Ατομική Διπλωματική Εργασία

TERMINATION ANALYSIS OF REASONING

Δήμητρα Αβραμοπούλου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Termination Analysis of Reasoning

Δήμητρα Αβραμοπούλου

Επιβλέπων Καθηγητής
Ανδρέας Πιερής

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Περίληψη

Η παρούσα διπλωματική εργασία εξετάζει το πρόβλημα του τερματισμού της διαδικασίας Semi-oblivious Chase για Simple Linear TGDs και Linear TGDs, εστιάζοντας στην έννοια της ιδιότητας weak-acyclicity. Το Chase Procedure αποτελεί ένα θεμελιώδες εργαλείο για την επεξεργασία και ανάλυση δεδομένων σε συστήματα βάσεων δεδομένων, όμως η εξασφάλιση του τερματισμού του παραμένει μια πρόκληση. Ο κύριος στόχος της εργασίας είναι η ανάπτυξη και αξιολόγηση ενός αλγορίθμου που προσδιορίζει αν το Chase Procedure τερματίζει ή όχι, εντοπίζοντας κρίσιμους κύκλους στον εξαρτημένο γράφο. Η υλοποίηση του αλγορίθμου πραγματοποιήθηκε στη γλώσσα προγραμματισμού Java, ενώ για την πειραματική αξιολόγηση και την ανάλυση των αποτελεσμάτων χρησιμοποιήθηκε η Python, αξιοποιώντας εργαλεία όπως οι βιβλιοθήκες NumPy και Matplotlib για την οπτικοποίηση των δεδομένων. Τα πειράματα περιλάμβαναν την ανάλυση μεγάλων συνόλων TGDs με στόχο τη μελέτη της απόδοσης του αλγορίθμου και την αναγνώριση των βημάτων που απαιτούν τον περισσότερο χρόνο. Τα αποτελέσματα έδειξαν ότι η διαδικασία parsing αποτελεί τη μεγαλύτερη πηγή καθυστέρησης, ,ειδικά για Linear TGDs, όπου απαιτείται μεγαλύτερη επεξεργασία λόγω της ανάγκης για εξειδίκευση και απλοποίηση των κανόνων.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Η Διαδικασία Chase

Η διαδικασία Chase αποτελεί ένα από τα πιο θεμελιώδη αλγοριθμικά εργαλεία στον τομέα των βάσεων δεδομένων. Χρησιμοποιείται για την επίλυση διαφόρων προβλημάτων, όπως ο έλεγχος περιορισμών, η ανταλλαγή δεδομένων και η απάντηση ερωτημάτων υπό περιορισμούς. Στη βασική της μορφή, η διαδικασία Chase δέχεται ως είσοδο μια βάση δεδομένων και ένα σύνολο περιορισμών (TGDs – Tuple Generating Dependencies) και παράγει ένα νέο σύνολο δεδομένων που ικανοποιεί αυτούς τους περιορισμούς, εφόσον η διαδικασία τερματίσει. Τα TGDs είναι κανόνες συνεπαγωγής που δηλώνουν ότι η ύπαρξη ορισμένων δεδομένων σε μια βάση δεδομένων συνεπάγεται την ύπαρξη άλλων δεδομένων. Η διαδικασία Chase προσθέτει νέα δεδομένα στη βάση για να ικανοποιήσει αυτούς τους κανόνες. Είναι χρήσιμη για την κατασκευή καθολικών προτύπων (universal models), τα οποία λειτουργούν ως αντιπροσωπευτικά όλων των πιθανών μοντέλων που ικανοποιούν τα δεδομένα και τους περιορισμούς.

1.2 Η Πρόκληση του Απείρου

Παρόλο που η διαδικασία Chase είναι ιδιαίτερα ισχυρή, δεν εγγυάται πάντα τον τερματισμό της. Υπάρχουν περιπτώσεις όπου η διαδικασία μπορεί να συνεχιστεί επ' άπειρον λόγω αλυσιδωτών προσθηκών δεδομένων που προκύπτουν από τους περιορισμούς. Αυτό δημιουργεί μια μεγάλη πρόκληση, καθώς για ορισμένους τύπους TGDs η διαδικασία δεν μπορεί να ολοκληρωθεί, ακόμα και αν η βάση δεδομένων είναι απλή. Για την αντιμετώπιση αυτής της πρόκλησης, έχουν προταθεί κάποιες συντακτικές ιδιότητες των κανόνων που διασφαλίζουν τον τερματισμό της διαδικασίας Chase. Μια από τις σημαντικότερες ιδιότητες είναι η ασθενής ακυκλικότητα (weak-acyclicity), η οποία διασφαλίζει τον τερματισμό της διαδικασίας Chase εφόσον οι κανόνες είναι Simple Linear όπως θα δούμε στη συνέχεια. Παρόλα αυτά, η ύπαρξη μιας συνθήκης που να είναι ταυτόχρονα αναγκαία και ικανή για τον τερματισμό της διαδικασίας Chase παραμένει ένα ανοιχτό και δύσκολο πρόβλημα.

1.3 Στόχοι Εργασίας

Η παρούσα διπλωματική εργασία εστιάζει στη μελέτη του τερματισμού της διαδικασίας Chase για δύο κατηγορίες περιορισμών, τα γραμμικά TGDs (linear TGDs) και τα απλά γραμμικά TGDs (simple linear TGDs). Στόχος της εργασίας είναι η υλοποίηση ενός αλγορίθμου που να ελέγχει αν η διαδικασία Chase θα τερματίσει ή όχι για διάφορα σύνολα από TGDs. Έπειτα ο αλγόριθμος θα αξιολογηθεί πειραματικά αναλύοντας ποια μέρη της διαδικασίας απαιτούν τον περισσότερο χρόνο. Τα αποτελέσματα των πειραμάτων θα παρουσιαστούν με scatter plots και δίνοντας λεπτομερή ανάλυση της συμπεριφοράς του αλγορίθμου.

1.4 Οργάνωση Κειμένου

Η υφιστάμενη διπλωματική εργασία αποτελείται από πέντε κεφάλαια.

Στο Κεφάλαιο 2 εξετάζουμε το πρόβλημα τερματισμού ενός Chase Procedure για Simple Linear και Linear TGDs. Αρχικά περιγράφουμε την διαδικασία Chase, ακολούθως εισάγουμε το πρόβλημα τερματισμού της διαδικασίας και στο τέλος προτείνουμε δύο αλγορίθμους που επιλύουν το πρόβλημα τερματισμού για τις δύο κατηγορίες TGDs.

Στο Κεφάλαιο 3 περιγράφουμε τον τρόπο υλοποίησης του αλγορίθμου. Αρχικά, παρουσιάζεται η γενική προσέγγιση και τα εργαλεία ανάπτυξης που χρησιμοποιήθηκαν. Στη συνέχεια, γίνεται λεπτομερής περιγραφή της υλοποίησης, η οποία περιλαμβάνει τη μετατροπή των δεδομένων, τη δημιουργία του εξαρτημένου γράφου και τον εντοπισμό κρίσιμου κύκλου.

Στο Κεφάλαιο 4 παρουσιάζουμε την πειραματική αξιολόγηση του αλγορίθμου. Αρχικά περιγράφουμε τον τρόπο υλοποίησης του αλγορίθμου και ποια εργαλεία χρησιμοποιήσαμε για την κατασκευή των γραφικών παραστάσεων. Ακολούθως, παρουσιάζουμε τα αποτελέσματα των πειραμάτων σε μορφή scatter plots και τέλος γίνεται σχολιασμός των με στόχο την ανάλυση της απόδοσης του αλγορίθμου για κάθε στάδιο που θα μελετήσουμε.

Τέλος, το Κεφάλαιο 5 συνοψίζει τα κύρια ευρήματα της εργασίας εξάγοντας τα γενικά συμπεράσματα που προκύπτουν από την έρευνα και την υλοποίηση του αλγορίθμου.

Με αυτή τη δομή, η διπλωματική εργασία στοχεύει να παρουσιάσει με σαφήνεια και ολοκληρωμένο τρόπο τη μελέτη, την ανάπτυξη και την αξιολόγηση του αλγορίθμου για την επίλυση του προβλήματος τερματισμού της διαδικασίας Semi-oblivious Chase για Simple Linear και Linear TGDs.

1.5 Γενικοί Ορισμοί

- Σύνολο Σταθερών (C): Αντιπροσωπεύουν τις «κανονικές» τιμές μιας βάσης δεδομένων.

- Σύνολο Κενών (N): Πρόκειται για ετικετοποιημένα κενά (nulls) που δηλώνουν ότι η τιμή είναι άγνωστη ή μη διαθέσιμη.

- Σύνολο Μεταβλητών (V): Χρησιμοποιούνται σε εξαρτήσεις και λειτουργούν ως κανονικές μεταβλητές.

- Σχεσιακό Σχήμα (R): Ένα πεπερασμένο σύνολο γνωρισμάτων (predicates) το καθένα με την δική του πληθικότητα. p/n: το predicate p έχει n γνωρίσματα

- Θέση (Position)**:** Η θέση p[i] αφορά το i-οστό γνώρισμα του predicate p.

- Όρος (Term)**:** Ένας όρος μπορεί να είναι μια σταθερά, null ή μεταβλητή.

- Άτομο (Atom): p(t), όπου p είναι ένα predicate και t είναι μια πλειάδα από όρους.

- Βάση Δεδομένων (Database): Μια βάση δεδομένων D είναι ένα πεπερασμένο σύνολο ατόμων της μορφής p(t), με την προϋπόθεση ότι $dom(D) \subset C$.

