\in T$:

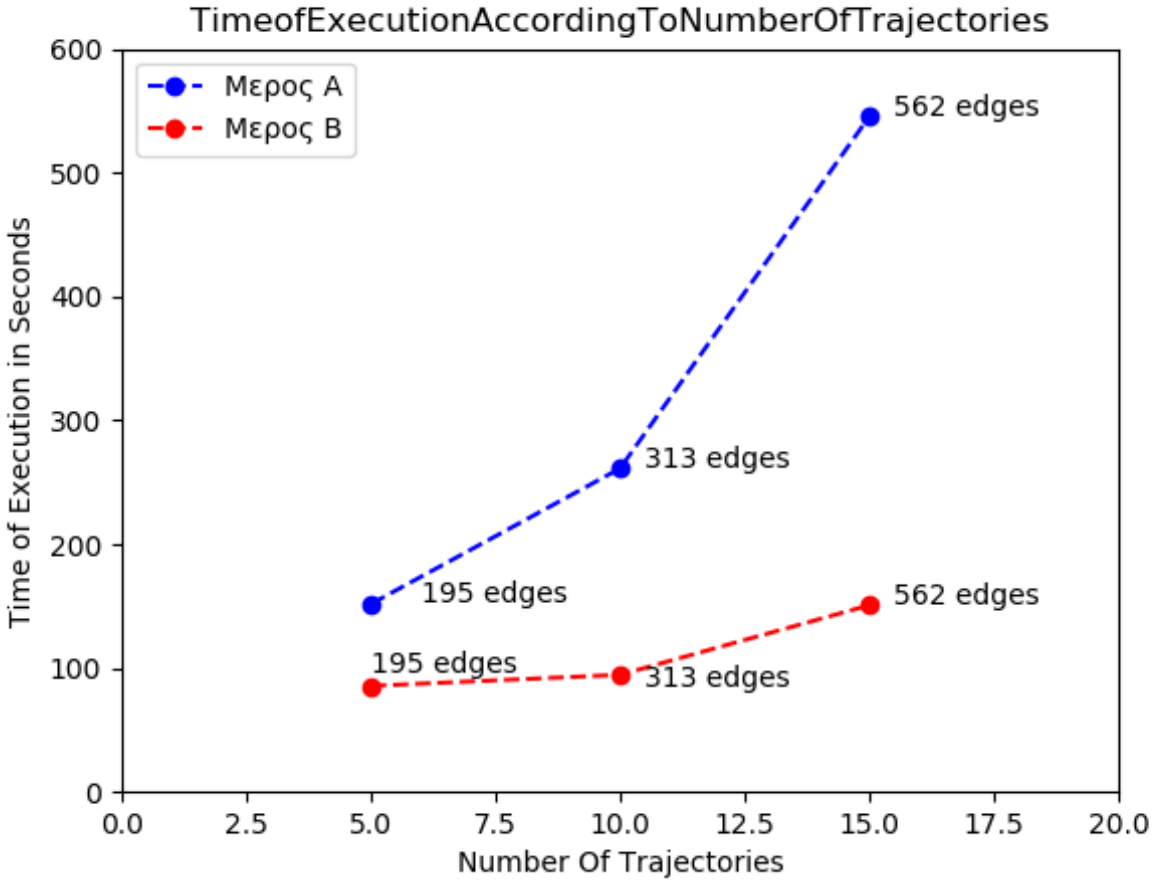
$$
\begin{aligned}
&\min_{x_{t,p}\in\{0,1\}} \sum_{p\in\mathcal{P}(t)} (\lambda + \frac{1}{|\mathcal{T}(p)|}) x_{t,p} \\
&s.t. \sum_{e\in p, p\in\mathcal{P}(t)} x_{t,p} = 1, \quad , \forall e \in t.
\end{aligned}
$$

Μετά την βελτιστοποίηση των $x_{t,p}$ για όλες τις τροχιές, η βέλτιστη τιμή για το $x_p = \max_{t \in \mathcal{T}(p)} x_{t,p}$.

Στη συνέχεια θα χρησιμοποιήσουμε τις ίδιες τροχιές που χρησιμοποιήσαμε στο αλγόριθμο του Μερους Α με σκοπό να δείξουμε ότι καταφέραμε με τις μετατροπές αυτές να μειώσουμε την πολυπλοκότητα και να κλιμακώσουμε το πρόβλημα μας ώστε να έχουμε αποτέλεσμα σε πολύ καλύτερο χρόνο.

Πρακτικά όμως τη λύση του Μερους Β μπορεί να είναι χειρότερη του Μερους Α, εφόσον δεν εξετάζει τις σχέσεις των αποτελεσμάτων κάθε τροχίας με όλες τις υπόλοιπες, αλλά είναι πολύ πιο πρακτική η επίλυση του προβλήματος.

Το αντίστοιχο διάγραμμα, με τα δεδομένα που χρησιμοποιήσαμε στο Μέρος Α, είναι το παρακάτω.



Σχήμα 2: Χρόνος Εκτέλεσης Ανάλογα με Πλήθος Τροχιών σε σχέση με το ΜΕΡΟΣ Α

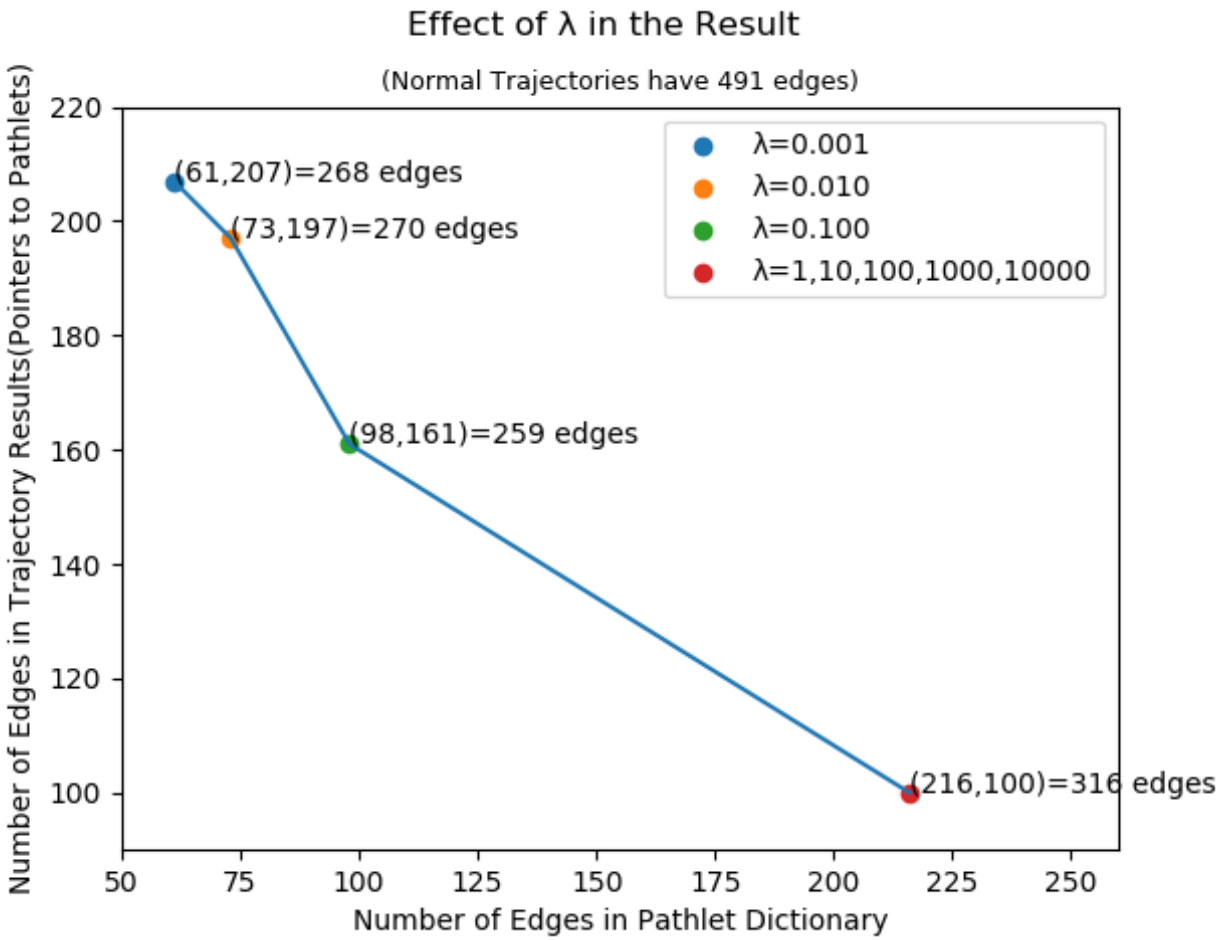
Παρατηρούμε ότι ο χρόνος εκτέλεσης είναι πολύ καλύτερος αλλά η κλιμάκωση παρ'όλο που είναι καλύτερη δεν μας επιτρέπει την χρήση του αλγορίθμου αυτού για Pathlet

Learning σε μεγάλης κλίμακας δεδομένα λόγω της πολυπλοκότητας σε μεταβλητές στο γραμμικό πρόγραμμα.

Στην συνέχεια αξίζει να δούμε την επηροή του λ στο αποτέλεσμα μας, δηλαδή στην σχέση μεταξύ μήκους των τροχιών(δεικτών σε Pathlets) και του πλήθους των Pathlets στο Pathlet Dictionary, κατά την εφαρμογή του αλγορίθμου μας γραμμικού προγραμματισμού.

Για να έχουμε καλά δεδομένα, ώστε να δούμε τις αλλαγές σύμφωνα με το λ, πρέπει να έχουμε κοινές υποδιαδρομές, ώστε αναλόγως να δράσει με τον ένα ή τον άλλον τρόπο σύμφωνα με την τιμή του. Έτσι αντί για map-matched τροχιές χρησιμοποίησα την παρακάτω λογική για τροχιές.

Χρησιμοποιώντας την βιβλιοθήκη random πήρα μια λίστα με 100 τροχιές που έχουν τυχαία τιμές απο [0,6] και με μήκος τροχιάς απο [3,7]. Έπειτα τις τιμές αυτές για κάθε τροχιά της ταξινόμησα αριθμητικά. Έτσι οι λίστα μας περιλαμβάνει τροχιές που θα έχουν κοινές υποδιαδρομές. Χρησιμοποιώντας τον αλγόριθμο μας προκύπτει το παρακάτω γράφημα όσον αφορά το λ:



Σχήμα 3: Διαφορά των Ακμών μεταξύ Trajectory Results και Pathlet Dictionary ανάλογα με το λ

Απο το διάγραμμα παρατηρούμε ότι ανάλογα με το λ είτε έχουμε μεγαλύτερο Pathlet Dictionary ή μεγαλύτερο Trajectory Results. Η ενδιάμεση λύση του λ = 0.1 μας δίνει το καλύτερο αποτέλεσμα για συμπίεση των 491 ακμών πριν την εφαρμογη του αλγορίθμου σε 259 ακμές, το οποίο περιλαμβάνει 98 ακμές στο Pathlet Dictionary και 161 ακμές στο Trajectory Results. Η συμπίεση είναι του μεγέθους 47.25%.

