ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΩΝ ΕΞΟΡΥΞΗΣ ΓΝΩΣΗΣ ΑΠΟ ΚΕΙΜΕΝΑ ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ SPARK

Χαροκόπειο Πανεπιστήμιο
Τμήμα Πληροφορικής και Τηλεματικής

Ιωάννης Κοντόπουλος

Επιβλέπων Καθηγητής
Ηρακλής Βαρλάμης

ΙΩΑΝΝΗΣ ΚΟΝΤΟΠΟΥΛΟΣ

2016

Μέλη τριμελούς επιτροπής

Μέλος 1ο: κ. Ηρακλής Βαρλάμης

Μέλος 2ο: κ. Δημήτριος Μιχαήλ

Μέλος 3ο: κα Μάρα Νικολαΐδου

ΙΩΑΝΝΗΣ ΚΟΝΤΟΠΟΥΛΟΣ

Η συγκεκριμένη πτυχιακή εργασία υλοποιήθηκε στο πλαίσιο των πτυχιακών εργασιών της περιόδου 2015-2016 για το τμήμα Πληροφορικής και Τηλεματικής του Χαροκοπείου Πανεπιστημίου Αθηνών. Απαγορεύεται ρητά κάθε προσπάθεια αντιγραφής, διανομής και χρήσης της, ενώ ενδείκνυται η χρήση της ως αναδιατυπωμένο υλικό σε εκπαιδευτικού, ερευνητικού και μη κερδοσκοπικού χαρακτήρα έργα πάντοτε με την επισήμανση αυτού του μηνύματος και του αρχικού τίτλου και συγγραφέα της εργασίας

Περιεχόμενα

Πίνακας Διαγραμμάτων

Πίνακες

Ευχαριστίες

Η συγκεκριμένη έρευνα έγινε στα πλαίσια της πτυχιακής μου στο Χαροκόπειο Πανεπιστήμιο και είναι συνέχεια της πρακτικής μου στο Ε.Κ.Ε.Φ.Ε. «Δημόκριτος».

Θέλω να ευχαριστήσω τον κ. Ηρακλή Βαρλάμη για τη βοήθειά του κατά τη διάρκεια της ανάπτυξης της πτυχιακής. Τις συμβουλές του πάνω στα θέματα της δημιουργίας εφαρμογής για την εξαγωγή περιλήψεων σε κατανεμημένες πλατφόρμες και την υπομονή του για τη διόρθωση της πτυχιακής αυτής μέσα σε όλες τις υποχρεώσεις του. Επίσης, θέλω να ευχαριστήσω τον κ. Γιώργο Γιαννακόπουλο και τη SciFY για την παροχή των υπολογιστών για τα πειράματα και την υπομονή που έδειξαν για όσο χρόνο διήρκησαν τα πειράματα αυτά.

1. Εισαγωγή

Από την αρχή των υπολογιστών, η κοινότητα της πληροφορικής έχει δείξει ενδιαφέρον στο να εξάγει περιλήψεις από κείμενα, είτε πρόκειται για επιστημονικά κείμενα, είτε για ειδησεογραφικά. Υπάρχουν δύο προσεγγίσεις στις αυτόματες περιλήψεις: extraction και abstraction. Οι μέθοδοι που χρησιμοποιούν την πρώτη προσέγγιση λειτουργούν επιλέγοντας ένα τμήμα από υπάρχουσες λέξεις, φράσεις ή προτάσεις από το αρχικό κείμενο για να σχηματίσουν την περίληψη. Αντίθετα, οι τεχνικές που ακολουθούν τη δεύτερη προσέγγιση, χτίζουν μια σημασιολογική αναπαράσταση και μετά χρησιμοποιούν τεχνικές δημιουργίας φυσικής γλώσσας για να δημιουργήσουν μια περίληψη που μοιάζει αρκετά σε μία περίληψη που θα έγραφε ένας άνθρωπος. Τέτοιες περιλήψεις μπορεί να περιέχουν λέξεις που δεν υπάρχουν στο αρχικό κείμενο. Η έρευνα για τις abstractive μεθόδους έχει αυξανόμενη σημαντικότητα και είναι ένα ενεργό πεδίο έρευνας, αλλά λόγω περιορισμών πολυπλοκότητας, η έρευνα μέχρι σήμερα έχει εστιάσει κυρίως στις extractive τεχνικές. Η τεχνική που έχει αναπτυχθεί στα πλαίσια αυτή της πτυχιακής είναι extractive.

Αυξανόμενο ενδιαφέρον, επίσης, έχει εμφανιστεί, όσον αφορά την εξαγωγή περιλήψεων από πολλαπλά κείμενα. Αυτή όμως η εξαγωγή έχει κάποια ζητήματα που πρέπει να λυθούν, όπως τον εντοπισμό και την επιλογή κατάλληλης πληροφορίας, τη «συμπίεση» των κειμένων χωρίς να χαθεί πληροφορία και πως επιβεβαιώνεται ότι η τελική περίληψη δεν θα έχει επαναλαμβανόμενη πληροφορία. Τέλος, ένα ακόμη ζήτημα έγκειται στο γεγονός ότι μπορεί να υπάρχουν κείμενα από διάφορες γλώσσες και άρα πρέπει να αναπτυχθεί μία τεχνική που είναι ανεξάρτητη της γλώσσας.

Υπάρχουν όμως και άλλα ζητήματα που πρέπει να λυθούν. Με τη συνεχώς αυξανόμενη δημιουργία δεδομένων αλλά και τη συσσώρευση αυτών, όπως για παράδειγμα κοινωνικά δίκτυα, δίκτυα αισθητήρων, αυξανόμενος αριθμός ειδήσεων κλπ., υπάρχει ανάγκη για τεχνικές που θα εκμεταλλευτούν περισσότερη επεξεργαστική ισχύ όταν αυτή είναι διαθέσιμη. Αρκετά εργαλεία που χρησιμοποιούν παράλληλη ή/και κατανεμημένη επεξεργασία έχουν αναπτυχθεί για να αντιμετωπίσουν ακριβώς αυτό το πρόβλημα και να μπορέσουν να μειώσουν το χρόνο εκτέλεσης ή να αυξήσουν την αποδοτικότητα των εφαρμογών.

Η παράλληλη επεξεργασία αναφέρεται στη μέθοδο όπου το ίδιο πρόβλημα μπορεί να διαιρεθεί σε μικρότερα και κάθε μικρό πρόβλημα μπορεί να εκτελεστεί ταυτόχρονα, μειώνοντας έτσι το χρόνο εκτέλεσης. Από την αρχή των μηχανών πολλών πυρήνων, εφαρμογές έχουν αναπτυχθεί για να χρησιμοποιούν όλους τους πυρήνες και να εκμεταλλεύονται όσο περισσότερη υπολογιστική ισχύ γίνεται σε ένα μηχάνημα. Παρά το γεγονός αυτό, με τη συνεχόμενη αύξηση των δεδομένων, είναι απαραίτητη μία τεχνική που θα μπορεί να εκμεταλλευτεί την υπολογιστική ισχύ πολλών μηχανημάτων. Η κατανεμημένη επεξεργασία είναι μία τέτοια τεχνική, όπου εφαρμογές μπορούν να εκμεταλλευτούν κάθε πυρήνα από κάθε υπολογιστή που είναι διαθέσιμος σε ένα δίκτυο ή σε ένα cluster.

Αυτή η πτυχιακή, ξεκινάει αναλύοντας και εξηγώντας την αναπαράσταση γράφων ν-γραμμάτων και τους αλγόριθμους που σχετίζονται με αυτούς. Στη συνέχεια και χτίζοντας πάνω στις τεχνικές γράφων ν-γραμμάτων, παρουσιάζεται μία τεχνική για εξαγωγή περιλήψεων, η οποία είναι ανεξάρτητη της γλώσσας των κειμένων, επιλέγει την καταλληλότερη πληροφορία από τα κείμενα και εξάγει προτάσεις από τα κείμενα χωρίς να επαναλαμβάνει την πληροφορία που υπάρχει. Ύστερα, παρέχεται και προτείνεται μία κατανεμημένη υλοποίηση των γράφων ν-γραμμάτων και της τεχνικής εξαγωγής περίληψης, αναλύοντας σχηματικά πως επιτυγχάνεται αυτή. Έπειτα, δίνονται παραδείγματα χρήσης αυτής της εφαρμογής, αναφερόμενη στους προγραμματιστές και εξηγώντας τη χρήση πολλών μεθόδων που υπάρχουν στην εφαρμογή αυτή. Τέλος, μελετάται η

κλιμάκωση της εφαρμογής που αναπτύχθηκε σε πολλά μηχανήματα και πυρήνες όσον αφορά το μέγεθος των κειμένων (μεγάλα δεδομένα) και όχι το πλήθος.

2. Σχετική Δουλειά

Στο παρελθόν, πολλοί έχουν προσπαθήσει να εξάγουν αυτόματα περιλήψεις από κείμενα, είτε από ένα (single document summarization), είτε από πολλά κείμενα (multi-document summarization). Οι περισσότερες τεχνικές χρησιμοποιούν τη συχνότητα εμφάνισης λέξεων (Term Frequency) και το πόσο σημαντικές είναι οι λέξεις σε ένα σύνολο κειμένων (Inverse Document Frequency), δηλαδή χρησιμοποιούν στατιστικές μεθόδους ή μεθόδους μηχανικής μάθησης (Das & Martins, 2007). Οι τεχνικές μηχανικής μάθησης συνήθως αφορούν αλγορίθμους Naïve Bayes, Δέντρα Απόφασης (Decision Trees), Νευρωνικά Δίκτυα (Neural Networks). Η διαφορά στην έρευνα της πτυχιακής με τις άλλες δουλειές είναι ότι χρησιμοποιεί τα ν-γράμματα χαρακτήρων, τη γειτνίασή τους και τους αλγορίθμους που αφορούν τους γράφους ν-γραμμάτων, χωρίς μεθόδους μηχανικής μάθησης για τον εντοπισμό της κατάλληλης πληροφορίας.

Ένα ακόμη ενδιαφέρον θέμα είναι η εξαγωγή καλής περίληψης όταν πρόκειται για κείμενα διαφορετικών τύπων. Διάφορες έρευνες έχουν γίνει πάνω σε αυτό το πρόβλημα (Elfayoumy & Thoppil, 2014). Το πλεονέκτημα των τεχνικών των γράφων ν-γραμμάτων είναι ότι έχουν δείξει καλά αποτελέσματα πάνω σε διαφορετικούς τύπους κειμένων (Giannakopoulos, Mavridi, Paliouras, Papadakis, & Tserpes, 2012) ακόμη και σε προβλήματα classification. Άρα είναι λογικό να εξεταστούν και τα αποτελέσματα αυτών σε προβλήματα αυτόματων περιλήψεων.

Σε αυτήν την πτυχιακή όμως δεν εξετάζεται τόσο η αποτελεσματικότητα των γράφων ν-γραμμάτων στις περιλήψεις αλλά η κατανεμημένη επεξεργασία της διαδικασίας εξαγωγής περίληψης και η κλιμάκωσή της με την προσθήκη περισσότερων πυρήνων. Μία προσπάθεια κατανεμημένης εξαγωγής περίληψης[1] έχει γίνει χρησιμοποιώντας τον αλγόριθμο Lex Rank ο οποίος βασίζεται σε γράφους (Erkan & Radev, 2004). Σε αυτήν τη μέθοδο επίσης, εντοπίζονται στερεότυπες προτάσεις χρησιμοποιώντας την τεχνική Locality Sensitive Hash Signatures (Van Durme & Lall, 2010). Μία από τις διαφορές είναι ότι σε αυτό εξάγονται πάντα περιλήψεις μεγέθους 5 προτάσεων, χάνοντας έτσι σημαντική πληροφορία ειδικά όταν πρόκειται για μεγάλα δεδομένα.

[1] https://github.com/karlhigley/lexrank-summarizer

3. Θεωρητικό Υπόβαθρο

Σε αυτό το κεφάλαιο θα αναλυθούν οι γράφοι ν-γραμμάτων, πως δημιουργούνται, πως μετριέται η ομοιότητα μεταξύ τους και τι λειτουργίες μπορούν να γίνουν σε αυτούς (Giannakopoulos, Automatic Summarization from Multiple Documents, 2009). Στο επόμενο κεφάλαιο θα εξηγηθεί πως με αυτές τις λειτουργίες που περιγράφονται μπορούν να εξαχθούν οι περιλήψεις από τα κείμενα.

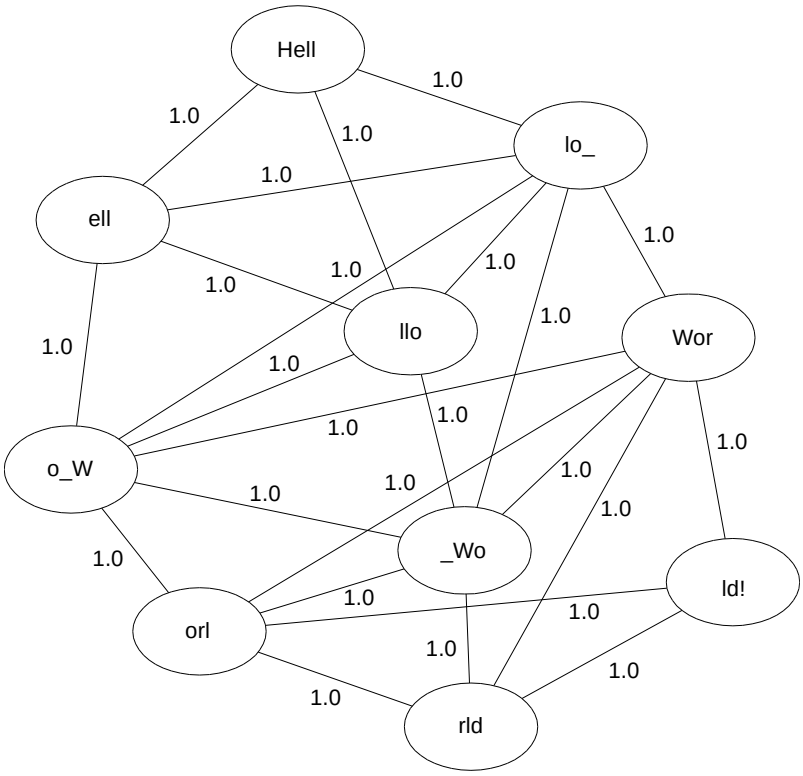
3.1 Αναπαράσταση Γράφων Ν-Γραμμάτων

Πριν αναφερθούν λεπτομέρειες για την παράλληλη και κατανεμημένη επεξεργασία, θα εξηγηθούν τα ν-γράμματα, η αναπαράσταση των γράφων ν-γραμμάτων και οι αλγόριθμοι γύρω από αυτούς τους γράφους που τους κάνουν εφαρμόσιμους σε εξαγωγή αυτόματων περιλήψεων. Στη βασική τους χρήση οι γράφοι ν-γραμμάτων αναπαριστούν κείμενο. Με δεδομένη την αναπαράσταση ενός κειμένου σαν μία ακολουθία χαρακτήρων, τα ν-γράμματα εξάγονται με τον εξής τρόπο:

Κείμενο εισόδου: Hello World!
Εξαγόμενα ν-γράμματα: Hel, ell, llo, lo_, o_W, _Wo, Wor, orl, rld, ld!

