Πανεπιστήμιο Ιωαννίνων

Τμήμα Πληροφορικής

Επεξεργασία και δρομολόγηση XML ερωτήσεων σε συστήματα ομότιμων κόμβων

Πτυχιακή Εργασία

Μαρίνα Δρόσου

Επιβλέπουσα καθηγήτρια: Ευαγγελία Πιτουρά

Αύγουστος 2006

Πρόλογος

Στην Πτυχιακή αυτή εργασία εξετάζουμε κάποιες από τις πτυχές του ενδιαφέροντος προβλήματος της αναζήτησης αποτελεσμάτων μέσα σε μεγάλο όγκο δεδομένων. Παρουσιάζουμε μία πρόσφατη δομή, το ιστόγραμμα bloom, που αποτελεί μία περίληψη των αρχικών δεδομένων και μπορεί να χρησιμοποιηθεί αποτελεσματικά στην αναζήτηση αποτελεσμάτων μέσα σε αυτά. Στη συνέχεια επιχειρούμε να επεκτείνουμε την εφαρμογή του και στη δρομολόγηση XML ερωτήσεων σε συστήματα ομότιμων κόμβων.

Η εργασία αυτή εκπονήθηκε στα πλαίσια του προπτυχιακού προγράμματος σπουδών του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων υπό την επίβλεψη της Αναπληρώτριας Καθηγήτριας Ευαγγελίας Πιτουρά την οποία θα ήθελα να ευχαριστήσω για τη συνεργασία.

Ιωάννινα, Αύγουστος 2006

Μαρίνα Δρόσου

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1: Εισαγωγή

Τα τελευταία χρόνια ο όγκος της πληροφορίας που είναι προσβάσιμος από τους χρήστες ηλεκτρονικών υπολογιστών και άλλων ανάλογων μέσων έχει αυξηθεί σημαντικά. Κρίνεται λοιπόν απαραίτητη η εύρεση αποδοτικών τρόπων διαχείρισης και αναζήτησης αυτής της πληροφορίας.

Ένας συνήθης τρόπος πρόσβασης στην πληροφορία στις μέρες μας είναι η διακίνησή της μέσω συστημάτων ομότιμων κόμβων. Πρόκειται για συστήματα υπολογιστών στα οποία κάθε ένας από αυτούς είναι απευθείας συνδεδεμένος μόνο με ένα μικρό σχετικά αριθμό άλλων υπολογιστών μέσω των οποίων μπορεί έμμεσα να συνδεθεί και με τους υπόλοιπους.

Η ποικιλομορφία των διάφορων υπολογιστικών συστημάτων και λοιπών συσκευών έχει οδηγήσει στην ανάγκη περιγραφής της πληροφορίας με κάποιον τρόπο κατανοητό από όλους. Μία προσέγγιση που τείνει να εδραιωθεί ευρέως τα τελευταία χρόνια είναι η χρήση της γλώσσας XML (eXtensible Mark-up Language) [14] για το σκοπό αυτό. Πρόκειται για μία σημασιολογική (περιγραφική) γλώσσα η οποία χρησιμοποιεί ετικέτες για να περιγράψει την πληροφορία, όπως θα εξηγήσουμε στη συνέχεια.

1.1 XML δεδομένα

Όπως αναφέραμε, η XML χρησιμοποιεί ετικέτες για να περιγράψει τα διάφορα δεδομένα [1]. Οι ετικέτες αποτελούνται από συμβολοσειρές που περικλείονται σε αγκύλες. Οι συμβολοσειρές αυτές μπορούν να αποτελούνται από τυχαία σύμβολα (όπως γράμματα και

αριθμούς) αλλά συνήθως προτιμούνται λέξεις οι οποίες περιγράφουν την πληροφορία που εσωκλείεται στις ετικέτες.

Για παράδειγμα, μπορούμε να αναπαραστήσουμε την πληροφορία ότι υπάρχει ένα βιβλίο με τίτλο «Fundamentals of Database Systems» με συγγραφείς τους R. Elmasri και S. B. Navathe που εκδόθηκε από τον εκδοτικό οίκο Addison-Wesley το 2000 όπως φαίνεται στο Σχήμα 1-1:

```
<book>
      <title> Fundamentals of Database Systems </title>
      <author> R. Elmasri </author>
      <author> S. B. Navathe </author>
      <publisher> Addison-Wesley </publisher>
      <year> 2000 </year>
</book>
```

Σχήμα 1-1: Περιγραφή δεδομένων μέσω XML.

Η αναζήτηση σε δεδομένα αυτής της μορφής μπορεί να πραγματοποιηθεί με ερωτήσεις της μορφής μονοπατιών. Ένα *μονοπάτι p* ορίζεται ως μία ακολουθία $/p_1/p_2/\ldots/p_n$ όπου $p_1, p_2,\ldots, p_n$ είναι ετικέτες XML. Για παράδειγμα, η αναζήτηση του εκδοτικού οίκου στα παραπάνω δεδομένα γίνεται με την ερώτηση «/book/publisher» το οποίο επιστρέφει ως αποτέλεσμα το «Addison-Wesley». Υπάρχουν και πιο σύνθετες δομές ερωτήσεων τέτοιας μορφής [16] οι οποίες όμως μπορούν να αναλυθούν σε ένα σύνολο απλών ερωτήσεων όπως αυτή του παραδείγματος.

Ο τρόπος με τον οποίο δίνονται απαντήσεις σε ερωτήσεις της παραπάνω μορφής παρουσιάζει ιδιαίτερο ενδιαφέρον καθώς πρέπει να βρεθούν τρόποι αποτελεσματικής και αποδοτικής αναζήτησης αποτελεσμάτων μέσα σε ένα μεγάλο όγκο δεδομένων. Κάτι τέτοιο αποδεικνύεται ακόμα πιο πολύπλοκο (αλλά και ενδιαφέρον) όταν η αναζήτηση πρέπει να πραγματοποιηθεί σε δεδομένα τα οποία είναι αποθηκευμένα σε διάφορες τοποθεσίες, όπως στην περίπτωση των συστημάτων ομότιμων κόμβων. Στα πλαίσια της εργασίας αυτής θα ασχοληθούμε με κάποιες πτυχές αυτού του προβλήματος.

1.2 Σκοπός της εργασίας

Η ανάπτυξη των υπολογιστικών συστημάτων και η ευρεία χρήση τους έχει φέρει ως αποτέλεσμα και τη συσσώρευση μεγάλου όγκου δεδομένων που βρίσκονται αποθηκευμένα σε αυτά. Εξαιτίας αυτού, τεχνικές αναζήτησης που εφαρμόζονται απευθείας στα δεδομένα είναι τις περισσότερες φορές χρονοβόρες και κατά συνέπεια μη αποδοτικές. Για να αυξήσουμε την αποτελεσματικότητα της αναζήτησης στρεφόμαστε πλέον σε τεχνικές που αναζητούν αποτελέσματα όχι στα ίδια τα δεδομένα αλλά σε περιλήψεις αυτών.

Στην εργασία αυτή θα ασχοληθούμε κυρίως με μία ενδιαφέρουσα δομή, το ιστόγραμμα bloom, και τους τρόπους με τους οποίους αυτή μπορεί να χρησιμοποιηθεί στην επεξεργασία και τη δρομολόγηση XML ερωτήσεων σε συστήματα ομότιμων κόμβων. Το ιστόγραμμα bloom είναι μία δομή που αποτελεί μία περίληψη των δεδομένων μας. Συνοπτικά μπορούμε να πούμε πως το ιστόγραμμα αυτό επιχειρεί να ομαδοποιήσει τα διάφορα μονοπάτια που εμφανίζονται στα XML δεδομένα μας με βάση τη συχνότητά τους, δηλαδή τον αριθμό εμφάνισής τους. Ακόμα δίνει μία εκτίμηση για τη συχνότητα των μονοπατιών κάθε ομάδας. Βασικό πλεονέκτημά του είναι το ότι μπορεί να δώσει μία ικανοποιητική περίληψη των δεδομένων καταλαμβάνοντας πολύ μικρότερο χώρο από αυτά.

Στόχος μας είναι η κατασκευή του ιστογράμματος bloom και η μελέτη της χρήσης και της απόδοσης του. Ακόμα προτείνουμε και κάποιες νέες εφαρμογές για τα ιστογράμματα bloom, όπως τη συγχώνευσή τους και τη χρήση τους ως ευρετήρια δρομολόγησης σε συστήματα ομότιμων κόμβων.

1.3 Διάρθρωση της εργασίας

Το υπόλοιπο αυτής της εργασίας είναι δομημένο ως εξής: Στο Κεφάλαιο 2 εισάγουμε τη δομή του ιστογράμματος bloom και περιγράφουμε τα χαρακτηριστικά του, τους αλγορίθμους κατασκευής και χρήσης του καθώς και μία θεωρητική προσέγγιση της απόδοσης του. Στο Κεφάλαιο 3 περιγράφουμε την υλοποίηση του ιστογράμματος bloom και παρουσιάζουμε διάφορα πειράματα που πραγματοποιήσαμε για τη μέτρηση της απόδοσής του και το ρόλο που οι διάφορες σχεδιαστικές παράμετροι του ιστογράμματος διαδραματίζουν σε αυτή. Στο Κεφάλαιο 4 προτείνουμε κάποιους αλγορίθμους συγχώνευσης δύο διαφορετικών ιστογραμμάτων bloom και παρουσιάζουμε πειραματικά αποτελέσματα

της σύγκρισης της απόδοσης των συγχωνευμένων ιστογραμμάτων με την απόδοση μη-συγχωνευμένων ιστογραμμάτων που αναφέρονται στα ίδια δεδομένα. Στο Κεφάλαιο 5 κάνουμε μια μικρή περιγραφή των συστημάτων ομότιμων κόμβων και των μεθόδων αναζήτησης πληροφορίας σε αυτά και εφαρμόζουμε τα συγχωνευμένα ιστογράμματα bloom στη δρομολόγηση ερωτήσεων. Τέλος, στο Παράρτημα Α είναι διαθέσιμα κάποια από τα σημαντικότερα τμήματα του πηγαίου κώδικα της υλοποίησής μας.

Κεφάλαιο 2: Ιστογράμματα Bloom

Στο κεφάλαιο αυτό θα περιγράψουμε τα ιστογράμματα bloom. Σκοπός των ιστογραμμάτων αυτών είναι η οργάνωση πληροφορίας σχετικά με τη συχνότητα εμφάνισης συγκεκριμένων μονοπατιών (όπως αυτά ορίστηκαν στην §1.1) σε ένα σύνολο XML δεδομένων, έτσι ώστε να μπορούμε να αναζητούμε απαντήσεις σε ερωτήσεις της μορφής μονοπατιών μέσω των ιστογραμμάτων bloom και όχι των αρχικών δεδομένων. Θα ξεκινήσουμε με μία εισαγωγή στα φίλτρα bloom (που αποτελούν βασικό συστατικό αυτών των ιστογραμμάτων) και τις ιδιότητές τους και θα συνεχίσουμε με τη δομή των ιστογραμμάτων bloom, την κατασκευή τους και το σφάλμα εκτίμησης της συχνότητας των ερωτήσεων της μορφής μονοπατιών που προκύπτει από τη χρήση τους.

2.1 Φίλτρα Bloom

Ένα *φίλτρο bloom* [2] είναι μία δομή που αναπαριστά ένα σύνολο στοιχείων $P = \{p_1, p_2, \ldots, p_n\}$ μέσω ενός πίνακα bit μήκους m συνοδευόμενο από k συναρτήσεις κατακερματισμού, έστω $h_1, h_2, \ldots, h_k$. Συγκεκριμένα, τα bit του πίνακα μπορούν να λάβουν τις τιμές 0 ή 1 ενώ υποθέτουμε ότι όλες οι συναρτήσεις κατακερματισμού κατακερματίζουν ένα τυχαίο στοιχείο x ομοιόμορφα στο διάστημα $[1, m]$. Αρχικά όλα τα bit του φίλτρου έχουν την τιμή 0. Για να εισάγουμε ένα στοιχείο p στο φίλτρο εφαρμόζουμε κάθε μία από τις k συναρτήσεις κατακερματισμού ξεχωριστά στο p και μεταβάλουμε την τιμή του bit στη θέση $h_i(p)$ του φίλτρου σε 1, για $1 \leq i \leq k$. Για να ταιριάζει ένα στοιχείο q στο φίλτρο θα πρέπει να ισχύει το εξής: Αν εφαρμόσουμε κάθε μία από τις k συναρτήσεις κατακερματισμού ξεχωριστά στο q, θα πρέπει τα bit στις θέσεις $h_i(q)$ του φίλτρου να έχουν τιμή ίση με 1, για $1 \leq i \leq k$.