Επομένως η επιλογή του λ μπορεί να επηρεάσει πολύ το αποτέλεσμα μας, και πρέπει να πάρουμε το κατάλληλο για την δουλειά που το θέλουμε. Αν αυτή είναι η καλύτερη συμπίεση τότε αυτό με τις λιγότερες ακμές ως σύνολο, αλλιώς αν θέλουμε να πάρουμε μεγαλύτερες υποδιαδρομές στα Pathlets για μελέτη τους εξετάζουμε με μικρές τιμές για το λ.

Στο επόμενο κεφάλαιο θα επιλύσουμε το πρόβλημα μας με χρήση δυναμικού προγραμματισμού ώστε να μειωθεί το τεράστιο πλήθος των μεταβλητών που χρησιμοποιούνται για την επίλυση με γραμμικό προγραμματισμό, ιδιαίτερα γιατί η αύξηση αυτών με τον παρών αλγόριθμο είναι πολλαπλάσια όσο αυξάνεται το πλήθος των δεδομένων μας.

3.2 ΥΛΟΠΟΙΗΣΗ ΜΕ ΔΥΝΑΜΙΚΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Εν τέλει, χρησιμοποιώντας την μελέτη μας απο τους προηγούμενους αλγορίθμους δημιουργούμε έναν δυναμικό αλγόριθμο που θα έχει την πιο αποτελεσματική χρήση.

3.2.1 ΜΕΡΟΣ Γ

Συγκεκριμένα, δηλώνοντας την βέλτιστη αντικειμενική τιμή του προηγούμενου αλγόριθμου,για κάθε υποδιαδρομη ${}^{U}i{}^{U}j$ μιας τροχιάς t = ${}^{U}1{}^{U}n$, ώς f*(${}^{U}i{}^{U}j$) , μπορούμε να την υπολογίσουμε αναδρομικά με τον παρακάτω τρόπο:

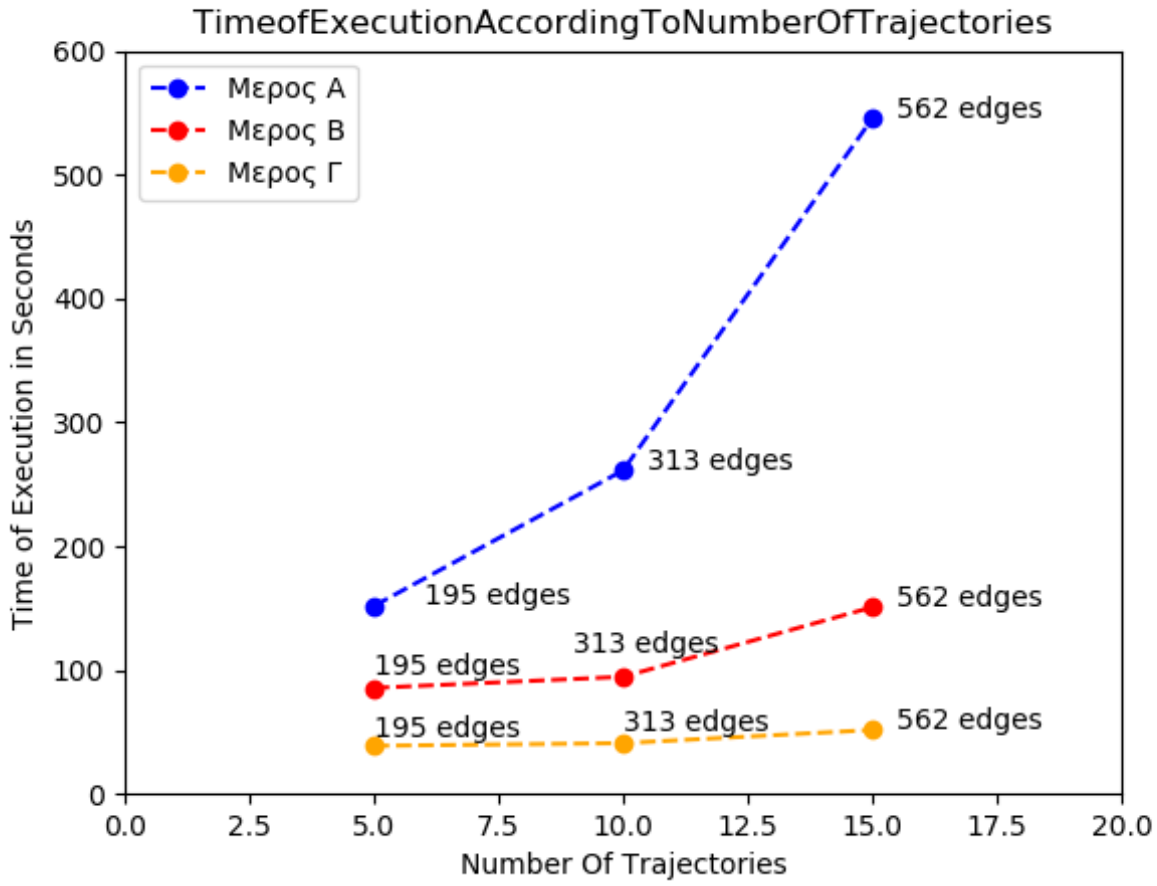
$$f^\star(v_i\hat{v}_j) = \begin{cases} \min\limits_{k\in\{i+1,\cdots,j-1\}} (f^\star(v_i\hat{v}_k) + f^\star(v_k\hat{v}_j)) & i < j-1 \\ \lambda + 1/|\mathcal{T}(v_i v_{i+1})| & \text{otherwise.} \end{cases}$$

Χρησιμοποιούμε τις βέλτιστες τιμές για κάθε υποδιαδρομή ώστε να λάβουμε την καλύτερη αποσύνδεση κάθε τροχιάς t. Η λογική εύρεσης καλύτερης αποσύνδεσης εξηγείται εν συνεχεία.

Ξεκινώντας απο το σύνολο τις τροχιάς συγκρίνουμε αναδρομικά ξεκινώντας απο τις μεγαλύτερες στις μικρότερες και αν βρούμε κάποια να έχει μικρότερη απο άλλη τότε την διαφοροποιούμε και την εξετάζουμε μόνη της. Ενώ αυτές που δεν την αφορούν τις ενώνουμε και τις εξετάζουμε διαφορετικά. Αν έχουν όλα τα ίδια μήκοι υποδιαδρομών ίδια τιμή τότε δεν χρειάζεται να διαιρέσουμε αυτή την υποδιαδρομή και την παίρνουμε ολόκληρη ως μέλος της αποσύνδεσης. Συγκεκριμένα ο κώδικας του αναφέρεται στο κεφάλαιο Προγραμματιστικός Κώδικας σελίδα 27.

Παρακάτω θα προσθέσω το αποτέλεσμα της δυναμικής υλοποίησης του παραδείγματος με τις πολύ λίγες τροχιές που χρησιμοποίησα στα προηγούμενα 2 κεφάλαια. Ο σκοπός είναι να δούμε την καλυτέρευση μας σε χρόνο και σε κλιμάκωση.

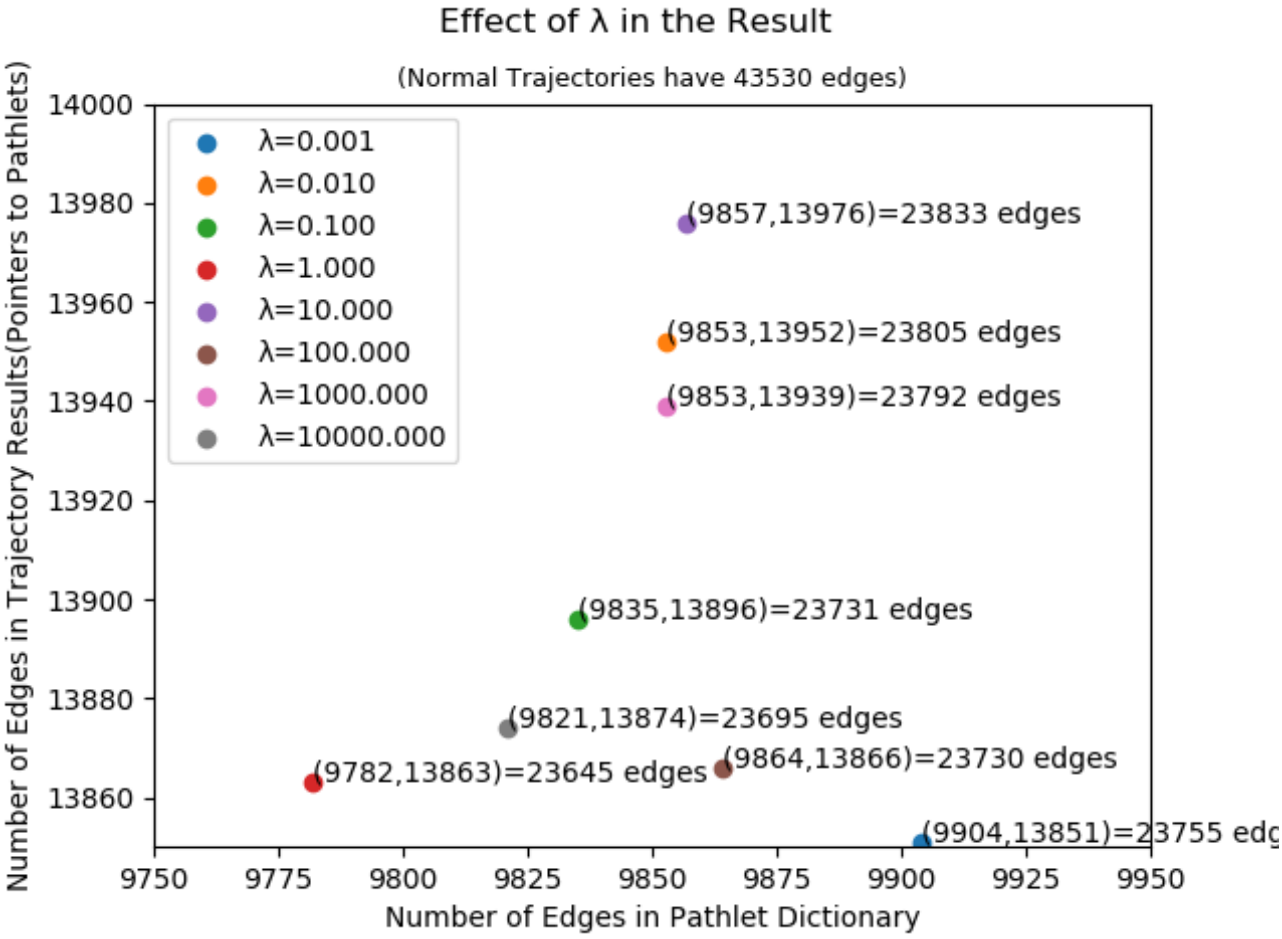
Πριν συνεχίσουμε έχω προσθέσει παραλληλία στον κώδικα του Μέρους Γ, αλλά για την σύγκριση με τα άλλα δύο Μέρη θα χρησιμοποιήσω έναν πυρήνα. Παρόλο που η χρήση παραλληλίας με ένα πυρήνα θα είναι ελαφρώς πιό αργό απ'ότι αν δεν έβαζα καθόλου παραλληλία.