Για λόγους κατανόησης, στα ν-γράμματα τα κενά έχουν αντικατασταθεί με κάτω παύλες. Στο παραπάνω παράδειγμα το μέγεθος του ν είναι 3 και πλέον ονομάζονται τριγράμματα. Δημιουργούνται χρησιμοποιώντας ένα «συρόμενο παράθυρο» που μετακινείται μόνο έναν χαρακτήρα κάθε φορά για κάθε ν-γραμμα. Το μέγεθος του ν δεν είναι σταθερό αλλά μπορούν να υπάρξουν διγράμματα ($\nu = 2$), τριγράμματα ($\nu = 3$) κ. ο. κ. Αυτά τα ν-γράμματα πλέον χρησιμοποιούνται σαν κόμβοι του γράφου που είναι και μοναδικοί. Αν δηλαδή βρεθεί κάποιο ν-γραμμα πολλές φορές, ο γράφος θα έχει μόνο έναν κόμβο που θα αντιπροσωπεύει αυτό το ν-γραμμα.

Δεδομένου του κειμένου του παραπάνω παραδείγματος, οι ακμές του γράφου μπορούν τώρα να δημιουργηθούν. Οι ακμές θα δημιουργηθούν με βάση ένα μέγεθος παραθύρου, *dwin*. Συνδέοντας τα τριγράμματα με μέγεθος παραθύρου $dwin = 3$, οι ακμές δημιουργούνται ως εξής:
Hel – ell, Hel – llo, Hel – lo_, ell – llo, ell – lo_, ell – o_W, κ. ο. κ. Κάθε τρίγραμμα συνδέεται με τα επόμενα 3 τριγράμματα ($dwin = 3$) για να δημιουργήσουν μοναδικές, μη κατευθυνόμενες ακμές. Κάθε ακμή έχει βάρος, το οποίο παίζει σημαντικό ρόλο στους αλγορίθμους που θα παρουσιαστούν αργότερα και δείχνει τη γειτνίαση μεταξύ τους, δηλαδή το πόσο κοντά βρίσκεται το ένα σε σχέση με το άλλο. Κάθε βάρος αντιστοιχεί σε συχνότητα εμφάνισης της κάθε ακμής μέσα στο κείμενο. Αν η ακμή, Hel – ell, εμφανιστεί στο κείμενο δύο φορές θα έχει βάρος δύο (2,0). Στο Σχήμα 1 φαίνεται πως μοιάζει ο γράφος που δημιουργήθηκε από το παραπάνω παράδειγμα.



Σχήμα 1: Η αναπαράσταση του γράφου του κειμένου "Hello World!"

3.2 Μέτρα Ομοιότητας Γράφων N-Γραμμάτων

Δεδομένων δύο γράφων $G_1<V_1,E_1>$, $G_2<V_2,E_2>$, μπορούν να εφαρμοστούν ορισμένες λειτουργίες και ορισμένοι αλγόριθμοι πάνω σε αυτούς, εκ των οποίων μία πολύ σημαντική λειτουργία είναι η σύγκριση μεταξύ τους. Υπάρχουν τέσσερα μέτρα ομοιότητας: ομοιότητα μεγέθους (size similarity), ομοιότητα περιεκτικότητας (containment similarity), ομοιότητα τιμής (value similarity), και κανονικοποιημένη ομοιότητα τιμής (normalized value similarity). Αυτά τα μέτρα ορίζονται ως εξής:

Αν |V| είναι το μέγεθος του γράφου (αριθμός ακμών) τότε η **ομοιότητα μεγέθους** των $G_1<V_1,E_1>$, $G_2<V_2,E_2>$ είναι:

$$SS = \frac{\min\ (|V1|, |V2|)}{\max\ (|V1|, |V2|)}$$

Η **ομοιότητα περιεκτικότητας** (CS) των $G_1<V_1,E_1>$, $G_2<V_2,E_2>$: Κάθε κοινή ακμή προσθέτει

$$\frac{1}{\min\ (|V1|, |V2|)}$$

σε ένα άθροισμα.

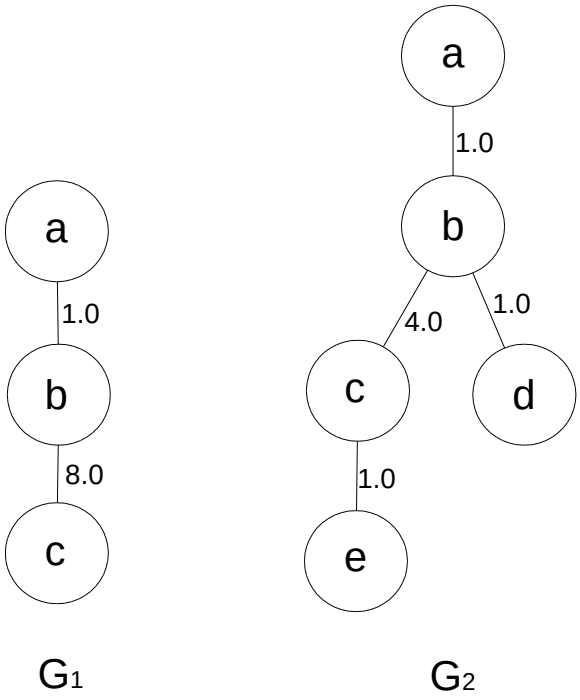
Η **ομοιότητα τιμής** (VS) των $G_1<V_1,E_1>$, $G_2<V_2,E_2>$: Κάθε κοινή ακμή προσθέτει

$$\frac{\frac{\min(w_{E1}, w_{E2})}{\max(w_{E1}, w_{E2})}}{\max\ (|V1|, |V2|)}$$

σε ένα άθροισμα, όπου w_{E1} είναι το βάρος της ακμής του $G_1<V_1,E_1>$ και w_{E2} είναι το βάρος της ακμής του $G_2<V_2,E_2>$. Τα ελάχιστα και μέγιστα βάρη που χρησιμοποιούνται στην εξίσωση είναι μόνο από τις κοινές ακμές και όχι από όλες τις ακμές. Τέλος, η **κανονικοποιημένη ομοιότητα τιμής** (NVS) είναι το πηλίκο των δύο προηγούμενων ομοιοτήτων:

$$NVS = \frac{VS}{SS}$$

Οι τιμές των ομοιοτήτων κυμαίνονται από 0 έως και 1 ($0 \leq similarity \leq 1$). Μία ομοιότητα τιμής 1 αντιστοιχεί σε πλήρη ομοιότητα των δύο γράφων. Παρακάτω φαίνεται ένα παράδειγμα για τον υπολογισμό ομοιοτήτων.



Σχήμα 2: Παράδειγμα δύο γράφων

Στο Σχήμα 2 φαίνονται δύο γράφοι από τους οποίους θα εξαχθούν οι προαναφερθείσες ομοιότητες και για τους οποίους έχουμε:

$$SS = \frac{2}{4} = 0.5 \text{ ή } 50\%$$

$$CS = \frac{1}{2} + \frac{1}{2} = 1.0 \text{ ή } 100\%$$

$$VS = \frac{\frac{1.0}{1.0}}{4} + \frac{\frac{4.0}{8.0}}{4} = \frac{1}{4} + \frac{1}{8} = 0.375 \text{ ή } 37.5\%$$

$$NVS = \frac{VS}{SS} = \frac{0.375}{0.5} = 0.75 \text{ ή } 75\%$$

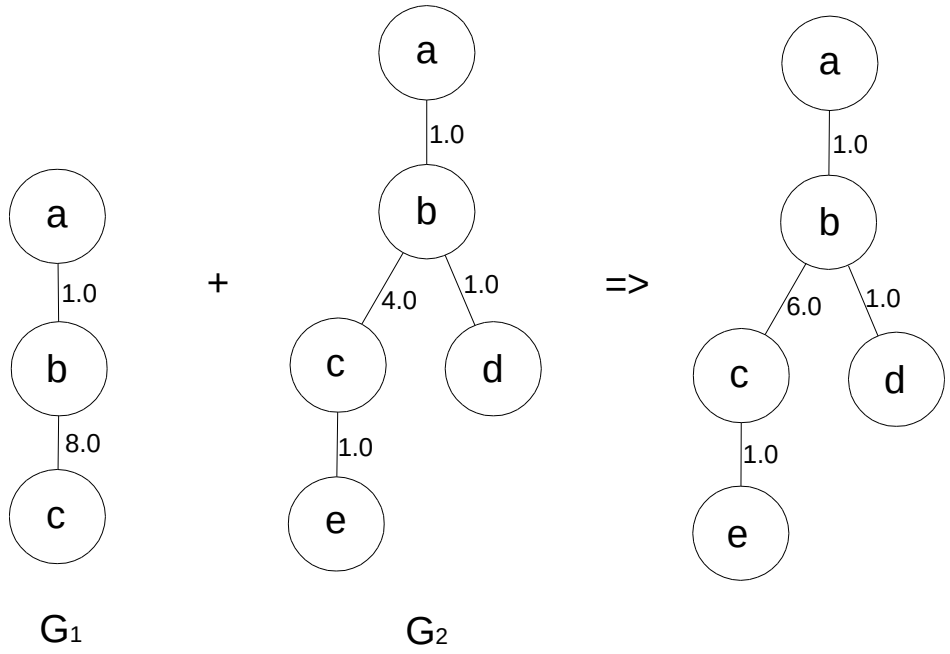
3.3 Αλγόριθμοι Γράφων N-Γραμμάτων

Δεδομένων δύο γράφων ν-γραμμάτων G_1, G_2, υπάρχουν τέσσερεις δυαδικές λειτουργίες που μπορούν να εφαρμοστούν μεταξύ τους και αυτές είναι οι: merge (or union), intersection, inverse intersection και λειτουργία delta (all-not-in) (Giannakopoulos, Automatic Summarization from Multiple Documents, 2009).

Η λειτουργία ένωσης δύο γράφων (merge), επιστρέφει έναν νέο γράφο που περιέχει όλες τις ακμές και από τους δύο, εφαρμόζοντας μία συνάρτηση μέσου όρου για τα βάρη των κοινών ακμών:

$$w_{Em} = w_{E1} + L(w_{E2} - w_{E1}),$$

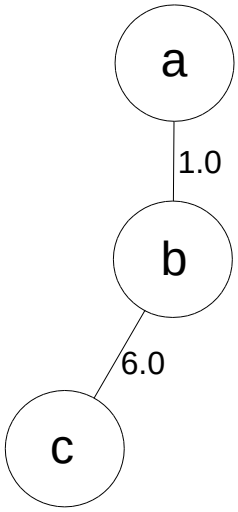
όπου η μεταβλητή L, $(0 \leq L \leq 1)$, είναι ο παράγοντας εκμάθησης (learning factor). Αυτός ο παράγοντας υποδεικνύει ποιο από τα δύο βάρη της κοινής ακμής πρέπει να ληφθεί περισσότερο υπόψιν. Η τιμή $L = 0.5$ υποδεικνύει τον μέσο όρο.



Σχήμα 3: Η ένωση των γράφων του Σχήματος 2 με τιμή L = 0.5

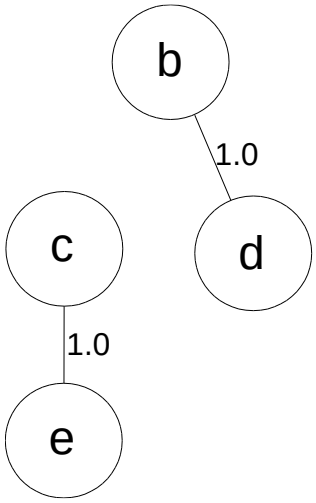
Στο Σχήμα 3 φαίνεται ο νέος γράφος που προκύπτει αν εφαρμόσουμε τη λειτουργία ένωσης (merge operator) στους δύο γράφους του Σχήματος 2. Ακριβώς επειδή η τιμή του L είναι 0.5 η κοινή ακμή $b - c$ έχει σαν βάρος τον μέσο όρο των δύο τιμών των προηγούμενων γράφων.

Η λειτουργία διασταύρωσης (intersect operator) επιστρέφει έναν νέο γράφο ο οποίος περιέχει μόνο τις κοινές ακμές των δύο προηγούμενων. Πάλι στα βάρη των ακμών αυτών εφαρμόζεται η ίδια συνάρτηση μέσου όρου όπως στην προηγούμενη λειτουργία, βλέπε Σχήμα 4.



Σχήμα 4: Η διασταύρωση των γράφων του Σχήματος 2 με L = 0.5

Η λειτουργία της αντίστροφης διασταύρωσης (inverse intersection) επιστρέφει έναν γράφο που περιέχει τις μη κοινές ακμές από τους δύο. Στο Σχήμα 5 φαίνεται αυτός ο γράφος αν εφαρμοστεί αυτή η λειτουργία στους γράφους του Σχήματος 2.



Σχήμα 5: Η αντίστροφη διασταύρωση των γράφων του Σχήματος 2

Τέλος, η λειτουργία δέλτα (delta operator) επιστρέφει έναν γράφο που περιέχει τις ακμές του πρώτου γράφου που δεν περιέχονται στον δεύτερο. Ανάλογα με το ποιος γράφος θα θεωρηθεί πρώτος ή δεύτερος αλλάζει και το αποτέλεσμα της λειτουργίας αυτής. Αν για παράδειγμα, σαν πρώτος γράφος θεωρηθεί ο G_1, από το Σχήμα 2, τότε θα επιστραφεί ένας κενός γράφος χωρίς ακμές εφόσον όλες οι ακμές του G_1 περιέχονται στον G_2. Αν θεωρηθεί αντίστροφα σαν πρώτος γράφος ο G_2 τότε θα επιστραφεί ο γράφος του Σχήματος 5.

4. Εξαγωγή Περιλήψεων

Σε αυτό το κεφάλαιο θα αναλυθεί η διαδικασία με την οποία χρησιμοποιώντας γράφους ν-γραμμάτων μπορούν να εξαχθούν περιλήψεις κειμένων. Η μέθοδος που παρουσιάζεται στην ουσία καταλήγει σε ένα σύνολο προτάσεων που καλύπτουν το νόημα των κειμένων, χωρίς επαναλαμβανόμενη πληροφορία. Η διαδικασία που ακολουθείται πρώτα ομαδοποιεί τα κείμενα, σε κάθε ομάδα κειμένου, εξάγονται οι προτάσεις και ομαδοποιούνται. Ύστερα, με τη χρήση των γράφων δημιουργείται το νόημα του κειμένου με βάση το οποίο μπορούν να αναγνωριστούν οι προτάσεις που το καλύπτουν περισσότερο. Στις επόμενες ενότητες περιγράφεται αναλυτικά η διαδικασία που ακολουθείται για τη δημιουργία του νοήματος και πως ορίζουμε ότι μία πρόταση το καλύπτει περισσότερο.

4.1 Ομαδοποίηση Κειμένων

Έχοντας σαν είσοδο ένα σύνολο από κείμενα από μία κατηγορία (π.χ. πολιτική, οικονομία κλπ.), πρέπει αυτά να ομαδοποιηθούν σε γεγονότα. Αυτή η ομαδοποίηση είναι απαραίτητη γιατί μία περίληψη βγάζει περισσότερο νόημα ανά γεγονός παρά σε όλα τα γεγονότα μεταξύ τους (Harabagiu & Lacatusu, 2005).

Η ιδέα είναι να χρησιμοποιηθεί μία γρήγορη, βασισμένη στην ομοιότητα διαδικασία στην εξαγωγή γεγονότων. Αυτό που μας ενδιαφέρει περισσότερο είναι τα κείμενα να βρίσκονται στην κατάλληλη ομάδα, δίνοντας έμφαση στην ακρίβεια (precision) και όχι τόσο στην ανάκληση (recall). Δηλαδή, δεν πειράζει αν μία ομάδα δεν περιέχει κάποιο κείμενο, αλλά πειράζει αν μία ομάδα περιέχει ένα «άσχετο» κείμενο. Το κείμενο που μπορεί να χαθεί από κάποια ομάδα θα βρεθεί σε μία άλλη (πιθανότατα μόνο του).