Υπάρχει περίπτωση ένα στοιχείο q να ταιριάζει στο φίλτρο bloom αλλά να μην ανήκει στο σύνολο P το οποίο αναπαρίσταται από το φίλτρο. Όμως, ένα στοιχείο που δεν ταιριάζει στο φίλτρο bloom δεν μπορεί σε καμία περίπτωση να ανήκει στο αντίστοιχο σύνολο P.

Για παράδειγμα, έστω ότι έχουμε τέσσερις συναρτήσεις κατακερματισμού h_1, h_2, h_3, h_4 και δύο μονοπάτια p και q. Ακόμα, ας υποθέσουμε ότι το φίλτρο bloom που θα χρησιμοποιήσουμε έχει μήκος $m = 8$ bits. Ας υποθέσουμε επίσης πως $h_1(p) = 1$, $h_2(p) = 3$, $h_3(p) = 4$, $h_4(p) = 7$ και $h_1(q) = 2$, $h_2(q) = 3$, $h_3(q) = 6$, $h_4(q) = 8$. Αρχικά το φίλτρο είναι το 00000000. Μετά την εισαγωγή του p το φίλτρο μετασχηματίζεται σε 10110010. Αν σε αυτό το σημείο ελέγξουμε τα bit στις θέσεις 2, 3, 6 και 8 βλέπουμε ότι δεν έχουν όλα την τιμή 1 και έτσι συμπεραίνουμε πως το q δεν ανήκει στο σύνολο που αντιπροσωπεύει το φίλτρο, κάτι που ισχύει. Έστω ότι στη συνέχεια εισάγουμε άλλο ένα μονοπάτι r στο φίλτρο για το οποίο $h_1(r) = 1$, $h_2(r) = 2$, $h_3(r) = 6$, $h_4(r) = 8$. Το φίλτρο πλέον θα είναι το 11110111. Αν σε αυτό το σημείο κάνουμε έναν έλεγχο για το q θα δούμε πως αυτό ταιριάζει στο φίλτρο και θα συμπεράνουμε λανθασμένα πως ανήκει και στο σύνολο που αντιπροσωπεύει το φίλτρο.

Η πιθανότητα τέτοιων σφαλμάτων κατά τη χρήση των φίλτρων bloom εξαρτάται από τα χαρακτηριστικά τους και έχει αποδεχθεί ότι είναι ίση με:

$$\varepsilon = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k \qquad (2.1)$$

όπου n είναι ο αριθμός των μονοπατιών που θα εισαχθούν στο φίλτρο. Έχει επίσης αποδειχθεί πως η πιθανότητα αυτή ελαχιστοποιείται όταν $k = \ln 2 \cdot \frac{m}{n}$. Αν θέσουμε $l = \frac{m}{n}$, η ελάχιστη τιμή της πιθανότητας ε όπως προκύπτει από τη Σχέση 2.1 είναι $0,6185^l$.

2.2 Ιστογράμματα Bloom

Στην παράγραφο αυτή θα περιγράψουμε τη δομή των ιστογραμμάτων bloom και τα διάφορα χαρακτηριστικά τους.

2.2.1 Περιγραφή δομής

Ορίζουμε ως *συχνότητα* ενός μονοπατιού τον αριθμό των εμφανίσεων του σε ένα σύνολο δεδομένων D (όπου D μπορεί να είναι ένα σύνολο XML αρχείων).

Ένα ιστόγραμμα bloom [7] είναι ένας δισδιάστατος πίνακας της μορφής $H(paths, v)$ όπου *paths* είναι ένα σύνολο μονοπατιών που συναντούνται στο D και v είναι μία αντιπροσωπευτική τιμή για τις όλες τις συχνότητες των μονοπατιών που ανήκουν στο *paths*.

Επομένως, δοσμένου ενός μονοπατιού p που ανήκει στο D, μπορούμε να βρούμε μία εγγραφή i του πίνακα H τέτοια ώστε το p να ανήκει στο $H_i.paths$ και να θεωρήσουμε την τιμή $H_i.v$ ως τη συχνότητα του p.

Σχεδιαστικές παράμετροι του ιστογράμματος αυτού είναι ο αριθμός εγγραφών του πίνακα H (έστω b) και ο τρόπος με τον οποίο τα μονοπάτια του D διαμοιράζονται στα υποσύνολα H_i.paths έτσι ώστε η συχνότητα $H_i.v$ που επιστρέφεται ως συχνότητα του κάθε μονοπατιού να είναι μία καλή εκτίμηση της πραγματικής του συχνότητας. Ακόμα, ένα άλλο ζήτημα είναι το πώς μπορούμε να αναπαραστήσουμε τα διάφορα υποσύνολα μονοπατιών του D με κάποιον αποδοτικό τρόπο.

Για να διευθετήσει αυτά τα ζητήματα, το ιστόγραμμα bloom έχει τα εξής χαρακτηριστικά:

1. Για τη βελτίωση της εκτίμησης της συχνότητας μονοπάτια με περίπου ίσες συχνότητες ομαδοποιούνται και τοποθετούνται στην ίδια εγγραφή του πίνακα H.
2. Τα μονοπάτια που ανήκουν στην ίδια εγγραφή του πίνακα H αναπαρίστανται μέσω ενός φίλτρου bloom έτσι ώστε να είναι εύκολο να διαπιστωθεί αν ένα τυχαίο μονοπάτι p ανήκει ή όχι στο υποσύνολο των μονοπατιών της εγγραφής.

Για παράδειγμα, εάν τα μονοπάτια που ανήκουν στο D και οι συχνότητες τους είναι αυτά που φαίνονται στο Σχήμα 2-1, τότε το αντίστοιχο ιστόγραμμα bloom θα είναι αυτό που φαίνεται στο Σχήμα 2-2 (όπου με $BF(P)$ συμβολίζουμε το φίλτρο bloom που αντιστοιχεί στο υποσύνολο μονοπατιών P).

Μονοπάτι	Συχνότητα
/a	10
/a/b	10
/a/f/c	99
/a/e	101
/a/z	999
/a/s	1001
/a/i	1499
/a/o	1501

Σχήμα 2-1: Παράδειγμα μονοπατιών - συχνοτήτων.

Φίλτρο bloom	Συχνότητα
BF(/a, /a/b)	10
BF(/a/f/c, /a/e)	100
BF(/a/z, /a/s)	1000
BF(/a/i, /a/o)	1500

Σχήμα 2-2: Ιστόγραμμα bloom για τα δεδομένα του Σχήματος 2-1.

2.2.2 Εκτίμηση συχνότητας μονοπατιού

Στη συνέχεια παρουσιάζουμε τον αλγόριθμο με τον οποίο μπορούμε να λάβουμε μία εκτίμηση της συχνότητας ενός μονοπατιού μέσω ενός ιστογράμματος bloom. Ο αλγόριθμος αυτός προτάθηκε στο [7]. Η είσοδος του αλγορίθμου είναι ένα συγκεκριμένο ιστόγραμμα bloom *BH*(*BF*, *count*) και ένα μονοπάτι *p*. Ως έξοδος του αλγορίθμου επιστρέφεται ένας αριθμός που αντιπροσωπεύει τη συχνότητα του μονοπατιού *p*.

Αλγόριθμος 2-1: QueryBloomHistogram(BH, p)

```
1: count = 0; k = 0;
2: for i = 1 to b do
3:  if IsMember(p, BH.BF[i]) then
4:   count += BH.count[i];
5:   k++;
6:  end if
7: end for
8: if k > 0 then
9:  return count/k;
10: else
11:  return 0;
12: end if
```

Αλγόριθμος 2-1: Εκτίμηση συχνότητας μονοπατιού.

Η συνάρτηση `IsMember` (Γραμμή 3, Αλγόριθμος 2-1) παίρνει ως είσοδο το μονοπάτι `p` και το φίλτρο bloom `BH.BF[i]`. Επιστρέφει `true` αν το μονοπάτι ταιριάζει στο φίλτρο και `false` διαφορετικά. Αν και οποιοδήποτε μονοπάτι θα ανήκει το πολύ σε ένα υποσύνολο μονοπατιών, υπάρχει περίπτωση να ταιριάζει σε παραπάνω από ένα από τα αντίστοιχα φίλτρα bloom. Σε αυτήν την περίπτωση επιστέφεται ο μέσος όρος των αντίστοιχων συχνοτήτων.

2.2.3 Σφάλμα εκτίμησης συχνότητας

Ένα από τα πλεονεκτήματα του ιστογράμματος bloom είναι ότι το σφάλμα της εκτίμησης της συχνότητας ενός μονοπατιού είναι φραγμένο και μπορεί να υπολογιστεί. Όπως αναφέραμε (§2.1), υπάρχει περίπτωση ένα στοιχείο q να ταιριάζει σε ένα φίλτρο bloom αλλά να μην ανήκει στο αντίστοιχο σύνολο P το οποίο αναπαρίσταται από το φίλτρο και έχει αποδειχθεί ότι η πιθανότητα αυτού του είδους σφάλματος καθορίζεται από τον αριθμό bits του φίλτρου m και τον αριθμό των συναρτήσεων κατακερματισμού k και μάλιστα είναι περίπου ίση με $\varepsilon \approx \left(1-e^{-kn/m}\right)^k$, όπου n το πλήθος των στοιχείων του P.

Στη συνέχεια υποθέτουμε πως έχουμε ένα ιστόγραμμα με b εγγραφές (τις οποίες στο εξής θα αναφέρουμε και ως *κάδους*) στο οποίο έχουν εισαχθεί συνολικά n μονοπάτια με συχνότητες που ανήκουν στο διάστημα $(0, M]$. Υποθέτουμε ακόμα πως η συχνότητα του κάδου i είναι ίση με V_i, με $1 \leq V_i \leq M$ και πως η πραγματική συχνότητα ενός μονοπατιού

είναι ίση με V. Μπορούμε να διακρίνουμε τις ερωτήσεις σε δύο είδη. Πρόκειται για τις *θετικές ερωτήσεις*, δηλαδή μονοπάτια τα οποία όντως υπάρχουν στο αρχικό σύνολο δεδομένων και για τα οποία $V \neq 0$, και τις *αρνητικές ερωτήσεις*, δηλαδή μονοπάτια που δεν εμφανίζονται στο αρχικό σύνολο δεδομένων και άρα έχουν πραγματική συχνότητα ίση με το μηδέν ($V = 0$).

Για τη μέτρηση του σφάλματος χρησιμοποιούμε την τιμή του απόλυτου σφάλματος. Το *απόλυτο σφάλμα* μιας εκτίμησης με τιμή $\hat{X}$ όταν η αντίστοιχη πραγματική τιμή είναι ίση με X είναι ίσο με $\left|X - \hat{X}\right|$.

Το σφάλμα στην τιμή που επιστρέφεται ως συχνότητα ενός μονοπατιού από ένα ιστόγραμμα bloom επηρεάζεται από δύο παράγοντες:

- Πρώτον, το γεγονός ότι το ιστόγραμμα bloom επιστρέφει ως συχνότητα του μονοπατιού όχι την πραγματική συχνότητα αλλά μία προσεγγιστική τιμή v.
- Δεύτερον, το γεγονός ότι ενώ ένα μονοπάτι μπορεί να έχει τοποθετηθεί σε έναν μόνο κάδο του ιστογράμματος bloom μπορεί μετά την τοποθέτηση όλων των μονοπατιών να ταιριάζει σε πολλά από τα φίλτρα bloom του ιστογράμματος.

Για τις αρνητικές ερωτήσεις το σφάλμα προέρχεται εξ ολοκλήρου από τον δεύτερο παράγοντα. Έστω ότι k κάδοι αναφέρουν λανθασμένα ότι το μονοπάτι ταιριάζει σε αυτούς. Τότε το σφάλμα θα είναι ίσο με $e_k = \left|\frac{1}{k}\sum_{i=1}^{k} V_i - V\right|$ (με V = 0). Επειδή $V_i \leq M$ θα είναι $e_k \leq M$. Επομένως, επειδή οποιοσδήποτε συνδυασμός κάδων μπορεί να αναφέρει λανθασμένα ότι το μονοπάτι ταιριάζει σε αυτόν, για μία αρνητική ερώτηση το αναμενόμενο σφάλμα θα είναι ίσο με:

$$E = \sum_{i=1}^{b} \binom{b}{i} \varepsilon^i (1-\varepsilon)^{b-i} e_i < b\varepsilon \cdot M \qquad (2.2)$$

(όπου ε το σφάλμα των φίλτρων bloom όπως ορίστηκε παραπάνω). Επομένως σε αυτήν την περίπτωση το σφάλμα είναι φραγμένο.