- Εξάρτηση δημιουργίας πλειάδων (Tuple-generating dependencies – TGD): Είναι ένας κανόνας της μορφής:

$$\forall X \forall Y(\phi(X,Y) \rightarrow \exists Z \psi(X,Z))$$

Όπου:

- φ(X, Y): Το σώμα (body) του κανόνα, το οποίο αποτελείται από σύζευξη ατόμων.
- ψ(X, Z): Η κεφαλή (head) του κανόνα, η οποία δηλώνει την ύπαρξη νέων δεδομένων.

Τα TGDs χρησιμοποιούνται για να δηλώσουν ότι η ύπαρξη ορισμένων δεδομένων συνεπάγεται την ανάγκη δημιουργίας νέων δεδομένων.

Κατηγορίες TGDs:

1. Γραμμικά TGDs (Linear TGDs): Έχουν μόνο ένα άτομο στο σώμα του κανόνα.
2. Απλά Γραμμικά TGDs (Simple Linear TGDs): Υποσύνολο των γραμμικών TGDs με τον περιορισμό ότι δεν επιτρέπεται η επανάληψη μεταβλητών στο σώμα.

Trigger: Ένα ζευγάρι (σ,h) όπου σ είναι ένα TGD και h είναι μια απεικόνιση που κάνει το body(σ) να ικανοποιείται στη βάση δεδομένων.

Ομομορφισμός (Homomorphism): Μια απεικόνιση h που χαρτογραφεί τις μεταβλητές του body(σ) σε τιμές ή nulls της βάσης δεδομένων .

Frontier ενός TGD (fr(σ)): Αποτελείται από τις μεταβλητές που εμφανίζονται τόσο στο body όσο και στο head του σ.

Existential ενός TGD (ex(σ)): Οι μεταβλητές που εμφανίζονται στο head(σ) και όχι στο body(σ).

Κεφάλαιο 2

Περιγραφή Semi-oblivious Chase Termination Problem για Simple Linear και Linear TGDs

---

2.1 Chase Procedure

Η διαδικασία Chase είναι ένας θεμελιώδης αλγοριθμικός μηχανισμός στον τομέα των βάσεων δεδομένων. Μπορεί να εφαρμοστεί σε διάφορα προβλήματα όπως Ανταλλαγή Δεδομένων (Data Exchange), Απάντηση Ερωτημάτων υπό Περιορισμούς (Query Answering under Constraints) και Έλεγχος Συνεκτικότητας (Constraint Checking).

Η διαδικασία δέχεται ως είσοδο μια βάση δεδομένων D και ένα σύνολο Σ από TGDs. Το chase λειτουργεί πάνω στο D εφαρμόζοντας το *trigger* για ένα σύνολο από TGDs πάνω στο D. Αν η διαδικασία τερματίσει (κάτι που δεν είναι εγγυημένο), το αποτέλεσμα είναι μια πεπερασμένη βάση δεδομένων DΣ που ονομάζεται καθολικό μοντέλο του D και Σ. Το DΣ έχει τις εξής ιδιότητες:

- Το DΣ είναι *μοντέλο* του D και του Σ, δηλαδή περιέχει το D και ικανοποιεί τους περιορισμούς του Σ.
- Το DΣ είναι *καθολικό*, δηλαδή μπορεί να ενσωματωθεί ομομορφικά σε κάθε άλλο μοντέλο των D και Σ.

Διαδικασία Chase:

Είσοδος:

1. Βάση Δεδομένων D: Ένα σύνολο αρχικών δεδομένων.
2. Σύνολο TGDs Σ: Ένα σύνολο κανόνων που επιβάλλουν σχέσεις μεταξύ των δεδομένων.

Βήματα της Διαδικασίας Chase:

1. Εντοπισμός Trigger:

   Εντοπίζουμε ένα trigger (σ,h), όπου:

   - σ είναι ένας κανόνας (TGD) από το σύνολο Σ.
   - h είναι ένας ομομορφισμός που απεικονίζει τις μεταβλητές του body(σ) σε τιμές της βάσης δεδομένων D, έτσι ώστε $h(body(\sigma)) \subseteq D$.

2. Εφαρμογή του Trigger:

   - Δημιουργούμε νέες εγγραφές που προκύπτουν από το head(σ).
   - Χρησιμοποιούμε νέες τιμές (nulls) για τις υπαρξιακές μεταβλητές που δεν μπορούν να αντιστοιχιστούν σε υπάρχουσες τιμές.

3. Ενημέρωση της Βάσης:
   Η βάση δεδομένων ενημερώνεται με τις νέες εγγραφές. Αν προκύψουν νέοι κανόνες που μπορούν να ενεργοποιηθούν, η διαδικασία συνεχίζεται.

4. Επανάληψη:
   Επαναλαμβάνουμε τη διαδικασία μέχρι να μην υπάρχει διαθέσιμο trigger ή μέχρι η διαδικασία να φτάσει σε μη-τερματισμό (infinite loop).

Παράδειγμα Chase:

Έστω TGD: $ancestor(X,Y) \rightarrow \exists Z(parent(X,Z) \wedge ancestor(Z,Y))$ και
$D=\{ancestor(Alice,Bob)\}$

1. Εντοπισμός Trigger: Εντοπίζουμε h(X=Alice,Y=Bob) που ικανοποιεί το body ancestor(X,Y).
2. Εφαρμογή Trigger: Προσθέτουμε parent(Alice,z1)), ancestor(z1,Bob) στη βάση, όπου z1 είναι μια νέα άγνωστη τιμή (null).
3. Ενημέρωση: Νέα βάση D1 ={ancestor(Alice,Bob), parent(Alice,z1)), ancestor(z1,Bob) }
4. Εντοπίζουμε νέο trigger με h'(X=z1,Y=Bob) και προσθέτουμε parent(z1,z2), ancestor(z2,Bob),
   όπου z2 είναι μια νέα άγνωστη τιμή (null).
5. Εντοπίζουμε h'(X=z1,Y=Bob) και προσθέτουμε parent(z1,z2), ancestor(z2,Bob),
   όπου z2 είναι μια νέα άγνωστη τιμή (null).

Τελική Βαση D_{Σ} = {ancestor(Alice,Bob),parent(Alice,z1),ancestor(z1,Bob)}∪

$$\bigcup_{i>0} \{ parent(zi, zi+1), ancestor(zi+1,Bob)\},$$

z1,z2,.. είναι nulls.

Όπως παρατηρούμε σε αυτό το παράδειγμα ο Chase δεν τερματίζει λόγω του αναδρομικού χαρακτήρα του κανόνα. Κάθε φορά που το trigger ενεργοποιείται, δημιουργούνται νέες άγνωστες τιμές (nulls) προκαλώντας ένα ατέρμονα βρόγχο.

Είδη Chase:

1. Oblivious Chase:
   Εφαρμόζει όλα τα διαθέσιμα triggers χωρίς να λαμβάνει υπόψη προηγούμενες εφαρμογές, οδηγώντας σε πιθανές περιττές προσθήκες.
2. Semi-Oblivious Chase:
   Αποφεύγει την εφαρμογή triggers που είναι ισοδύναμα με προηγούμενα, μειώνοντας τον αριθμό των περιττών εγγραφών.

Στην παρούσα εργασία θα ασχοληθούμε με το Semi-Oblivious Chase.

2.2 Chase Termination Problem

Το Chase Termination Problem αναφέρεται στο ερώτημα του αν η διαδικασία Chase τερματίζει όταν εφαρμόζεται σε μια δεδομένη βάση δεδομένων D και ένα σύνολο TGDs Σ. Το πρόβλημα αυτό είναι γνωστό πως δεν έχει γενική λύση. Ο τερματισμός της διαδικασίας Chase δεν είναι πάντα εγγυημένος λόγω του ότι μπορεί να οδηγήσει σε άπειρη ακολουθία νέων εγγραφών όταν οι περιορισμοί επιβάλλουν αναδρομικές σχέσεις μεταξύ των δεδομένων όπως είδαμε και στο πιο πάνω παράδειγμα. Ωστόσο, αν και το πρόβλημα δεν έχει γενική λύση όπως θα δούμε υπάρχουν κάποιες συντακτικές ιδιότητες των TGDs με τις οποίες η διαδικασία chase θα τερματίζει για κάθε βάση δεδομένων.

2.3 Περιγραφή Αλγορίθμου Semi-oblivious Chase Termination για Simple Linear TGDs

Simple Linear TGD: Ένα TGD σ είναι Simple Linear αν το body(σ) αποτελείται από μόνο ένα άτομο και δεν υπάρχει επανάληψη μεταβλητών στο body(σ).

Παράδειγμα: $\sigma = p(X, Y) \rightarrow \exists Z\ s(X, Z), p(X, Z)$.
Όπως παρατηρούμε το σ είναι simple linear διότι το body(σ) = p(X, Y) αποτελείται από μόνο ένα άτομο και δεν υπάρχουν επαναλήψεις μεταβλητών σε αυτό.