Σχήμα 4: Χρόνος Εκτέλεσης Ανάλογα με Πλήθος Τροχιών σε σχέση με τα ΜΕΡΟΙ Α & Β

Από το διάγραμμα είναι προφανές ότι η ταχύτητα εκτέλεσης και η κλιμάκωση στο πλήθος των δεδομένων είναι πολύ καλύτερες απο τους αλγορίθμους γραμμικού προγραμματισμού. Το μόνο αρνητικό, ενώ είναι ο μόνος αλγόριθμος που έχει πρακτική λογική αφού μπορεί να εφαρμοστεί σε μεγάλα σύνολα δεδομένων, είναι ότι δεν δίνει πάντα την βέλτιστη λύση. Αυτο οφείλετε στο ότι οι προηγούμενη δυο αλγόριθμοι δίνουν μεγαλύτερη σημασία στην λεπτομέρεια του αποτελέσματος, στην σωστή διάταξη και λειτουργία των περιορισμών για την λήψη λύσης. Ενώ ο αλγόριθμος δυναμικού προγραμμάτισμου για λόγους ταχύτητας και πολυπλοκότητας είναι πιό ανοιχτός. Τα αποτελέσματα που μας δίνει όμως είναι αρκετά ικανοποιητικά.

Παρακάτω έχουμε το διάγραμμα επηροής του λ στην υλοποίηση με δυναμικό προγραμματισμό. Χρησιμοποιήθηκαν 1000 τροχιές με 43530 ακμές στο σύνολο τους.

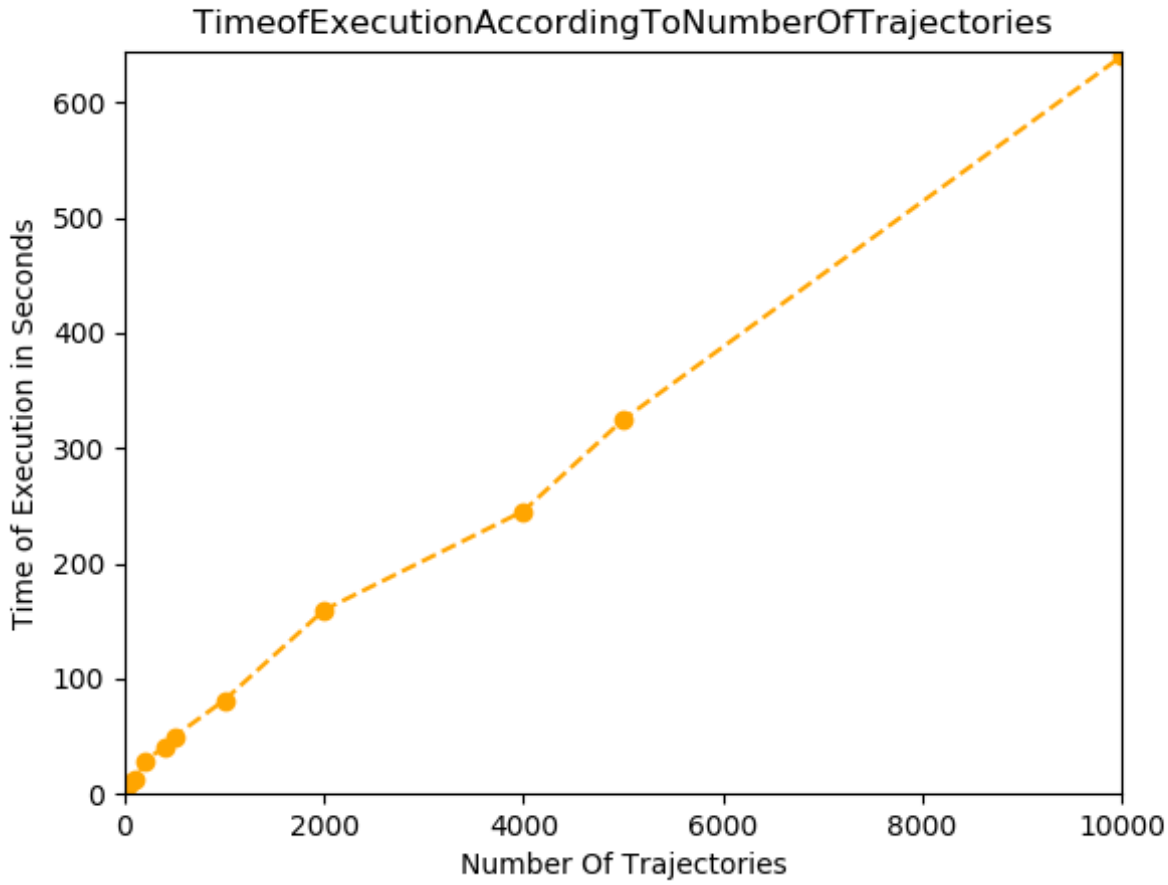


Σχήμα 5: Διαφορά των Ακμών μεταξύ Trajectory Results και Pathlet Dictionary ανάλογα με το λ

Από το διάγραμμα αυτό παρατηρούμε ότι το λ επηρεάζει το αποτέλεσμα μας και ανάλογα την τιμή του μπορούμε να βρούμε καλύτερο αποτέλεσμα συμπίεσης. Αλλά δεν παρατηρείτε κάποια λογική κατάταξη ανάλογα με την αριθμητική ταξινόμηση του όπως στους αλγορίθμου γραμμικού προγραμματισμού. Έτσι για λόγους ταχύτητας είναι

αποδεκτό να χρησιμοποιήσουμε μια προεπιλεγμένη(default) τιμή λ=1, όπου και στο συγκεκριμένο παράδειγμα παρέχει το βέλτιστο αποτέλεσμα με συμπίεση 45.68%.

Εν συνεχεία αξίζει να δούμε την χρόνο εκτέλεσης σε πολλαπλά μεγέθη συνόλων δεδομένων ώστε να βεβαιωθούμε ότι το πρόβλημα μας κλιμακώνει χρονικά.Η εκτέλεση θα γίνει με εφαρμογή της παραλληλίας(το σύστημα μου έχει 4 πυρήνες) και λ=1.

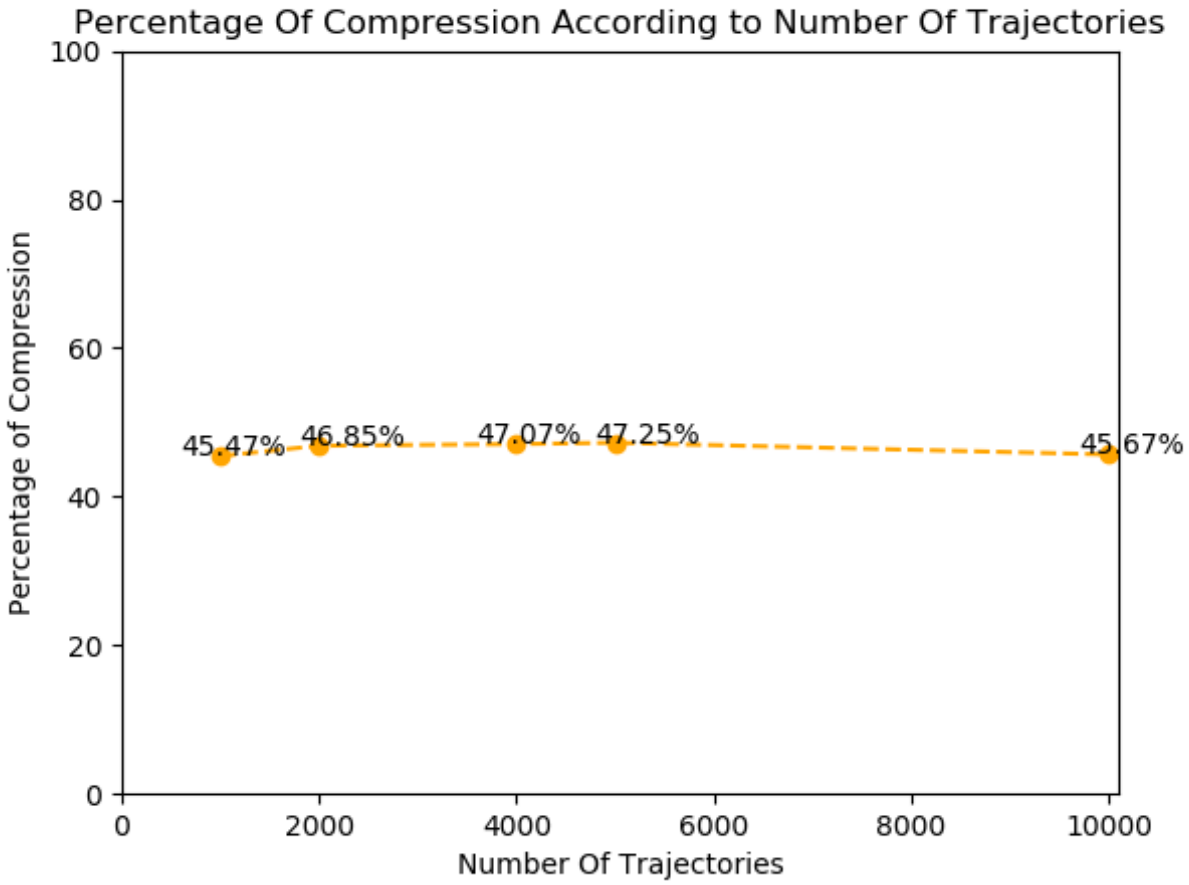


Σχήμα 6: Χρόνος Εκτέλεσης Ανάλογα με Πλήθος Τροχιών για Ελεγχο Κλιμάκωσης

Από το διάγραμμα μπορούμε να παρατηρήσουμε ότι ο χρόνος εκτέλεσης του Pathlet Learning είναι ανάλογος με το μέγεθος του συνόλου των δεδομένων γραμμικά. Δηλαδή η υλοποίηση μας με δυναμικό προγραμματισμό είναι κλιμακωτή.