Οι θετικές ερωτήσεις επηρεάζονται και από τους δύο παράγοντες. Αν V η πραγματική συχνότητα και V_* η εκτίμηση του ιστογράμματος bloom, τότε το σφάλμα είναι ίσο με:

$$E = (1-\varepsilon)^{b-1}\left|V - V_*\right| + \sum_{i=1}^{b-1}\binom{b-1}{i}\varepsilon^i(1-\varepsilon)^{b-i-1}e_{i+1} \qquad (2.3)$$

Ο δεύτερος όρος του αθροίσματος προκύπτει όπως και στην περίπτωση των αρνητικών ερωτήσεων, μόνο που τώρα λανθασμένη απάντηση μπορούν να δώσουν μέχρι b-1 κάδοι (αφού το μονοπάτι υπάρχει στο αρχικό σύνολο δεδομένων θα έχει σίγουρα τοποθετηθεί σε κάποιον από τους κάδους). Ο πρώτος όρος του αθροίσματος εκφράζει το σφάλμα που προκύπτει από τον πρώτο παράγοντα που αναφέρθηκε πριν. Και σε αυτήν την περίπτωση το σφάλμα συνεχίζει να είναι φραγμένο και μάλιστα $E < (1-\varepsilon)^{b-1}\left|V - V_*\right| + (b-1)\varepsilon \cdot M$.

2.2.4 Κατασκευή ιστογράμματος Bloom

Για την κατασκευή ενός ιστογράμματος bloom πρέπει να καθοριστούν δύο ζητήματα. Αυτά είναι το πώς θα χωριστούν τα μονοπάτια σε υποσύνολα (ή *κάδους*) και πώς θα υπολογιστεί η αντιπροσωπευτική τιμή συχνότητας του κάθε κάδου. Ας υποθέσουμε ότι όλα τα μονοπάτια και οι συχνότητές τους είναι αποθηκευμένα σε έναν πίνακα T κατά αύξουσα σειρά συχνότητας.

Η τιμή συχνότητας του κάθε κάδου επιλέγεται έτσι ώστε να είναι όσο το δυνατό πιο κοντά σε κάθε πραγματική συχνότητα που ανήκει στον κάδο. Είναι δηλαδή ίση με τη μέση τιμή όλων των συχνοτήτων των μονοπατιών του κάδου.

Η εύρεση των ορίων του κάθε κάδου είναι πιο πολύπλοκη. Αρχικά ορίζουμε ως $f(x, y)$ το σφάλμα που αντιστοιχεί σε έναν κάδο που περιέχει όλα τα μονοπάτια από το x-οστό ως το y-οστό (με τη βάση τη σειρά που αυτά είναι αποθηκευμένα στον T). Ακόμα ορίζουμε ως $OPT[x, b]$ το βέλτιστο σφάλμα που έχουμε όταν χρησιμοποιούμε b κάδους για να χωρίσουμε τα πρώτα x μονοπάτια που βρίσκονται στον T. Παρατηρούμε ότι:

$$OPT[n,b] = \min_{i=b-1}^{n-1}\{OPT[i,b-1] + f(i+1,n)\} \quad (2.4)$$

Εάν *PSUM* είναι ο αθροιστικός πίνακας συχνοτήτων με βάση τον *T* (δηλαδή *PSUM*[*i*] = *PSUM*[*i*-1] + *T*[*i*]) τότε μπορεί να αποδειχθεί (βλ. Παράρτημα Β) πως:

$$f(x,y) = PSUM[y] + PSUM[x-1] - 2PSUM[t] + (2t - x - y + 1)count[t] \quad (2.5)$$

όπου *t* είναι ο αύξων αριθμός του μονοπατιού που βρίσκεται στο μέσο του *x*-οστού και του *y*-οστού μονοπατιού και *count*[*i*] η συχνότητά του.

Στον Αλγόριθμο 2-2 παρουσιάζεται η διαδικασία εύρεσης του βέλτιστου ιστογράμματος bloom. Ο αλγόριθμος αυτός προτάθηκε στο [7]. Μπορεί να τροποποιηθεί έτσι ώστε να επιστρέφει και τα βέλτιστα όρια για τον κάθε κάδο (κάτι που έχει γίνει και στην υλοποίησή μας). Η είσοδος του αλγορίθμου αποτελείται από έναν ταξινομημένο κατά αύξουσα σειρά πίνακα συχνοτήτων *x*, τον αριθμό των μονοπατιών *n* και τον επιθυμητό αριθμό κάδων *b*. Ως έξοδο έχουμε το συνολικό σφάλμα του βέλτιστου ιστογράμματος.

Αλγόριθμος 2-2: BuildHistogram(x[], n, b)

```
1: PSUM[1] = x[1]
2: for i = 2 to n do
3:  PSUM[i] = PSUM[i - 1] + x[i]
4: end for
5: for i = 1 to n do
6:  OPT[i,1] = f(1,i)
7: end for
8: for k = 2 to b do
9:  for j = 1 to n do
10:   OPT[j,k] = +∞
11:   for i = k - 1 to j - 1 do
12:    OPT[j,k] = min(OPT[j,k], OPT[i,k-1]+f(i+1,j))
13:   end for
14:  end for
15: end for
16: return OPT[n,b]
```

Αλγόριθμος 2-2: Κατασκευή ιστογράμματος bloom.

2.2.5 Μέγεθος ιστογράμματος Bloom

Όπως είδαμε το ιστόγραμμα bloom αποτελείται από ένα σύνολο εγγραφών της μορφής (*φίλτρο bloom*, *συχνότητα*). Επομένως, το μέγεθός του εξαρτάται από το μέγεθος αυτών των εγγραφών. Εάν υποθέσουμε ότι το ιστόγραμμά μας έχει *b* κάδους, το μήκος των φίλτρων bloom είναι *m* bits και ότι ένας ακέραιος αριθμός έχει μέγεθος *a* bits, τότε το ιστόγραμμά μας θα έχει μέγεθος (σε bits) ίσο με:

$$(m+a)\cdot b \tag{2.6}$$

Θα μπορούσαμε να μειώσουμε το χώρο που απαιτείται αν χρησιμοποιούσαμε φίλτρα bloom μεταβλητού μεγέθους. Όπως αναφέραμε στην §2.1, η βέλτιστη πιθανότητα σφάλματος ε των φίλτρων bloom εξαρτάται από τον παράγοντα $l = \frac{m}{n}$ όπου *m* το μήκος τους και *n* το πλήθος των μονοπατιών που έχουν εισαχθεί σε αυτά. Αν καθορίσουμε ένα κοινό *l* για όλα τα φίλτρα του ιστογράμματος, το μήκος τους θα καθορίζεται από τον αριθμό των μονοπατιών που θα πρέπει να εισαχθούν σε καθένα από αυτά. Στην περίπτωση αυτή το συνολικό μέγεθος (σε bit) των φίλτρων bloom θα είναι ίσο με $l \cdot \sum_{1 \le i \le b} n_i \approx l \cdot n$, όπου n_i το πλήθος των μονοπατιών που έχουν εισαχθεί στον *i*-οστό κάδο. Στην περίπτωση αυτή το συνολικό μέγεθος του ιστογράμματος bloom (σε bit) θα είναι:

$$l \cdot n + a \cdot b \tag{2.7}$$

Με βάση τον παραπάνω τύπο μπορούμε να υπολογίσουμε το μέγιστο αριθμό κάδων που μπορούμε να έχουμε όταν διαθέτουμε έναν ορισμένο χώρο για το ιστόγραμμά μας. Πιο συγκεκριμένα, ο ελάχιστος χώρος που απαιτείται για ένα ιστόγραμμα bloom με παράγοντα *l* προκύπτει από τη Σχέση 2.7 για $b = 1$ και είναι ίσος με $S_{\min} = l \cdot n + a$. Εφόσον το πλήθος των κάδων του ιστογράμματος δεν έχει νόημα να είναι μεγαλύτερο από το πλήθος των μονοπατιών, ο μέγιστος χώρος που μπορεί να απαιτηθεί από ένα ιστόγραμμα δίνεται πάλι από τη Σχέση 2.7 για $b = n$ και είναι ίσος με $S_{\max} = l \cdot n + a \cdot n$. Αν χρησιμοποιήσουμε φίλτρα bloom σταθερού μεγέθους τα αντίστοιχα μεγέθη προκύπτουν από τη Σχέση 2.6 και είναι $S_{\min} = (m+a)$ και $S_{\max} = (m+a) \cdot n$.

Επομένως, αν έχουμε διαθέσιμο χώρο μεγέθους S για ένα ιστόγραμμα με n μονοπάτια και παράγοντα l, ο μέγιστος αριθμός κάδων b που μπορούμε να έχουμε είναι:

- Αν $S > S_{\max}$, τότε $b = n$ και δημιουργούμε ένα ιστόγραμμα που απαιτεί χώρο ίσο με $S_{\max}$. Εναλλακτικά μπορούμε να μεταβάλλουμε τον παράγοντα l ώστε να εκμεταλλευτούμε τον επιπλέον χώρο που μας δίνεται βελτιώνοντας την απόδοση του ιστογράμματος.
- Αν $S \in [S_{\min}, S_{\max}]$, τότε $b = \frac{S - S_{\min}}{a}$.
- Αν $S < S_{\min}$, τότε δεν υπάρχει αρκετός χώρος διαθέσιμος για την κατασκευή του ιστογράμματος και θα πρέπει να μεταβάλλουμε τον παράγοντα l, θυσιάζοντας ένα ποσοστό της απόδοσης του ιστογράμματος. Μία άλλη λύση που προτείνεται στο [7] είναι να μην εισάγουμε στο ιστόγραμμα μονοπάτια με μικρές συχνότητες εμφάνισης, μειώνοντας έτσι το n και κατά συνέπεια και το $S_{\min}$.

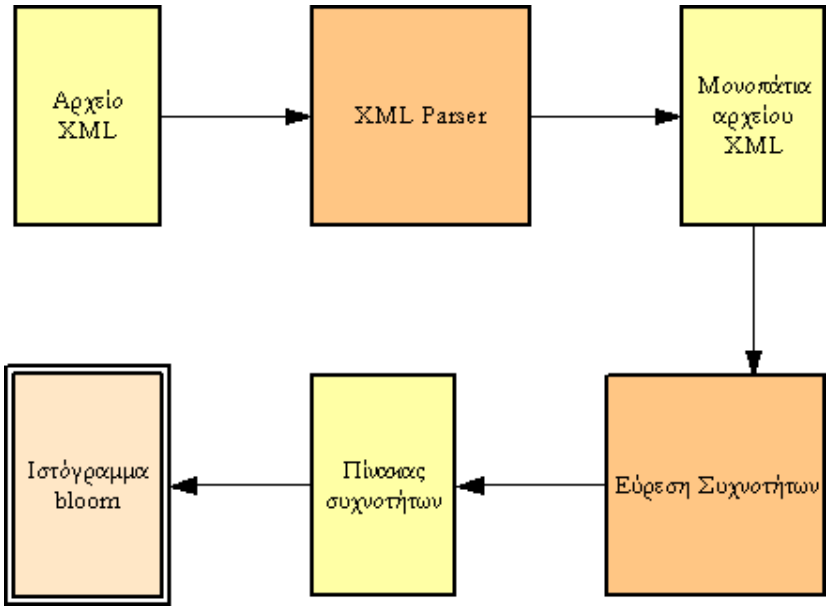
Παρόλα αυτά, η εύρεση του βέλτιστου αριθμού κάδων για ένα ιστόγραμμα bloom δεν είναι κάτι το προφανές. Μεγαλύτερος αριθμός κάδων οδηγεί γενικά σε λιγότερα μονοπάτια ανά κάδο και άρα σε ακριβέστερες εκτιμήσεις συχνοτήτων. Όμως η αύξηση των κάδων οδηγεί και σε περισσότερα φίλτρα bloom και κατά συνέπεια η πιθανότητα κάποια από αυτά να αναφέρουν λανθασμένα ότι ένα μονοπάτι ταιριάζει σε αυτά αυξάνεται (βλ. Σχέσεις 2.2 και 2.3).

Εφόσον δεν υπάρχει κάποια ακριβής σχέση εξάρτησης του συνολικού σφάλματος ενός ιστογράμματος bloom από τον αριθμό των κάδων του, ο βέλτιστος αριθμός των κάδων μπορεί μέχρι στιγμής να βρεθεί μόνο εμπειρικά. Επίσης, καθώς τα ιστογράμματα bloom έχουν γενικά μικρές απαιτήσεις σε χώρο, η χρήση μεταβλητών φίλτρων bloom δεν είναι συχνά προτιμητέα, καθώς αυξάνει και την πολυπλοκότητα υλοποίησης.

Κεφάλαιο 3: Υλοποίηση και πειραματική μελέτη

Στο κεφάλαιο αυτό θα περιγράψουμε την υλοποίηση του ιστογράμματος bloom. Επίσης θα το χρησιμοποιήσουμε για τον υπολογισμό της συχνότητας των μονοπατιών που αποτελούν τα δεδομένα μας. Σε αυτό το στάδιο θεωρούμε ότι το ιστόγραμμα αναφέρεται μόνο στα δεδομένα ενός υπολογιστή, αυτού που το χρησιμοποιεί.