Η διαδικασία ομαδοποίησης ακολουθεί τα εξής στάδια: Πρώτα, προ-επεξεργάζεται το κείμενο, συμπεριλαμβανομένου και του τίτλου και κρατούνται μόνο οι λέξεις που αρχίζουν με κεφαλαίο γράμμα και οι αριθμοί σε μία προσπάθεια να εντοπιστεί η πληροφορία που υπάρχει στο κείμενο με τη μορφή αριθμών ή κυρίων ονομάτων. Αυτό υποθέτει ότι αυτές οι λέξεις και οι αριθμοί φανερώνουν το νόημα της πληροφορίας που υπάρχει σε ένα κείμενο. Το αποτέλεσμα αυτής της διαδικασίας είναι μία σειρά από λέξεις και αριθμούς. Για παράδειγμα, δεδομένης της πρότασης “U.S. tells North Korea new missile launch would be a huge mistake”, το αποτέλεσμα θα είναι <”U.S.”, “North”, “Korea”>.

Στη συνέχεια χρησιμοποιείται γράφος ν-γραμμάτων για να αναπαραστήσουμε το αποτέλεσμα του προηγούμενο βήματος, με κάθε ν-γραμμα όμως να αντιστοιχεί σε μία λέξη ($\nu = 2$, $dwin = 3$). Αυτό δείχνει ότι και ο τρόπος με τον οποίο οι λέξεις και οι αριθμοί βρίσκονται και γειτονεύουν στο κείμενο είναι σημαντικός, όχι μόνο οι λέξεις και οι αριθμοί (Giannakopoulos, Automatic Summarization from Multiple Documents, 2009).

Με βάση αυτούς τους γράφους, συγκρίνονται όλα τα πιθανά ζευγάρια κειμένων. Για να αναγνωριστεί αν δύο κείμενα αναφέρονται στο ίδιο γεγονός χρησιμοποιείται ο εξής κανόνας: η κανονικοποιημένη ομοιότητα τιμής (normalized value similarity) πρέπει να είναι μεγαλύτερη του 0,2 (20%) και η ομοιότητα μεγέθους (size similarity) πρέπει να είναι μεγαλύτερη του 0,1 (10%). Αυτή η ευριστική (που έχει προκύψει από πειράματα προηγούμενων ερευνών (Giannakopoulos, Kioumourtzis, & Karkaletsis, NewSum: "N-Gram Graph"-Based Summarization in the Real World, 2014)**)** βασίζεται στην υπόθεση ότι τα κείμενα πρέπει να επικαλύπτονται σε έναν ορισμένο βαθμό (κατώφλι κανονικοποιημένης ομοιότητας τιμής), αλλά πρέπει να είναι συγκρίσιμα και σε μέγεθος (κατώφλι ομοιότητας μεγέθους). Τέλος, η προσέγγιση αυτή βασίζεται στην υπόθεση ότι

αν τα κείμενα Α και Β μιλούν για το ίδιο γεγονός και τα κείμενα Β και Γ μιλούν για το ίδιο γεγονός, τότε τα κείμενα Α και Γ μιλούν επίσης για το ίδιο γεγονός. Αυτή η υπόθεση ολοκληρώνει τη διαδικασία ομαδοποίησης και το αποτέλεσμα αυτής είναι ομάδες κειμένων, όπου όλα τα κείμενα μέσα στην ίδια ομάδα αναφέρονται στο ίδιο γεγονός.

Δεδομένων αυτών των ομάδων, τώρα πρέπει να εντοπιστούν τα θέματα που σχηματίζουν το «νόημα» του κάθε γεγονότος. Στις επόμενες παραγράφους περιγράφεται αυτή η διαδικασία εντοπισμού.

4.2 Δημιουργία Θεμάτων και Γεγονότων

Σε αυτό το στάδιο εντοπίζονται ομάδες προτάσεων μέσα στο ίδιο γεγονός. Στην προσέγγιση αυτής της πτυχιακής, πρώτα χωρίζουμε τα κείμενα σε προτάσεις. Για αποδοτικό εντοπισμό προτάσεων, χρησιμοποιείται ένα εργαλείο για να κάνει αυτό το διαχωρισμό (SentenceDetectorME class of the Apache OpenNLP java package[2]). Αυτό το εργαλείο εκπαιδεύεται ανά γλώσσα και μπορεί να αποφασίσει αν ένα σημείο στο κείμενο είναι σημείο διαχωρισμού πρότασης.

Στη συνέχεια, μετατρέπονται οι προτάσεις σε γράφους ν-γραμμάτων ($\nu = 3$, $dwin = 3$). Αυτές οι τιμές έχουν δείξει να δουλεύουν καλά σε διάφορα πειράματα (Giannakopoulos & Karkaletsis, AutoSummENG and MeMoG in Evaluating Guided Summaries, 2011)**.** Έχοντας αυτούς τους γράφους συγκρίνονται όλα τα πιθανά ζευγάρια. Έχοντας εξάγει αυτές τις ομοιότητες μεταξύ των γράφων δημιουργείται ένας πίνακας ομοιοτήτων (similarity matrix) με βάση την κανονικοποιημένη ομοιότητα τιμής (normalized value similarity).

	1	2	3	4	5
1	1.0	0.9	0.7	0.1	0.2
2	0.9	1.0	0.8	0.4	0.35
3	0.7	0.8	1.0	0.4	0.2
4	0.1	0.4	0.4	1.0	0.8
5	0.2	0.35	0.2	0.8	1.0

Σχήμα 6 Παράδειγμα Πίνακα Ομοιότητας

Στο Σχήμα 6 φαίνεται ένα παράδειγμα πίνακα ομοιότητας προτάσεων. Οι ομοιότητες (δεκαδικοί αριθμοί) έχουν εξαχθεί από τη σύγκριση του γράφου της αντίστοιχης πρότασης με τον γράφο της άλλης. Οι αριθμοί εκτός του πίνακα υποδηλώνουν το αναγνωριστικό (id) της κάθε πρότασης. Άρα, η πρόταση 1 μοιάζει κατά 0,2 ή 20% με την πρόταση 5. Τέλος, από το Σχήμα 6 φαίνεται ότι έχουν συγκριθεί όλα τα πιθανά ζευγάρια προτάσεων.

[2] https://opennlp.apache.org/

Ύστερα, εφαρμόζεται ο αλγόριθμος Markov Clustering – MCL (Dongen, 2000) πάνω σε αυτόν τον πίνακα. Το αποτέλεσμα του αλγορίθμου αυτού είναι ομάδες προτάσεων (καμία πρόταση δεν μπορεί να ανήκει ταυτόχρονα σε δύο ομάδες) οι οποίες θεωρείται ότι αναπαριστούν διάφορα θέματα των γεγονότων. Στην ουσία, όσες προτάσεις είναι όμοιες μεταξύ τους με βάση τους γράφους τους, μιλάνε για το ίδιο θέμα.

Ο MCL είναι ένας μη επιβλεπόμενος τρόπος για να οριστεί αυτόματα και αποδοτικά ένας καλός αριθμός θεμάτων, με βάση τις ομοιότητες των προτάσεων. Αυτός ο αλγόριθμος είναι η καλύτερη επιλογή εφόσον δεν χρειάζεται από πριν να του ορίσουμε κάποιο αριθμό ομάδων (clusters) και μπορεί να τελειώσει γρήγορα την ομαδοποίηση σε σχέση με άλλους πιο αργούς αλγορίθμους (Blei, Ng, & Jordan, 2003).

Έχοντας πλέον αυτά τα θέματα, που είναι σύνολα προτάσεων, τώρα πρέπει να εξαχθεί το «νόημα» του κάθε θέματος. Στην προσέγγιση των γράφων ν-γραμμάτων θεωρείται πως το «νόημα» ενός συνόλου από γράφους είναι ο μέγιστος κοινός υπο-γράφος του συνόλου. Άρα, για να εξαχθεί το «νόημα» χρησιμοποιείται ο αλγόριθμος διασταύρωσης, που περιεγράφηκε στο προηγούμενο κεφάλαιο, μεταξύ όλων των γράφων που ανήκουν στο ίδιο θέμα.

Για ένα γεγονός, η παραπάνω διαδικασία έχει σαν αποτέλεσμα ένα σύνολο από γράφους ν-γραμμάτων, όπου ο καθένας αντιπροσωπεύει το «νόημα» ενός θέματος. Για να εκφραστεί το «νόημα» ολόκληρου του γεγονότος πρέπει να συνδυαστούν οι επιμέρους γράφοι των θεμάτων. Για αυτό το λόγο, χρησιμοποιείται ο αλγόριθμος ένωσης μεταξύ όλων των «νοημάτων» των θεμάτων. Ο γράφος που προκύπτει από αυτήν την ένωση είναι πλέον ο αντιπροσωπευτικός γράφος του γεγονότος.

4.3 Εξαγωγή Κατάλληλων Προτάσεων για Περίληψη

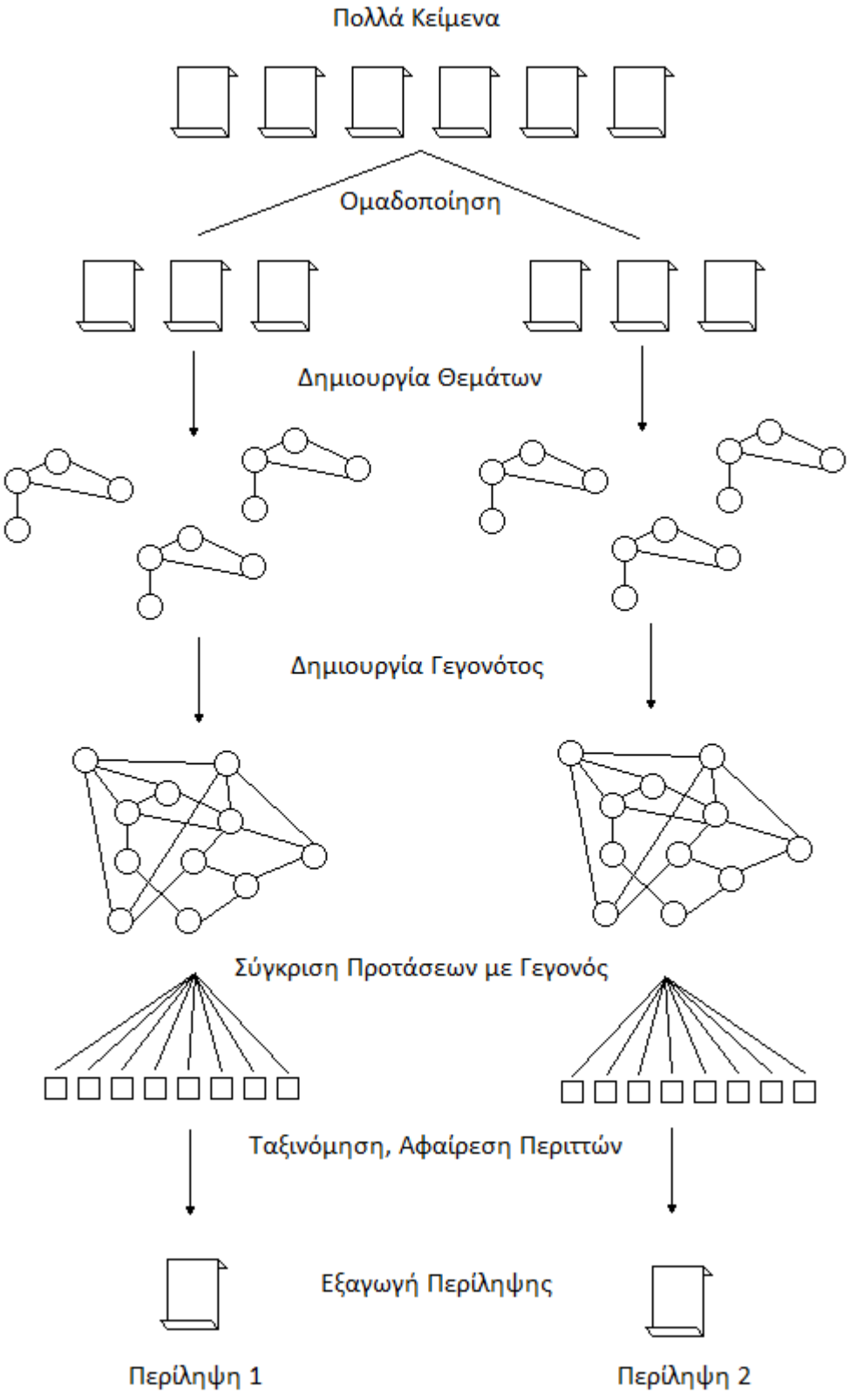
Για να θεωρηθεί μία πρόταση κατάλληλη για την περίληψη θα πρέπει να είναι όμοια με το γεγονός. Στην προσέγγιση των γράφων ν-γραμμάτων η ομοιότητα τιμής (value similarity) είναι αυτή που μετράει αν ένας γράφος είναι γενικά όμοιος με έναν άλλον. Με άλλα λόγια αυτό το μέτρο σύγκρισης είναι η καλύτερη επιλογή όταν θέλουμε να λάβουμε υπόψιν την επικάλυψη μεταξύ δύο γράφων (σε αντίθεση με τη μέγιστη πιθανή επικάλυψη με βάση το μέγεθος, που αντιπροσωπεύεται από την κανονικοποιημένη ομοιότητα τιμής).

Για να καταλήξουμε σε μία λίστα προτάσεων ταξινομημένων με βάση την καταλληλότητά τους να συμπεριληφθούν στην περίληψη, συγκρίνουμε τον γράφο της κάθε πρότασης με τον γράφο του γεγονότος μέσα στο οποίο ανήκει. Μετά την ταξινόμηση των προτάσεων με βάση την ομοιότητα καταλήγουμε σε μία λίστα υποψήφιων προτάσεων για την τελική περίληψη.

Μία απλή προσέγγιση θα ήταν να χρησιμοποιηθεί αυτή η λίστα σαν περίληψη, αλλά πολλές προτάσεις μπορεί να μιλούν για το ίδιο πράγμα ή και να είναι ίδιες. Για να αποφευχθεί αυτό το πρόβλημα και να αντιμετωπιστεί αυτός ο πλεονασμός συγκρίνονται οι προτάσεις μεταξύ τους και φεύγουν οι περιττές από τη λίστα.

Ο αλγόριθμος αρχίζει από την πιο κατάλληλη πρόταση και τη συγκρίνει (κανονικοποιημένη ομοιότητα τιμής) με όλες τις υπόλοιπες, ύστερα προχωράει στην επόμενη πιο κατάλληλη και συνεχίζει με τον ίδιο τρόπο. Αν η ομοιότητα είναι πάνω από ένα κατώφλι (0,2 ή 20%) σημαίνει ότι ο δεύτερος υποψήφιος επαναλαμβάνει την πληροφορία που δίνεται από τον πρώτο και εφόσον έρχεται δεύτερος στην ταξινομημένη λίστα αφαιρείται από αυτήν. Η διαδικασία συνεχίζεται μέχρι να φτάσουμε στο τέλος της λίστας και να έχουν αφαιρεθεί όλες οι περιττές προτάσεις.