Η εκτέλεση με 10000 τροχιές ολοκληρώνετε σε 10.66 λεπτά που είναι ο διπλάσιος σχεδόν με αυτόν των 5000 τροχιών, 5.41 λεπτά, ο οποίος είναι πολύ καλός χρόνος εκτέλεσης.

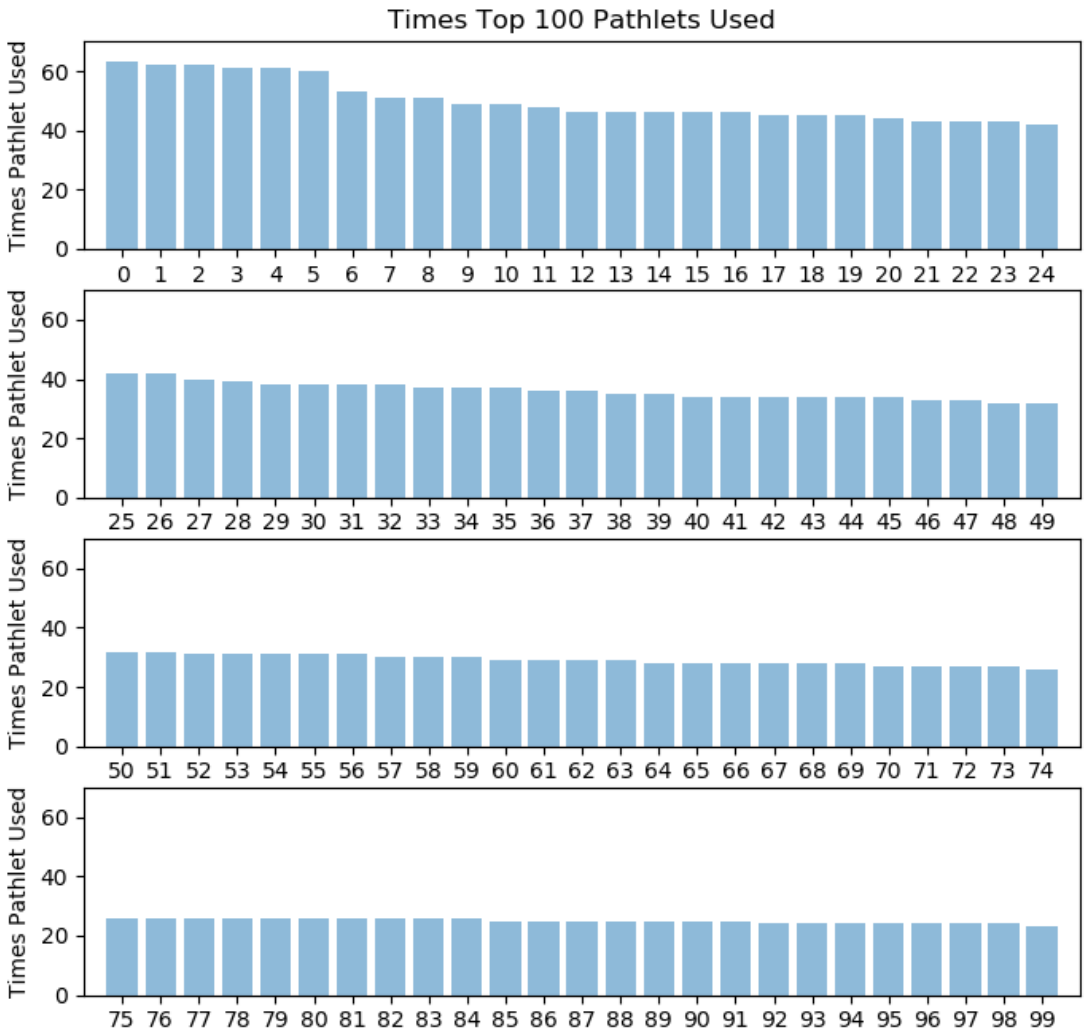
Παρακάτω προσθέτουμε διάγραμμα με το ποσοστό της συμπίεσης για 1000,2000,4000,5000 και 10000 τροχιές.



Σχήμα 7: Ποσοστό συμπίεσης ανάλογα με το πλήθος των τροχιών που χρησιμοποιήθηκαν

Βλέποντας το διάγραμμα παρατηρούμε ότι η συμπίεση στα δεδομένα μας μένει σχεδόν ίδια. Βέβαια το ποσοστό συμπίεσης απαιτεί και την σωστή επιλογή δεδομένων, αφού αν δεν περιλαμβάνουν καλό ποσοστό απο κοινές υποδιαδρομές δεν θα έχουμε καλό αποτέλεσμα. Στο διάγραμμα το καλύτερο ποσοστό το έχει στις 5000 τροχιές.

Τέλος αξίζει να παρατηρήσουμε τη χρήση των Pathlets που βρήκαμε με την εφαρμογή του αλγόριθμου και αν αυτά είναι αποτελεσματικά. Έτσι στην επόμενη σελίδα προσθέτουμε διάγραμμα χρήσης των 100 καλύτερων Pathlets για ανακατασκευή τροχιών σε παράδειγμα που αφορά την υλοποίηση του Pathlet Learning σε πλήθος 1000 τροχιών, ώστε να δούμε αν μπορέσαμε να βρούμε αρκετές κοινές υποδιαδρομές ώστε να δικαιολογείτε η χρήση του αλγορίθμου μας και ποιες είναι οι πιο πολυσύχναστες.



Σχήμα 8: Πλήθος Φορών Χρήσης των 100 πιο πολυσύχναστων Pathlets

Από τα αποτελέσματα βλέπουμε ότι έχουμε κοινές υποδιαδρομές μεταξύ τροχιών που φτάνουν μέχρι και 63 φορές. Έτσι παραδείγματος χάριν στο πιο πολυσύχναστο αυτό Pathlet, ενώ χωρίς την χρήση του αλγόριθμου η μνήμη που καταλαμβάνουν οι ακμές του στις τροχιές είναι 63*2 = 126 ακέραιοι, με την χρήση του Pathlet Learning καταλαμβάνει 63+2 = 65 ακέραιοι. Αφού τα Pathlets λειτουργούν σαν δείκτες στις τροχιές για ανακατασκευή.Ένα απο τα καλύτερα Pathlets είναι το Pathlet 48 το οποίο μείωσε την χρήση μνήμης από 11*32 = 352 ακέραιους σε 11+32 = 43 ακέραιους.

Στην συνέχεια αναφέρονται τα παραπάνω Pathlets με τα edges τους και τον ακριβή αριθμό που χρησιμοποιούνται για ανακατασκευή τροχιών.

Pathlet 0 [(818, 9129)] : 63 times used

Pathlet 1 [(817,)] : 62 times used

Pathlet 2 [(815, 816)] : 62 times used

Pathlet 3 [(9132, 9133, 9134)] : 61 times used

Pathlet 4 [(9130, 9131)] : 61 times used

Pathlet 5 [(5347, 5348)] : 60 times used

Pathlet 6 [(49152,)] : 53 times used

Pathlet 7 [(5340, 5341)] : 51 times used

Pathlet 8 [(49153, 48946)] : 51 times used

Pathlet 9 [(5346,)] : 49 times used

Pathlet 10 [(43369, 43370)] : 49 times used

Pathlet 11 [(5342, 5343)] : 48 times used

Pathlet 12 [(3300, 3301, 3302)] : 46 times used

Pathlet 13 [(47373, 47374)] : 46 times used

Pathlet 14 [(5334, 5335)] : 46 times used

Pathlet 15 [(5338, 5339)] : 46 times used

Pathlet 16 [(76157, 66682)] : 46 times used

Pathlet 17 [(9135, 48772)] : 45 times used

Pathlet 18 [(47727, 47728)] : 45 times used

Pathlet 19 [(47690, 54425)] : 45 times used

Pathlet 20 [(47815,)] : 44 times used

Pathlet 21 [(48773, 48774)] : 43 times used

Pathlet 22 [(54425, 10909)] : 43 times used

Pathlet 23 [(45382, 45383)] : 43 times used

Pathlet 24 [(63863, 40742)] : 42 times used

Pathlet 25 [(76153, 76154)] : 42 times used

Pathlet 26 [(45380, 45381)] : 42 times used

Pathlet 27 [(55446, 55447, 58677)] : 40 times used

Pathlet 28 [(48415, 48416)] : 39 times used

Pathlet 29 [(48161,)] : 38 times used

Pathlet 30 [(4058, 4059, 4060)] : 38 times used

Pathlet 31 [(63884, 63885)] : 38 times used

Pathlet 32 [(63881,)] : 38 times used

Pathlet 33 [(40743, 40744, 43389, 43390, 43391)] : 37 times used

Pathlet 34 [(47375, 5357)] : 37 times used

Pathlet 35 [(47812, 47813, 55445)] : 37 times used

Pathlet 36 [(1565, 1566)] : 36 times used

Pathlet 37 [(63887, 76950, 3259, 76944)] : 36 times used

Pathlet 38 [(58679, 58678)] : 35 times used

Pathlet 39 [(48775, 48776, 48777, 48778)] : 35 times used

Pathlet 40 [(566, 567, 63522)] : 34 times used

Pathlet 41 [(3036, 3035)] : 34 times used

Pathlet 42 [(70879, 70878)] : 34 times used

Pathlet 43 [(76126, 76133, 76127)] : 34 times used

Pathlet 44 [(67939, 67940)] : 34 times used

Pathlet 45 [(48202, 68254)] : 34 times used

Pathlet 46 [(3596, 3595, 3594, 3593)] : 33 times used

Pathlet 47 [(48779,)] : 33 times used

Pathlet 48 [(3592, 48068, 48067, 48069, 74893, 70984, 70983, 70982, 70981, 70980, 70979)] : 32 times used