Weak-Acyclicity: Το weak-acyclicity είναι μια συντακτική ιδιότητα των TGDs όπου διασφαλίζει ότι η διαδικασία chase θα τερματίσει για κάθε βάση δεδομένων. Απαραίτητη προϋπόθεση είναι τα TGDs να είναι Simple Linear. Η συνθήκη για τα Simple Linear είναι ικανή και αναγκαία ενώ για τα τα Linear TGDs η συνθήκη είναι ικανή και όχι αναγκαία. Αυτό σημαίνει ότι αν ένα σύνολο από Linear TGDs δεν είναι weak-acyclic δεν σημαίνει απαραίτητα ότι η διαδικασία chase δεν θα τερματίσει, ενώ αν είναι weak-acyclic γνωρίζουμε ότι η διαδικασία σίγουρα θα τερματίσει. Για να ελέγξουμε αν ένα σύνολο από Simple Linear TGDs είναι weak-acyclic, κατασκευάζουμε τον εξαρτημένο γράφο (dependency graph) των Simple Linear TGDs και ελέγχουμε πως δεν υπάρχει κύκλος που να περιέχει ειδική ακμή. Οι ειδικές ακμές προκύπτουν από τις υπαρξιακές μεταβλητές των TGDs και είναι υπεύθυνες για τη δημιουργία νέων άγνωστων τιμών (nulls) κατά τη διαδικασία Chase.

Ο εξαρτημένος γράφος (dependency graph) ενός συνόλου TGDs Σ είναι ένας κατευθυνόμενος πολυγράφος με ετικέτες που ορίζεται ως $DG(\Sigma)=(N,E,\lambda)$ όπου:

- $N=pos(sch(\Sigma))$: Το σύνολο των κόμβων του γράφου είναι οι θέσεις του σχήματος της βάσης δεδομένων που ορίζεται από τους κανόνες Σ.
- $\lambda:E\rightarrow\Sigma\times N$: Η συνάρτηση λ ετικετοποιεί τις ακμές του γράφου με βάση τους κανόνες από το Σ και τις θέσεις του σχήματος.
- E: Το σύνολο των ακμών E ορίζεται ως εξής, με $head(\sigma)=a1,\ldots,ak$ και $[k]=\{1,2..,k\}$:

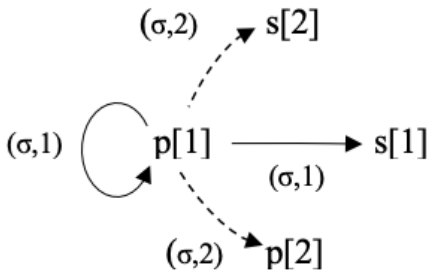
  Για κάθε κανόνα $\sigma\in\Sigma$, για κάθε μεταβλητή $V\in fr(\sigma)$ του και για κάθε θέση $\pi\in body(\sigma,V)$:

Κανονικές Ακμές (Normal Edges): Για κάθε i∈[k] και για κάθε θέση π′ στο ai που περιέχει τη μεταβλητή V, υπάρχει μια κανονική ακμή e=(π,π′)∈E με ετικέτα λ(e)=(σ,i)

Ειδικές Ακμές (Special Edges): Για κάθε υπαρξιακή μεταβλητή W∈ex(σ), για κάθε i∈[k] και για κάθε θέση π′ στο ai που περιέχει τη μεταβλητή W, υπάρχει μια ειδική ακμή e=(π,π′)∈E με ετικέτα λ(e)=(σ,i).

Οι κανονικές ακμές καταγράφουν τη διάδοση ενός όρου από μια θέση π σε μια θέση π′ κατά τη διάρκεια της διαδικασίας chase, ενώ οι ειδικές ακμές καταγράφουν ότι η διάδοση από τη θέση π σε π′ δημιουργεί τιμή null στη θέση π′ εξαιτίας μιας υπαρξιακής μεταβλητής.

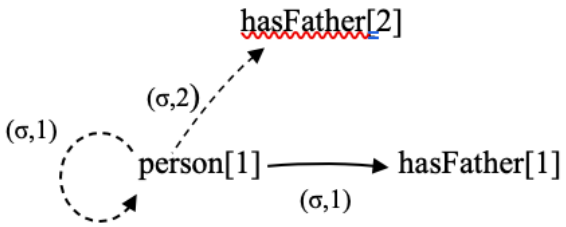
Παράδειγμα: σ: P(X, Y) → ∃Z S(X, Z), P(X, Z)



Σχήμα 3.1

Το σχήμα 3.1 παρουσιάζει τον εξαρτημένο γράφο του TGD σ όπου όπου οι διακεκομμένες γραμμές αποτελούν τις ειδικές ακμές. Όπως παρατηρούμε το σ είναι weak-acyclic αφού δεν υπάρχει κύκλος που να περιέχει ειδική ακμή.

Παράδειγμα: σ: person(X) → ∃Y hasFather(X, Y), person(Y)



Σχήμα 3.2

Το σχήμα 3.2 παρουσιάζει τον εξαρτημένο γράφο του TGD σ όπου όπου οι διακεκομμένες γραμμές αποτελούν τις ειδικές ακμές. Όπως παρατηρούμε το σ δεν είναι weak-acyclic αφού υπάρχει κύκλος στον κόμβο person[1] που περιέχει ειδική ακμή.

Ψευδοκώδικας για το Πρόβλημα Τερματισμού του Chase για simple linearTGDs:

```
Είσοδος: Σύνολο Σ simple linear TGDs
Έξοδος: Απόφαση για τον τερματισμό του Chase για κάθε TGD στο Σ

Διαδικασία:
1. Για κάθε simple linear TGD σ στο σύνολο Σ:
    α. Δημιούργησε τον εξαρτημένο γράφο του σ.
    β. Ελέγξε αν ο γράφος είναι weakly acyclic:
        i. Αν ναι, τότε ο Chase τερματίζει για το TGD σ.
        ii. Αν όχι, ο Chase δεν τερματίζει για το TGD σ.
```

2.4 Περιγραφή Αλγορίθμου Semi-oblivious Chase Termination για Linear TGDs

Linear TGD: Ένα TGD σ είναι Linear αν το body(σ) αποτελείται από μόνο ένα άτομο.

Παράδειγμα: $\sigma = p(X, X) \rightarrow \exists Z\ s(X, Z), p(X, Z)$.

Όπως παρατηρούμε το σ είναι linear διότι το body(σ) = p(X, X). Βλέπουμε ότι στο body(σ) υπάρχει επανάληψη της μεταβλητής X κάτι που επιτρέπεται στα linear TGDs.

Ο αλγόριθμος Semi-oblivious Chase Termination για linear TGDs είναι παρόμοιος με τον αλγόριθμο για simple linear TGDs, με την διαφορά ότι τα linear TGDs θα πρέπει πρώτα να μετατραπούν σε simple linear και μετά η διαδικασία είναι η ίδια. Η μετατροπή ενός linear tgd σε simple linear επιτυγχάνεται με την μέθοδο της εξειδίκευσης (specialization) και της απλοποίησης (simplification) όπως περιγράφονται πιο κάτω.

Διαδικασία μετατροπής linear TGD σε simple linear:

1. Εξειδίκευση (Specialization):

Μια εξειδίκευση ενός συνόλου μεταβλητών X=(x1,x2,…,xn) είναι μια συνάρτηση f που αντιστοιχίζει κάθε μεταβλητή xi σε μια από τις παρακάτω τιμές:

- f(x1)= x1: Η πρώτη μεταβλητή παραμένει πάντα αμετάβλητη.
- Για κάθε i∈{2,…,n} ισχύει: f(xi) ∈ {f(x1), f(x2),…,f(xi−1)}

Δηλαδή, η μεταβλητή xi μπορεί να αντιστοιχιστεί είτε στον εαυτό της είτε σε οποιαδήποτε προηγούμενη τιμή που έχει ήδη αντιστοιχιστεί.

Με απλά λόγια, μια εξειδίκευση ορίζει όλους τους δυνατούς συνεπείς τρόπους αντικατάστασης των μεταβλητών στο σώμα ενός γραμμικού TGD, διατηρώντας τη δομή των επαναλήψεων των μεταβλητών.

Σκοπός της Εξειδίκευσης:
Η συνάρτηση εξειδίκευσης f:

- Καταγράφει όλους τους πιθανούς τρόπους αντικατάστασης των μεταβλητών στο σώμα ενός γραμμικού TGD.
- Επιτρέπει τη συστηματική δημιουργία όλων των απλοποιήσεων ενός γραμμικού TGD, εφαρμόζοντας διαφορετικούς έγκυρους τρόπους αντιστοίχισης των μεταβλητών.

Παράδειγμα Εξειδίκευσης: Έστω X=(x,y,x,z).