Το αποτέλεσμα αυτής της διαδικασίας είναι ένα σύνολο προτάσεων που καλύπτουν πλήρως το «νόημα» του γεγονότος, χωρίς επαναλαμβανόμενη πληροφορία και έτσι αυτές οι προτάσεις αποτελούν την περίληψη. Στο Σχήμα 7 φαίνεται ολόκληρη η διαδικασία εξαγωγής περίληψης βηματικά.



Σχήμα 7 Διαδικασία Εξαγωγής Περίληψης

5. Υλοποίηση

Σε αυτό το κεφάλαιο θα αναλυθεί ακριβώς πως υλοποιήθηκε το πρόβλημα, πως παραλληλοποιήθηκε και θα αναφερθούν παραδείγματα χρήσης. Εν συντομία, τα επιμέρους προβλήματα που παραλληλοποιήθηκαν στην διαδικασία εξαγωγής περιλήψεων είναι η εξαγωγή ν-γραμμάτων (είτε πρόκειται για λέξεις, είτε για χαρακτήρες), ο υπολογισμός ομοιότητας γράφων, οι αλγόριθμοι μεταξύ των γράφων ν-γραμμάτων, ο διαχωρισμός προτάσεων των κειμένων, η σύγκριση όλων των προτάσεων μεταξύ τους και ο αλγόριθμος Markov Clustering. Για την υλοποίηση χρησιμοποιήθηκε το εργαλείο Apache Spark[3], ένα εργαλείο για επεξεργασία δεδομένων μεγάλης κλίμακας. Συγκεκριμένα, χρησιμοποιήθηκαν οι βιβλιοθήκες GraphX και MLlib, για επεξεργασία γράφων και αλγορίθμων μηχανικής μάθησης αντίστοιχα. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η Scala. Το project μπορεί να βρεθεί στο GitHub[4].

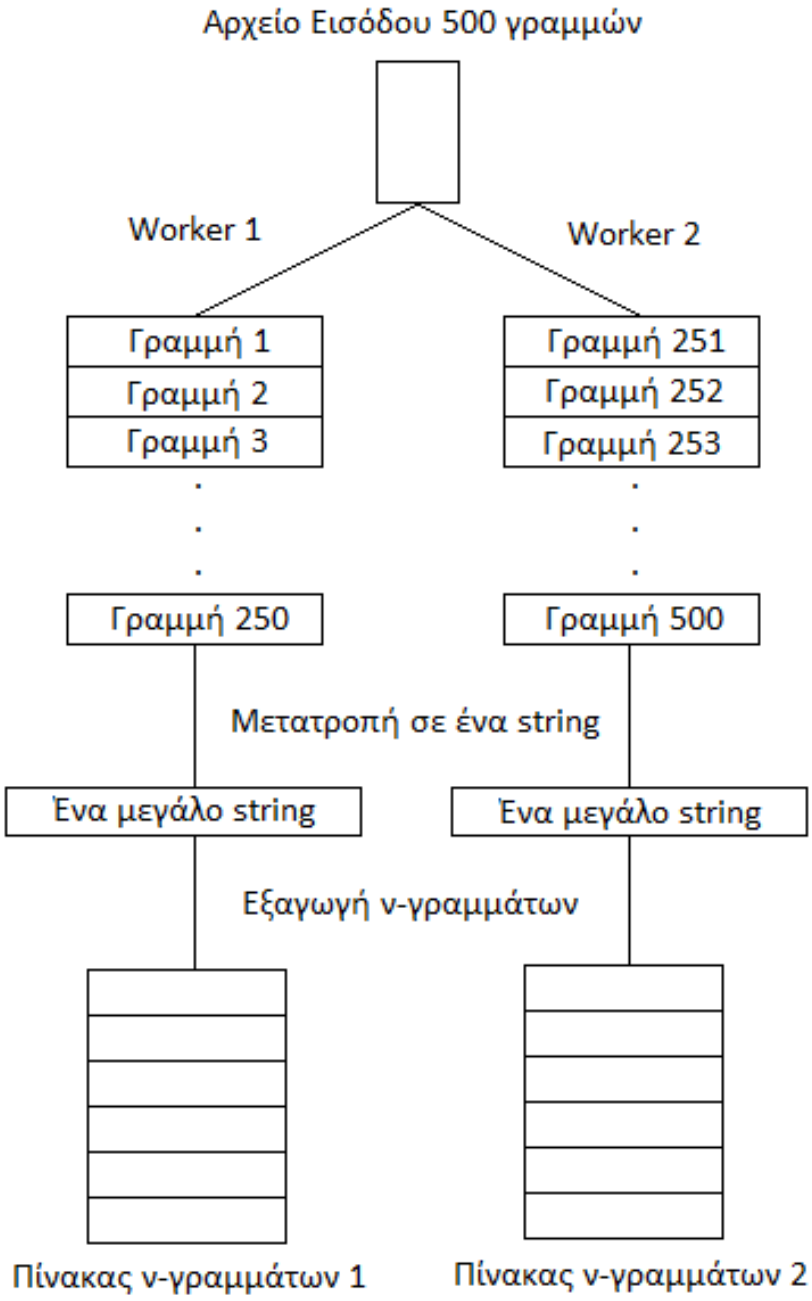
5.1 Παραλληλοποίηση του προβλήματος

5.1.1 Εξαγωγή Ν-Γραμμάτων

Αρχικά, τα κείμενα διαβάζονται από αρχεία και το Spark αναλαμβάνει να «σπάσει» το string του αρχείου αυτού σε partitions, αλλά σε επίπεδο γραμμών. Συγκεκριμένα, αν υποθέσουμε ότι έχουμε 2 workers και ένα αρχείο με 500 γραμμές, κάθε worker θα πάρει από 250 γραμμές. Ο καθένας έχει τις γραμμές σε μορφή πίνακα, όπου κάθε στοιχείο του πίνακα αντιστοιχεί σε μία γραμμή. Σε επίπεδο εξαγωγής ν-γραμμάτων, ο κάθε worker/πυρήνας θα κάνει το εξής: θα συγχωνεύσει τις γραμμές σε ένα string και πάνω σε αυτό το string θα τρέξει τον αλγόριθμο εξαγωγής ν-γραμμάτων ή πιο απλά θα τρέξει έναν κυλιόμενο παράθυρο και θα εξάγει τα επικαλυπτόμενα ν-γράμματα. Το αποτέλεσμα αυτής της διαδικασίας είναι μία κατανεμημένη συλλογή από ν-γράμματα που δεν θα είναι 100% ακριβή, εφόσον χάνονται μερικά στην αρχή και στο τέλος του κάθε partition. Δεδομένου όμως ότι κάθε partition μπορεί να περιέχει μερικά εκατομμύρια χαρακτήρες, η απώλεια αξιοπιστίας θα πρέπει να είναι αμελητέα. Το κύριο πλεονέκτημα της διαδικασίας εδώ είναι ότι γίνεται παράλληλα. Στο Σχήμα 8 φαίνεται ένα παράδειγμα εξαγωγής ν-γραμμάτων από αρχείο 500 γραμμών με δύο workers.

[3] http://spark.apache.org/

[4] https://github.com/ioannis-kon/ParallelNGG



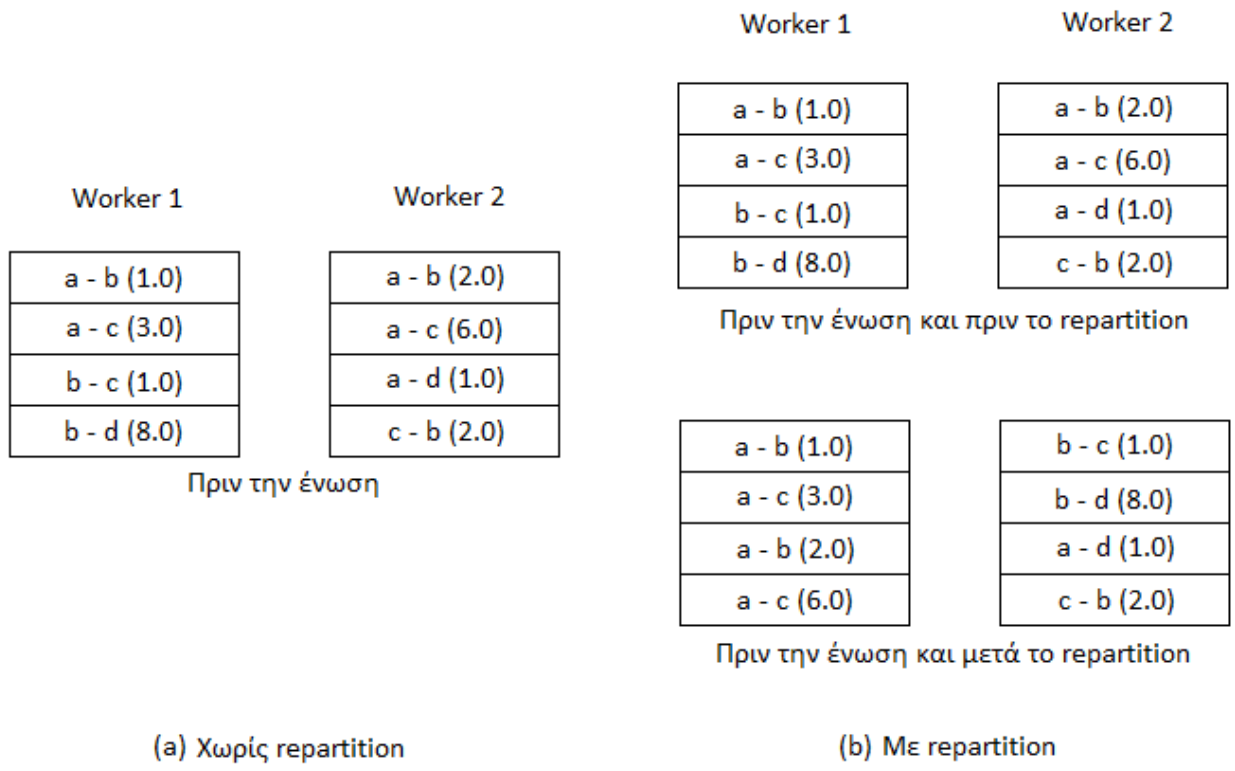
Σχήμα 8: Παράδειγμα Εξαγωγής ν-γραμμάτων παράλληλα

Αυτά τα εξαγόμενα ν-γράμματα χρησιμοποιούνται μετά για τη δημιουργία του γράφου ν-γραμμάτων.

5.1.2 Λειτουργίες Γράφων N-Γραμμάτων

Στη συνέχεια, ο γράφος «σπάει» και κάθε worker έχει μοιρασμένους τους κόμβους και τις ακμές του. Αν για παράδειγμα ο γράφος έχει 100 κόμβους και 500 ακμές, με την ίδια λογική όπως παραπάνω, ο worker 1 θα έχει τους πρώτους 50 κόμβους και τις πρώτες 250 ακμές και ο δεύτερος worker θα έχει τα υπόλοιπα. Το Spark μετά αναλαμβάνει να υπολογίσει τις ομοιότητες, άρα και αυτή η διαδικασία γίνεται παράλληλα.

Η λειτουργία συγχώνευσης ή ένωσης των γράφων γίνεται επίσης παράλληλα, αλλά σε αυτήν την περίπτωση πρέπει πρώτα να γίνει μία επανατοποθέτηση (repartition) των κόμβων και ακμών μεταξύ των workers για να μπορέσει να παραλληλοποιηθεί και να μπορέσουν να βγουν σωστά αποτελέσματα στα βάρη των ακμών. Πιο συγκεκριμένα, ας υποθέσουμε ότι έχουμε τις ακμές a – b με βάρος 1,0 του πρώτου γράφου και την ακμή a – b με βάρος 2,0 του δεύτερου γράφου, η πρώτη ακμή βρίσκεται στον πρώτο worker και η δεύτερη στον δεύτερο worker. Αν αυθαίρετα εφαρμοστεί η ένωση σε κάθε worker τότε θα προκύψει ένας γράφος που θα έχει και τις δύο αυτές ακμές, ενώ θα έπρεπε να περιέχει μία με βάρος 1,5. Για να λυθεί αυτό το πρόβλημα θα πρέπει να γίνει πρώτα ένα repartition έτσι ώστε οι ακμές που έχουν τους ίδιους κόμβους να βρίσκονται στον ίδιο worker και να μπορέσει ο καθένας ανεξάρτητα να εφαρμόσει την ένωση.



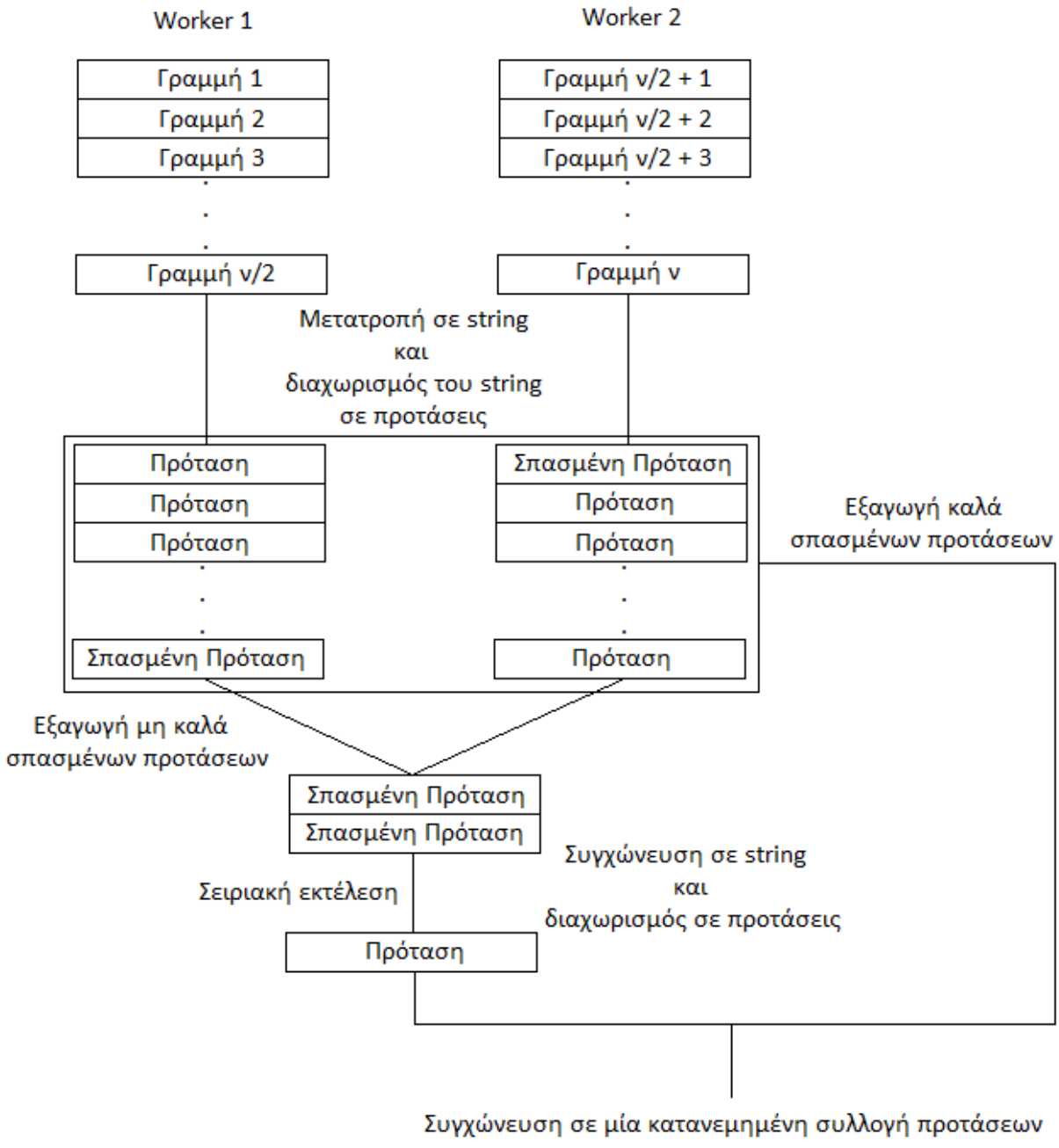
Σχήμα 9: Κατανομή ενός μέρους των ακμών των δύο γράφων μεταξύ δύο workers πριν την λειτουργία ένωσης

Στο Σχήμα 9 φαίνεται ένα παράδειγμα αυτής της διαδικασίας επανατοποθέτησης ακμών. Κάθε worker περιέχει ακμές και από τους δύο γράφους. Στην εικόνα (a) του σχήματος φαίνεται ότι αν

αρχίσει κάθε worker να συγχωνεύει τις ακμές, αυτές που έχουν κοινούς κόμβους (a – b, a – c) δεν θα αλλάξει καθόλου τα βάρη τους αφού βρίσκονται σε ξεχωριστά partitions. Στην εικόνα (b) του σχήματος φαίνεται τι γίνεται πριν και μετά το repartition. Όλες οι ακμές με κοινούς κόμβους έχουν μεταφερθεί στον worker 1 και οι υπόλοιπες έχουν μεταφερθεί στον worker 2. Αν εφαρμοστεί τώρα η λειτουργία ένωσης, το αποτέλεσμα θα είναι ένας γράφους με την ακμή a – b να έχει βάρος 1,5 και την ακμή a – c να έχει βάρος 4,5, που είναι το επιθυμητό αποτέλεσμα. Το ίδιο ακριβώς γίνεται και στον αλγόριθμο διασταύρωσης (intersection), μόνο που σε αυτήν την περίπτωση μας ενδιαφέρουν μόνο οι κοινές ακμές και απορρίπτουμε τις υπόλοιπες. Τις υπόλοιπες δύο λειτουργίες αναλαμβάνει το Spark πάλι να τις εφαρμόσει.