Επιγραμματικά, με τη βοήθεια ενός XML parser βρίσκονται όλα τα μονοπάτια που υπάρχουν σε ένα αρχείο (ή σύνολο αρχείων) XML. Αφού βρεθούν όλα τα μονοπάτια μετρούνται οι αντίστοιχες συχνότητες οι οποίες στη συνέχεια χρησιμοποιούνται για την κατασκευή του ιστογράμματος bloom. Τα παραπάνω βήματα φαίνονται συνοπτικά στο Σχήμα 3-1.



Σχήμα 3-1: Κατασκευή ιστογράμματος bloom.

3.1 Προεπεξεργασία δεδομένων και δημιουργία ιστογράμματος bloom

Το σύστημά μας έχει υλοποιηθεί στη γλώσσα προγραμματισμού C++. Ο κώδικας για όσα αναφέρονται παρακάτω μπορεί να βρεθεί στο Παράρτημα Α.

Σκοπός μας είναι η κατασκευή του ιστογράμματος bloom που αντιστοιχεί σε ένα συγκεκριμένο σύνολο XML δεδομένων. Όπως έχουμε αναφέρει (βλ. §2.2.4), αφετηρία για την κατασκευή του ιστογράμματος bloom είναι ο πίνακας συχνοτήτων των μονοπατιών, δηλαδή ο πίνακας T της μορφής T(*μονοπάτι*, *συχνότητα*). Επομένως, πριν προχωρήσουμε στην κατασκευή του ιστογράμματος bloom πρέπει να δημιουργήσουμε τον πίνακα αυτό.

Αρχικά απομονώνουμε όλα τα διακριτά μονοπάτια που εμφανίζονται στο σύνολο των XML δεδομένων. Το parsing των XML αρχείων γίνεται με τη βοήθεια της βιβλιοθήκης Xerces-C++ [12], η οποία υλοποιεί διάφορες μεθόδους για τη διάσχιση XML δεδομένων, συμπεριλαμβανομένης και της μεθόδου DOM που χρησιμοποιούμε. Στο τέλος αυτού του πρώτου σταδίου είμαστε σε θέση να γνωρίζουμε όλα τα μονοπάτια που υπάρχουν στα XML δεδομένα μας, ανεξαρτήτως μήκους και θέσης εμφάνισής τους στα δεδομένα (έχουμε στην διάθεσή μας δηλαδή όχι μόνο τα μονοπάτια που ξεκινούν από το στοιχείο-ρίζα των XML δεδομένων αλλά και όλα τα υπο-μονοπάτια που εμφανίζονται σε κάποια τυχαία θέση στα δεδομένα).

Στη συνέχεια μετρούμε τη συχνότητα εμφάνισης του κάθε μονοπατιού έτσι ώστε να καταλήξουμε σε μία λίστα από μονοπάτια όπου το καθένα εμφανίζεται μόνο μία φορά και συνοδεύεται από τη συχνότητα εμφάνισής του στο αρχικό σύνολο των XML δεδομένων. Η λίστα αυτή είναι επίσης ταξινομημένη κατά αύξουσα σειρά συχνότητας, κάτι που απαιτείται από τον αλγόριθμο κατασκευής του ιστογράμματος bloom (βλ. §2.2.4), και αποτελεί ουσιαστικά τον ζητούμενο πίνακα T.

Έχοντας πλέον διαθέσιμο τον πίνακα T, υλοποιούμε τον Αλγόριθμο 2-2 της §2.2.4. Η υλοποίησή του δε διαφέρει σημαντικά από τη θεωρητική προσέγγιση. Χρησιμοποιήσαμε συναρτήσεις κατακερματισμού που αποτελούν μέρος της βιβλιοθήκης General Purpose Hash Function [9].

Μετά την εφαρμογή του αλγορίθμου έχουμε στη διάθεσή μας το ιστόγραμμα bloom που αντιστοιχεί στα αρχικά XML δεδομένα και μπορούμε πλέον να το χρησιμοποιήσουμε για την εκτίμηση της συχνότητας ενός τυχαίου μονοπατιού, βασιζόμενοι στον Αλγόριθμο 2-1 που παρουσιάστηκε στην παράγραφο §2.2.4.

3.2 Πειραματική μελέτη ιστογράμματος bloom

Στην ενότητα αυτή παρουσιάζουμε τις μετρήσεις που πραγματοποιήσαμε με σκοπό τη μελέτη της απόδοσης των ιστογραμμάτων bloom. Θα ξεκινήσουμε με μία αναφορά στα σύνολα δεδομένων που χρησιμοποιήσαμε και θα συνεχίσουμε με τρεις σειρές πειραμάτων που δείχνουν πως επηρεάζεται η απόδοση των ιστογραμμάτων bloom από τα χαρακτηριστικά των φίλτρων bloom που αυτά διαθέτουν, τον αριθμό των κάδων τους και το χώρο που τους διατίθεται.

Ως μέτρο της απόδοσης του ιστογράμματος χρησιμοποιούμε το μέσο απόλυτο σφάλμα το οποίο ορίζεται ως εξής: Έστω ότι υποβάλλουμε στο ιστόγραμμα ερωτήσεις για τη συχνότητα n μονοπατιών. Αν η πραγματική συχνότητα του i-στού μονοπατιού στα δεδομένα μας είναι X_i και η εκτίμηση του ιστογράμματος για τη συχνότητά του είναι V_i (με $1 \le i \le n$), τότε το μέσο απόλυτο σφάλμα είναι ίσο με $\frac{1}{n}\sum_{i=1}^{n}\left|X_i - V_i\right|$.

Στόχος μας είναι η ελαχιστοποίηση του μέσου απόλυτου σφάλματος. Όπως είδαμε στο Κεφάλαιο 2 το σφάλμα αυτό εξαρτάται από το σφάλμα των φίλτρων bloom και την αντιπροσωπευτική τιμή συχνότητας του κάθε κάδου. Στα πειράματα του κεφαλαίου αυτού μελετούμε τον τρόπο με τον οποίο οι παράμετροι αυτοί επηρεάζουν το μέσο απόλυτο σφάλμα των ιστογραμμάτων bloom καθώς και το ποιος παράγοντας είναι ο πιο καθοριστικός. Για το λόγο αυτό πραγματοποιήσαμε τρεις σειρές πειραμάτων: μία στην οποία μεταβάλουμε το σφάλμα των φίλτρων bloom των ιστογραμμάτων, μία στην οποία μεταβάλουμε των αριθμό των κάδων (δηλαδή των αριθμό των ομάδων και κατά συνέπεια την ακρίβεια της εκτίμησης της συχνότητας για κάθε ομάδα) και μία στη οποία επιχειρούμε να προσδιορίσουμε το βαθμό κατά τον οποίο κάθε παράγοντας επηρεάζει το τελικό αποτέλεσμα.

3.2.1 Σύνολα δεδομένων

Χρησιμοποιήσαμε τρία διαφορετικά σύνολα δεδομένων για τις μετρήσεις μας, αρκετά διαφορετικά μεταξύ τους:

- Το πρώτο από αυτά είναι το «SIGMOD record» [15], ένα ευρετήριο άρθρων του ομότιτλου περιοδικού. Πρόκειται για ένα σύνολο δεδομένων με αρκετά επαναληπτική δομή (λίγα διαφορετικά μεταξύ τους μονοπάτια που εμφανίζονται το καθένα πολλές φορές).
- Το δεύτερο είναι το «SwissProt» [15] το οποίο περιέχει πληροφορίες για τη δομή διάφορων πρωτεϊνών. Πρόκειται για ένα αρκετά μεγάλο και πολύπλοκο σύνολο δεδομένων που χαρακτηρίζεται από συχνότητες μονοπατιών που απέχουν πολύ η μία από την άλλη.
- Για το τρίτο σύνολο δεδομένων χρησιμοποιήσαμε μία γεννήτρια τυχαίων XML δεδομένων, την XMark [13]. Το τελευταίο αυτό σύνολο χαρακτηρίζεται από μεγάλο αριθμό διαφορετικών μονοπατιών.

Στον Πίνακα 3-1 παρουσιάζονται συνοπτικά κάποια στοιχεία για τα σύνολα δεδομένων που χρησιμοποιήσαμε.

Πίνακας 3-1: Χαρακτηριστικά συνόλων δεδομένων.

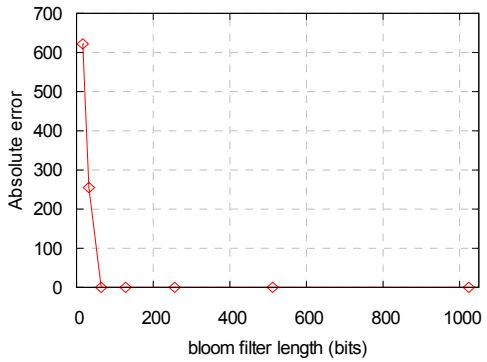
	Μέγεθος	# μονοπατιών	Μέγιστη διαφορά συχνοτήτων	# διαφορετικών συχνοτήτων	Ελάχιστη συχνότητα	Μέγιστη συχνότητα
SIGMOD record	483 KB	31	3670	3	67	3737
SwissProt	112130 KB	335	566307	85	1	566308
XMark	75501 KB	1747	40924	250	1	40925

3.2.2 Μήκος φίλτρων bloom

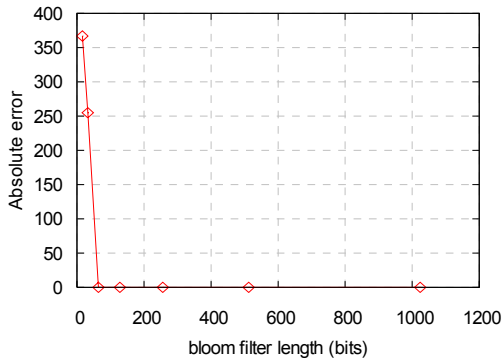
Ενδιαφέρον παρουσιάζει ο τρόπος με τον οποίο τα χαρακτηριστικά των φίλτρων bloom που έχουμε επιλέξει να ενσωματώσουμε στο ιστόγραμμά μας μπορούν να επηρεάσουν την απόδοση αυτού. Στην πρώτη σειρά πειραμάτων που ακολουθεί διατηρήσαμε σταθερό τον

αριθμό των κάδων του ιστογράμματος έτσι ώστε τα μονοπάτια να τοποθετούνται στους ίδιους κάδους κάθε φορά και μεταβάλλαμε το σφάλμα των φίλτρων bloom. Σύμφωνα με την Εξίσωση 2.1, το σφάλμα αυτό μπορεί να μεταβληθεί επηρεάζοντας τις παραμέτρους k και m. Επιλέξαμε εδώ να μεταβάλλουμε το μήκος m των φίλτρων bloom. Επειδή το ιστόγραμμα bloom περιέχει πολλά φίλτρα bloom σταθερού μεγέθους και κάθε φίλτρο έχει διαφορετικό σφάλμα (λόγω του διαφορετικού αριθμού μονοπατιών που εισάγεται στο καθένα), θεωρούμε το μέσο όρο των σφαλμάτων αυτών ως το σφάλμα των φίλτρων του ιστογράμματος.

Στα Σχήματα 3-2 και 3-3 φαίνονται τα αποτελέσματα για το σύνολο δεδομένων «SIGMOD record» για θετικές ερωτήσεις. Στον οριζόντιο άξονα σημειώνεται το μήκος των φίλτρων bloom του ιστογράμματος και στον κάθετο άξονα η τιμή του απόλυτου σφάλματος.

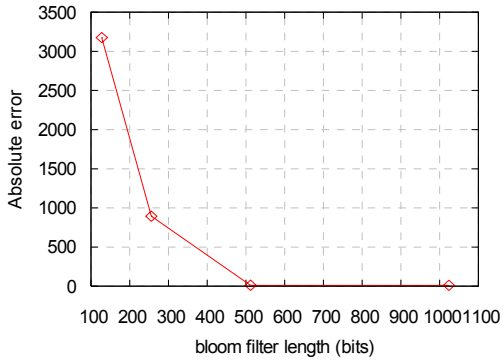


Σχήμα 3-2: SIGMOD record - 5 κάδοι.

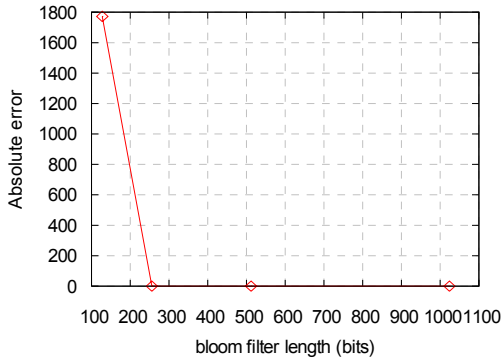


Σχήμα 3-3: SIGMOD record - 10 κάδοι.