Pathlet 49 [(10910,)] : 32 times used

Pathlet 50 [(658, 67946)] : 32 times used

Pathlet 51 [(520, 521)] : 32 times used

Pathlet 52 [(70855, 70854)] : 31 times used

Pathlet 53 [(54982,)] : 31 times used

Pathlet 54 [(809, 47943)] : 31 times used

Pathlet 55 [(76134, 45377)] : 31 times used

Pathlet 56 [(58680,)] : 31 times used

Pathlet 57 [(47294, 70856)] : 30 times used

Pathlet 58 [(63952, 63951)] : 30 times used

Pathlet 59 [(518, 519)] : 30 times used

Pathlet 60 [(48787, 48786, 77009, 48752)] : 29 times used

Pathlet 61 [(67955, 67956)] : 29 times used

Pathlet 62 [(805, 806)] : 29 times used

Pathlet 63 [(67920, 67921)] : 29 times used

Pathlet 64 [(47729,)] : 28 times used

Pathlet 65 [(731, 53079)] : 28 times used

Pathlet 66 [(70878, 70879)] : 28 times used

Pathlet 67 [(67958, 67959, 67960, 67961)] : 28 times used

Pathlet 68 [(67922, 67923, 67924, 67925, 67926)] : 28 times used

Pathlet 69 [(63862,)] : 28 times used

Pathlet 70 [(48789, 48788)] : 27 times used

Pathlet 71 [(4058,)] : 27 times used

Pathlet 72 [(807, 808)] : 27 times used

Pathlet 73 [(67962, 67963)] : 27 times used

Pathlet 74 [(63507, 48790)] : 26 times used

Pathlet 75 [(48168, 48169, 48170)] : 26 times used

Pathlet 76 [(14249, 14250)] : 26 times used

Pathlet 77 [(43392, 66746)] : 26 times used

Pathlet 78 [(76147,)] : 26 times used

Pathlet 79 [(47479, 47480)] : 26 times used

Pathlet 80 [(16564, 47478)] : 26 times used

Pathlet 81 [(67964,)] : 26 times used

Pathlet 82 [(67927, 67928, 67929, 67930, 67931, 67932)] : 26 times used

Pathlet 83 [(67941, 76148)] : 26 times used

Pathlet 84 [(51, 52, 53)] : 26 times used

Pathlet 85 [(48751, 48750, 76988, 54968)] : 25 times used

Pathlet 86 [(28826, 28827, 28828)] : 25 times used

Pathlet 87 [(70628, 10894, 4525)] : 25 times used

Pathlet 88 [(63863, 63862, 61312)] : 25 times used

Pathlet 89 [(729, 730)] : 25 times used

Pathlet 90 [(4060, 4059)] : 25 times used

Pathlet 91 [(74392, 74393, 74394, 74395)] : 25 times used

Pathlet 92 [(3396, 3397, 3398, 63926)] : 24 times used

Pathlet 93 [(564, 565)] : 24 times used

Pathlet 94 [(54601, 28823, 28824, 28825)] : 24 times used

Pathlet 95 [(47726,)] : 24 times used

Pathlet 96 [(66747, 40744, 40743, 40742)] : 24 times used

Pathlet 97 [(67934, 67935)] : 24 times used

Pathlet 98 [(76943, 72657, 72658, 47940, 50)] : 24 times used

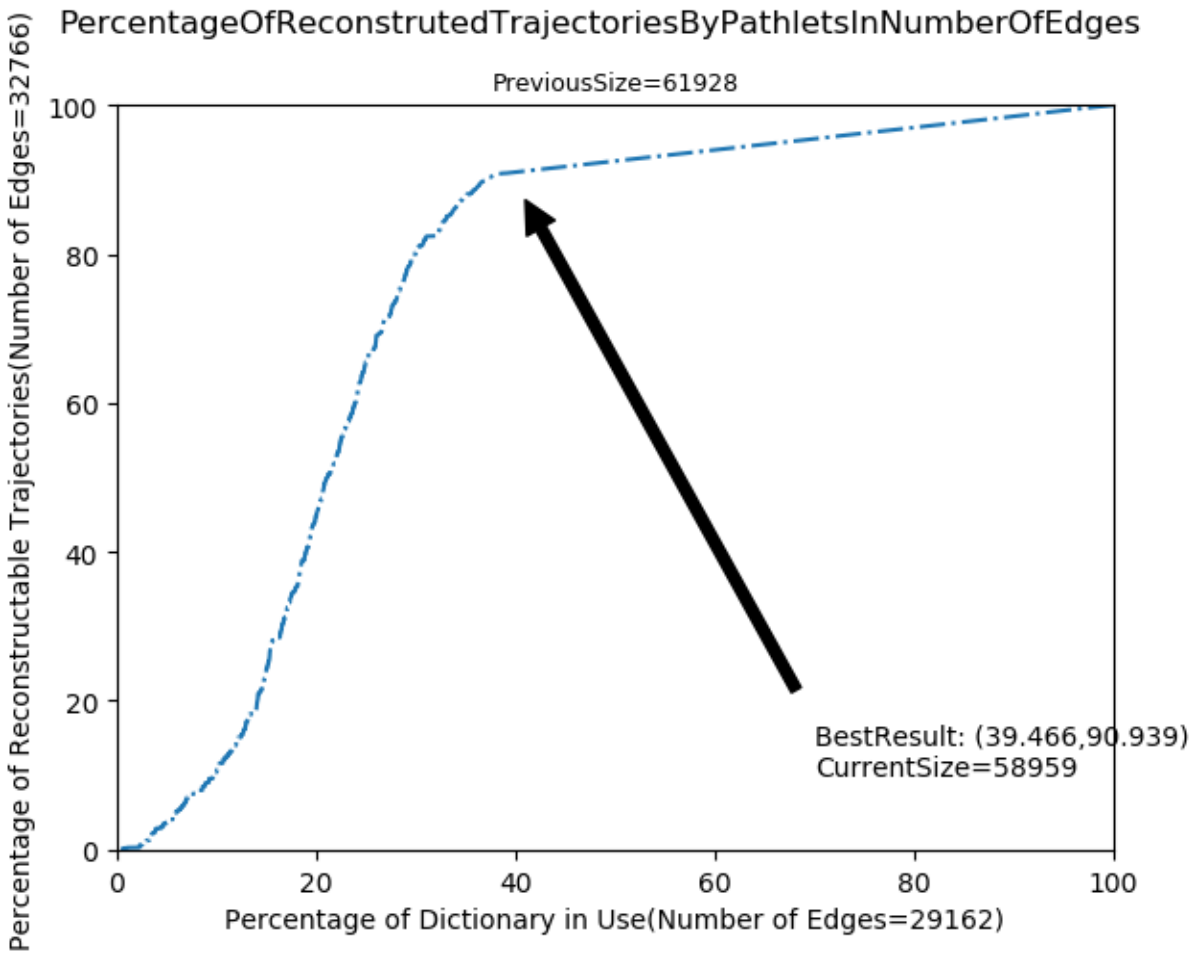
Pathlet 99 [(47190, 47191)] : 23 times used

Αυτές οι διαδρομές είναι οι πιο πολυσύχναστες.Έτσι μπορούν να εξέταστούν και για άλλους λόγους, όπως αξιοποίηση της πληροφορίας για λόγους διαφήμισης, έλεγχος κίνησης κ.α, εφόσον κάποιος θέλει να τις εξετάσει για αυτούς τους λόγους.

3.3. ΑΞΙΟΠΟΙΗΣΗ ΣΧΕΣΗΣ ΔΕΔΟΜΕΝΩΝ ΤΟΥ PATHLET DICTIONARY

Γνωρίζουμε ότι το Pathlet Dictionary μπορεί να ανακατασκευάσει όλες τις τροχιές των δεδομένων μας. Όμως ανάλογα με τα δεδομένα που του δώσαμε μπορεί να μην μοιράζονται υποδιαδρομές με άλλες. Έτσι μεγαλώνει το τελικό μας αποτέλεσμα, γι'αυτό τον λόγο μπορούμε να εξετάσουμε το ποσοστό ανακατασκευής των τροχιών αν αφαιρέσουμε Pathlets, τα οποία δεν χρησιμοποιούνται σε πολλές. Και αν το αποτέλεσμα μας είναι καλύτερο να αφαιρέσουμε αυτές τις τροχιές και να έχουμε κανονικές τροχιές, οι οποίες εξαιρέθηκαν αφού δεν μοιράζονται αρκετές κοινές υποδιαδρομές, μαζί με αυτές που προέκυψαν με Pathlet Learning στην λύση μας.

Αυτός ο έλεγχος γίνεται σύμφωνα με το μέγεθος σε μεταβλητές του αποτελέσματος και όχι απλά στο πλήθος των Pathlets και των TrajectoryResults, δηλαδή των δεικτών σε Pathlets για κάθε τροχιά. Μπορούμε να δούμε την χρησιμότητα του στο παρακάτω διάγραμμα στο οποίο χρησιμοποιήθηκαν και τυχαίες τροχιές με σκοπό να μην μοιράζονται πολλές υποδιαδρομές μεταξύ τους και με τις άλλες τροχιές.Πριν το Pathlet Learning οι ακμές των τροχιών είναι 106932.



Σχήμα 9: Ποσοστό ορθά επανακατασκευασμένων τροχιών ανάλογα με τα χρησιμοποιούμενα Pathlets σύμφωνα με το πλήθος των ακμών και επίδειξη του καλύτερο ποσοστού

Παρατηρούμε ότι το 39.46% των Pathlets μπορούνε να ανακατασκευάσουνε το 90.93% των τροχιών. Έτσι αν κάνουμε την παραπάνω βελτιστοποίηση μειώνουμε το μέγεθος των ακμών του συνολικού αποτελέσματος απο 61928 σε 58959, δηλαδή βελτίωση 4.79% σε σχέση με το αποτέλεσμα του Pathlet Learning. Στο σύνολο τους ο Pathlet Learning και ο PercentageOrderOptimizer έκαναν συμπίεση των δεδομένων σε ποσοστό 44.86% , ενώ ο Pathlet Learning μόνος του 42.08%.

4. ΣΥΜΠΕΡΑΣΜΑΤΑ

Η ραγδαία αύξηση των βάσεων δεδομένων επιβάλει την χρήση αλγορίθμων συμπίεσης. Συγκεκριμένα για την συμπίεση διαδρομών/τροχιών, παραδείγματος χάριν στον χάρτη google maps, μπορεί να χρησιμοποιηθεί ο αλγόριθμος Pathlet Learning που αναφέρθηκε σε αυτή την εργασία.

Ειδικότερα η χρήση του προτείνετε για δεδομένα που περιλαμβάνουν κοινές υποδιαδρομές, για παράδειγμα τροχιές που αντιστοιχούν σε μία πόλη. Ανάλογα με τις κοινές υποδιαδρομές μπορούμε να πετύχουμε τεράστια συμπίεση.

Η χρήση για τεράστιο πλήθος τροχιών επιβάλει την χρήση του αλγορίθμου σε υλοποίηση δυναμικού προγραμματισμού επειδή είναι χρονικά και χωρικά κλιμακωτός.

Τα Pathlets που βρήκαμε με την χρήση του αλγορίθμου μπορούν να εξεταστούν και για άλλους λόγους αφού αποτελούν πολυσύχναστες διαδρομές. Διάφοροι τέτοιοι λόγοι είναι η εξέταση τους για λόγους διαφήμισης, π.χ. ταμπέλες διαφημίσεων, ή για τοπικές προτιμήσεις απο εφαρμογές τουρισμού ή χαρτών GPS, ή για μελέτη κίνησης στους δρόμους και για άλλους λόγους.