Οι πιθανές εξειδικεύσεις του X περιλαμβάνουν:

1. f1(x) = x, f1(y) = x, f1(z) = x → f1(X)=(x,x,x,x)
2. f2(x) = x, f2(y) = x, f2(z) = z → f2(X)=(x,x,x,z)
3. f3(x) = x, f3(y) = y, f3(z) = x → f3(X)=(x,y,x,x)
4. f4(x) = x, f4(y) = y, f4(z) = z → f4(X)=(x,y,x,z)

2. Απλοποίηση (Simplification):

Η απλοποίηση ενός γραμμικού TGD είναι μια διαδικασία που μετατρέπει κάθε εξειδίκευση σε ένα simple linear tgd, διατηρώντας τη σημασιολογική του ισοδυναμία, αλλά απλοποιώντας τη δομή του. Η απλοποίηση επιτυγχάνεται μέσω των εξής βημάτων:

1. Εύρεση μοναδικών μεταβλητών (unique(t)): Για κάθε όρο t=(t1,t2,…,tn) στο σώμα και στην κεφαλή του tgd βρίσκουμε το σύνολο των μοναδικών μεταβλητών που εμφανίζονται διατηρώντας τη σειρά της πρώτης εμφάνισής τους.
2. Αναγνώριση θέσεων (IDs): Για κάθε όρο t, έχουμε μια μοναδική τιμή που αναπαριστά τη θέση του στο unique(t).
3. Απλοποίηση ατόμου (simple(a)): Για ένα άτομο α=R(t1,t2,…,tn) η απλοποίησή του ορίζεται ως: simple(R(t1,t2,…tn)) = Rid(t)(unique(t))

Παράδειγμα Απλοποίησης: Έστω το γραμμικό TGD: $P(x,y,x) \rightarrow Q(x,z)$

Βήμα 1: Εύρεση εξειδικέυσεων (specialization)

- $f1=\{x \mapsto x, y \mapsto x\}$
- $f2=\{x \mapsto x, y \mapsto y\}$
- $f3=\{x \mapsto y, y \mapsto y\}$

Βήμα 2: Απλοποίηση (simplification):

Για κάθε εξειδίκευση εφαρμόζουμε τα βήματα για την απλοποίηση του TGD:

1. Εύρεση μοναδικών μεταβλητών (unique(t)):

   Για το σώμα P(x,y,x) ανάλογα με την εξειδίκευση:

   - f1: unique(t)=[x]
   - f2: unique(t)=[x,y]
   - f3: unique(t)=[y]

   Για την κεφαλή Q(x,z) ανάλογα με την εξειδίκευση:

   - f1: unique(t)=[x,z]

- f2: unique(t)=[x,z]
- f3: unique(t)=[x,z]

2. Αναγνώριση θέσεων (IDs):

   Για το σώμα P(x,y,x) ανάλογα με την εξειδίκευση:

   - f1: IDs=[1,1,1]
   - f2: IDs=[1,2,1]
   - f3: IDs=[1,1,1]

   Για την κεφαλή Q(x,z) ανάλογα με την εξειδίκευση:

   - f1: IDs=[1,3]
   - f2: IDs=[1,3]
   - f3: IDs=[1,3]

3. Απλοποίηση ατόμου (simple(a)):

   - f1: P111(x)→Q12(x,z)
   - f2: P121(x,y)→Q12(x,z)
   - f3: P111(y)→Q12(y,z)

Ψευδοκώδικας για το Πρόβλημα Τερματισμού του Chase για linear TGDs:

Είσοδος: Σύνολο Σ από linear TGDs

Έξοδος: Απόφαση για τον τερματισμό του Chase για κάθε TGD στο Σ

Διαδικασία:

Για κάθε γραμμικό TGD σ στο σύνολο Σ:

1. Εύρεση Εξειδικεύσεων:
   - Υπολόγισε όλες τις εξειδικεύσεις spec(σ) του body(σ).
2. Για κάθε εξειδίκευση spec(σ):

α. Δημιούργησε την απλοποίηση simple(spec(σ)).

β. Κατασκεύασε τον εξαρτημένο γράφο του simple(spec(σ)).

γ. Έλεγξε αν ο γράφος εξαρτήσεων είναι ασθενώς ακυκλικός (weakly acyclic):

- Αν ο γράφος είναι ασθενώς ακυκλικός:
  - Το Chase τερματίζει για αυτή την απλοποίηση.
- Αλλιώς:
  - Το Chase δεν τερματίζει για αυτή την απλοποίηση

Κεφάλαιο 3

Υλοποίηση Semi-oblivious Chase Termination Problem

3.1 Περιγραφή Τρόπου Υλοποίησης

Για την υλοποίηση του αλγορίθμου Semi-oblivious Chase χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java. Αρχικά, ο κώδικας διαβάζει τα TGDs από ένα αρχείο εισόδου, αφαιρεί κανόνες που δεν είναι simple linear και τα μετατρέπει σε αντικείμενα κατάλληλης μορφής για να μπορούν να υπέχουν επεξεργασίας. Στη συνέχεια, κατασκευάζει τον Εξαρτημένο Γράφο (Dependency Graph) των TGDs, όπου οι κόμβοι αναπαριστούν τις θέσεις των TGDs, ενώ οι ακμές δηλώνουν τις εξαρτήσεις μεταξύ τους (κανονικές και ειδικές ακμές). Ο γράφος ελέγχεται για ασθενή ακυκλικότητα (weak acyclicity), μια ιδιότητα που εξασφαλίζει ότι η διαδικασία Chase θα τερματίσει. Παράλληλα, ο κώδικας μετράει και καταγράφει τον χρόνο εκτέλεσης διαφόρων σταδίων, όπως η ανάγνωση και ανάλυση των TGDs, η δημιουργία του εξαρτημένου γράφου, ο έλεγχος ακυκλικότητας και ο συνολικός χρόνος με σκοπό την αξιολόγηση της αποδοτικότητας της διαδικασίας. Στο τέλος της εκτέλεσης, το πρόγραμμα εμφανίζει αναλυτικά στατιστικά στοιχεία, όπως τον αριθμό των TGDs κάθε αρχείου και τους χρόνους εκτέλεσης όλων των αρχείων για κάθε στάδιο. Τα δεδομένα αυτά εκτυπώνονται σε μορφή που μπορεί να χρησιμοποιηθεί για περαιτέρω ανάλυση, όπως η δημιουργία scatter plots με τη χρήση Python που θα χρησιμοποιηθούν στην πειραματική αξιολόγηση όπως θα δούμε αργότερα.

3.2 Εργαλεία Ανάπτυξης

3.2.1 Εργαλείο Intellij

Το IntelliJ IDEA είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού (IDE) που αναπτύχθηκε από την JetBrains και χρησιμοποιείται ευρέως για την ανάπτυξη εφαρμογών σε Java αλλά και σε άλλες γλώσσες όπως Kotlin, Scala, Groovy, και JavaScript. Αποτελεί ένα από τα πιο ισχυρά και δημοφιλή IDEs λόγω της έξυπνης συμπλήρωσης κώδικα, των εργαλείων αναδόμησης (refactoring), και της ανάλυσης κώδικα σε πραγματικό χρόνο που βοηθούν στην αποφυγή σφαλμάτων και τη βελτιστοποίηση της ανάπτυξης λογισμικού. Διαθέτει ενσωματωμένο debugger, διαχείριση εξαρτήσεων μέσω Maven και Gradle, καθώς και ενσωμάτωση με συστήματα ελέγχου εκδόσεων όπως το Git και το SVN. Είναι διαθέσιμο σε δύο εκδόσεις: τη δωρεάν Community Edition, η οποία καλύπτει βασικές ανάγκες ανάπτυξης, και την Ultimate Edition, η οποία προσφέρει προηγμένες λειτουργίες για ανάπτυξη εταιρικών και web εφαρμογών.

3.2.2 Γλώσσα Προγραμματισμού Java

Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού που αναπτύχθηκε από την Sun Microsystems το 1995 και ανήκει πλέον στην Oracle. Διακρίνεται για τη φιλοσοφία "Write Once, Run Anywhere" (WORA), επιτρέποντας την εκτέλεση του ίδιου κώδικα σε οποιαδήποτε συσκευή με Java Virtual Machine (JVM). Χρησιμοποιείται για ανάπτυξη επιτραπέζιων εφαρμογών, web εφαρμογών, και εφαρμογών Android. Η Java διαθέτει πλούσιες βιβλιοθήκες, αυτοματοποιημένη διαχείριση μνήμης με garbage collection, και χαρακτηριστικά όπως πολυμορφισμό, κληρονομικότητα και ενθυλάκωση. Είναι ευρέως διαδεδομένη λόγω της ασφάλειας, της ευελιξίας και της σταθερότητάς της σε μεγάλες και σύνθετες εφαρμογές.

3.3 Λεπτομέρειες Υλοποίησης

Η υλοποίηση αποτελείται από διάφορες κλάσεις που συνεργάζονται για να επιτελέσουν την επεξεργασία των TGDs, τη δημιουργία εξαρτημένου γράφου και τον εντοπισμό κρίσιμων κύκλων. Παρακάτω περιγράφονται οι κλάσεις που χρησιμοποιούνται, καθώς και οι μέθοδοι που περιέχουν και οι λειτουργίες που επιτελούν.

1. Κλάση ChaseTermination

Μέθοδοι:

1. void main(String[] args):

Η κύρια μέθοδος εκκίνησης του προγράμματος που διαχειρίζεται τη ροή των δεδομένων, μετρά τον χρόνο εκτέλεσης για κάθε βήμα και εκτυπώνει τα αποτελέσματα. Αρχικά διαβάζει τα αρχεία .txt από το φάκελο δεδομένων που καθορίζεται από το datasetDirPath. Ακολούθως επεξεργάζεται κάθε αρχείου αναλύοντας τα TGDs που περιέχει. Ανάλογα με το flag isLinear, είτε επεξεργάζεται Linear TGDs, είτε Simple Linear TGDs.
Για κάθε αρχείο, μετρά:

- Χρόνο ανάλυσης των TGDs (parsing time).
- Χρόνο δημιουργίας του εξαρτημένου γράφου (graph creation time).
- Χρόνο ελέγχου ασθενούς ακυκλικότητας (weak acyclicity check time).
- Συνολικό χρόνο εκτέλεσης.

Τέλος, οι χρόνοι εκτέλεσης που μετρήθηκαν αποθηκεύονται σε λίστες και εκτυπώνονται για τη δημιουργία γραφημάτων αργότερα.