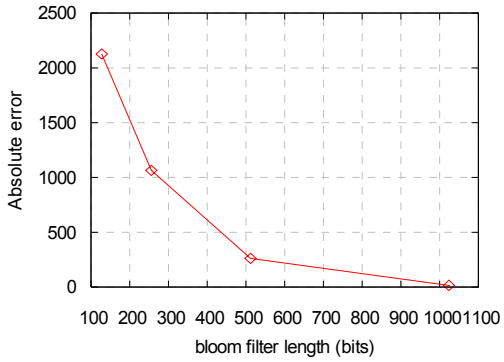
Παρατηρούμε πως καθώς αυξάνεται το μήκος των φίλτρων bloom μειώνεται το σφάλμα της εκτίμησης. Για το συγκεκριμένο σύνολο δεδομένων μάλιστα, για μήκος φίλτρων bloom ίσο ή μεγαλύτερο των 64 bits και για θετικές ερωτήσεις, η εκτίμηση είναι απόλυτα ακριβής (ενώ το μέγεθος του ιστογράμματος είναι αρκετά μικρότερο από αυτό του πλήρους πίνακα συχνοτήτων). Αντίστοιχα αποτελέσματα λαμβάνουμε και για τα άλλα δύο σύνολα δεδομένων, όπου το σφάλμα μειώνεται επίσης κατά πολύ καθώς αυξάνουμε τον αριθμό των bits των φίλτρων bloom των ιστογραμμάτων. Οι αντίστοιχες μετρήσεις φαίνονται στα Σχήματα 3-4 έως 3-7.



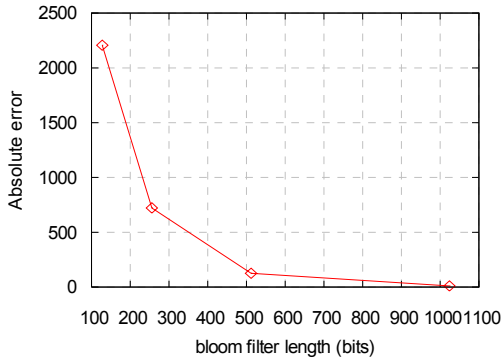
Σχήμα 3-4: SwissProt - 70 κάδοι.



Σχήμα 3-5: SwissProt - 100 κάδοι.

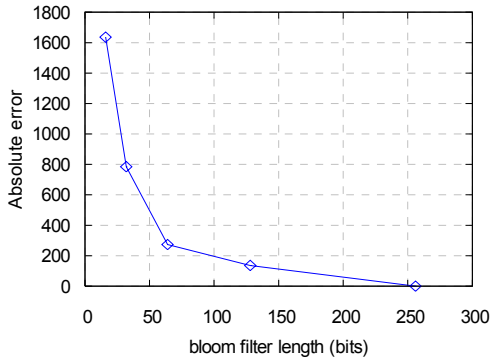


Σχήμα 3-6: XMark- 150 κάδοι.

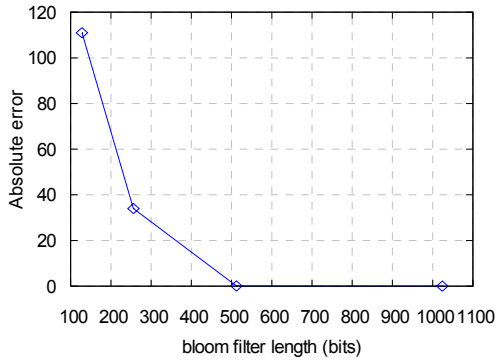


Σχήμα 3-7: XMark- 250 κάδοι.

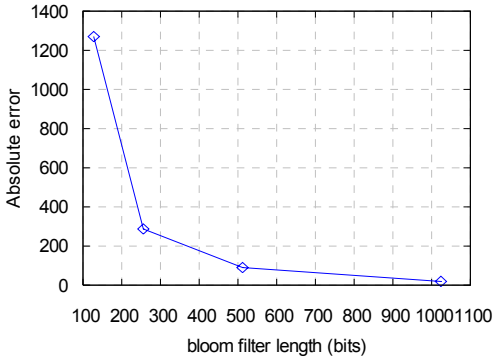
Αντίστοιχα αποτελέσματα λαμβάνουμε και για αρνητικές ερωτήσεις. Να σημειώσουμε πως ως αρνητικές ερωτήσεις χρησιμοποιούμε μονοπάτια κατασκευασμένα από τις ίδιες XML ετικέτες που εμφανίζονται στο αρχικό σύνολο δεδομένων τα οποία όμως δεν εμφανίζονται αυτούσια στο σύνολο αυτό. Με αυτόν τον τρόπο τα αποτελέσματα είναι πιο ρεαλιστικά καθώς μονοπάτια κατασκευασμένα από τις ίδιες ετικέτες έχουν περισσότερες πιθανότητες να χρησιμοποιηθούν ως ερωτήσεις από τους πραγματικούς χρήστες και επίσης να κατακερματιστούν στις ίδιες θέσεις των φίλτρων bloom που κατακερματίζονται και τα μονοπάτια των θετικών ερωτήσεων. Κι εδώ το μέσο απόλυτο σφάλμα μειώνεται καθώς αυξάνεται το μήκος των φίλτρων bloom. Παρουσιάζουμε εδώ ενδεικτικά τη μεταβολή του σφάλματος όταν διατηρούμε το πλήθος των κάδων σταθερό για κάθε σύνολο δεδομένων.



Σχήμα 3-8: SIGMOD record - 10 κάδοι.



Σχήμα 3-9: SwissProt - 70 κάδοι.



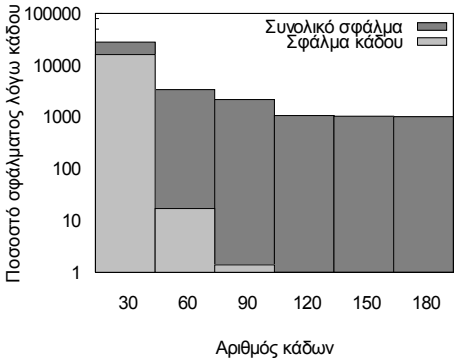
Σχήμα 3-10: XMark- 150 κάδοι.

3.2.3 Αριθμός κάδων

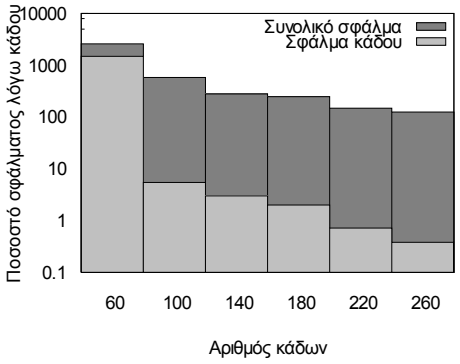
Όπως είδαμε στην §2.2.5, η εύρεση του βέλτιστου αριθμού κάδων δεν είναι μία προφανής διαδικασία.

Αν θεωρήσουμε μηδενικό το σφάλμα των φίλτρων bloom, σε ένα ιδανικό ιστόγραμμα bloom θα είχαμε τόσους κάδους όσες και οι διαφορετικές συχνότητες που εντοπίζονται στο αρχικό σύνολο XML δεδομένων. Με αυτόν τον τρόπο το σφάλμα λόγω της προσέγγισης της πραγματικής συχνότητας θα μηδενιζόταν καθώς κάθε μονοπάτι θα ταίριαζε σε έναν μόνο κάδο ο οποίος θα είχε και την ακριβή του συχνότητα.

Στις γραφικές παραστάσεις των Σχημάτων 3-11 και 3-12 φαίνεται το ποσοστό του σφάλματος εκτίμησης της συχνότητας που οφείλεται στο γεγονός ότι ως συχνότητα ενός μονοπατιού θεωρούμε την αντιπροσωπευτική τιμή του κάδου στο οποίο αυτό ανήκει. Χρησιμοποιήσαμε δύο από τα σύνολα δεδομένων μας - τα «SwissProt» και «XMark» - και φίλτρα bloom μήκους 128 και 512 bits αντίστοιχα. Όπως περιμέναμε, το ποσοστό αυτό μειώνεται καθώς ο αριθμός των κάδων αυξάνεται. Από ένα σημείο και μετά μάλιστα, το σφάλμα εκτίμησης της συχνότητας οφείλεται σχεδόν αποκλειστικά στα φίλτρα bloom που αναφέρουν λανθασμένα ότι τα μονοπάτια ταιριάζουν σε αυτά. Παρατηρούμε μάλιστα πως αυτό αρχίζει να συμβαίνει από τη στιγμή που ο αριθμός των κάδων αρχίζει να υπερβαίνει τον αριθμό των διαφορετικών συχνοτήτων που υπάρχουν στο αρχικό σύνολο δεδομένων (βλ. Πίνακα 3-1). Στις γραφικές παραστάσεις που ακολουθούν ο άξονας *y* παρουσιάζεται σε λογαριθμική κλίμακα.



Σχήμα 3-11: SwissProt.



Σχήμα 3-12: XMark.

3.2.4 Συνολικό σφάλμα

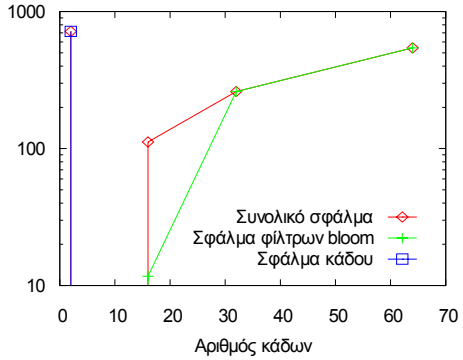
Αν υποθέσουμε ότι διαθέτουμε ορισμένο χώρο για το ιστόγραμμά μας, πώς μπορούμε να επιλέξουμε συγκεκριμένα χαρακτηριστικά για αυτό; Όπως δείξαμε στις προηγούμενες ενότητες, το σφάλμα του ιστογράμματος bloom μειώνεται αν:

- Αυξήσουμε το μήκος των φίλτρων bloom
- Αυξήσουμε τον αριθμό των κάδων

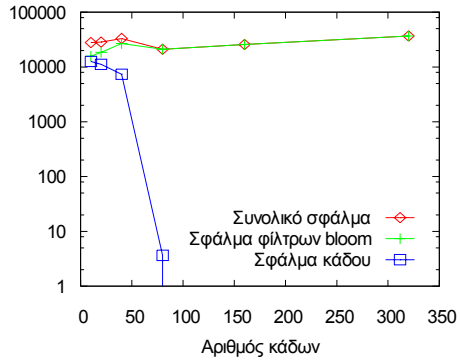
Όταν διαθέτουμε περιορισμένο χώρο όμως δεν μπορούμε να αυξήσουμε και τις δύο αυτές παραμέτρους ταυτόχρονα, επομένως πρέπει να αποφασίσουμε ποια θα επιλέξουμε να αυξήσουμε περισσότερο.

Στη σειρά πειραμάτων που ακολουθεί κατασκευάσαμε έναν αριθμό ιστογραμμάτων bloom για καθένα από τα σύνολα δεδομένων μας. Τα ιστογράμματα κάθε συνόλου δεδομένων έχουν όλα το ίδιο μέγεθος αλλά διαφέρουν στον αριθμό των κάδων και το μήκος των φίλτρων bloom. Πιο συγκεκριμένα, καθώς αυξάνεται ο αριθμός των κάδων μειώνεται το μήκος των φίλτρων bloom (άρα αυξάνεται και το σφάλμα τους) με τέτοιον τρόπο ώστε το συνολικό μέγεθος της κάθε εγγραφής των ιστογραμμάτων να παραμένει σταθερό.

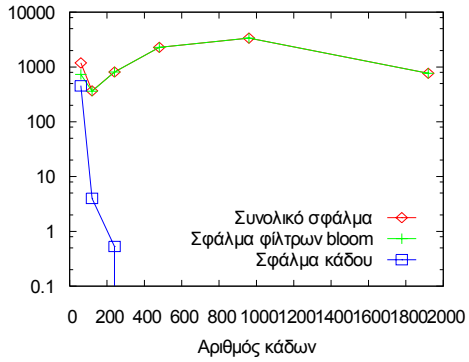
Στις γραφικές παραστάσεις των Σχημάτων 3-13 έως 3-15 φαίνεται το συνολικό σφάλμα που εμφανίζουν τα ιστογράμματα bloom καθώς και τα ποσοστά αυτού που οφείλονται αποκλειστικά στα φίλτρα bloom και στους κάδους του ιστογράμματος. Ο χώρος που αφιερώσαμε για τα φίλτρα bloom των ιστογραμμάτων είναι 64 byte για το σύνολο δεδομένων «SIGMOD record», 320 byte για το «SwissProt» και 7,5 KB για το «XMark».



Σχήμα 3-13: SIGMOD record.



Σχήμα 3-14: SwissProt.



Σχήμα 3-15: XMark.