Ορισμένες τροχιές που δεν βοηθάνε και δεν προκαλούν καλή συμπίεση μπορούν να ξεχωριστούν με την χρήση του αλγορίθμου PercentageOrderOptimizer, που αναφέρεται στο κεφάλαιο Αξιοποίηση Σχέσης Δεδομένων του Pathlet Dictionary. Έτσι αυτές οι τροχιές μπορούν είτε να προστεθούν σε άλλο πλήθος δεδομένων τροχιών που πιθανότατα να περιλαμβάνει κοινές υποδιαδρομές με αυτές ή να τις έχουμε ώς κανονικές τροχιές αντί σε μορφή που χρησιμοποιει ο Pathlet Learning.

5. ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Εκτός του ότι μπορούνε να χρησιμοποιηθούνε τα Pathlet για διάφορους λόγους μελέτης, υπάρχουν και διάφοροι πιθανοί τρόποι βελτιστοποίησης του αλγορίθμου Pathlet Learning.

Όσον αφορά την υλοποίηση του αλγορίθμου σε δυναμικό προγραμματισμό μπορεί να επαναξεταστεί ώστε πιθανώς να βρεθεί ένας αλγόριθμος καλύτερης ανακατασκευής των τροχιών χρησιμοποιώντας της βέλτιστες τιμές των υποδιαδρομών. Ίσως μπορεί να γίνει βελτιστοποίηση της παραλληλίας ώστε να περιλαμβάνει μεγαλύτερο πλήθος της εκτέλεσης του αλγορίθμου, αφού από θεωρία της παραλληλίας, για να γίνει ένα πρόγραμμα πραγματικά παράλληλο πρέπει να είναι παράλληλο στο σύνολό του.

Επίσης μπορούμε να προσθέσουμε παραλληλία στον PercentageOrderOptimizer με τον ίδιο τρόπο που εφαρμόσαμε στην υλοποίηση με δυναμικό προγραμματισμό ώστε να βελτιστοποιηθεί αισθητά ο χρόνος εκτέλεσης. Καθώς και πιθανότατα ανακατασκευή του σηριακού ώστε να μην υπάρχουν άσκοπες πράξεις.

Τέλος μπορούμε μετά την εκτέλεση του PercentageOrderOptimizer να επαναχρησιμοποιήσουμε τις τροχιές που αφαιρέθηκαν απο τη λογική του Pathlet Learning, με άλλες τροχιές που πιθανότατα να περιλαμβάνουν περισσότερες κοινές υποδιαδρομές, ώστε να προκαλέσουμε συμπίεση και σε αυτές. Αυτό σημαίνει την εφαρμογή του Pathlet Learning σε ποσοστά του πλήθους των δεδομένων μας ώστε να μπορέσουμε να παραλληλήσουμε τον αλγόριθμο σε πολλαπλά υπολογιστικά συστήματα μέσω δικτύου και όχι μόνο σε σχέση με τους πυρήνες που υπάρχουν σε ένα σύστημα/υπολογιστή, ώστε να βελτιωθεί πολύ ο χρόνος εκτέλεσης σε τεράστια μεγέθη δεδομένων.

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟΣ ΚΩΔΙΚΑΣ

Παρακάτω έχουμε σε Python την υλοποίηση κάποιων βασικών και σχετικών με τον αλγόριθμο Pathlet Learning μέρη του κώδικα αντίστοιχα και με την σειρά των κεφαλαίων μας.Στην σελίδα https://github.com/FliPPyPwN/PaphletLearningForTrajCompression, το οποίο είναι public repository στο github, υπάρχουν όλα τα αρχεία που χρησιμοποιήθηκαν για να πάρουμε τα αποτελέσματα εκτός των δεδομένων μας.

Στην υλοποίηση με γραμμικό προγραμματισμό χρησιμοποιήται η παρακάτω συνάρτηση για υπολογισμό όλων των πιθανών υποδιαδρομών που κάποιες απο αυτές θα περιλαμβάνουν το Pathlet Dictionary.

```
def FindAllPossiblePathlets(self, trajectories) :
    AllPossiblePathlets = []
    TpIndexesNeededForPathletLearning = []
    SubIndexesNeededForPathletLearning = []

    seen = dict()

    trajCounter = 0
    for traj in trajectories :
        trajIndexTemp = []
        for i in range(len(traj)) :
            trajIndexTemp.append([])

        for i in range(len(traj) + 1):

            for j in range(i + 1, len(traj) + 1):

                sub = tuple(traj[i:j])
                if (sub not in seen) :
                    for k in range(i,j) :
                        trajIndexTemp[k].append(len(AllPossiblePathlets))

                    TpIndexesNeededForPathletLearning.append(1)
                    seen[sub] = len(AllPossiblePathlets)
                    AllPossiblePathlets.append(sub)
                else :
                    index = seen[sub]
                    TpIndexesNeededForPathletLearning[index] =
                                TpIndexesNeededForPathletLearning[index] + 1

                    for k in range(i,j) :
                        trajIndexTemp[k].append(index)

        trajCounter = trajCounter + 1
        SubIndexesNeededForPathletLearning.append(trajIndexTemp)

    return AllPossiblePathlets,
    TpIndexesNeededForPathletLearning,SubIndexesNeededForPathletLearning
```

Ο κώδικας στο Μέρος Α δεν περιλαμβάνει το TpIndexesNeededForPathletLearning αλλά τα υπόλοιπα είναι παρόμοια όπου επιστρέφει όλες τις υποδιαδρομές των τροχιών και

πληροφορίες, όπως οι σχέσεις μεταξύ των υποδιαδρομών και των τροχιών, που χρησιμοποιούνται για την υλοποίηση του αλγορίθμου με γραμμικό προγραμματισμό.

Στην συνέχεια έχουμε τον κώδικα που αφορά τον υπολογισμό των αποτελεσμάτων που ψάχνουμε με την λογική του Μέρους Α.

```
def SolvePathletLearningLinearly(self,trajectories,IndexesForConstraints) :
    problem = LpProblem("PathletLearning", LpMinimize)

    Xp = LpVariable.dicts("Xp", list(range(len(self.Pathlets))), cat="Binary")

    Xtp = []
    #-------------------
    #Constraints
    print("Adding Constraints\n1st Set Of Constraints")
    for i in range(len(trajectories)) :
        Xtp.append(LpVariable.dicts("Xtp"+str(i), list(range(len(self.Pathlets))),
          cat="Binary"))

        for j in range(len(Xtp[i])) :
            problem += Xtp[i][j] <= Xp[j]


    print("2nd Set Of Constraints")
    for i in range(len(trajectories)) :

        for j in range(len(trajectories[i])) :
            temp = lpSum(Xtp[i][k] for k in IndexesForConstraints[i][j])

            problem += temp == 1

    #-------------------
    #objective function
    print("Adding Objective Function")
    temp = lpSum(Xp[i]  for i in range(len(Xp)))

    temp += lpSum(self.lamda*Xtp[i][j] for j in range(len(Xtp[i])) for i in range(len(Xtp)))

    problem += temp
    print("SolvingStarts!")
    problem.solve() #!!!

    PathletResults = []
    for i in range(len(self.Pathlets)) :
        PathletResults.append(Xp[i].varValue)

    TrajsResults = []
    for i in range(len(trajectories)) :
        trajResult = []
        for j in range(len(self.Pathlets)) :
            trajResult.append(Xtp[i][j].varValue)
        TrajsResults.append(trajResult)

    return PathletResults,TrajsResults
```

Για το Μέρος Β χρησιμοποιούμε τον παρακάτω.

```
def SolvePathletLearningScalableLinearly(self,trajIndex,
TpIndexesNeededForPathletLearning, SubIndexesNeededForPathletLearning) :
        problem = LpProblem("PathletLearning", LpMinimize)

        #-------------------
        #Constraints
        Xtp = LpVariable.dicts("Xtp"+str(trajIndex), list(range(len(self.Pathlets))),
        cat="Binary")

        PathletsUsing = set()
        for indexes in SubIndexesNeededForPathletLearning[trajIndex] :
            PathletsUsing.update(indexes)

            temp = lpSum(Xtp[index]  for index in indexes)

            problem += temp == 1

        temp = lpSum((self.lamda + 1/TpIndexesNeededForPathletLearning[i])*Xtp[i]
             for i in   PathletsUsing)

        problem += temp
        problem.solve() #!!!

        trajResults = []

        for i in range(len(self.Pathlets)) :
            if i not in PathletsUsing :
                trajResults.append(0)
            else :
                trajResults.append(Xtp[i].varValue)

            if trajResults[i] == 1 :
                self.Xp[i] = 1

        return trajResults
```

Αυτές οι δυό συναρτήσεις υλοποιούν την βασική λειτουργία του Pathlet Learning για του αλγορίθμους υλοποίησης με γραμμικό προγραμματισμό.

Τα παρακάτω χρησιμοποιούνται και στις δυο υλοποιήσεις με γραμμικό προγραμματισμό. Τέλος χρησιμοποιούμε την συνάρτηση MinimizePathletLearningResults με σκοπό να το αφαιρέσουμε τις άσκοπες πληροφορίες και να το φέρουμε στην τελική μορφή συμπίεσης των δεδομένων.Η συνάρτηση γράφεται στην επόμενη σελίδα.

```
def MinimizePathletLearningResults(self,PathletResults,trajectories) :
        #---------------------------------------------
        #Parakatw meiwnw to megethos twn dedomenwn mas
        indexes = []
        for i in range(len(PathletResults)) :
            if PathletResults[i] == 0 :
                indexes.append(i)

        self.Pathlets = np.array(self.Pathlets)
        PathletResults = np.array(PathletResults)
        self.TrajsResults = np.array(self.TrajsResults)

        self.Pathlets = list(np.delete(self.Pathlets, indexes))
        PathletResults = np.delete(PathletResults, indexes)
        self.TrajsResults = np.delete(self.TrajsResults, indexes, 1)

        NewTrajsResults = []
        for i in range(len(self.TrajsResults)) :
            NewTraj = np.where(self.TrajsResults[i] == 1)[0]

            #Swsth topothethsh twn index sta Pathlets
            if not(self.ReturnRealTraj(NewTraj) == trajectories[i]) :
                for PossibleOrder in list(permutations(NewTraj, len(NewTraj))) :
                    if self.ReturnRealTraj(PossibleOrder) == trajectories[i] :
                        NewTraj = PossibleOrder
                        break
                NewTrajsResults.append(list(NewTraj))
            else :
                NewTrajsResults.append(NewTraj.tolist())

        self.TrajsResults = NewTrajsResults
```

Έτσι έχουμε το τελικό μας αποτέλεσμα και χρησιμοποιούμε απλό τρόπο για επιστροφή της αρχικής λίστας τροχιών μας με την χρήση του ReturnAllTrajectoriesInAList.

```
def ReturnRealTraj(self,TrajResult) :
        RealTraj = []
        for i in range(len(TrajResult)) :
            index = TrajResult[i]

            RealTraj = RealTraj + list(self.Pathlets[index])

        return RealTraj

def ReturnAllTrajectoriesInAList(self) :
        RealTrajs = []
        for i in range(len(self.TrajsResults)) :
            RealTrajs.append(self.ReturnRealTraj(self.TrajsResults[i]))

        RealTrajs = RealTrajs + self.NormalTrajectories

        return RealTrajs
```

Παρακάτω θα αναφερθούμε και θα δούμε τα βασικά μέρη υλοποίησης με δυναμικό προγραμματισμό. Έχει χρησιμοποιηθεί παραλληλία για την επιτάχυνση της εκτέλεσης αλλά δεν θα δώσουμε σημασία σε αυτό.