5.1.3 Διαχωρισμός Προτάσεων

Ο διαχωρισμός προτάσεων είναι και αυτός μία λειτουργία που μπορεί να γίνει παράλληλα με τον ίδιο τρόπο που γίνεται και η εξαγωγή ν-γραμμάτων, μόνο που εδώ δεν πρέπει να χαθούν κάποιες προτάσεις γιατί μπορεί να κάποια να είναι αντιπροσωπευτική για την περίληψη. Σε αυτήν την περίπτωση πάλι συγχωνεύονται οι γραμμές του αρχείου σε κάθε worker σε ένα string. Ύστερα, χρησιμοποιείται ο Apache OpenNLP Sentence Splitter για να διαχωρίσει τις προτάσεις. Μία πρόταση όμως μπορεί να έχει σπάσει μεταξύ των partitions και κάθε worker να έχει από μισή πρόταση. Για να λυθεί αυτό το πρόβλημα, παίρνουμε τις προτάσεις που βρίσκονται στην αρχή και στο τέλος του κάθε partition. Για να γίνει αυτό, χρησιμοποιούνται κάποιοι κανόνες: πρέπει ο τελευταίος χαρακτήρας της πρότασης να μην είναι τερματικός χαρακτήρας “[.?;!:]” (πρόταση σπασμένη στο τέλος του partition) και ο πρώτος χαρακτήρας να μην είναι κεφαλαίο γράμμα (πρόταση σπασμένη στην αρχή του partition). Αυτές συγχωνεύονται και πλέον ο διαχωρισμός γίνεται σειριακά. Δεδομένου ότι οι προτάσεις θα είναι λίγες, δεν υπάρχει κάποια επιβάρυνση στην απόδοση. Τέλος, συγχωνεύονται όλες οι προτάσεις σε μία κατανεμημένη συλλογή. Στο Σχήμα 10 φαίνεται η διαδικασία εξαγωγής προτάσεων.

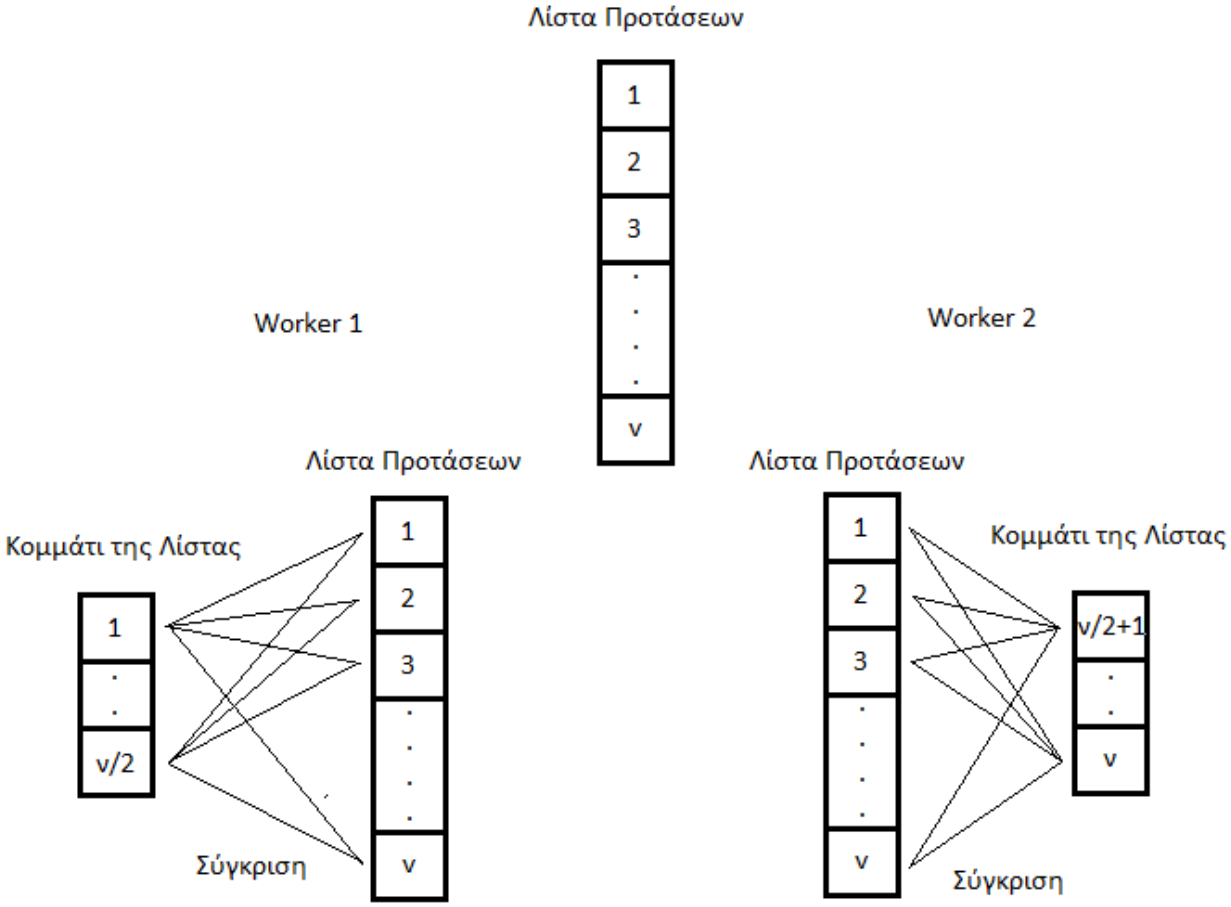


Σχήμα 10: Διαδικασία εξαγωγής προτάσεων

5.1.4 Σύγκριση Προτάσεων

Ένα ακόμη πρόβλημα που παραλληλοποιήθηκε ήταν η σύγκριση όλων των προτάσεων μεταξύ τους. Ένας απλός σειριακός αλγόριθμος θα ήταν να ξεκινήσουμε από την πρώτη και να τη συγκρίνουμε με όλες τις υπόλοιπες, να συνεχίσουμε στη δεύτερη κ. ο. κ. Ο παράλληλος αλγόριθμος μοιάζει πάρα πολύ με την έννοια ότι κάθε worker παίρνει όλη αυτή τη μεγάλη λίστα

των προτάσεων και παίρνει και ένα κομμάτι που του αντιστοιχεί. Συγκρίνει κάθε πρόταση από το κομμάτι της λίστας που του αντιστοιχεί με όλες τις προτάσεις της λίστας που έχει μοιραστεί σε όλους τους workers. Το Apache Spark προσφέρει έτοιμη μια μέθοδο με όνομα cartesian και βρίσκει όλους αυτούς τους συνδυασμούς προτάσεων (για τη δική μας περίπτωση). Αυτή όμως η μέθοδος θα επιστρέψει και περιττές συγκρίσεις. Για παράδειγμα, δεν θα επιστρέψει μόνο τον συνδυασμό προτάσεων (1,2) όπου 1 η πρώτη πρόταση και 2 η δεύτερη πρόταση αλλά θα επιστρέψει και τον συνδυασμό (2,1). Για να δημιουργηθεί ο πίνακας ομοιοτήτων χρειάζεται μόνο ο ένας εκ των δύο, για αυτόν το λόγο αφαιρούνται όλοι οι συνδυασμοί όπου το δεύτερο στοιχείο είναι μικρότερο του πρώτου και για το παραπάνω παράδειγμα κρατάμε μόνο τον (1,2). Στο Σχήμα 11 φαίνεται η διαδικασία σύγκρισης όλων των προτάσεων μεταξύ τους με δύο workers. Κάθε πρόταση αναπαρίσταται με ένα id, γι` αυτό και στην εικόνα αντί να αναγράφεται Πρόταση 1, γράφεται μόνο ο αριθμός 1.



Σχήμα 11: Σύγκριση όλων των προτάσεων μεταξύ τους

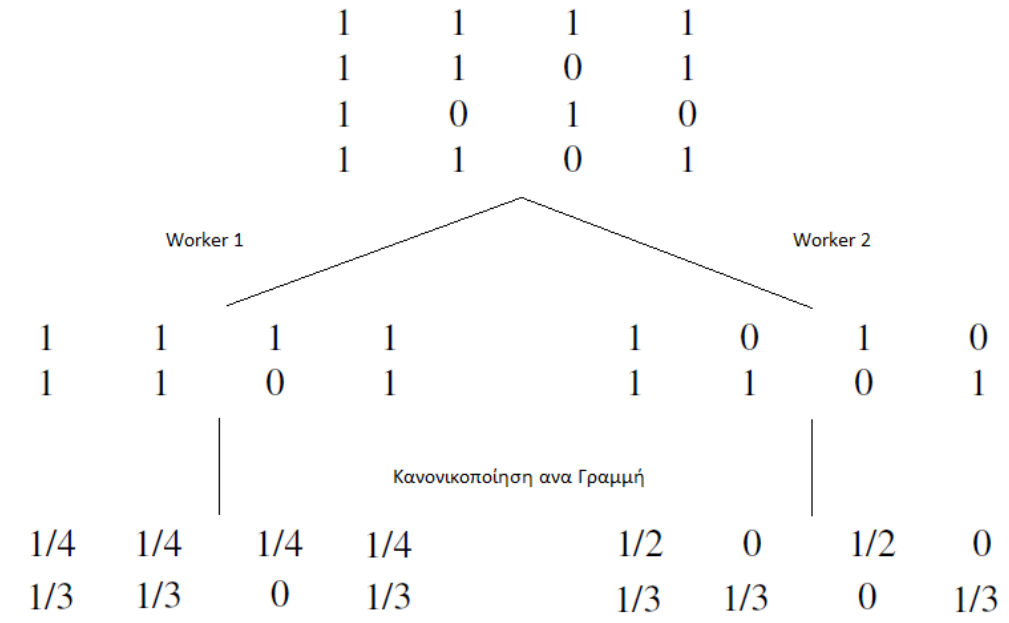
Στην υλοποίηση που έχει προταθεί ως τώρα οι γράφοι είναι υλοποιημένοι με το Apache Spark GraphX, που σημαίνει ότι κάθε γράφος είναι κατανεμημένος, πράγμα το οποίο δεν βολεύει για το

παραπάνω πρόβλημα. Εφόσον εδώ θα έχουμε πολλές προτάσεις και εφόσον οι προτάσεις πάντα θα είναι μικρές από άποψη χαρακτήρων και μεγέθους δεν συμφέρει καθόλου το «σπάσιμο» του γράφου. Για αυτόν το λόγο, χρησιμοποιήθηκε η υλοποίηση γράφων ν-γραμμάτων του εργαλείου ανοιχτού λογισμικού JInsect. Σε αυτό το εργαλείο ο γράφος δεν είναι κατανεμημένος και έτσι μπορεί κάθε worker να έχει πολλούς γράφους μαζί για να συγκρίνει.

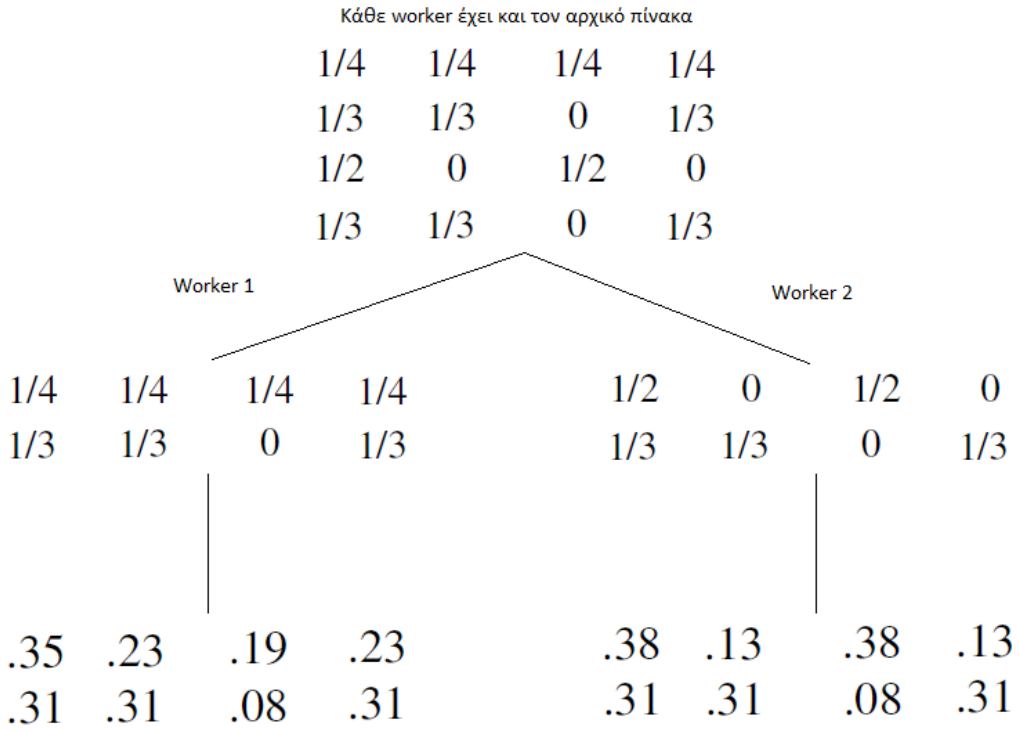
5.1.5 Markov Clustering

Το τελευταίο πρόβλημα που παραλληλοποιήθηκε είναι ο αλγόριθμος Markov Clustering. Ο αλγόριθμος αυτός έχει τέσσερα μέρη, την κανονικοποίηση (normalization), την επέκταση (expansion), inflation και επανάληψη των τελευταίων δύο βημάτων αυτών μέχρι ο πίνακας να παραμείνει ίδιος από τη μία επανάληψη στην άλλη ή έχουν ολοκληρωθεί 100 επαναλήψεις. Η παραλληλοποίηση έγινε ανά γραμμή και μέρος του κώδικα πάρθηκε από το GitHub[5]. Για την κανονικοποίηση η παραλληλοποίηση είναι εύκολη εφόσον μπορεί να γίνει ανά γραμμή, αλλά για την επέκταση που στην ουσία είναι πολλαπλασιασμός πινάκων ο κάθε κόμβος πρέπει να έχει και ολόκληρο τον πίνακα εκτός από ένα μέρος του. Το inflation γίνεται πάλι ανά γραμμή και μπορεί επίσης να παραλληλοποιηθεί εύκολα. Στο Σχήμα 12 φαίνεται η παράλληλη κανονικοποίηση του πίνακα (matrix normalization) με 2 workers, όπου ο καθένας παίρνει 2 γραμμές από τον πίνακα τεσσάρων γραμμών. Εφόσον η διαδικασία αυτή γίνεται ανά γραμμή, κάθε worker μπορεί να δουλεύει ανεξάρτητα. Στο Σχήμα 13 φαίνεται η παράλληλη επέκταση του πίνακα με 2 workers. Σε αυτήν την περίπτωση κάθε worker εκτός από το κομμάτι που του αντιστοιχεί (2 γραμμές ο καθένας) έχει και ολόκληρο τον αρχικό πίνακα, για να μπορέσει να κάνει τον πολλαπλασιασμό.