Όπως παρατηρούμε, το σφάλμα που προέρχεται αποκλειστικά από το σωστό κάδο είναι γενικά μικρότερο από αυτό που προέρχεται από το λανθασμένο ταίριασμα της ερώτησης σε διάφορα φίλτρα bloom. Το σφάλμα αυτό μάλιστα μηδενίζεται καθώς ο αριθμός των κάδων γίνεται ίσος ή μεγαλύτερος από τον αριθμό των διαφορετικών συχνοτήτων που εμφανίζονται στο σύνολο δεδομένων. Καθώς όμως αφιερώνουμε όλο και περισσότερο χώρο για τους κάδους και μειώνουμε το μήκος των φίλτρων bloom η ακρίβειά τους μειώνεται και άρα το συνολικό σφάλμα γενικά αυξάνεται.

Το πια παράμετρο θα επιλέξουμε να αυξήσουμε περισσότερο δεν είναι πάντα εύκολο να προκαθοριστεί καθώς στην πράξη εξαρτάται κατά πολύ από το σύνολο δεδομένων που θέλουμε να εισάγουμε στο ιστόγραμμα bloom και ιδιαίτερα από το πλήθος των διακριτών μονοπατιών και τον αριθμό των διαφορετικών συχνοτήτων. Επειδή το ιστόγραμμα bloom ως δομή έχει γενικά μικρές απαιτήσεις σε χώρο (ενδεικτικά αναφέρουμε πως ενώ τα - σχετικά καλής απόδοσης - ιστογράμματα που κατασκευάσαμε για το σύνολο δεδομένων «XMark» έχουν μέγεθος μικρότερο των 10 KB, ο αντίστοιχος πίνακας με όλα τα μονοπάτια και τις συχνότητές τους έχει μέγεθος 98 KB), πιστεύουμε πως ένας αριθμός κάδων σχετικά κοντά στον αριθμό διαφορετικών συχνοτήτων του αρχικού συνόλου δεδομένων σε συνδυασμό με φίλτρα bloom αρκούντως μεγάλου μεγέθους (σε σχέση με το πλήθος των μονοπατιών) είναι συνήθως εφικτός και αποδοτικός.

Κεφάλαιο 4: Συγχώνευση ιστογραμμάτων bloom

Τα ιστογράμματα bloom μπορούν να χρησιμοποιηθούν και για τη δρομολόγηση ερωτήσεων σε συστήματα ομότιμων κόμβων (p2p). Συγκεκριμένα, θεωρούμε ένα σύστημα ομότιμων κόμβων στο οποίο οι κόμβοι αποθηκεύουν δεδομένα σε μορφή XML. Τα δεδομένα αυτά μπορεί να αντιστοιχούν για παράδειγμα σε διάφορες υπηρεσίες ή αρχεία που προσφέρει ο κάθε κόμβος. Σε ένα τέτοιο σύστημα κάθε κόμβος είναι απευθείας συνδεδεμένος με ένα πλήθος άλλων κόμβων. Όταν λοιπόν υποβάλλεται μία ερώτηση σε κάποιον κόμβο ο τρόπος με τον οποίο η ερώτηση αυτή θα δρομολογηθεί μέσα στο σύστημα ώστε να απαντηθεί σύντομα και σωστά δεν είναι προφανής.

Μία ιδέα είναι να χρησιμοποιηθεί η πληροφορία που βρίσκεται αποθηκευμένη στο ιστόγραμμα bloom του κάθε κόμβου. Ο κάθε κόμβος θα μπορούσε να διατηρεί κάποιες πληροφορίες για τα ιστογράμματα bloom των γειτόνων του ή να αναζητά απευθείας πληροφορίας σε αυτά μόλις χρειάζεται. Έτσι θα μπορούσε να δρομολογήσει την ερώτηση στους γείτονες εκείνους που παρουσιάζουν μεγαλύτερη συχνότητα αποτελεσμάτων για τη συγκεκριμένη ερώτηση.

Ενδιαφέρον παρουσιάζει ο τρόπος με τον οποίο ένας κόμβος θα αναζητήσει πληροφορία στα ιστογράμματα των γειτόνων του. Η αναζήτηση σε καθέναν από αυτούς μόλις ο κόμβος δέχεται μία ερώτηση δεν είναι ιδιαίτερα αποδοτική καθώς απαιτεί την συχνή επικοινωνία μεταξύ των κόμβων και τη συνεχή ανταλλαγή μηνυμάτων, κάτι που εισάγει μεγάλη καθυστέρηση στην επιστροφή αποτελέσματος ως απάντηση στην ερώτηση. Μία ιδέα είναι να διατηρεί ο κάθε κόμβος αντίγραφα των ιστογραμμάτων των γειτόνων του. Καθώς τα ιστογράμματα έχουν γενικά μικρό μέγεθος και σε ένα σύστημα ομότιμων κόμβων ο κάθε κόμβος έχει ένα γενικά μικρό αριθμό γειτόνων αυτή η λύση θα μπορούσε να είναι αρκετά

αποδοτική. Μία άλλη ιδέα είναι ο κάθε κόμβος να συγχωνεύει όλα τα ιστογράμματα των γειτόνων του σε ένα και να αντλεί πληροφορία από αυτό. Το πώς θα γίνει αυτή η συγχώνευση είναι κάτι που θα μας απασχολήσει στη συνέχεια.

4.1 Ορισμός του προβλήματος

Πιο συγκεκριμένα, ας υποθέσουμε ότι έχουμε διαθέσιμα δύο ιστογράμματα bloom $BH_1(paths_1, v_1)$ και $BH_2(paths_2, v_2)$ – με το συμβολισμό που εισάγαμε στην §2.2.1 – τα οποία αντιπροσωπεύουν τα σύνολα δεδομένων D_1 και D_2 αντίστοιχα και έστω ότι τα ιστογράμματα αυτά καταλαμβάνουν χώρο ίσο με s_1 και s_2 αντίστοιχα έχοντας και τα δύο φίλτρα bloom ίσου μήκους m.

Στόχος μας είναι η κατασκευή ενός νέου ιστογράμματος $BH(paths, v)$ το οποίο θα αντιπροσωπεύει το σύνολο δεδομένων $D_1 \cup D_2$ καταλαμβάνοντας χώρο όχι μεγαλύτερο από $s_{max} = \max\{s_1, s_2\}$.

4.2 Αλγόριθμοι συγχώνευσης

Αυτό που μας ενδιαφέρει κυρίως στα ιστογράμματα bloom είναι το σφάλμα της εκτίμησης της συχνότητας των μονοπατιών το οποίο θέλουμε να διατηρήσουμε σε όσο το δυνατόν χαμηλότερα επίπεδα. Στη συνέχεια θα προτείνουμε μερικούς τρόπους συγχώνευσης δύο ιστογραμμάτων bloom έχοντας υπόψη την παρατήρηση αυτή και υποθέτοντας ότι ο χώρος που έχουμε διαθέσιμος για το συγχωνευμένο ιστόγραμμα είναι ίσος με το χώρο που καταλαμβάνει ένα από τα δύο αρχικά ιστογράμματα, έστω το μεγαλύτερο από αυτά.

4.2.1 Συγχώνευση με βάση τις τιμές των κάδων

Η κεντρική ιδέα πίσω από τη συγχώνευση δύο ιστογραμμάτων bloom με βάση τις τιμές των κάδων είναι η εξής: προσπαθούμε να συγχωνεύσουμε κάδους από τα δύο ιστογράμματα οι οποίοι έχουν όσο το δυνατό πιο κοντινές τιμές. Ένας τρόπος για να γίνει αυτό είναι να εξετάσουμε για κάθε κάδο του δεύτερου ιστογράμματος ποιος κάδος του πρώτου

ιστογράμματος έχει την πιο κοντινή σε αυτόν τιμή και στη συνέχεια να τον συγχωνεύσουμε με αυτόν.

Όταν συγχωνεύουμε δύο κάδους ο νέος κάδος έχει ως φίλτρο bloom την συγχώνευση των φίλτρων bloom BF_1, BF_2 των δύο αρχικών κάδων (δηλαδή το $BF = (BF_1)$ BOR (BF_2), όπου BOR η πράξη του δυαδικού OR) και ως τιμή τον μέσο όρο των τιμών των δύο αρχικών κάδων.

Ο βασικός αλγόριθμός για τη συγχώνευση δύο ιστογραμμάτων bloom BH_1 και BH_2 με βάση τις τιμές των κάδων παρουσιάζεται στη συνέχεια (b_1 και b_2 είναι το πλήθος των κάδων των δύο ιστογραμμάτων αντίστοιχα).

Αλγόριθμος 4-1: MergeBloomHistograms_buckets(BH1, BH2)

```
1: int min, target
2:  for i = 1 to b2 do
3:   min = +∞;, target = 0;
4:   for j = 1 to b1 do
5:    if BH2.value[i] – BH1.value[j] < min then
6:     min = BH2.value[i] – BH1.value[j];
7:     target = j;
8:    end if
9:   end for
10:   MergeBuckets(i, target);
11:  end for
```

Αλγόριθμος 4-1: Συγχώνευση ιστογραμμάτων bloom με βάση τις τιμές των κάδων.

Στον Αλγόριθμο 4-1 η συνάρτηση `MergeBuckets` αναλαμβάνει να συγχωνεύσει τους δύο κάδους *i* και *target* σε έναν (με τις ιδιότητες που περιγράψαμε πριν) και να τον προσθέσει στο νέο ιστόγραμμα. Ο αλγόριθμος που περιγράψαμε δημιουργεί ένα νέο ιστόγραμμα bloom το οποίο καταλαμβάνει τον ίδιο χώρο με το ιστόγραμμα BH_1, δηλαδή χώρο $s = s_1 \leq s_{max}$.

4.2.2 Συγχώνευση με βάση τα φίλτρα bloom

Μία άλλη προσέγγιση στη συγχώνευση δύο ιστογραμμάτων bloom είναι να στηριχθούμε στην ομοιότητα των φίλτρων bloom των δύο ιστογραμμάτων συγχωνεύοντας κάδους με παρόμοια φίλτρα bloom.

Αρχικά θα πρέπει να ορίσουμε έναν τρόπο μέτρησης της ομοιότητας δύο φίλτρων bloom και αυτός είναι ο εξής: Έστω τα φίλτρα bloom B_1 και B_2 με μήκος m. Με $B[i]$ συμβολίζουμε το i-οστό bit του φίλτρου B. Ορίζουμε την απόσταση $dist(B_1, B_2)$ των φίλτρων B_1 και B_2 να είναι ίση με τον αριθμό των bit στα οποία αυτά διαφέρουν, δηλαδή:

$$dist(B_1, B_2) = |B_1[1] - B_2[1]| + |B_1[2] - B_2[2]| + ... + |B_1[m] - B_2[m]| \quad (4.1)$$

Στη συνέχεια ορίζουμε την ομοιότητα των δύο φίλτρων B_1 και B_2 ως την ποσότητα

$$similarity(B_1, B_2) = m - dist(B_1, B_2) \quad (4.2)$$

Όσο πιο μεγάλη είναι αυτή η ποσότητα, τόσο πιο παρόμοια είναι τα δύο φίλτρα.

Στον αλγόριθμο συγχώνευσης με βάση τα φίλτρα bloom προσπαθούμε να εντοπίσουμε και να συγχωνεύσουμε κάδους με όσο το δυνατόν πιο παρόμοια φίλτρα bloom, στηριζόμενοι στο γεγονός ότι αυτό θα μειώσει τον αριθμό των φίλτρων του συγχωνευμένου ιστογράμματος που θα αναφέρουν λανθασμένα ότι η ερώτηση-μονοπάτι ταιριάζει σε αυτούς.

Ο αλγόριθμος αυτός είναι ο εξής:

Αλγόριθμος 4-2: MergeBloomHistograms_filters(BH1, BH2)

```
1: int max, target
2:  for i = 1 to b2 do
3:   max = -∞;, target = 0;
4:   for j = 1 to b1 do
5:    if similarity(BH2.BF[i], BH1.BF[j]) > max then
6:     max = similarity(BH2.BF[i], BH1.BF[j]);
7:     target = j;
8:    end if
9:   end for
```

```
10:   MergeBuckets(i, target);
11:  end for
```

Αλγόριθμος 4-2: Συγχώνευση ιστογραμμάτων bloom με βάση τα φίλτρα bloom.

4.2.3 Βελτιώσεις στους αλγορίθμους συγχώνευσης

Οι αλγόριθμοι που παρουσιάστηκαν στις ενότητες §4.2.1 και §4.2.2 μπορούν να δώσουν ακόμα καλύτερα αποτελέσματα αν λάβουμε υπόψη κάποια από τα στοιχεία που αναφέρονται στην ενότητα αυτή.