Αρχικά πρέπει να βρούμε τις βέλτιστες τιμές των υποδιαδρομών. Αυτό το κάνουμε με τις εξής συναρτήσεις. Αρχικά βρίσκουμε τα Tp όλων των υποδιαδρομών, δηλαδή το πλήθος των φορών που μια υποδιαδρομή υπάρχει στις τροχιές, το οποίο υλοποιείτε με τον παρακάτω κώδικα.

```
def FindTpCounterOfPathlets(self, trajectories) :
        for traj in trajectories :
            for i in range(len(traj) + 1):
                for j in range(i + 1, i + 3):

                    sub = tuple(traj[i:j])

                    if (sub not in self.TpCounterNeededForPathletLearning) :
                        self.l.acquire()

                        if (sub not in self.TpCounterNeededForPathletLearning) :
                            self.TpCounterNeededForPathletLearning[sub] = 1
                        else :
                            self.TpCounterNeededForPathletLearning[sub] =
                                    self.TpCounterNeededForPathletLearning[sub] + 1

                        self.l.release()
                    else :
                        self.TpCounterNeededForPathletLearning[sub] =
                                    self.TpCounterNeededForPathletLearning[sub] + 1
```

Αφού βρήκαμε τα παρακάτω βρίσκουμε αναδρομικά τα F* που αποτελούν τις βέλτιστες τιμές των υποδιαδρομών. Με τις παρακάτω δυο συναρτήσεις.

```
def FindFStarForAllSubTrajs(self,traj) :

        FoundValuesOfSubPaths = dict()

        def RecursiveCalculationOfFStar(i,j) : {#δηλώνετε μέσα εδώ η άλλη συνάρτηση}

        for i in range(len(traj)):
                for j in range(i + 1, len(traj) + 1):

                    subtraj = traj[i:j]

                    FoundValuesOfSubPaths[tuple(subtraj)] = RecursiveCalculationOfFStar(i,j-1)

        return FoundValuesOfSubPaths
```

Η RecursiveCalculationOfFStar δηλώνετε μέσα στην FindFStarForAllSubTrajs όπου και εκεί χρησιμοποιείτε και έχει την παρακάτω υλοποίηση.

```
def RecursiveCalculationOfFStar(i,j) :
        if i < j-1:
            sub = tuple(traj[i:j+1])

            if sub in FoundValuesOfSubPaths :
                return FoundValuesOfSubPaths[sub]

            minValue = float('inf')
            for k in range(i+1,j) :
                val1 = RecursiveCalculationOfFStar(i,k)
                val2 = RecursiveCalculationOfFStar(k,j)
                ReturnValue = val1 + val2
                if ReturnValue < minValue :
                    minValue = ReturnValue

            FoundValuesOfSubPaths[sub] = minValue

            return minValue
        else :
            sub = tuple(traj[i:i+2])

            if sub in FoundValuesOfSubPaths :
                return FoundValuesOfSubPaths[sub]

            TpResult = self.TpCounterNeededForPathletLearning[sub]
            Value = self.lamda + 1.0/TpResult

            FoundValuesOfSubPaths[sub] = Value

            return Value
```

Έτσι έχουμε όλες τις βέλτιστες τιμές για κάθε πιθανή υποδιαδρομή.Τώρα αρκεί να τις χρησιμοποιήσουμε ώστε να βρούμε την καλύτερη ανακατασκευή σύμφωνα με αυτές. Έτσι χρησιμοποιούμε τις εξής δυο συναρτήσεις.

```
def ReturnTrajResultAfterFindingDecomposition(self,traj,FoundValuesOfSubPaths) :

    def BacktrackingToFindBestDecomposition(Path) : {#δηλώνετε μέσα εδώ η άλλη
        συνάρτηση}

    BestDecTrajViaPathlet = BacktrackingToFindBestDecomposition(traj)

    return BestDecTrajViaPathlet
```

Όπου η υλοποίηση του BacktrackingToFindBestDecomposition είναι η εξής.

```
def BacktrackingToFindBestDecomposition(Path) :
        if len(Path) == 2 :
            Value1 = FoundValuesOfSubPaths[(Path[0],)]
            Value2 = FoundValuesOfSubPaths[(Path[1],)]

            if Value1 == Value2 :
                return [self.PathToPathletIndex((Path[0],Path[1]))]
            return [self.PathToPathletIndex((Path[0],)),self.PathToPathletIndex((Path[1],))]
        elif len(Path) == 1 :
            return [self.PathToPathletIndex(tuple(Path))]
        elif len(Path) == 0 :
            return []

        BestpathDec = []
        left = []
        right = []

        counter = len(Path) - 1
        flag = True
        while flag :
            MinValue = float('inf')
            Min_i = -1
            FoundBetterSubPath = False
            for i in range(len(Path) - counter + 1) :
                subpath = Path[i:i+counter]

                Value = FoundValuesOfSubPaths[tuple(subpath)]
                if MinValue == Value :
                    continue
                elif Value < MinValue and MinValue == float('inf') :
                    MinValue = Value
                    Min_i = i
                elif Value > MinValue :
                    FoundBetterSubPath = True
                else :
                    MinValue = Value
                    Min_i = i
                    FoundBetterSubPath = True

            if FoundBetterSubPath :
                if Min_i > 0 :
                    left = left + Path[0:Min_i]
                if Min_i + counter < len(Path) :
                    right =  Path[Min_i+counter:len(Path)]+right
                Path = Path[Min_i:Min_i+counter]

            counter = counter - 1
            if counter == 1 :
                flag = False

        left = BacktrackingToFindBestDecomposition(left)
        right = BacktrackingToFindBestDecomposition(right)

        if left :
            BestpathDec = left
        BestpathDec = BestpathDec + [self.PathToPathletIndex(tuple(Path))]
        if right :
            BestpathDec = BestpathDec + right

        return BestpathDec
```

Χρησιμοποιείτε η συνάρτηση PathToPathletIndex για να βρεθεί η καλύτερη ανακατασκευή στο BacktrackingToFindBestDecomposition.Παρατηρούμε ότι έχει και κάποια locks αλλά αυτά αφορούν την παραλληλοποίηση του αλγορίθμου και δεν έχει σημασία στην λογική του αλγορίθμου.

```
def PathToPathletIndex(self,path) :
    index = -1

    if path not in self.Pathlets :
        self.l.acquire()

        if path not in self.Pathlets :
            index = len(self.Pathlets)
            self.Pathlets[path] = index
        else :
            index = self.Pathlets[path]

        self.l.release()
    else :
        index = self.Pathlets[path]

    return index
```

Όπου εν τέλη βρίσκουμε την καλύτερη σύμφωνα με τον αλγόριθμο ανακατασκευή της τροχιάς σύμφωνα με τις βέλτιστες τιμές των Pathlets. Και αυτές οι υποδιαδρομές που χρησιμοποιούνται έστω σε μια ανακατασκευή τροχιάς θα περιλαμβάνονται στο τελικό Pathlet Dictionary.

Η FindFStarAndTrajRes είναι αυτή που ενώνει τον υπολογισμό των βέλτιστων τιμών υποδιαδρομών και της εύρεσης της καλύτερης ανακατασκευής και παρουσιάζετε παρακάτω. Περιλαμβάνει και εξέταση καθαρισμού για να μειώσουμε τον χώρο που χρειάζεται η εκτέλεση του αλγορίθμου αλλά δεν έχει σχέση με τον ίδιο τον αλγόριθμο Pathlet Learning.

```
def FindFStarAndTrajRes(self,traj) :
    FoundValuesOfSubPaths = self.FindFStarForAllSubTrajs(traj)
    TrajResult = self.ReturnTrajResultAfterFindingDecomposition(traj,
        FoundValuesOfSubPaths)

    del FoundValuesOfSubPaths

    if time.time() - self.ListForClean[0] > 180.0 :
        proc = os.getpid()
        if  proc not in self.SetForProcs :
            if self.ListForClean[1] == 1 :
                self.ListForClean[1] = multiprocessing.cpu_count()
                self.SetForProcs.clear()
                self.ListForClean[0] = time.time()
            else :
                self.ListForClean[1] = self.ListForClean[1] - 1
                self.SetForProcs[proc] = 0

            gc.collect()

    return TrajResult
```

Χρησιμοποιείτε το παρακάτω για επιστροφή της λίστας των τροχιών στην μορφή που αρχικά δώθηκαν.

```
def ReturnRealTraj(self,TrajResult) :
        RealTraj = []
        for i in range(len(TrajResult)) :
            index = TrajResult[i]

            RealTraj = RealTraj + list(self.Pathlets[index])

        return RealTraj

def ReturnAllTrajectoriesInAList(self) :
        RealTrajs = []
        for i in range(len(self.TrajsResults)) :
            RealTrajs.append(self.ReturnRealTraj(self.TrajsResults[i]))

        RealTrajs = RealTrajs + self.NormalTrajectories

        return RealTrajs
```