2. List<SimpleLinearTGD> readLinearTGDsFromFile(String filePath):

Αυτή η μέθοδος διαβάζει τα Linear TGDs από το καθορισμένο αρχείο. Για κάθε γραμμή του αρχείου: Αν η γραμμή δεν είναι κενή ή σχόλιο, την αναλύει και την προσθέτει σε μια λίστα τύπου SimpleLinearTGD. Χρησιμοποιεί τη μέθοδο parseTGD() για να αναλύσει το TGD και να το μετατρέψει σε αντικείμενο SimpleLinearTGD.

3. List<SimpleLinearTGD>
convertLinearToSimpleLinear(List<SimpleLinearTGD> linearTGDs):

Μετατρέπει τα Linear TGDs σε Simple Linear TGDs. Για κάθε TGD στη λίστα δημιουργεί εξειδικεύσεις των μεταβλητών και εφαρμόζει αυτές τις εξειδικεύσεις για να παράξει απλοποιημένα TGDs.

4. List<Map<String,String>> generateSpecializations(List<String> variables): H

Η μέθοδος δημιουργεί μια νέα λίστα specializations που θα αποθηκεύσει όλες τις εξειδικεύσεις για τις μεταβλητές που δίνονται στη στη λίστα variables. Στη συνέχεια καλεί την αναδρομική μέθοδο generateSpecializationsRec, η οποία χειρίζεται την αναδρομική διαδικασία δημιουργίας αυτών των εξειδικεύσεων.

5. void generateSpecializationsRec(Map<String, String> current, List<String> variables, int index, List<Map<String, String>> specializations):

Είναι μια αναδρομική διαδικασία που δημιουργεί όλες τις πιθανές εξειδικεύσεις για ένα σύνολο μεταβλητών. Ξεκινώντας από τη πρώτη μεταβλητή στη λίστα, η μέθοδος

αναθέτει σε αυτήν όλες τις δυνατές τιμές που έχουν ανατεθεί σε προηγούμενες μεταβλητές, καθώς και την ίδια την τιμή της. Κάθε νέα εξειδίκευση αποθηκεύεται στον χάρτη (Map<String, String>), όπου το κλειδί είναι το όνομα της μεταβλητής και η τιμή είναι η τιμή στην οποία έχει ανατεθεί. Ο χάρτης διατηρεί τις εξειδικεύσεις των μεταβλητών καθώς η αναδρομή προχωρά, επιτρέποντας τη δημιουργία διαφορετικών συνδυασμών αναθέσεων. Όταν ολοκληρωθεί η διαδικασία για όλες τις μεταβλητές, η τρέχουσα εξειδίκευση προστίθεται στη λίστα των εξειδικεύσεων, και η αναδρομή επιστρέφει για να εξετάσει άλλες μεταβλητές.

6. SimpleLinearTGD parseTGD(String tgdString):

Αναλύει ένα συγκεκριμένο string που αναπαριστά ένα TGD και το μετατρέπει σε ένα αντικείμενο SimpleLinearTGD. Αρχικά διαχωρίζει το TGD σε head και body. Η γραμμή που περιέχει το TGD διαχωρίζεται σε δύο μέρη με βάση το σύμβολο ":-" με την εντολή String[] parts = tgdString.split(":-"). Στη συνέχεια ελέγχει εάν η δομή του TGD είναι σωστή. Πρέπει να υπάρχουν ακριβώς δύο μέρη μετά το split: Αν δεν υπάρχουν ακριβώς δύο μέρη, πετάει μια εξαίρεση IllegalArgumentException. Από τα δύο μέρη που δημιουργήθηκαν, το πρώτο (parts[0]) είναι το head και το δεύτερο (parts[1]) είναι το body. Καλείται η parseAtoms(String atomsString) για κάθε μέρος ώστε να αναλυθούν τα άτομα που περιέχονται στο head και το body αντίστοιχα: Έχουμε List<Atom> bodyAtoms = parseAtoms(parts[1]); και List<Atom> headAtoms = parseAtoms(parts[0]); Ελέγχει επίσης ότι το body περιέχει ακριβώς ένα άτομο και ότι δεν υπάρχει επανάληψη μεταβλητών στο body, κάτι το οποίο είναι απαραίτητο για ένα Simple Linear TGD. Με τα άτομα που αναλύθηκαν από το head και το body, δημιουργείται ένα νέο αντικείμενο SimpleLinearTGD και επιστρέφεται: return new SimpleLinearTGD(bodyAtom, headAtoms);

2. Κλάση SimpleLinearTGD

Αντιπροσωπεύει έναν Απλό Γραμμικό TGD που αποτελείται από έναν όρο (Atom) στο σώμα και μία λίστα από όρους στην κεφαλή.

Ιδιωτικές Μεταβλητές :

1. Atom body: Ένα άτομο στο σώμα του TGD.
2. List<Atom> head: Λίστα ατόμων στην κεφαλή του TGD.

Μέθοδοι:

1. Atom getBody(): Επιστρέφει το άτομο του σώματος του TGD.

2. List<Atom> getHead(): Επιστρέφει την κεφαλή του TGD όπου είναι μια λίστα από άτομα.

3. Set<String> getFrontier(): Επιστρέφει το σύνολο των frontier μεταβλητών του TGD, δηλαδή των μεταβλητών που εμφανίζονται τόσο στο σώμα όσο και στην κεφαλή. Δημιουργεί δύο σύνολα, το bodyVariables και το headVariables και με την μέθοδο retainAll() βρίσκει τις κοινές μεταβλητές των δύο συνόλων και τις επιστρέφει.

4. Set<String> getExistential(): Επιστρέφει το σύνολο των υπαρξιακών μεταβλητών που εμφανίζονται στην κεφαλή αλλά όχι στο σώμα. Δημιουργεί δύο σύνολα, το bodyVariables και το headVariables και με την μέθοδο removeAll() αφαιρεί από το σύνολο headVariables όλες τις μεταβλητές που περιέχονται στο bodyVariables.

3. Κλάση Atom

Αντιπροσωπεύει ένα άτομο που αποτελείται από ένα κατηγόρημα (predicate) και μια λίστα από όρους.

Ιδιωτικές Μεταβλητές:

1. String predicate: Το όνομα του κατηγορήματος.
2. List<String> terms: Η λίστα των όρων.

Μέθοδοι:

1. getPredicate(): Επιστρέφει το κατηγόρημα του ατόμου.

2. List<String> getTerms(): Επιστρέφει τη λίστα των όρων του ατόμου.

3. Atom(String predicate, List<String> terms): Ο constructor δημιουργεί ένα αντικείμενο της κλάσης Atom.

4. Κλάση DependencyGraph

Η κλάση DependencyGraph έχει ως στόχο την αναπαράσταση ενός εξαρτημένου γράφου για τα SimpleLinearTGDs. Ο κύριος στόχος αυτής της κλάσης είναι να κατασκευάσει και να διαχειριστεί έναν γράφο που αναπαριστά τις εξαρτήσεις μεταξύ των όρων των TGDs και να προσδιορίσει αν ο γράφος είναι ασθενώς ακυκλικός (weakly acyclic).

Ιδιωτικές Μεταβλητές:

1. adjList (Map<String, List<Edge>>): Είναι μια λίστα γειτνίασης που χρησιμοποιείται για την αποθήκευση των ακμών του γράφου. Κάθε κλειδί στον πίνακα είναι ο κόμβος του γράφου και η τιμή του είναι μια λίστα από ακμές που ξεκινούν από αυτόν τον κόμβο.
Μέθοδοι:

1. void addEdge(String from, String to, String type): Αυτή η μέθοδος προσθέτει μια νέα ακμή στον γράφο. Η ακμή καθορίζεται από τον κόμβο προέλευσης from, τον κόμβο προορισμού to, και τον τύπο της ακμής (κανονική ή ειδική).

2. boolean isWeaklyAcyclic(): Η μέθοδος αυτή ελέγχει αν ο εξαρτημένος γράφος είναι ασθενώς ακυκλικός (weakly acyclic), δηλαδή ότι δεν υπάρχει κύκλος στον εξαρτημένο γράφο που να περιέχει ειδική ακμή. Αρχικά, η μέθοδος αρχικοποιεί δύο σύνολα: το σύνολο visited, το οποίο αποθηκεύει τους κόμβους που έχουν ήδη επισκεφτεί ώστε να μην εξεταστούν ξανά και το σύνολο recStack, το οποίο αποθηκεύει τους κόμβους που βρίσκονται στη στοίβα αναδρομής για την ανίχνευση κύκλων στον γράφο. Στη συνέχεια, για κάθε κόμβο στον πίνακα γειτνίασης (adjList), η μέθοδος καλεί τη βοηθητική μέθοδο detectDangerousCycle() για να ελέγξει αν υπάρχει επικίνδυνος κύκλος, δηλαδή κύκλος που να περιλαμβάνει τουλάχιστον μία ειδική ακμή. Αν βρεθεί τέτοιος κύκλος, η μέθοδος επιστρέφει false, υποδεικνύοντας ότι ο γράφος δεν είναι weak acyclic. Αν όλοι οι κόμβοι και οι ακμές έχουν εξεταστεί και δεν έχει εντοπιστεί επικίνδυνος κύκλος, η μέθοδος επιστρέφει true, δηλώνοντας ότι ο γράφος είναι weak acyclic.