[5] https://github.com/joandre/MCL_spark

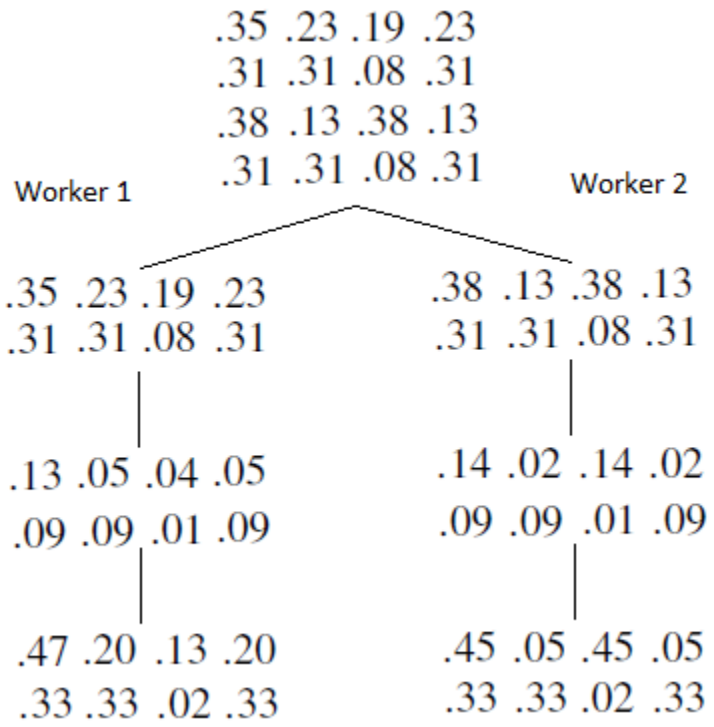


Σχήμα 12: Παράλληλη κανονικοποίηση πίνακα με δύο workers



Σχήμα 13: Παράλληλη επέκταση του πίνακα με δύο workers

Τέλος, μένει το inflation που γίνεται ανά γραμμή του πίνακα και μπορεί ο κάθε worker να δουλεύει ανεξάρτητα. Στο Σχήμα 14 φαίνεται αυτή η διαδικασία με 2 workers.



Σχήμα 14: Παράλληλο inflation του πίνακα με δύο workers

Ο αλγόριθμος Markov Clustering χρησιμοποιήθηκε με τιμές $e = 2$ και $r = 2$ για expansion και inflation, αντίστοιχα. Τέλος, για να αναγνωριστούν ποιες προτάσεις ανήκουν σε κάποια ομάδα (cluster), σε κάθε πρόταση έχει ανατεθεί ένα matrix id πριν την εκκίνηση του αλγορίθμου. Η βιβλιοθήκη MLlib του Spark, βοήθησε σε αυτήν τη διαδικασία.

Στη συνέχεια της διαδικασίας ακολουθεί η κατανεμημένη διασταύρωση και ένωση των γράφων. Επειδή οι γράφοι που θα διασταυρώνονται είναι πάλι μικρού μεγέθους (προτάσεις) όπως στην περίπτωση σύγκρισης προτάσεων και εφόσον η διασταύρωση θα δίνει σαν αποτέλεσμα έναν μικρό γράφο, ίσως συμφέρει να χρησιμοποιηθεί πάλι η υλοποίησης της βιβλιοθήκης JInsect. Ύστερα, να πρέπει να κατανεμηθούν οι γράφοι για να μπορέσουν να συγχωνευθούν παράλληλα. Μέχρι τώρα, οι γράφοι έχουν οριστεί με αριθμό partitions σε 1 μόνο για τη διασταύρωση και ύστερα γίνεται repartition σε όλους τους διαθέσιμους πόρους. Στο μέλλον ίσως γίνει και εδώ συνδυασμός των δύο βιβλιοθηκών για βελτίωση της απόδοσης του αλγορίθμου.

5.2 Παραδείγματα Χρήσης

Σε αυτό το κεφάλαιο θα αναφερθούν μερικά παραδείγματα χρήσης της εφαρμογής σχετικά με την εξαγωγή περιλήψεων αλλά και τη δημιουργία γράφων, ομαδοποίηση κειμένων κλπ. Στην

υλοποίηση της εφαρμογής αυτή μία οντότητα που μπορεί να σπάσει σε μικρότερα κομμάτια λέγεται Entity και συγκεκριμένα για τα string, String Entity. Αυτή η οντότητα σπάει σε άτομα (Atoms ή String Atoms). Αν για παράδειγμα έχουμε κείμενα, οντότητα θεωρείται το κείμενο και άτομα θεωρούνται οι λέξεις, τα ν-γράμματα ή και οι προτάσεις. Αν θεωρήσουμε τις προτάσεις σαν οντότητες τότε τα άτομα είναι πάλι οι λέξεις ή τα ν-γράμματα. Για τη δημιουργία μιας οντότητας υπάρχουν δύο τρόποι, είτε από ένα string είτε από ένα αρχείο. Παρακάτω φαίνονται τα παραδείγματα χρήσης:

```
// δημιουργία οντότητας
val e = new StringEntity.

// διάβασμα οντότητας από string
e.fromString("Hello World!").

// διάβασμα οντότητας από αρχείο
e.fromFile(sc,"path/to/file",numPartitions).
```

Όταν διαβάζεται η οντότητα από ένα αρχείο πρέπει επίσης να δώσουμε σαν είσοδο και το περιβάλλον του Spark (SparkContext), αλλά και τον αριθμό των κομματιών που θα σπάσει η οντότητα. Για να πάρουμε το περιεχόμενο της οντότητας απλά καλούμε τη μέθοδο getPayload:

```
// ανάκτηση του περιεχομένου της οντότητας
val payload = e.getPayload.
```

Αυτή η οντότητα μπορεί να σπάσει όπως προαναφέρθηκε σε άτομα. Αυτό επιτυγχάνεται με την κλάση StringEntityTokenizer. Συγκεκριμένα, αυτή η κλάση μπορεί να σπάσει την οντότητα σε άτομα λέξεων, ν-γραμμάτων, ν-γραμμάτων αποτελούμενα από λέξεις, ν-γραμμάτων αποτελούμενα από κεφαλαίες λέξεις, κεφαλαίες λέξεις και τέλος κεφαλαίες λέξεις και αριθμούς. Η δημιουργία της κλάσης αυτής και η εξαγωγή των αντίστοιχων ατόμων γίνεται ως εξής:

```
// δημιουργία αντικειμένου tokenizer
val set = new StringEntityTokenizer.

// ανάκτηση λέξεων από την οντότητα e
val tokens = set.getTokens(e).

// ανάκτηση ν-γραμμάτων με μέγεθος ν = 3
val ngrams = set.getCharacterNGrams(e,3).

// ανάκτηση ν-γραμμάτων λέξεων
val wordNgrams = set.getWordNGrams(e,3).
```

```
// ανάκτηση ν-γραμμάτων κεφαλαίων λέξεων
val capWordNgrams = set.getCapWordNGrams(e,3).


// ανάκτηση κεφαλαίων λέξεων
val capitalizedWords = set.getCapitalizedWords(e).

// ανάκτηση κεφαλαίων λέξεων και αριθμών
val capAndNums = set.getCapitalizedAndNumbers(e).
```

Από μία οντότητα μπορούν να εξαχθούν άτομα προτάσεων. Για να γίνει αυτό πρέπει να δημιουργηθεί το αντίστοιχο αντικείμενο της κλάσης και να κληθεί η κατάλληλη μέθοδος:

```
// δημιουργία αντικειμένου
val ss = new OpenNLPSentenceSplitter("path/to/model/file").

// εξαγωγή προτάσεων
val sentences = ss.getSentences(e).
```

Το αντικείμενο του OpenNLPSentenceSplitter παίρνει σαν όρισμα και ένα αρχείο μοντέλο το οποίο έχει εκπαιδευτεί στον διαχωρισμό προτάσεων για μια συγκεκριμένη γλώσσα.

Έχοντας μία οντότητα μπορούμε να δημιουργήσουμε γράφους ν-γραμμάτων. Πάλι πρέπει να δημιουργηθούν τα κατάλληλα αντικείμενα:

```
// δημιουργία αντικειμένου που δημιουργεί γράφο από οντότητα έχοντας ως ορίσματα
// το μέγεθος του ν και το μέγεθος του παραθύρου γειτνίασης με βάση το οποίο θα
// δημιουργηθούν οι ακμές.
val nggc = new NGramGraphCreator(3,3).

// ανάκτηση του γράφου από μία οντότητα e
val g = nggc.getGraph(e).

// αντικείμενο που δημιουργεί γράφο με ν-γράμματα κεφαλαίων λέξεων και αριθμών
val wggc = new WordNGramGraphCreator(3,3).

// ανάκτηση του γράφου
val g = wggc.getGraph(e).
```

Μία βασική λειτουργία είναι η αποθήκευση του γράφου και το φόρτωμα του γράφου από αρχείο. Αυτές οι λειτουργίες επιτυγχάνονται ως εξής:

```
// δημιουργία αντικειμένου με ορίσματα το SparkContext και τον αριθμό των partitions
val st = new NGramGraphStorage(sc,numPartitions).

// αποθήκευση γράφου σε αρχείο
st.saveGraph(g).

// αποθήκευση γράφου σε αρχείο τύπου dot
st.saveGraphToDotFormat(g).

// ανάκτηση γράφου από αρχείο
val g = st.loadGraph("path/to/graph").
```

Κάποιες πολύ βασικές λειτουργίες είναι η εξαγωγή ομοιοτήτων και οι αλγόριθμοι ένωσης, διασταύρωσης κλπ. Μεταξύ των γράφων. Έχοντας δύο γράφους g1 και g2 τα παραπάνω μπορούν να εφαρμοστούν ως εξής:

```
// δημιουργία αντικειμένου
val ngc = new GraphSimilarityCalculator.

// ανάκτηση ομοιότητας
val gs = gnc.getSimilarity(g1,g2).

// ανάκτηση ολικής ομοιότητας
val ovs = gs.getOverallSimilarity.

// ανάκτηση ομοιότητας μεγέθους
val sSimil = gs.getSimilarityComponents("size").

// ανάκτηση ομοιότητας τιμής
val vSimil = gs.getSimilarityComponents("value").

// ανάκτηση ομοιότητας περιεκτικότητας
val cSimil = gs.getSimilarityComponents("containment").

// ανάκτηση κανονικοποιημένης ομοιότητας τιμής
val nSimil = gs.getSimilarityComponents("normalized").
```

Η μέθοδος getSimilarity του αντικειμένου GraphSimilarityCalculator επιστρέφει ένα αντικείμενο GraphSimilarity. Από αυτό μπορεί να παρθεί η ολική ομοιότητα καθώς και οι υπόλοιπες προαναφερθείσες ομοιότητες μέσω ενός Hash Map. Στη συνέχεια, φαίνονται παραδείγματα χρήσης για τις λειτουργίες μεταξύ δύο γράφων. Αρχικά πρέπει να δημιουργηθούν τα κατάλληλα αντικείμενα:

```
// δημιουργία αντικειμένου
val mo = new MergeOperator(0.5).

// ανάκτηση ενωμένου γράφου
val mg = mo.getResult(g1,g2).

// δημιουργία αντικειμένου
val io = new IntersectOperator(0.5).

// ανάκτηση διασταυρωμένου γράφου
val ig = io.getResult(g1,g2).

// δημιουργία αντικειμένου
val ivo = new InverseIntersectOperator.

// ανάκτηση αντίστροφα διασταυρωμένου γράφου
val ivg = ivo.getResult(g1,g2).

// δημιουργία αντικειμένου
val do = new DeltaOperator.

// ανάκτηση γράφου μετά της λειτουργία δέλτα
val dg = do.getResult(g1,g2).
```

Στις λειτουργίες ένωσης και διασταύρωσης η παράμετρος είναι ο παράγοντας εκμάθησης (learning factor), ο οποίος ορίζει ποιο βάρος από τους δύο γράφους θα ληφθεί περισσότερο υπόψιν.

Παρακάτω φαίνονται παραδείγματα χρήσης των αλγορίθμων ομαδοποίησης (clustering) όπως για ομαδοποίηση κειμένων και του Markov Clustering:

```
// δημιουργία αντικειμένου με SparkContext και αριθμό partitions
val doc = new DocumentEventClustering(sc,numPartitions).

// εξαγωγή ομάδων από πίνακα που περιέχει τη διαδρομή προς τα αρχεία
val docClusters = doc.getClusters(files).

// αποθήκευση ομάδων σε αρχείο csv
doc.saveClustersToCsv(docClusters).
```

```
// ανάκτηση ομάδων από csv αρχείο
val retrievedClusters = doc.loadClustersFromCsv("path/to/csv/file").

// δημιουργία αντικειμένου με παραμέτρους που ορίζουν τον αριθμό των επαναλήψεων,
// expansionRate, inflationRate, epsilon
val mcl = new MatriMCL(100,2,2.0,0.05).

// εξαγωγή ομάδων από πίνακα ομοιότητας
val clusters = mcl.getMarkovClusters(sMatrix).

// αποθήκευση ομάδων σε αρχείο csv
mcl.saveClustersToCsv(clusters).

// ανάκτηση ομάδων από αρχείο csv
val clusters = mcl.loadClustersFromCsv(sc,"path/to.csv/file").
```

Τέλος, θα δειχτεί η εξαγωγή περιλήψεων από κείμενα:

```
// δημιουργία αντικειμένου
val sum = new NGGSummarizer(sc,numPartitions,false).

// εξαγωγή περιλήψεων από φάκελο που περιέχει τα κείμενα
val sums = sum.getSummary("folder").

// εξαγωγή περίληψης από πίνακα που περιέχει διαδρομές προς τα κείμενα
val summary = sum.getTopicSummary(documents).

// αποθήκευση περιλήψεων
sum.saveSummaries(sums).

// αποθήκευση περίληψης
sum.saveSummary(summary).
```

Στα παραπάνω παραδείγματα χρήσης φάνηκε πως εξάγονται περιλήψεις από πολλά κείμενα (getSummary), τα οποία μπορεί να μιλούν για πολλά γεγονότα, άρα αυτή η μέθοδος χρησιμοποιεί και την ομαδοποίηση κειμένων. Επίσης, φάνηκε πως εξάγεται περίληψη από κείμενα που ανήκουν στο ίδιο γεγονός (getTopicSummary) και πως αποθηκεύονται οι περιλήψεις. Η Boolean μεταβλητή στην δημιουργία του NGGSummarizer δείχνει αν το Spark θα αποθηκεύσει τα ενδιάμεσα στάδια δημιουργίας του γράφου σε αρχείο σε περίπτωση που κάποιος κόμβος «πεθάνει». Αυτό γίνεται μόνο σε περίπτωση που υπάρχει κάποιο κατανεμημένο σύστημα αρχείων (nfs, hdfs[6]). Τέλος, θα δειχτεί πως αφαιρούνται οι περιττές προτάσεις από έναν πίνακα προτάσεων:

[6] https://hadoop.apache.org/

```
// δημιουργία αντικειμένου με SparkContext
val rem = new RedundancyRemover(sc).

// αφαίρεση περιττών προτάσεων
val filtered = rem.getFilteredSentences(sentencesArray).
```

Η παραπάνω μέθοδος χρησιμοποιεί τη μεθοδολογία που αναφέρθηκε σε προηγούμενο κεφάλαιο.