Εν τέλη πρέπει να δώσουμε τον κώδικα για το PercentageOrderOptimizer. Πρέπει να λάβουμε υπ'όψιν μας ότι δεν υπολογίζετε μόνο με το πλήθος των φορών χρήσειμοποιείτε ένα Pathlet αλλά και τον χρόνο που ποιάνει αυτό και οι τροχιές όταν αυτό αφαιρεθεί.

```
def PercentageOrderOptimizer(self,flag) :
        if not self.Pathlets :
            print("There are no Pathlets!")
            return

        PathletCounter = 0
        TrajsResultsCounter = 0
        for P in self.Pathlets :
            PathletCounter = PathletCounter + len(P)

        for T in self.TrajsResults :
            TrajsResultsCounter = TrajsResultsCounter + len(T)

        TrajectoriesThatUsePathlet = [[] for _ in range(len(self.Pathlets))]

        PathletsSizeUsed = [0]*len(self.Pathlets)
        TimesPathletsUsed = [0]*len(self.Pathlets)
        for trajIndex in range(len(self.TrajsResults)) :
            for PathletIndex in self.TrajsResults[trajIndex] :
                TimesPathletsUsed[PathletIndex] = TimesPathletsUsed[PathletIndex] +
                  len(self.TrajsResults[trajIndex])
                PathletsSizeUsed[PathletIndex] = len(self.Pathlets[PathletIndex])

                TrajectoriesThatUsePathlet[PathletIndex].append(trajIndex)
         #calculate percentage of reconstructable trajectories by deleting methodically from   pathlets
        TrajectoriesDeclined = set()
        CalculatedResult = [(100,100,0)]

        PathletsRemovedSizeCounter = 0
```

```
    PathletsRemovedCounter = 1
    while TimesPathletsUsed :
       minIndex = TimesPathletsUsed.index(min(TimesPathletsUsed))

       PathletsRemovedSizeCounter = PathletsRemovedSizeCounter +
                PathletsSizeUsed[minIndex]

       TrajectoriesDeclined.update(TrajectoriesThatUsePathlet[minIndex])

       del TrajectoriesThatUsePathlet[minIndex]
       del TimesPathletsUsed[minIndex]
       del PathletsSizeUsed[minIndex]

       TrajsDeclinedCounter = 0
       for index in TrajectoriesDeclined :
          TrajsDeclinedCounter = TrajsDeclinedCounter + len(self.TrajsResults[index])

       TrajsRemovedSizeCounter = 0
       for index in TrajectoriesDeclined :
          for Pindex in self.TrajsResults[index] :
             TrajsRemovedSizeCounter = TrajsRemovedSizeCounter +
               len(self.Pathlets[Pindex])

       if ((PathletCounter - PathletsRemovedSizeCounter)/PathletCounter) == 0 or
        ((TrajsResultsCounter - TrajsDeclinedCounter)/TrajsResultsCounter) == 0 :
          break

       CalculatedResult.append((((PathletCounter -
        PathletsRemovedSizeCounter)/PathletCounter)*100, ((TrajsResultsCounter -
        TrajsDeclinedCounter)/TrajsResultsCounter)*100,TrajsRemovedSizeCounter))

       PathletsRemovedCounter = PathletsRemovedCounter + 1

    CalculatedResult.append((0,0,self.RealTrajListCounter))


    xaxis = list()
    yaxis = list()
    zaxis = list()

    for (x,y,z) in CalculatedResult :
       xaxis.append(x)
       yaxis.append(y)
       zaxis.append(z)

    BestDifference = self.RealTrajListCounter
    BestDifResult = ()
    for index in range(len(xaxis)) :
       if yaxis[index]/100*TrajsResultsCounter + xaxis[index]/100*PathletCounter + zaxis[index]
        < BestDifference or not BestDifResult:
          BestDifference = yaxis[index]/100*TrajsResultsCounter +
                   xaxis[index]/100*PathletCounter + zaxis[index]
          BestDifResult = (xaxis[index],yaxis[index])

    if flag  and BestDifResult != self.RealTrajListCounter:
        self.OptimizeAccordingToResultPercentageOfPathletsAndTrajectories(BestDifResult)
```

Όπου στο BestDifResult έχουμε την καλύτερη σύμφωνα με τα αποτελέσματα διαφορά για αφαίρεση Pathlet και τροχιών, τα οποία δεν χρησιμοποιούνται καλώς, αφού πιθανότατα δεν έχουν πολλές ή καμία κοινή υποδιαδρομή με άλλες. Έπειτα χρησιμοποιείτε το παρακάτω για υπολογισμό των δεδομένων που θα αφαιρεθούν απο την Pathlet Learning δομή μας.

```
def OptimizeAccordingToResultPercentageOfPathletsAndTrajectories(self,BestDifResult) :
        (x,y) = BestDifResult

        PathletCounter = 0
        TrajsResultsCounter = 0

        for P in self.Pathlets :
            PathletCounter = PathletCounter + len(P)

        for T in self.TrajsResults :
            TrajsResultsCounter = TrajsResultsCounter + len(T)


        TrajectoriesThatUsePathlet = [[] for _ in range(len(self.Pathlets))]

        PathletsSizeUsed = [0]*len(self.Pathlets)
        TimesPathletsUsed = [0]*len(self.Pathlets)
        for trajIndex in range(len(self.TrajsResults)) :
            for PathletIndex in self.TrajsResults[trajIndex] :
                TimesPathletsUsed[PathletIndex] = TimesPathletsUsed[PathletIndex] +
                        len(self.TrajsResults[trajIndex])
                PathletsSizeUsed[PathletIndex] = len(self.Pathlets[PathletIndex])

                TrajectoriesThatUsePathlet[PathletIndex].append(trajIndex)


        #Xrhsh gia euresh twn pathlets p tha petaksoume
        PathletsDeclinedTemp = copy.deepcopy(TimesPathletsUsed)
        PathletsDeclinedTemp = np.argsort(PathletsDeclinedTemp)
        #-----------------------------------------------

        TrajectoriesDeclined = set()
        PathletsDeclined = list()

        CalculatedResult = [(100,100)]

        PathletsRemovedSizeCounter = 0

        PathletsRemovedCounter = 0
        while TimesPathletsUsed :
            TrajsDeclinedCounter = 0
            for index in TrajectoriesDeclined :
                TrajsDeclinedCounter = TrajsDeclinedCounter + len(self.TrajsResults[index])

            if ((PathletCounter - PathletsRemovedSizeCounter)/PathletCounter)*100 == x or
((TrajsResultsCounter - TrajsDeclinedCounter)/TrajsResultsCounter)*100 == y :
                break

            PathletsRemovedCounter = PathletsRemovedCounter + 1
```

```
            minIndex = TimesPathletsUsed.index(min(TimesPathletsUsed))

            TrajectoriesDeclined.update(TrajectoriesThatUsePathlet[minIndex])

           PathletsRemovedSizeCounter = PathletsRemovedSizeCounter +
            PathletsSizeUsed[minIndex]
           PathletsDeclined.append(minIndex)

           del TrajectoriesThatUsePathlet[minIndex]
           del TimesPathletsUsed[minIndex]
           del PathletsSizeUsed[minIndex]


           CalculatedResult.append((((PathletCounter -
            PathletsRemovedSizeCounter)/PathletCounter)*100,
            ((TrajsResultsCounter - TrajsDeclinedCounter)/TrajsResultsCounter)*100))

       PathletsDeclined = PathletsDeclinedTemp[0:PathletsRemovedCounter]

       self.FindAndAskForPercentageOptimization(PathletsDeclined,TrajectoriesDeclined)
```

Έτσι εν τέλη έχουμε την FindAndAskForPercentageOptimization όπου βρίσκει τα τελικά αποτελέσματα με τις αλλαγές και ρωτάει αν θέλουμε να κάνουμε την αλλαγή και πόσο συμπίεση αυτή προκαλεί.

```
def FindAndAskForPercentageOptimization(self,PathletsDeclined,TrajectoriesDeclined) :
       PreviousRes = 0 #Check if Optimization worth it
       for P in self.Pathlets :
              PreviousRes = PreviousRes + len(P)
       for T in self.TrajsResults :
              PreviousRes = PreviousRes + len(T)

       NormalTrajectories = list()
       for i in TrajectoriesDeclined :
          NormalTrajectories.append(self.ReturnRealTraj(self.TrajsResults[i]))

       #-----------------------------------------------------
       DecreaseOfPointersToPathlets = [0]*len(self.Pathlets)
       DecreaseCounter = 0
       for i in range(len(DecreaseOfPointersToPathlets)) :
          if i in PathletsDeclined :
             DecreaseCounter = DecreaseCounter + 1
          else :
             DecreaseOfPointersToPathlets[i] = i - DecreaseCounter

       TrajectoriesDeclined = list(TrajectoriesDeclined)

       TrajsResultsTemp = copy.deepcopy(self.TrajsResults)
       TrajsResultsTemp = np.array(TrajsResultsTemp)

       TrajsResultsTemp = np.delete(TrajsResultsTemp, TrajectoriesDeclined,0)

       for i in range(len(TrajsResultsTemp)) :
          for j in range(len(TrajsResultsTemp[i])) :
             TrajsResultsTemp[i][j] = DecreaseOfPointersToPathlets[TrajsResultsTemp[i][j]]

       #-----------------------------------------------------
```

```
        PathletsTemp = copy.deepcopy(self.Pathlets)
    PathletsTemp = np.array(PathletsTemp)

    PathletsTemp = np.delete(PathletsTemp, PathletsDeclined,0)

    PathletsTemp.tolist()
    PathletsTemp = list(PathletsTemp)
    TrajsResultsTemp.tolist()
    TrajsResultsTemp = list(TrajsResultsTemp)

    CurrentRes = 0
    for P in PathletsTemp :
        CurrentRes = CurrentRes + len(P)

    for T in TrajsResultsTemp :
        CurrentRes = CurrentRes + len(T)
    for T in NormalTrajectories :
        CurrentRes = CurrentRes + len(T)

    if CurrentRes < PreviousRes :
      print("Optimization via UsePercentage is worth it. Previous:"+str(PreviousRes)+" _
Current:"+str(CurrentRes))
    else :
      print("Optimization via UsePercentage is NOT worth it.
      Previous:"+str(PreviousRes)+" _ Current:"+str(CurrentRes))

    implement = ""
    while not(implement == 'yes') and not(implement == 'no') :
      implement = input("Do you want to implement Optimization? yes or no: ")
      print(implement)

    if implement == 'yes' :
      self.TrajsResults = TrajsResultsTemp
      self.Pathlets = PathletsTemp
      self.NormalTrajectories = NormalTrajectories
```

Αφού απαντήσουμε yes γίνεται η μετατροπή και όσα Pathlets αφαιρέσαμε αφαιρούνται απο την δομή και όσες τροχιές δεν μπορούν να ανακατασκευαστούν χωρίς αυτά προστέθονται στην λίστα NormalTrajectories της κλάσης ως κανονικές τροχιές.

Αυτές οι τελευταίες τρεις συναρτήσεις για το κεφάλαιο Αξιοποίηση Σχέσης Δεδομένων του Pathlet Dictionary υλοποιείτε ακριβως με τον ίδιο τρόπο και στα τρία Μέρη υλοποίησης του αλγορίθμου Pathlet Learning.

Τα υπόλοιπα μέρη του κώδικα που βρίσκονται όλα στο github έχουν δευτερεύουσα σημασία αφού δεν αφορούν την βασική υλοποίηση της εργασίας και γι’αυτό δεν αναφέρονται.

ΑΝΑΦΟΡΕΣ

[1] Pathlet Learning for Compressing and Planning Trajectories - Chen Chen, Hao Su, Qixing Juang, Lin Zhang, Leonidas Guibas (https://dl.acm.org/citation.cfm?id=2525443 - https://geometry.stanford.edu/papers/cshzg-gis-13/cshzg-gis-13.pdf)