3. boolean detectDangerousCycle(String node, Set<String> visited, Set<String> recStack, boolean containsSpecialEdge): Η μέθοδος detectDangerousCycle χρησιμοποιεί Αναζήτηση Κατά Βάθος (DFS) για να εντοπίσει επικίνδυνους κύκλους στον εξαρτημένο γράφο. Αρχικά, ελέγχει αν ο τρέχον κόμβος υπάρχει ήδη στη στοίβα αναδρομής (recStack). Αν ναι, αυτό σημαίνει ότι υπάρχει κύκλος στον γράφο και η μέθοδος επιστρέφει την τιμή του containsSpecialEdge. Αν ο κύκλος περιέχει ειδική ακμή, επιστρέφεται true, αλλιώς επιστρέφεται false. Στη συνέχεια, η μέθοδος ελέγχει αν ο κόμβος έχει ήδη επισκεφτεί (αν βρίσκεται στη λίστα visited). Αν έχει ήδη επισκεφτεί, η μέθοδος επιστρέφει false. Αν δεν έχει επισκεφτεί, προστίθεται στο σύνολο visited και στη στοίβα αναδρομής (recStack). Ακολούθως, η μέθοδος εξετάζει αναδρομικά όλους τους γείτονες του τρέχοντος κόμβου. Για κάθε γείτονα, η μέθοδος επαναλαμβάνει την αναδρομική κλήση με την ενημερωμένη τιμή του containsSpecialEdge, που ενημερώνεται σε true εάν εντοπιστεί κάποια ειδική ακμή στην πορεία. Αν κάποιος γείτονας προκαλεί έναν επικίνδυνο κύκλο, η μέθοδος επιστρέφει true, δηλώνοντας ότι έχει βρεθεί επικίνδυνος κύκλος. Εάν όλοι οι γείτονες εξεταστούν και δεν εντοπιστεί επικίνδυνος κύκλος, ο τρέχον κόμβος αφαιρείται από τη στοίβα αναδρομής και η μέθοδος επιστρέφει false.

Εσωτερική Κλάση Edge:

- Η κλάση Edge αναπαριστά μια ακμή στο γράφο. Κάθε ακμή έχει έναν προορισμό (to) και έναν τύπο (type).
- Η μέθοδος isSpecial() της κλάσης Edge ελέγχει αν η ακμή είναι ειδική.

4. Κλάση DependencyGraphBuilder

Η κλάση DependencyGraphBuilder έχει ως στόχο τη δημιουργία ενός εξαρτημένου γράφου από μια λίστα αντικειμένων τύπου SimpleLinearTGD. Ο εξαρτημένος γράφος αυτός περιλαμβάνει κόμβους που αναπαριστούν τους όρους στο σώμα και στην κεφαλή κάθε TGD, καθώς και ακμές που δηλώνουν τις εξαρτήσεις μεταξύ τους.

Μέθοδος buildDependencyGraph(List<SimpleLinearTGD> tgds): Η μέθοδος αυτή κατασκευάζει τον εξαρτημένο γράφο. Αρχικά, δημιουργεί ένα νέο αντικείμενο DependencyGraph, το οποίο περιέχει έναν πίνακα γειτνίασης (adjList) που θα αποθηκεύσει τις ακμές του γράφου. Στη συνέχεια, επεξεργάζεται κάθε SimpleLinearTGD που δίνεται ως όρισμα στη μέθοδο και για κάθε TGD, παίρνει το σώμα (body) και την κεφαλή (head), καθώς και τις μεταβλητές frontier και existential που προκύπτουν από το TGD. Για κάθε μεταβλητή του frontier, η μέθοδος εντοπίζει τις θέσεις της στο σώμα και στην κεφαλή του TGD. Όταν η μεταβλητή εμφανίζεται στο σώμα, δημιουργείται μια ακμή από τον αντίστοιχο κόμβο του σώματος προς τους κόμβους της κεφαλής που περιέχουν τη μεταβλητή αυτή και οι ακμές αυτές προστίθενται στον γράφο ως κανονικές ακμές. Όταν η μεταβλητή frontier εμφανίζεται στο σώμα, η μέθοδος αναζητά τις μεταβλητές του existential στην κεφαλή. Αν η μεταβλητή του existential εμφανίζεται σε κάποιον κόμβο της κεφαλής, δημιουργείται μια ειδική ακμή από τον κόμβο του σώματος προς τον αντίστοιχο κόμβο της κεφαλής. Μετά την επεξεργασία όλων των TGDs, η μέθοδος επιστρέφει το αντικείμενο depGraph, το οποίο περιλαμβάνει τις ακμές και τις συνδέσεις που δημιουργήθηκαν κατά τη διάρκεια της διαδικασίας.

Κεφάλαιο 4

Πειραματική Αξιολόγηση Αλγορίθμου

4.1 Περιγραφή Τρόπου Υλοποίησης

Για την υλοποίηση των πειραμάτων παίρνουμε τους χρόνους εκτέλεσης που παράγει το πρόγραμμα Java (χρόνοι ανάλυσης των TGDs, δημιουργίας του εξαρτημένου γράφου, ελέγχου ασθενούς ακυκλικότητας και συνολικός χρόνος εκτέλεσης) και τα εισάγουμε σε ένα πρόγραμμα Python ως λίστες. Χρησιμοποιούμε τη βιβλιοθήκη Matplotlib για να δημιουργήσουμε 4 scatter plots που αναπαριστούν τη σχέση μεταξύ του μεγέθους των TGDs και κάθε ένα από το χρόνο εκτέλεσης. Συνολικά θα δημιουργήσουμε 8 scatter plots διότι τα πειράματα θα αφορούν linear και simple linear tgds. Θα εστιάσουμε σε αρχεία με αριθμό tgds μέχρι 10000 για να έχουμε καλύτερα οπτικά αποτελέσματα. Η βιβλιοθήκη NumPy χρησιμοποιείται για να διαχειριστεί και να επεξεργαστεί τα φιλτράροντας τα δεδομένα από την αναπαράσταση τους. Οι γραφικές παραστάσεις αποτελούν αναλύσεις του χρόνου εκτέλεσης σε σχέση με το μέγεθος των δεδομένων

και παρέχουν σαφή εικόνα για την απόδοση του αλγορίθμου και την κατανόηση των χρόνων που απαιτούνται για τα διάφορα βήματα της διαδικασίας.

4.2 Εργαλεία Ανάπτυξης

4.2.1 Εργαλείο Visual Studio

Το Visual Studio είναι ένα πλήρες περιβάλλον ανάπτυξης (IDE) της Microsoft που χρησιμοποιείται για την ανάπτυξη λογισμικού σε πολλές γλώσσες προγραμματισμού, όπως C++, C#, Python και άλλες. Παρέχει προηγμένα εργαλεία για συγγραφή, εκτέλεση, αποσφαλμάτωση και βελτιστοποίηση του κώδικα.

4.2.2 Γλώσσα Προγραμματισμού Python

Η Python είναι μια ευρέως χρησιμοποιούμενη γλώσσα προγραμματισμού που είναι ιδιαίτερα δημοφιλής για την ανάλυση δεδομένων, τη μηχανική μάθηση, την επιστημονική υπολογιστική και την επεξεργασία μεγάλων συνόλων δεδομένων. Η Python είναι γνωστή για την ευχρηστία της, την αναγνωρίσιμη σύνταξη και την πληθώρα βιβλιοθηκών που επιτρέπουν την εύκολη εκτέλεση προηγμένων αναλύσεων.

4.2.2.1 Βιβλιοθήκη Matplotlib

Η Matplotlib είναι μια βιβλιοθήκη γραφημάτων για τη γλώσσα προγραμματισμού Python. Χρησιμοποιείται για τη δημιουργία διαγραμμάτων και γραφημάτων 2D και 3D.

4.2.2.2 Βιβλιοθήκη Numpy

Η Numpy είναι μια βιβλιοθήκη για τη γλώσσα Python που παρέχει υποστήριξη για πολυδιάστατους πίνακες και λειτουργίες με αριθμητικά δεδομένα. Είναι χρήσιμη για τη διαχείριση και την επεξεργασία μεγάλων δεδομένων.

4.3 Λεπτομέρειες Υλοποίησης

Ο κώδικας ξεκινά με την αρχικοποίηση των λιστών που περιλαμβάνουν τους χρόνους εκτέλεσης καθώς και τα αντίστοιχα μεγέθη των αρχείων, τα οποία προκύπτουν από την εκτέλεση του προγράμματος Java. Συγκεκριμένα, αρχικοποιούνται οι λίστες:

dataset_sizes, total_times, parsing_times, graph_creation_times και weak_acyclicity_check_times. Στην συνέχεια, ταξινομούνται οι χρόνοι βάσει του μεγέθους των συνόλων δεδομένων (dataset_sizes) σε αύξουσα σειρά. Ακολούθως, γίνεται ένα φιλτράρισμα των δεδομένων έτσι ώστε να εξετάσουμε μόνο τα αρχεία με μέγεθος μέχρι 10000 TGDs. Τέλος, με τη χρήση της βιβλιοθήκης Matplotlib κατασκευάζουμε ένα σύνολο 4 γραφημάτων (scatter plots) που τοποθετούνται σε έναν πίνακα 2x2. Κάθε γράφημα απεικονίζει τη σχέση μεταξύ το αριθμό των TGDs και των χρόνων εκτέλεσης για τα διάφορα στάδια (Total Time, Parsing Time, Graph Creation Time, Weak Acyclicity Check Time). Η συνάρτηση plt.subplots() χρησιμοποιείται για να δημιουργήσει το 2x2 πλέγμα των γραφημάτων. Για κάθε γράφημα, χρησιμοποιείται η μέθοδος scatter() για να δημιουργηθούν τα σημεία δεδομένων (σε μορφή κουκίδων) και η μέθοδος plot() για να συνδεθούν αυτά τα σημεία με γραμμές. Οι γραμμές αυτές βοηθούν στην αναγνώριση των τάσεων που προκύπτουν από τα δεδομένα. Κάθε γράφημα έχει τίτλο και ετικέτες για τους άξονες, χρησιμοποιώντας τις μεθόδους set_title(), set_xlabel() και set_ylabel(). Αφού δημιουργηθούν τα γραφήματα, η μέθοδος plt.tight_layout() διασφαλίζει ότι τα γραφήματα δεν επικαλύπτονται μεταξύ τους και μέθοδος plt.show() χρησιμοποιείται για την εμφάνιση των γραφημάτων στην οθόνη.