4.2.3.1 Καλύτερη προσέγγιση τιμών κάδων

Ένα μειονέκτημα των αλγορίθμων που παρουσιάστηκαν είναι ότι η τιμή ενός κάδου του συγχωνευμένου ιστογράμματος είναι ίση με το μέσο όρο των τιμών των δύο αρχικών κάδων. Κάτι τέτοιο μπορεί να οδηγήσει σε αύξηση του σφάλματος στην περίπτωση που στον έναν από τους δύο αρχικούς κάδους έχουν εισαχθεί πολύ περισσότερα μονοπάτια από ότι στον άλλο. Μία καλύτερη προσέγγιση κατά τον υπολογισμό της τιμής του νέου κάδου θα ήταν να λαμβάνεται υπόψη το πόσα μονοπάτια έχουν εισαχθεί σε καθέναν από τους αρχικούς κάδους. Για να γίνει κάτι τέτοιο θα πρέπει να τροποποιήσουμε ελαφρά τα ιστογράμματά μας έτσι ώστε για κάθε κάδο να γνωρίζουμε και τον αριθμό των μονοπατιών που έχουν εισαχθεί σε αυτόν. Αν αποφασίσουμε να συγχωνεύσουμε τους δύο κάδους με τιμές v_1, v_2 και αριθμό μονοπατιών k_1, k_2 αντίστοιχα τότε για τον νέο κάδο θα ισχύει:

$$v = \frac{k_1 \cdot v_1 + k_2 \cdot v_2}{k_1 + k_2} \text{ και } k = k_1 + k_2$$

(όπου v η τιμή και k ο αριθμός μονοπατιών του νέου κάδου.)

4.2.3.2 Άθροισμα τιμών κάδων

Υπάρχει περίπτωση να συγχωνευτούν δύο οι περισσότεροι κάδοι που διαθέτουν ίδια φίλτρα bloom. Για παράδειγμα, έστω ότι τα δύο αρχικά ιστογράμματα bloom είναι τα παρακάτω και εφαρμόζουμε συγχώνευση με βάση τις τιμές των κάδων:

	Φίλτρο bloom	**Συχνότητα**
A_1	BF(/a)	1000
A_2	BF(/b)	4000
A_3	BF(/c)	7000
A_4	BF(/d)	9000

	Φίλτρο bloom	**Συχνότητα**
B_1	BF(/a)	2000
B_2	BF(/y)	5000
B_3	BF(/w)	8000
B_4	BF(/z)	11000

Στην περίπτωση αυτή ο κάδος B_2 θα συγχωνευτεί με τον κάδο A_2 και ο νέος κάδος θα έχει τιμή 4500, τιμή που είναι μία καλή προσέγγιση των δύο αρχικών τιμών εφόσον ένα μονοπάτι που ταιριάζει στο νέο κάδο μπορεί να είναι είτε */b* (με τιμή 4000) είτε */y* (με τιμή 5000). Με παρόμοιο τρόπο ο κάδος B_1 θα συγχωνευτεί με τον κάδο A_1 και θα δημιουργήσουν ένα κάδο με τιμή 1500. Όμως, ένα μονοπάτι που ταιριάζει στο νέο αυτό κάδο θα είναι σίγουρα το */a*, οπότε ο κάδος αυτός θα δίνει πολύ καλύτερες εκτιμήσεις αν έχει την τιμή 3000, δηλαδή το άθροισμα των τιμών των δύο αρχικών κάδων και όχι το μέσο όρο τους.

4.2.3.3 Επαναληπτική κατασκευή ιστογράμματος

Μία ακόμα ιδέα για καλύτερα αποτελέσματα είναι να μη συγχωνεύονται ένας-ένας με τη σειρά οι κάδοι του δεύτερου ιστογράμματος με κάδους του πρώτου αλλά η συγχώνευση να γίνεται ως εξής: Ελέγχουμε έναν-έναν τους κάδους του δεύτερου ιστογράμματος και υπολογίζουμε τη μικρότερη διαφορά τιμής που έχουν από τους κάδους του πρώτου ιστογράμματος, έστω x_j με $1 \leq j \leq b_2$, όπου b_2 ο αριθμός των κάδων του δεύτερου ιστογράμματος. Στη συνέχεια συγχωνεύουμε τον κάδο i του δεύτερου ιστογράμματος για τον οποίο $x_i = \min_{j=1}^{b_2} x_j$ με τον αντίστοιχο κάδο του πρώτου ιστογράμματος και

επαναλαμβάνουμε τη διαδικασία μέχρι να συγχωνευτούν όλοι οι κάδοι του δεύτερου ιστογράμματος.

4.3 Πειραματική μελέτη των αλγορίθμων συγχώνευσης

Προκειμένου να μελετήσουμε την απόδοση των συγχωνευμένων ιστογραμμάτων bloom υλοποιήσαμε τους αλγορίθμους που παρουσιάστηκαν στις ενότητες §4.2.1 και §4.2.2 με κάποιες από τις βελτιώσεις που αναφέραμε στην ενότητα §4.2.3. Στη συνέχεια κατασκευάσαμε συγχωνευμένα ιστογράμματα για κάποια από τα σύνολα δεδομένων που παρουσιάστηκαν στον Πίνακα 3-1 και συγκρίναμε την απόδοσή τους σε σχέση με μη-συγχωνευμένα ιστογράμματα που αναφέρονται στα ίδια σύνολα δεδομένων και έχουν κατασκευαστεί έχοντας διαθέσιμο όλο το αρχικό σύνολο δεδομένων και όχι μόνο την περίληψή του (δηλαδή το αντίστοιχο ιστόγραμμα bloom). Θεωρούμε αυτά τα μη-συγχωνευμένα ιστογράμματα βέλτιστα.

Πιο συγκεκριμένα, για κάθε σύνολο δεδομένων ακολουθήσαμε την εξής διαδικασία: Αρχικά, χρησιμοποιήσαμε ολόκληρο το σύνολο δεδομένων και με βάση τον Αλγόριθμο 2-2 που οδηγεί στη δημιουργία βέλτιστων ιστογραμμάτων κατασκευάσαμε ένα ιστόγραμμα bloom *BH*. Στη συνέχεια χωρίσαμε το σύνολο δεδομένων σε δύο τμήματα και με βάση το κάθε τμήμα και τον Αλγόριθμο 2-2 κατασκευάσαμε δύο ιστογράμματα BH_1 και BH_2. Συγχωνεύσαμε τα ιστογράμματα BH_1 και BH_2 με βάση τους δύο αλγορίθμους συγχώνευσης παράγοντας έτσι τα BH_b (με βάση τον Αλγόριθμο 4-1) και BH_f (με βάση τον Αλγόριθμο 4-2). Με αυτόν τον τρόπο τα *BH*, BH_b και BH_f περιέχουν όλα πληροφορία για τα ίδια ακριβώς μονοπάτια.

Από τις βελτιώσεις που αναφέρθηκαν στην §4.2.3 υλοποιήσαμε αυτή της καλύτερης προσέγγισης των τιμών των κάδων. Η βελτίωση του αθροίσματος των τιμών των κάδων δε χρησιμοποιήθηκε εδώ καθώς λόγω του τρόπου κατασκευής των BH_1 και BH_2 ένα μονοπάτι του αρχικού συνόλου δεδομένων μπορεί να έχει τοποθετηθεί μόνο σε ένα από αυτά. Ακόμα αν υπάρχουν περισσότεροι του ενός κάδοι που απέχουν το ίδιο από τον κάδο που θέλουμε να συγχωνεύσουμε, επιλέγουμε έναν από αυτούς με τυχαίο τρόπο.

Στη συνέχεια υποβάλλαμε το ίδιο σύνολο ερωτήσεων και στα τρία αυτά ιστογράμματα και σημειώσαμε το σφάλμα που παρουσίασε το καθένα από αυτά. Το σύνολο

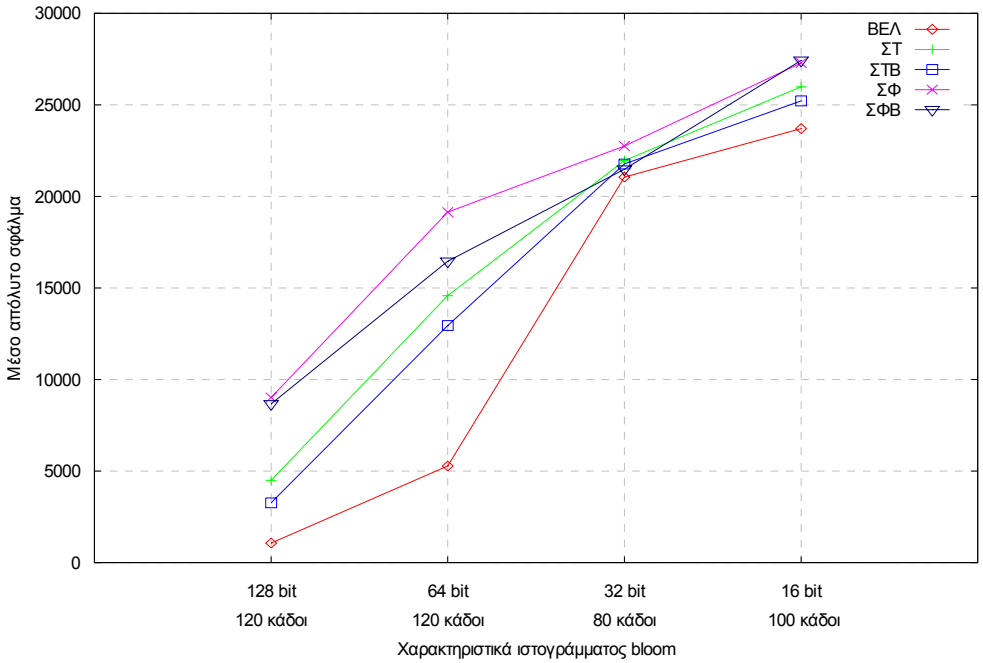
των ερωτήσεων αυτών αποτελείται από θετικές ερωτήσεις που αντιστοιχούν στο 50% των μονοπατιών που συναντούνται στο σύνολο δεδομένων. Ως σφάλμα θεωρήσαμε την τιμή του απόλυτου σφάλματος, όπως αυτό ορίστηκε στην §3.2.

Οι συντομογραφίες που χρησιμοποιούνται στην ενότητα αυτή για να περιγράψουν τους διάφορους τύπους ιστογραμμάτων bloom φαίνονται στον Πίνακα 4-1:

Πίνακας 4-1: Τύποι ιστογραμμάτων bloom.

ΒΕΛ	Βέλτιστο (μη-συγχωνευμένο) ιστόγραμμα bloom
ΣΤ	Συγχώνευση με βάση τις τιμές των κάδων
ΣΤΒ	Συγχώνευση με βάση τις τιμές των κάδων με βελτιώσεις
ΣΦ	Συγχώνευση με βάση τα φίλτρα bloom
ΣΦΒ	Συγχώνευση με βάση τα φίλτρα bloom με βελτιώσεις

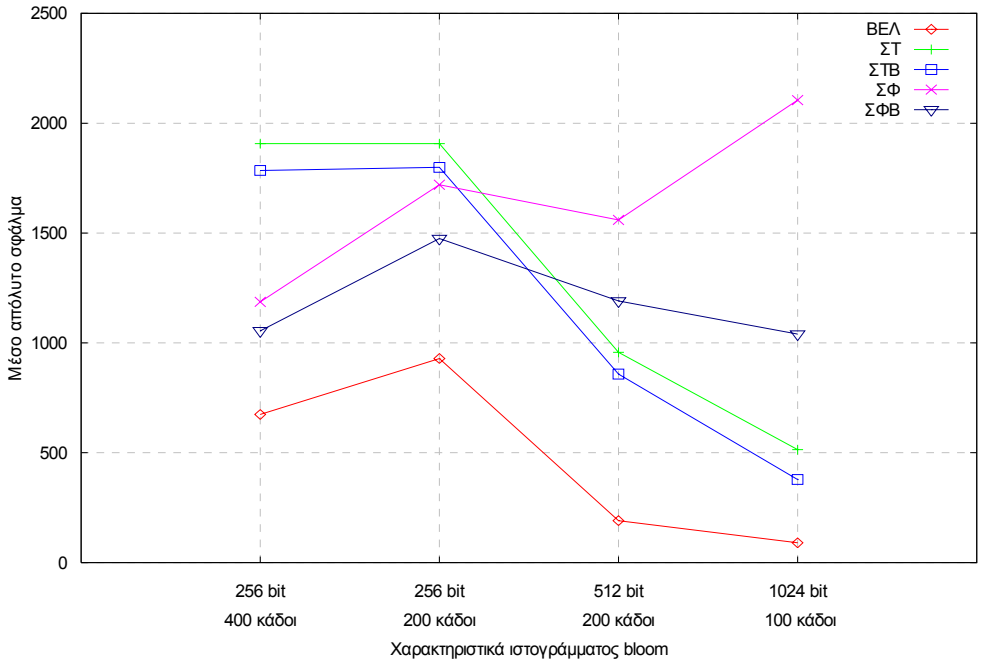
Στη γραφική παράσταση του Σχήματος 4-1 παρουσιάζεται το μέσο απόλυτο σφάλμα των διάφορων ιστογραμμάτων bloom για το σύνολο δεδομένων «SwissProt» και για διάφορους συνδυασμούς μήκους φίλτρων bloom και αριθμού κάδων. Την καλύτερη απόδοση παρουσιάζει, όπως και περιμέναμε, το βέλτιστο ιστόγραμμα bloom που έχει δημιουργηθεί απευθείας από ολόκληρο το σύνολο δεδομένων. Φαίνεται ακόμα πως οι βελτιώσεις που υλοποιήσαμε βελτιώνουν όντως την απόδοση τόσο του αλγορίθμου συγχώνευσης με βάση τις τιμές των κάδων όσο και του αλγορίθμου συγχώνευσης με βάση τα φίλτρα bloom. Τέλος, παρατηρούμε πως για αυτό σύνολο δεδομένων ο αλγόριθμος συγχώνευσης με βάση τις τιμές των κάδων παρουσιάζει σταθερά καλύτερη απόδοση από τον αλγόριθμο συγχώνευσης με βάση τα φίλτρα bloom και μάλιστα η απόδοση αυτή δε διαφέρει γενικά πάρα πολύ από την απόδοση του βέλτιστου ιστογράμματος. Αυτό συμβαίνει επειδή η συγχώνευση με βάση τις τιμές των κάδων οδηγεί σε κάδους που έχουν συχνότητες παραπλήσιες με τις συχνότητες των αρχικών κάδων από τους οποίους προήλθαν.