6. Αποτελέσματα και Αξιολόγηση

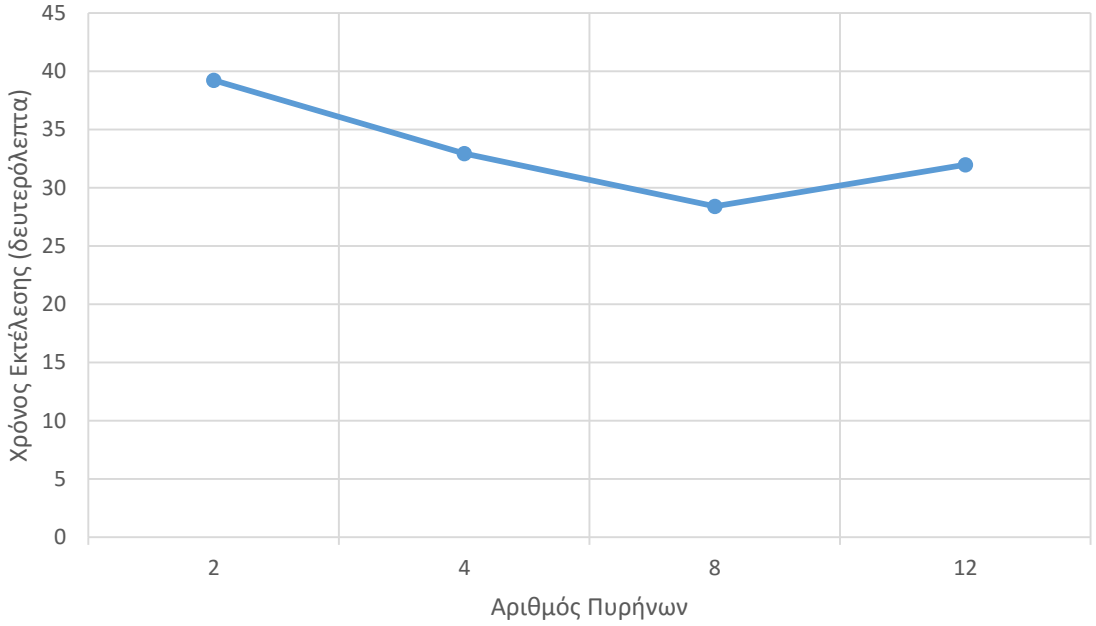
Σε αυτή την ενότητα θα παρουσιαστούν τα πειράματα που έγιναν για να αποδείξουν ότι η κατανομή του προβλήματος σε πολλούς workers μειώνει τον χρόνο εκτέλεσης του προβλήματος. Συγκεκριμένα, έγιναν πειράματα για να δείξουν τη μείωση του χρόνου στα πιο σημαντικά και χρονοβόρα προβλήματα όπως η ομαδοποίηση των κειμένων, ο διαχωρισμός προτάσεων, η σύγκριση όλων των προτάσεων μεταξύ τους μαζί με τον αλγόριθμο Markov Clustering, η δημιουργία του νοήματος του κειμένου και ο συνολικός χρόνο εκτέλεσης. Για τα πειράματα χρησιμοποιήθηκε το σύνολο δεδομένων των Tran et al (2013). Συγκεκριμένα, κείμενα από διαφορετικές κατηγορίες συγχωνεύθηκαν για δημιουργήσουν 5 μεγάλα κείμενα που το καθένα είχε περίπου 1000 γραμμές (με 214000 χαρακτήρες το κάθε κείμενο). Εφόσον, το πρόβλημα παραλληλοποιείται περισσότερο στον γράφο (σπάσιμο του γράφου σε περισσότερα κομμάτια), δεν μας ενδιαφέρει το πλήθος των κειμένων αλλά το μέγεθος αυτών. Για τη δημιουργία του cluster υπολογιστών χρησιμοποιήθηκαν τρεις υπολογιστές με 4 πυρήνες ο καθένας και 3GB μνήμη RAM. Ο Master είχε επίσης 4 πυρήνες με 2GB μνήμη RAM. Ο κάθε υπολογιστής έτρεχε λειτουργικό Debian. Παρακάτω παρατίθενται τα αποτελέσματα των πειραμάτων που προέκυψαν από τους μέσους όρους 3 εκτελέσεων για κάθε επιμέρους διαδικασία.

6.1 Ομαδοποίηση κειμένων

Στον Πίνακα 1 και στο Σχήμα 16 φαίνεται ο χρόνος εκτέλεσης της ομαδοποίησης κειμένων όταν χρησιμοποιείται διαφορετικός αριθμός πυρήνων.

Πίνακας 1 Ομαδοποίηση Κειμένων και Ποσοστό Βελτίωσης για διαφορετικό αριθμό πυρήνων

Αριθμός Πυρήνων	Χρόνος Εκτέλεσης (δευτερόλεπτα)	Ποσοστό Βελτίωσης με βάση τους 2 πυρήνες
2	39,224	-
4	32,953	15,99 %
8	28,419	27,55 %
12	31,992	18,44 %



Σχήμα 15 Χρόνος εκτέλεσης ομαδοποίησης κειμένων για διαφορετικό αριθμό πυρήνων

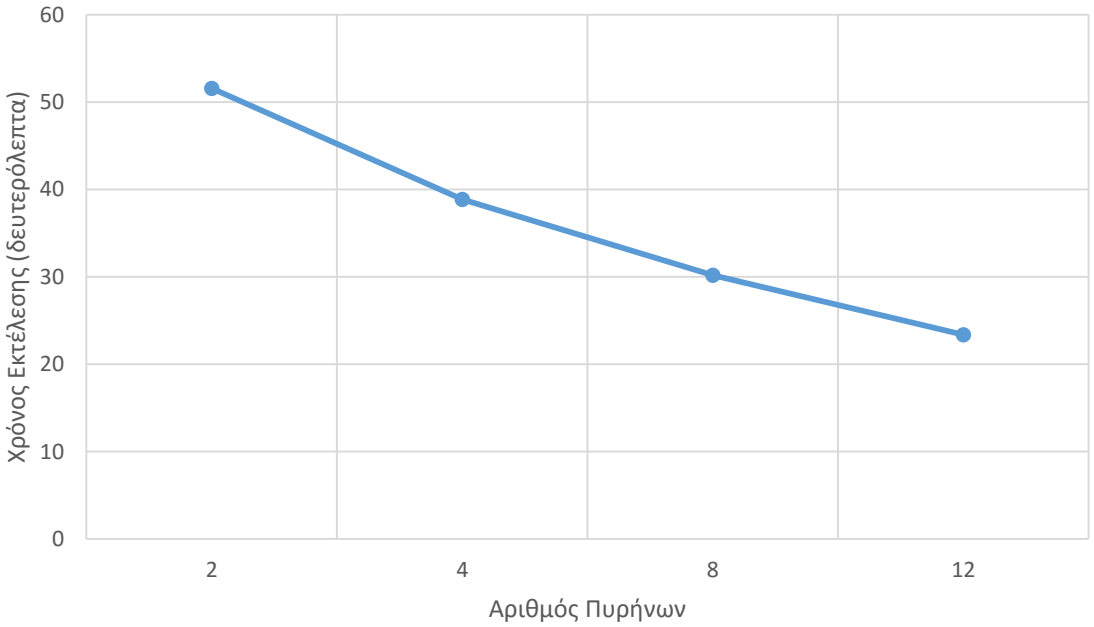
Από τα παραπάνω στοιχεία φαίνεται ότι αυτό το μέρος της διαδικασίας εξαγωγής περιλήψεων δεν είναι το περισσότερο χρονοβόρο. Επίσης, φαίνεται ότι από 8 πυρήνες μέχρι 12 ο χρόνος αυξάνεται. Αυτό συμβαίνει γιατί ενώ μπορεί τα κείμενα να είναι μεγάλα, οι κεφαλαίες λέξεις και οι αριθμοί που υπάρχουν να είναι λίγα σε σχέση με το κείμενο, άρα η παραλληλία μετά από ένα σημείο και μετά δεν συμφέρει γιατί χάνεται χρόνος στην επικοινωνία μεταξύ των partitions.

6.2 Διαχωρισμός Προτάσεων

Στον Πίνακα 2 και στο Σχήμα 17 φαίνεται ο χρόνος εκτέλεσης για τον διαχωρισμό προτάσεων.

Πίνακας 2 Διαχωρισμός Προτάσεων και Ποσοστό Βελτίωσης για διαφορετικό αριθμό πυρήνων

Αριθμός Πυρήνων	**Χρόνος Εκτέλεσης (δευτερόλεπτα)**	**Ποσοστό Βελτίωσης με βάση τους 2 πυρήνες**
2	51,567	-
4	38,868	24,63 %
8	30,188	41,46 %
12	23,388	54,65 %



Σχήμα 16 Χρόνος εκτέλεσης διαχωρισμού προτάσεων για διαφορετικό αριθμό πυρήνων

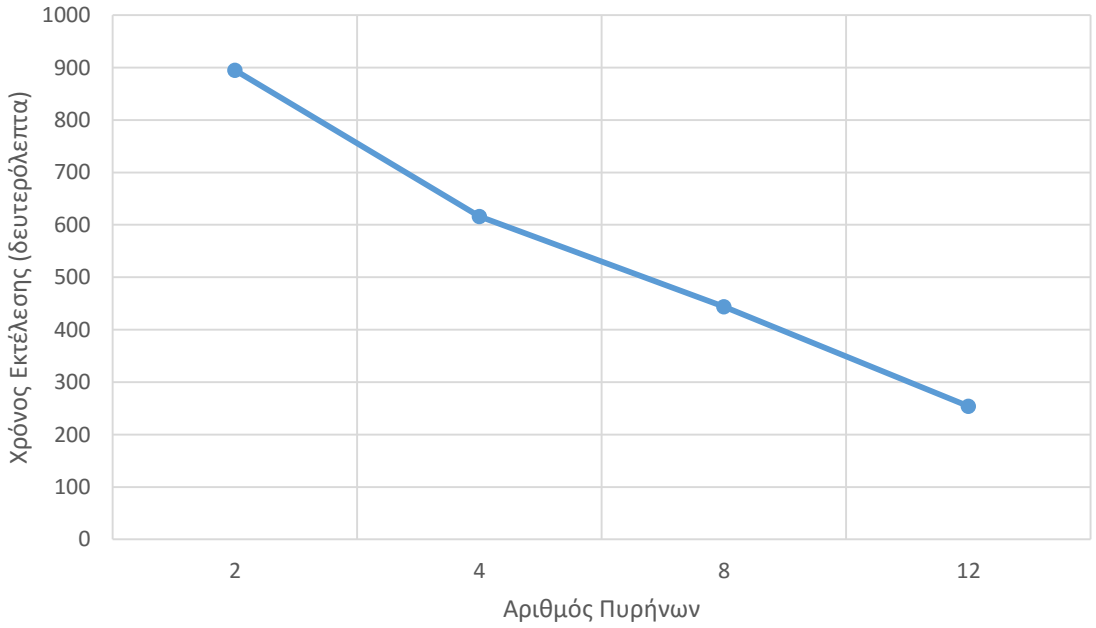
Από τα παραπάνω φαίνεται ότι το πρόβλημα διαχωρισμού προτάσεων παραλληλοποιήθηκε επιτυχώς και ότι η εξαγωγή των μη καλά σπασμένων προτάσεων σε έναν κόμβο για να σπάσουν επιτυχώς ξανά, δεν προσδίδει κάποια επιβάρυνση στο χρόνο εκτέλεσης.

6.3 Σύγκριση Προτάσεων και Markov Clustering

Στον Πίνακα 3 και στο Σχήμα 18 φαίνονται οι χρόνοι εκτέλεσης της σύγκρισης όλων των προτάσεων μεταξύ τους για τη δημιουργία του πίνακα ομοιοτήτων μαζί με τον αλγόριθμο Markov Clustering.

Πίνακας 3 Σύγκριση Προτάσεων και MCL, Ποσοστό Βελτίωσης για διαφορετικό αριθμό πυρήνων

Αριθμός Πυρήνων	**Χρόνος Εκτέλεσης (δευτερόλεπτα)**	**Ποσοστό Βελτίωσης με βάση τους 2 πυρήνες**
2	895,34	-
4	616,406	31,16 %
8	444,184	50,39 %
12	254,093	71,62 %



Σχήμα 17 Χρόνος εκτέλεσης της σύγκρισης προτάσεων και του αλγορίθμου Markov Clustering για διαφορετικό αριθμό πυρήνων

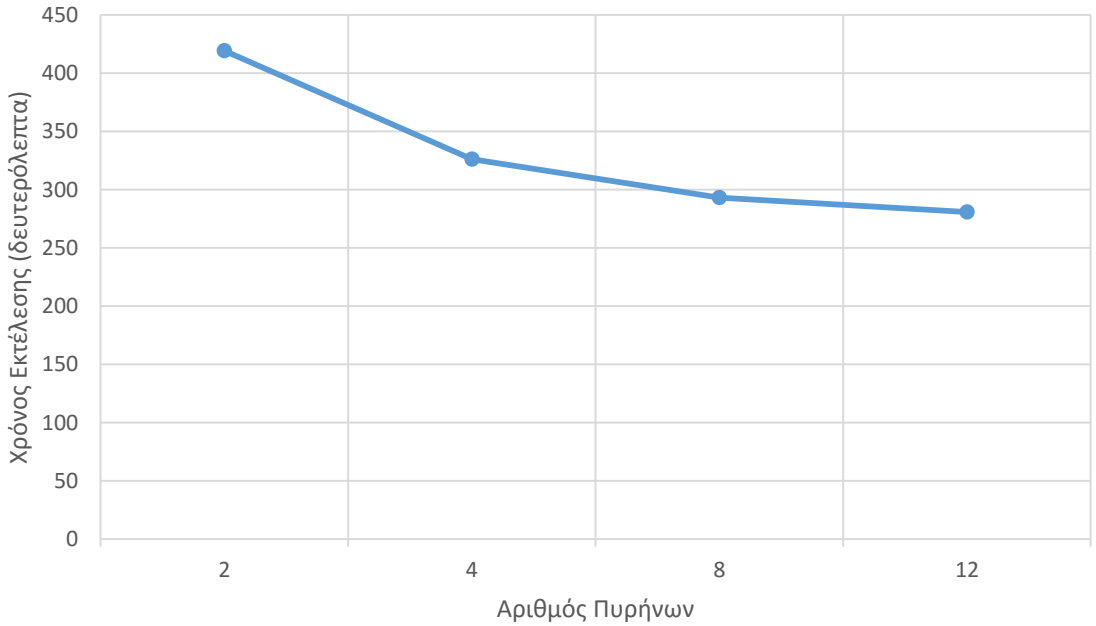
Το παραπάνω πρόβλημα, όπως και ο διαχωρισμός προτάσεων κατανεμήθηκε με επιτυχία και αποδεικνύει ότι το Apache Spark καταφέρνει επιτυχώς να κατανείμει από τον master σε όλους τους workers αυτόν το μεγάλο πίνακα προτάσεων. Συγκεκριμένα, είναι το πρόβλημα που παραλληλοποιήθηκε με μεγαλύτερη επιτυχία εφόσον έχει μεγαλύτερη κλιμάκωση στο διάγραμμα.