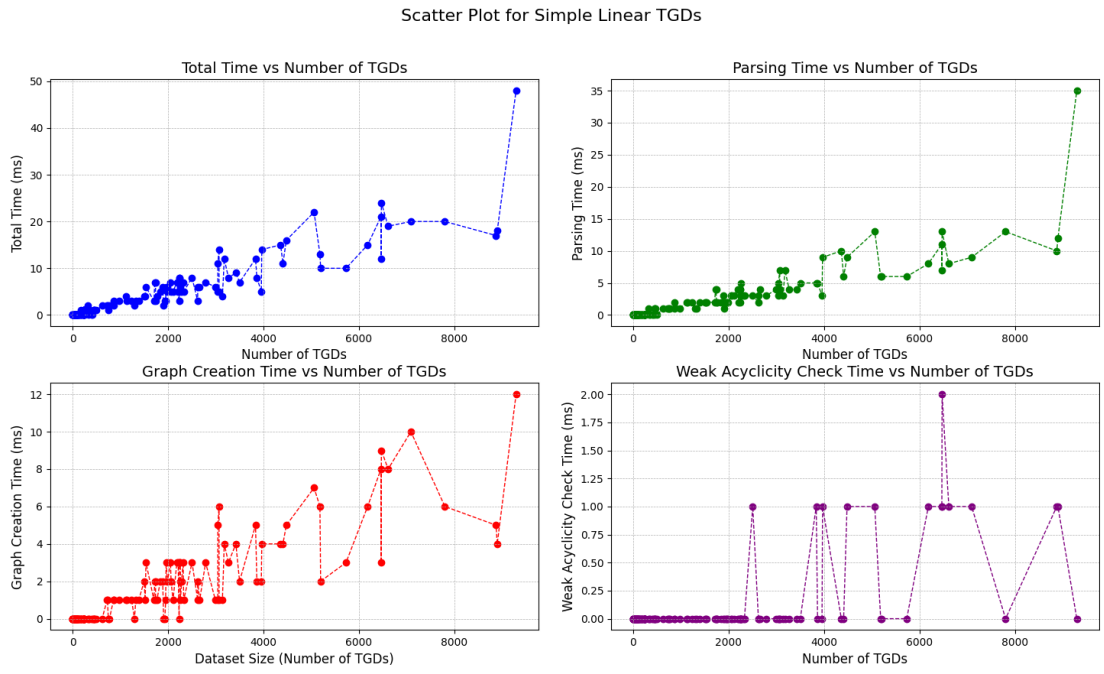
4.4 Αποτελέσματα Πειραμάτων

Τα πειράματα που πραγματοποιήθηκαν εστιάζουν στην πειραματική αξιολόγηση της απόδοσης του αλγορίθμου chase termination για τις δύο εκδοχές του, linear και simple linear TGDs. Το dataset που χρησιμοποιήσαμε αποτελείται από 179 αρχεία με TGDs. Κάθε αρχείο περιέχει κανόνες διαφορετικού μεγέθους και διαφορετικών τύπων. Οι χρόνοι εκτέλεσης μετρήθηκαν σε χιλιοστά του δευτερολέπτου (ms). Τα πειράματα εκτελέστηκαν σε υπολογιστή MacBook Air με επεξεργαστή Apple M2, που διαθέτει συνολικά 8 πυρήνες (4 υψηλής απόδοσης και 4 ενεργειακής απόδοσης), και 8 GB μνήμης RAM. Το σύστημα λειτουργούσε με το λειτουργικό macOS Ventura 13.5.1.

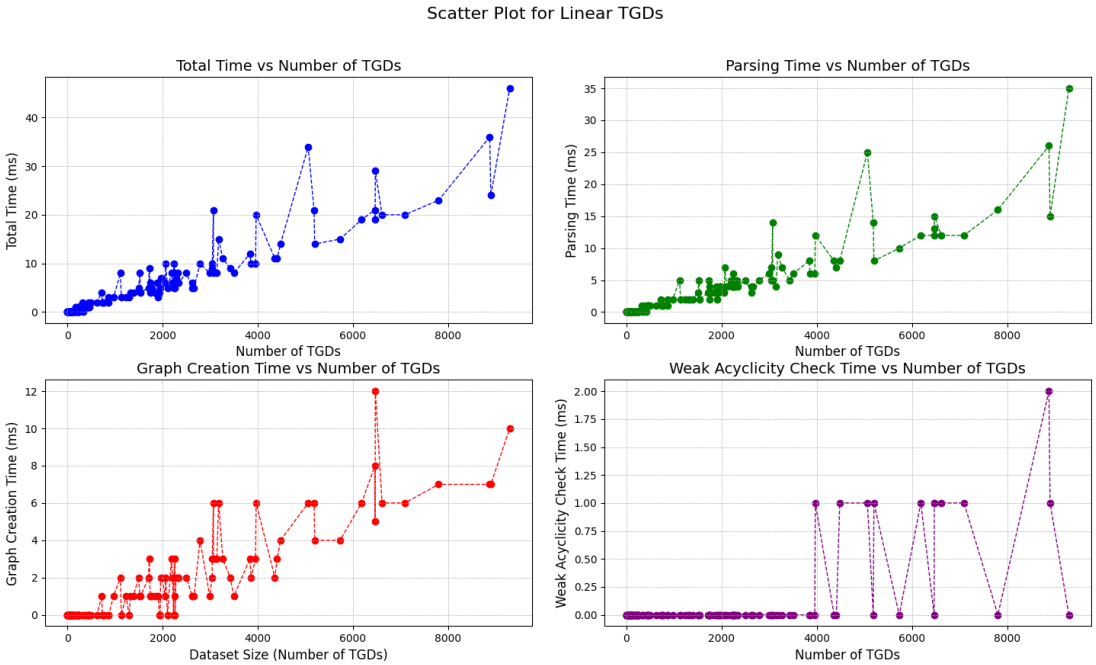
Συγκεκριμένα, εξετάστηκαν τα εξής στάδια της επεξεργασίας:

1. Parsing Time: Ο χρόνος που απαιτείται για την ανάλυση και την ανάγνωση των TGDs από τα αρχεία δεδομένων. Στην περίπτωση των linear TGDs, η διαδικασία περιλαμβάνει επιπλέον βήματα, όπως η εξειδίκευση (specialization) και η απλοποίηση (simplification).
2. Graph Creation Time: Ο χρόνος που απαιτείται για τη δημιουργία του Εξαρτημένου Γράφου (Dependency Graph). Ο γράφος αυτό αποτελεί βασικό εργαλείο για τον έλεγχο του weak acyclicity όπου θα μας πει αν τελικά ο αλγόριθμος τερματίζει ή όχι.
3. Weak Acyclicity Check Time: Ο χρόνος που απαιτείται για να ελεγχθεί αν ο εξαρτημένος γράφος είναι ασθενώς ακυκλικός (weakly acyclic). Αυτό είναι κρίσιμο για την ανίχνευση των επικίνδυνων κύκλων στον γράφο όπου θα οδηγήσουν σε μη τερματισμό του chase.
4. Total Time: Αφορά τον συνολικό χρόνο που απαιτείται για την ολοκλήρωση όλων των παραπάνω σταδίων για κάθε αρχείο από TGDs.

Για την αναπαράσταση των πειραμάτων υλοποιήθηκαν τα πιο κάτω scatter plots.



Σχήμα 5.1



Σχήμα 5.2

4.5 Σχολιασμός Αποτελεσμάτων

Στο Σχήμα 5.1 βλέπουμε τις γραφικές παραστάσεις για simple linear TGDs.

Όπως παρατηρούμε, ο χρόνος μετατροπής δεδομένων (Parsing Time), αποτελεί την πιο χρονοβόρα διαδικασία σε σχέση με το Graph Creation time και το Weak Acyclicity Check time. Αυτό συμβαίνει επειδή η συγκεκριμένη διαδικασία περιλαμβάνει τόσο την ανάγνωση και την ανάλυση του περιεχομένου του αρχείου όσο και τη δημιουργία των απαραίτητων εσωτερικών δομών για την αναπαράσταση των TGDs. Επίσης, για κάθε γραμμή του αρχείου ελέγχεται αν περιέχει ένα έγκυρο simple linear TGD. Παρατηρούμε ότι ο χρόνος εκτέλεσης για αυτή την διαδικασία είναι γραμμικός ως προς τον αριθμό των TGDs. Αυτό είναι κάτι που περιμέναμε αφού η ανάγνωση γίνεται μέσω BufferedReader έτσι κάθε γραμμή διαβάζεται ξεχωριστά. Η χρονική πολυπλοκότητα για την ανάγνωση των TGDs είναι O(n), όπου n ο αριθμός των γραμμών κάθε αρχείου.