6.4 Δημιουργία Θεμάτων και Νοήματος

Στον Πίνακα 4 και στο Σχήμα 19 φαίνονται οι χρόνοι εκτέλεσης για τη δημιουργία θεμάτων, δηλαδή η λειτουργία διασταύρωσης μεταξύ των προτάσεων που βρίσκονται στην ίδια ομάδα προτάσεων και η λειτουργία ένωσης αυτών των θεμάτων για τη δημιουργία του νοήματος του γεγονότος.

Πίνακας 4 Δημιουργία Θεμάτων και Νοήματος, Ποσοστό Βελτίωσης για διαφορετικό αριθμό πυρήνων

Αριθμός Πυρήνων	**Χρόνος Εκτέλεσης (δευτερόλεπτα)**	**Ποσοστό Βελτίωσης με βάση τους 2 πυρήνες**
2	419,166	-
4	326,101	22,21 %
8	293,133	30,07 %
12	280,819	33,01 %



Σχήμα 18 Χρόνοι εκτέλεσης για τη δημιουργία νοήματος γεγονότος για διαφορετικό αριθμό πυρήνων

Μία άλλη χρονοβόρα διαδικασία στην εξαγωγή περίληψης και φαίνεται πως αυτή βελτιώνεται με την προσθήκη περισσότερων πυρήνων, αλλά όχι τόσο πολύ σαν τις προηγούμενες. Αυτό γίνεται γιατί αρχικά οι γράφοι είναι μικροί (γράφοι προτάσεων) και ύστερα μετά από την ένωση αρχίζουν και μεγαλώνουν. Μέχρι να μεγαλώσουν αρκετά ώστε να συμφέρει η παραλληλία, χάνεται χρόνος στην επικοινωνία των partitions. Αναφέρθηκε, επίσης, σε προηγούμενο κεφάλαιο ότι όταν γίνεται

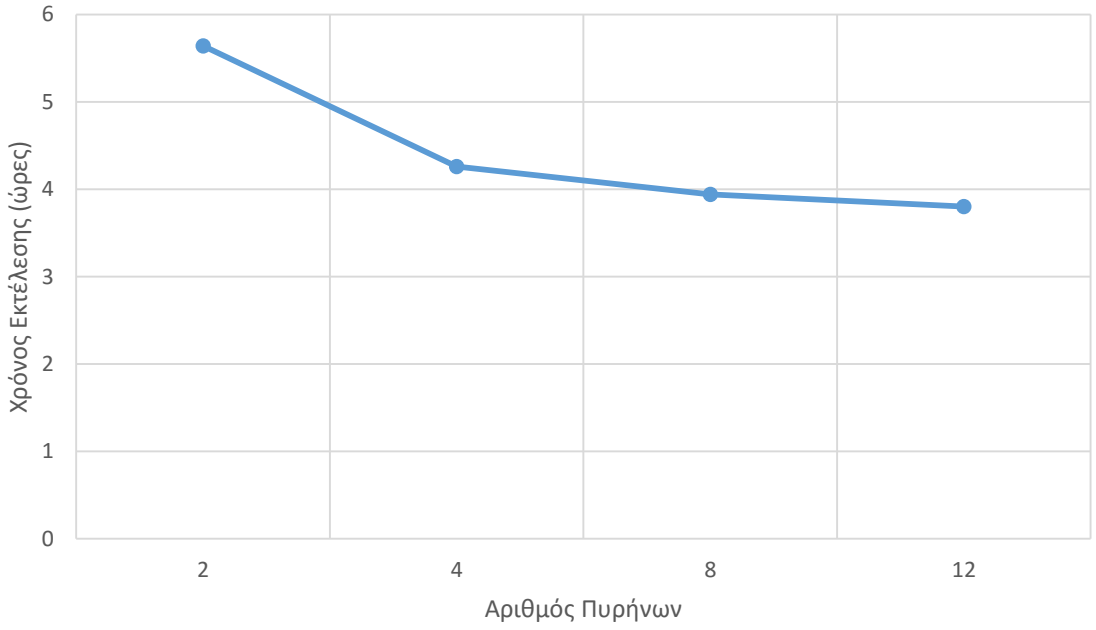
η διασταύρωση των γράφων πρώτα αυτοί οι γράφοι διασταυρώνονται σε έναν worker και μετά γίνεται repartition έτσι ώστε να μπορούν να ενωθούν σε πολλούς γιατί τότε είναι που συνεχώς θα αυξάνεται το μέγεθος του γράφου. Σε αυτήν την περίπτωση ίσως βοηθούσε πάλι να χρησιμοποιηθεί η βιβλιοθήκη JInsect ώστε να διασταυρωθούν οι μικροί γράφοι προτάσεων σειριακά και ύστερα να κατανεμηθούν για να ενωθούν παράλληλα. Σε περιπτώσεις όπου τα δεδομένα είναι λίγα η παραλληλία δεν συμφέρει καθόλου.

6.5 Συνολικός Χρόνος και Αποτελέσματα

Στον Πίνακα 5 και στο Σχήμα 20 φαίνονται οι συνολικοί χρόνοι εκτέλεσης για τη διαδικασία εξαγωγής περιλήψεων.

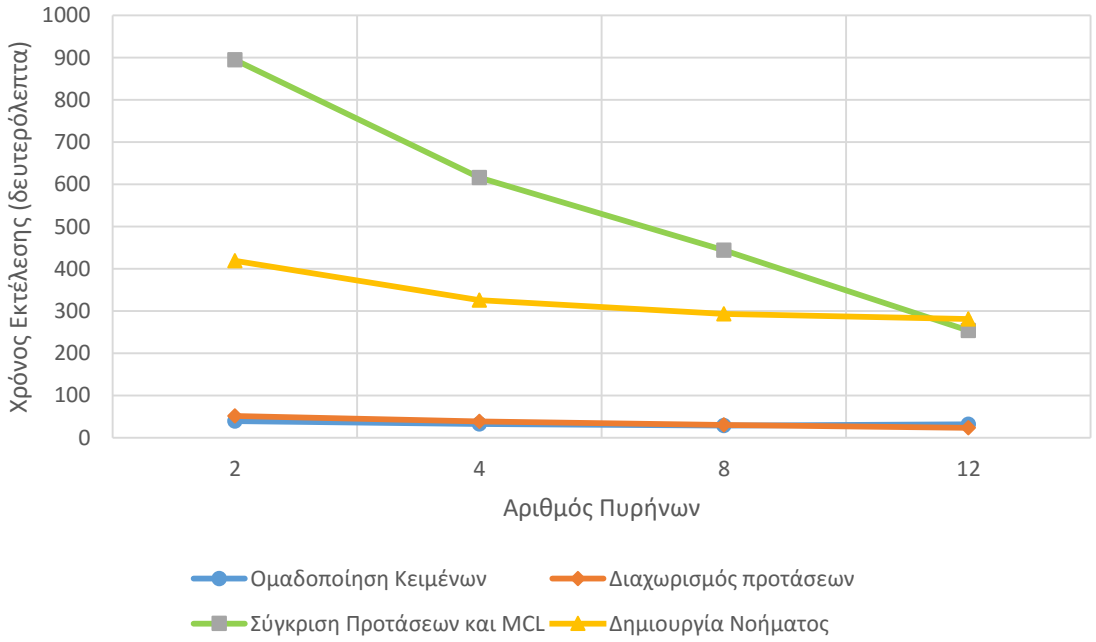
Πίνακας 5 Συνολικός Χρόνος Εκτέλεσης και Ποσοστό Βελτίωσης για διαφορετικό αριθμό πυρήνων

Αριθμός Πυρήνων	**Χρόνος Εκτέλεσης (ώρες)**	**Ποσοστό Βελτίωσης με βάση τους 2 πυρήνες**
2	5,64	-
4	4,26	24,47 %
8	3,94	30,14 %
12	3,8	32,63 %



Σχήμα 19 Συνολικός χρόνος εκτέλεσης εξαγωγής περίληψης για διαφορετικό αριθμό πυρήνων

Από τα παραπάνω φαίνεται ότι ο συνολικός χρόνος εξαγωγής περίληψης ανέρχεται σε ώρες σε αντίθεση με κάποιες από τις επιμέρους λειτουργίες που ανέρχονται σε λεπτά. Αυτό συμβαίνει γιατί σπαταλάτε αρκετός χρόνος στη σύγκριση της κάθε πρότασης με το νόημα του γεγονότος, άρα όσες περισσότερες προτάσεις υπάρχουν σε ένα γεγονός τόσο πιο χρονοβόρα θα είναι η διαδικασία αυτής της σύγκρισης, εφόσον σπάμε τους γράφους σε πολλά κομμάτια και δεν παραλληλοποιείται η σύγκριση αυτή. Ίσως, τελικά, σε αυτό το κομμάτι της εξαγωγής περιλήψεων συμφέρει να σπάμε τη σύγκριση σε πολλά κομμάτια όπως έγινε και στη σύγκριση όλων των προτάσεων μεταξύ τους χρησιμοποιώντας τη βιβλιοθήκη JInsect. Σε κάθε περίπτωση όμως χρειάζονται περισσότερα πειράματα. Στο Σχήμα 21 φαίνονται στο ίδιο διάγραμμα όλες οι επιμέρους διαδικασίες της εξαγωγής περίληψης για να φανεί περισσότερο ποιο μέρος παίρνει περισσότερο χρόνο.



Σχήμα 20 Χρόνος εκτέλεσης όλων των επιμέρους διαδικασιών εξαγωγής περίληψης

Πίνακας 6 Ποσοστό Χρόνου εκτέλεσης απο τον Συνολικό για κάθε επιμέρους διαδικασία

Επιμέρους Διαδικασία	**Ποσοστό του Συνολικού Χρόνου**
Ομαδοποίηση Κειμένων	0,19 %
Διαχωρισμός Προτάσεων	0,25 %
Σύγκριση Προτάσεων και MCL	4,4 %
Δημιουργία Θεμάτων και Νοήματος	2,06 %
Σύγκριση Προτάσεων με το Νόημα	93,1 %

Όπως και με τα προηγούμενα διαγράμματα, το ίδιο και σε αυτό φαίνεται ότι η ομαδοποίηση κειμένων και ο διαχωρισμός των προτάσεων αποτελούν το μικρότερο μέρος της συνολικής διαδικασίας. Τέλος, από τον Πίνακα 6 φαίνεται πως η σύγκριση όλων των προτάσεων με το νόημα του γεγονότος αποτελεί την πιο χρονοβόρα διαδικασία. Αυτό είναι λογικό εφόσον όσο περισσότερες προτάσεις υπάρχουν τόσο περισσότερο χρόνο θα παίρνει αυτή η διαδικασία.

7. Σύνοψη

Σε αυτήν την πτυχιακή αναπτύχθηκε μία εφαρμογή χρησιμοποιώντας το Apache Spark και τη γλώσσα προγραμματισμού Scala για εξαγωγή περιλήψεων από πολλά κείμενα με τη χρήση γράφων ν-γραμμάτων. Συγκεκριμένα, αναπτύχθηκαν οι μέθοδοι και αλγόριθμοι για τη δημιουργία και σύγκριση των γράφων, καθώς και αλγόριθμοι για ένωση, διασταύρωση κλπ. αυτών των γράφων. Επίσης, αναπτύχθηκαν μέθοδοι για ομαδοποίηση (clustering) κειμένων, αφαίρεση περιττών προτάσεων από μία λίστα προτάσεων, ομαδοποίηση με βάση τον αλγόριθμο Markov Clustering και εξαγωγή περίληψης.

Στη συνέχεια, αναλύθηκε το θεωρητικό υπόβαθρο των γράφων ν-γραμμάτων, περιεγράφηκε ολόκληρος ο αλγόριθμος που οδηγεί στην εξαγωγή των περιλήψεων και δόθηκε αναλυτική περιγραφή του πως παραλληλοποιήθηκε το πρόβλημα αυτό. Ύστερα, δόθηκαν παραδείγματα χρήσης και εκτελέστηκαν μερικά πειράματα που δείχνουν ότι το πρόβλημα αυτό παραλληλοποιήθηκε επιτυχώς, μειώνοντας το χρόνο εκτέλεσης ολόκληρης της διαδικασίας ή/και των επιμέρους διαδικασιών.

Μελετήθηκε το πώς ο συνδυασμός δύο βιβλιοθηκών (JInsect και της παρούσας) δίνει βέλτιστα αποτελέσματα (σύγκριση προτάσεων σε συνδυασμό με την JInsect), γεγονός που υποδεικνύει ότι μεγαλύτερος συνδυασμός των δύο ίσως επιφέρει καλύτερα αποτελέσματα.

8. Αναφορές

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation.

Das, D., & Martins, A. F. (2007). *A Survey on Automatic Text Summarization.*

Dongen, S. v. (2000). A cluster algorithm for graphs.

Elfayoumy, S., & Thoppil, J. (2014). *A survey of Unstructured Text Summarization Techniques.*

Erkan, G., & Radev, D. R. (2004). LexRank: Graph-based Lexical Centrality as Salience in Text Summarization.

Giannakopoulos, G. (2009). Automatic Summarization from Multiple Documents.

Giannakopoulos, G., & Karkaletsis, V. (2011). AutoSummENG and MeMoG in Evaluating Guided Summaries.

Giannakopoulos, G., Kioumourtzis, G., & Karkaletsis, V. (2014). NewSum: "N-Gram Graph"-Based Summarization in the Real World. Στο *Innovative Document Summarization Techniques: Revolutionizing Knowledge Understanding* (σ. Chapter 9).

Giannakopoulos, G., Mavridi, P., Paliouras, G., Papadakis, G. A., & Tserpes, K. (2012). Representation Models for Text Classification: a comparative analysis over three Web document types.

Harabagiu, S., & Lacatusu, F. (2005). *Topic Themes for Multi-Document Summarization.*

Tran, G. B., Tran, T. A., Tran, N. K., Alrifai, M., & Kanhabua, N. (2013). *Leverage Learning to rank in an optimization framework for timeline summarization.* In Proc. TAIA workshop, SIGIR 2013. Available online at: http://www.l3s.de/~gtran/timeline/

Van Durme, B., & Lall, A. (2010). Online Generation of Locality Sensitive Hash Signatures.