Η συνάρτηση parseTGD() διαχειρίζεται την ανάλυση κάθε γραμμής, διασπώντας την σε body και head και στη συνέχεια αναλύοντας κάθε άτομο ξεχωριστά. Η πολυπλοκότητα για την ανάλυση του body και του head εξαρτάται από τον αριθμό των ατόμων και των μεταβλητών που περιέχονται σε αυτά. Για κάθε άτομο, η πολυπλοκότητα είναι O(m), όπου m το πλήθος των όρων σε κάθε άτομο. Επομένως, η θεωρητική γραμμική χρονική πολυπλοκότητα είναι κάτι που επιβεβαιώνεται.

Η δημιουργία του εξαρτημένου γραφήματος παρουσιάζει μια γενικά γραμμική αύξηση σχετικά με τον αριθμό των TGDs, αλλά υπάρχουν και κάποιες διακυμάνσεις στον χρόνο που μπορεί να οφείλονται στην πολυπλοκότητα κάθε TGD. Αυτό οφείλεται στη διαδικασία της αναζήτησης και δημιουργίας συνδέσεων μεταξύ των μεταβλητών του body και του head. Οι κανονικές ακμές δημιουργούνται συνδέοντας κάθε frontier μεταβλητή του body με κάθε εμφάνισή της στο head, ενώ οι ειδικές ακμές συνδέουν κάθε frontier μεταβλητή του body με κάθε υπαρξιακή μεταβλητή στο head. Αυτό απαιτεί επαναλήψεις που αυξάνονται με τον αριθμό των μεταβλητών και των συνδυασμών τους στα TGDs. Η θεωρητική πολυπλοκότητα για τη δημιουργία των κανονικών ακμών μπορεί να φτάσει το O(m^2) ανά TGD, όπου m ο αριθμός των μεταβλητών ανά TGD, αφού η διαδικασία απαιτεί την αναζήτηση και σύνδεση κάθε συνδυασμού μεταβλητών του body με τις αντίστοιχες μεταβλητές του head. Για τις ειδικές ακμές, η πολυπλοκότητα είναι O(me) ανά TGD, όπου e ο αριθμός των υπαρξιακών μεταβλητών ανά TGD. Συνεπώς, η συνολική πολυπλοκότητα για τη δημιουργία του γραφήματος είναι O(n(m^2 + me)), όπου n ο αριθμός των TGDs. Αυτή η ανάλυση επιβεβαιώνει ότι η πολυπλοκότητα αυξάνεται γραμμικά με τον αριθμό των TGDs, αλλά επίσης επηρεάζεται από την εσωτερική πολυπλοκότητα κάθε TGD λόγω της διαχείρισης των μεταβλητών.

Ο Χρόνος Ελέγχου Ασθενούς Ακυκλικότητας (Weak Acyclicity Check Time) παρατηρούμε ότι είναι αρκετά χαμηλός. Αυτό ήταν κάτι που περιμέναμε αφού ο αλγόριθμος DFS, που χρησιμοποιείται για την ανίχνευση επικίνδυνων κύκλων, έχει θεωρητική χρονική πολυπλοκότητα O(V+E), όπου V είναι ο αριθμός των κόμβων και E των ακμών στο γράφημα. Παρατηρούμε ότι ο χρόνος αυξάνεται με την πολυπλοκότητα του γράφου. Γενικά, ο αλγόριθμος είναι αποδοτικός στον εντοπισμό κρίσιμων κύκλων, ακόμη και για μεγάλα σύνολα δεδομένων.

Ο συνολικός χρόνος εκτέλεσης αυξάνεται αναμενόμενα καθώς μεγαλώνει το μέγεθος των TGDs, αλλά παραμένει σε γραμμικά πλαίσια, αποδεικνύοντας την αποδοτικότητα της διαδικασίας.

Στο Σχήμα 5.2 βλέπουμε τις γραφικές παραστάσεις για linear TGDs. Οι χρόνοι εκτέλεσης για τον έλεγχο ακυκλικότητας και την δημιουργία του εξαρτημένου γράφου έχουν παρόμοια τάση με τα simple linear TGDs. Παρατηρούμε ότι ο συνολικός χρόνος εκτέλεσης και ο χρόνος μετατροπής δεδομένων (Parsing Time) είναι εμφανώς μεγαλύτερος και αυτό οφείλεται στο επιπλέον βήμα της μετατροπής των linear TGDs σε simple linear με τη διαδικασία του specialization και simplification. Αυτό επιτυγχάνεται με τη δημιουργία όλων των δυνατών εξειδικεύσεων των μεταβλητών και την εφαρμογή τους στα TGDs. Αυτή η διαδικασία απαιτεί την αντιστοίχιση εξειδικευμένων τιμών στις αντίστοιχες μεταβλητές κάθε TGD, αυξάνοντας έτσι τον χρόνο ανάλυσης. Ωστόσο, παρατηρούμε ότι η διαφορά αυτή δεν είναι μεγάλη επομένως ο αλγόριθμος παραμένει αποδοτικός και στην εκδοχή με τα linear TGDs.

Συνολικά, τα αποτελέσματα των πειραμάτων επιβεβαιώνουν την αποδοτικότητα των αλγορίθμων και για τις δύο εκδοχές. Ο χρόνος μετατροπής δεδομένων, παρόλο που είναι ο πιο χρονοβόρος, διατηρείται γραμμικός ως προς τον αριθμό των TGDs. Ο χρόνος δημιουργίας ακολουθεί επίσης γραμμική τάση και ο χρόνος ελέγχου ασθενούς ακυκλικότητας παραμένει ιδιαίτερα χαμηλός λόγω της αποδοτικότητας του αλγορίθμου DFS. Τα Simple Linear TGDs απαιτούν λιγότερο χρόνο σε σχέση με τα Linear λόγω της απουσίας διαδικασιών εξειδίκευσης και απλοποίησης. Η γενική χρονική πολυπλοκότητα του αλγορίθμου είναι γραμμική, διασφαλίζοντας την αποδοτική εφαρμογή του σε μεγάλα σύνολα δεδομένων.

Κεφάλαιο 5

Συμπεράσματα

5.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία επικεντρώθηκε στην ανάπτυξη και πειραματική αξιολόγηση ενός αλγορίθμου για την επίλυση του προβλήματος τερματισμού της διαδικασίας Semi-oblivious chase για Simple Linear και Linear TGDs ελέγχοντας την ιδιότητα weak-acyclicity. Βασικός στόχος ήταν η ανίχνευση επικίνδυνων κύκλων στον εξαρτημένο γράφο, που ενδέχεται να οδηγήσουν σε μη τερματισμό.

Η πειραματική ανάλυση έδειξε ότι αλγόριθμος που υλοποιήθηκε αποδείχθηκε αποτελεσματικός στην επεξεργασία και τον εντοπισμό κρίσιμων κύκλων σε μεγάλα σύνολα δεδομένων TGDs. Ο χρόνος μετατροπής δεδομένων (parsing time) ήταν ο μεγαλύτερος, καθώς περιλάμβανε την ανάγνωση και ανάλυση των TGDs. Ωστόσο, παρέμεινε αρκετά γρήγορος, γεγονός που υποδεικνύει τη αποδοτικότητα του αλγορίθμου. Επιπλέον, ο χρόνος δημιουργίας του εξαρτημένου γράφου (graph creation time) αυξανόταν αναμενόμενα με την πολυπλοκότητα των TGDs αλλά παρέμεινε γραμμικός. Σημαντικό είναι το γεγονός ότι ο χρόνος εντοπισμού κρίσιμων κύκλων (weak acyclicity check time) στον εξαρτημένο γράφο ήταν εξαιρετικά μικρός, αποδεικνύοντας την ικανότητα του αλγορίθμου να εντοπίζει κρίσιμους κύκλους γρήγορα και αποδοτικά. Τα αποτελέσματα δείχνουν ότι το weak-acyclicity αποτελεί βασικό παράγοντα για τον επιτυχή τερματισμό της διαδικασίας Semi-oblivious chase.

Παράλληλα, παρατηρήθηκε ότι οι χρόνοι εκτέλεσης και ειδικά το parsing ήταν μεγαλύτεροι στις περιπτώσεις linear TGDs σε σύγκριση με τα simple linear TGDs,

λόγω της διαδικασίας εξειδίκευσης (specialization) και απλοποίησης (simplification) που απαιτείται για την μετατροπή τους. Η επιπρόσθετη αυτή διαδικασία αυξάνει τον υπολογιστικό φόρτο, με το parsing time να είναι εμφανώς υψηλότερο για τα linear TGDs. Παρά το γεγονός αυτό, όλοι οι χρόνοι εκτέλεσης παραμένουν αρκετά καλοί.

Συμπερασματικά, ο αλγόριθμος επιβεβαιώθηκε ως αποτελεσματικός, προσφέροντας γρήγορους χρόνους επεξεργασίας ακόμη και σε μεγάλα σύνολα δεδομένων.

Βιβλιογραφία

[1] "Matplotlib: Visualization with Python". [Ηλεκτρονικό]. Διαθέσιμο από: https://matplotlib.org

[2] "IntelliJ IDEA: The Java IDE for Professional Developers". [Ηλεκτρονικό]. Διαθέσιμο από: https://www.jetbrains.com/idea/

[3] "NumPy: The fundamental package for scientific computin with Python". [Ηλεκτρονικό]. Διαθέσιμο από: https://numpy.org

[4] Marco Calautti & Georg Gottlob & Andreas Pieris. "Chase Termination for Guarded Existential Rules". PDF File.

[5] Marco Calautti & Georg Gottlob & Andreas Pieris. "Non-Uniformly Terminating Chase: Size and Complexity"

[6] "Visual Studio Code — Code Editing. Redefined". [Ηλεκτρονικό]. Διαθέσιμο από: https://code.visualstudio.com/