Σχήμα 4-1: Απόδοση συγχωνευμένων ιστογραμμάτων για το σύνολο δεδομένων «SwissProt».

Στη γραφική παράσταση του Σχήματος 4-2 παρουσιάζονται τα αντίστοιχα αποτελέσματα για το σύνολο δεδομένων «XMark». Και εδώ την καλύτερη απόδοση σημειώνει το βέλτιστο ιστόγραμμα bloom. Ακόμα, παρατηρούμε και εδώ πως οι βελτιώσεις που υλοποιήσαμε βελτιώνουν όντως την απόδοση των δύο αλγορίθμων. Σε αυτό το σύνολο δεδομένων όμως δεν υπάρχει ένας από τους δύο αλγορίθμους ο οποίος να υπερτερεί σταθερά από τον άλλο. Παρατηρούμε πως όταν γενικά έχουμε μικρό μήκος φίλτρων bloom ο αλγόριθμος συγχώνευσης με βάση τα φίλτρα bloom έχει καλύτερη απόδοση, κάτι που δε συμβαίνει καθώς αυξάνουμε το μήκος των φίλτρων bloom, ακόμα και όταν παράλληλα μειώνουμε τον αριθμό των κάδων του ιστογράμματος.

Το σύνολο δεδομένων «Xmark» αποτελείται από μεγάλο πλήθος διαφορετικών μονοπατιών (βλ. Πίνακα 3-1). Επομένως, η συγχώνευση με βάση τις τιμές των κάδων σε συνδυασμό με μικρό μήκος φίλτρων bloom οδηγεί σε «υπερφορτωμένα» φίλτρα bloom, δηλαδή σε φίλτρα bloom που περιλαμβάνουν πολλά bit με την τιμή 1. Αυτό αυξάνει το μέσο σφάλμα καθώς κάδοι με τέτοια φίλτρα αναφέρουν πολύ συχνά λανθασμένα ότι διάφορα μονοπάτια ανήκουν σε αυτούς. Η συγχώνευση με βάση τα φίλτρα bloom αντιμετωπίζει αυτό το πρόβλημα και συνεπώς εμφανίζει καλύτερη απόδοση σε αυτές τις περιπτώσεις.



Σχήμα 4-2: Απόδοση συγχωνευμένων ιστογραμμάτων για το σύνολο δεδομένων «XMark».

4.4 Συμπεράσματα

Όπως είναι αναμενόμενο τα συγχωνευμένα ιστογράμματα bloom έχουν χαμηλότερη απόδοση από τα ιστογράμματα που προκύπτουν απευθείας από το αρχικό σύνολο δεδομένων. Παρόλα αυτά όμως η απόδοσή τους είναι συγκρίσιμη με αυτή των βέλτιστων ιστογραμμάτων - κάτι που καθιστά τη χρήση τους ιδιαίτερα ελκυστική, ιδιαίτερα σε περιπτώσεις που το αρχικό σύνολο δεδομένων δεν είναι πάντα διαθέσιμο. Μία τέτοια περίπτωση είναι η δρομολόγηση ερωτήσεων σε συστήματα ομότιμων κόμβων την οποία θα εξετάσουμε στο Κεφάλαιο 5.

Ακόμα, όπως προκύπτει από την πειραματική μελέτη, οι βελτιώσεις που προτάθηκαν στην §4.2.3 οι οποίες υλοποιήσαμε βελτιώνουν όντως την απόδοση και των δύο αλγορίθμων, περίπου κατά 10%. Κανένας αλγόριθμος ωστόσο δεν υπερτερεί ξεκάθαρα έναντι του άλλου σε όλες τις περιπτώσεις. Σε γενικές γραμμές μπορούμε να πούμε πως η συγχώνευση με βάση τις τιμές των κάδων είναι αρκετά αποτελεσματική, αρκεί το σφάλμα των φίλτρων bloom του τελικού συγχωνευμένου ιστογράμματος να μην είναι μεγάλο. Σε αυτές τις περιπτώσεις η συγχώνευση με βάση τα φίλτρα bloom φαίνεται πως είναι πιο αποδοτική.

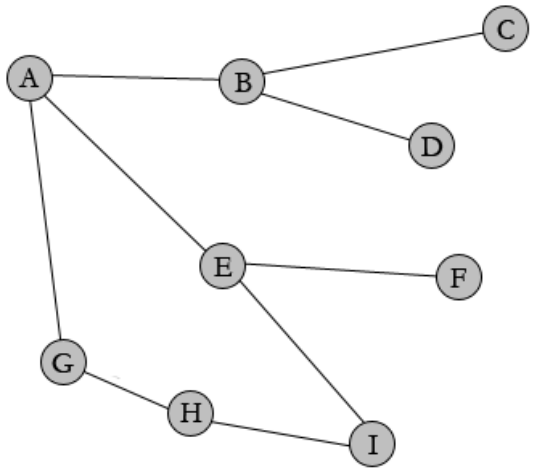
Κεφάλαιο 5: Χρήση ιστογραμμάτων Bloom στη δρομολόγηση ερωτήσεων

Στο Κεφάλαιο 4 είδαμε πως μπορούμε να συγχωνεύσουμε ιστογράμματα bloom και να δημιουργήσουμε νέα τα οποία μπορούμε στη συνέχεια να χρησιμοποιήσουμε αντί των αρχικών ιστογραμμάτων. Ένα από τα πλεονεκτήματα της χρήσης συγχωνευμένων ιστογραμμάτων bloom είναι ότι για την κατασκευή τους απαιτούνται μόνο τα αρχικά ιστογράμματα bloom και όχι το αρχικό σύνολο δεδομένων. Στο κεφάλαιο αυτό θα δούμε πως μπορούμε να χρησιμοποιήσουμε τα συγχωνευμένα ιστογράμματα bloom ως ευρετήρια για στη δρομολόγηση ερωτήσεων σε συστήματα ομότιμων κόμβων.

5.1 Συστήματα ομότιμων κόμβων

Τα *συστήματα ομότιμων κόμβων* (*peer-to-peer systems* ή p2p) αποτελούνται από έναν μεγάλο αριθμό υπολογιστών (ή άλλων παρεμφερών συσκευών) που ονομάζουμε *κόμβους*. Όλοι οι κόμβοι έχουν γενικά παρόμοιες δυνατότητες όσον αφορά χαρακτηριστικά όπως η υπολογιστική ισχύς κ.τ.λ. Κάθε κόμβος είναι συνδεδεμένος απευθείας με ένα σχετικά μικρό αριθμό άλλων κόμβων (τους οποίους ονομάζουμε *γείτονες* του κόμβου) οι οποίοι με τη σειρά τους είναι συνδεδεμένοι απευθείας με άλλους κόμβους κ.ο.κ. Με αυτόν τον τρόπο κάθε κόμβος μπορεί έμμεσα μέσω των γειτόνων του να συνδεθεί με οποιονδήποτε άλλο κόμβο του δικτύου. Σε ένα σύστημα ομότιμων κόμβων διάφοροι κόμβοι μπορούν να συνδέονται στο σύστημα και να αποσυνδέονται από αυτό χωρίς το σύστημα να παύει να λειτουργεί.

Ένα παράδειγμα συστήματος ομότιμων κόμβων φαίνεται στο Σχήμα 5-1. Σε αυτήν την περίπτωση οι γείτονες του κόμβου A είναι οι κόμβοι B, E και G. Μέσω αυτών ο A μπορεί να συνδεθεί και με τους υπόλοιπους κόμβους. Για παράδειγμα, με τον κόμβο C μπορεί μα συνδεθεί μέσω της διαδρομής $A \rightarrow B \rightarrow C$. Επίσης σε ένα σύστημα ομότιμων κόμβων είναι δυνατό να σχηματίζονται και κύκλοι μεταξύ των κόμβων, όπως στην περίπτωση της διαδρομής $A \rightarrow E \rightarrow I \rightarrow H \rightarrow G \rightarrow A$ στο Σχήμα 5-1.



Σχήμα 5-1: Παράδειγμα συστήματος p2p.

5.2 Αναζήτηση πληροφορίας σε συστήματα ομότιμων κόμβων

Ένας από τους κύριους λόγους χρήσης των συστημάτων ομότιμων κόμβων είναι η δυνατότητα αναζήτησης και ανταλλαγής πληροφορίας με όλους τους κόμβους του συστήματος. Καθώς ένα σύστημα ομότιμων κόμβων δεν έχει σταθερή δομή αφού οποιαδήποτε στιγμή κόμβοι μπορούν να προστεθούν ή να αποχωρήσουν η αναζήτηση πληροφορίας γίνεται συνήθως ως εξής:

Κάθε κόμβος διαθέτει ένα τοπικό ευρετήριο που περιγράφει τα δεδομένα που διαθέτει ο ίδιος ο κόμβος. Το ευρετήριο αυτό μπορεί να έχει οποιαδήποτε δομή που μπορεί να συνοψίσει αποτελεσματικά τα δεδομένα, όπως για παράδειγμα ένα ιστόγραμμα bloom. Κάθε φορά που ένας κόμβος δέχεται μία ερώτηση αρχικά ελέγχει το τοπικό του ευρετήριο και επιστρέφει μία εκτίμηση για τον αριθμό των δεδομένων του που απαντούν την ερώτηση. Στη συνέχεια επιλέγει κάποιους από τους γειτονικούς του κόμβους και τους προωθεί την ερώτηση. Οι κόμβοι αυτοί ακολουθούν την ίδια διαδικασία. Συνήθως η

διαδικασία αυτή σταματά όταν έχει επιστραφεί ένας επαρκής αριθμός αποτελεσμάτων αλλά τα διάφορα συστήματα ομότιμων κόμβων μπορεί να διευθετούν με διαφορετικό τρόπο αυτό το ζήτημα.

Σε ποιους από τους γείτονές του θα προωθήσει όμως ο κόμβος την ερώτηση; Και σε αυτό το ζήτημα υπάρχουν διαφορές μεταξύ των διάφορων συστημάτων ομότιμων κόμβων. Σε συστήματα όπως το Gnutella [10] οι κόμβοι προωθούν την ερώτηση σε όλους τους γείτονές τους. Η μέθοδος αυτή είναι απλή αλλά δημιουργεί πολλά μηνύματα μεταξύ των κόμβων αυξάνοντας έτσι κατά πολύ το κόστος απάντησης μίας ερώτησης. Μία άλλη προσέγγιση είναι αυτή που χρησιμοποιείται από το σύστημα Napster [11]. Στο σύστημα αυτό χρησιμοποιούνται κεντρικά ευρετήρια. Υπάρχουν κάποιοι κεντρικοί κόμβοι οι οποίοι κρατούν πληροφορία για όλα τα δεδομένα που είναι αποθηκευμένα σε οποιονδήποτε κόμβο του δικτύου. Όταν ένας κόμβος δέχεται μία ερώτηση απευθύνεται σε έναν από τους κεντρικούς κόμβους ο οποίος του απαντά με μία λίστα από κόμβους που διαθέτουν αποτελέσματα σχετικά με την ερώτηση. Στη συνέχεια ο κόμβος επικοινωνεί με αυτούς και ανταλλάσσει δεδομένα. Η μέθοδος αυτή είναι επίσης απλή αλλά απαιτεί κόμβους με αυξημένες δυνατότητες (κεντρικοί κόμβοι) και είναι ευάλωτη καθώς βλάβες στους κεντρικούς κόμβους επηρεάζουν τη λειτουργία ολόκληρου του συστήματος. Το Freenet [8] είναι ένα άλλο σύστημα στο οποίο κάθε κόμβος διαθέτει ευρετήρια με λίστες από κόμβους που διαθέτουν απαντήσεις για ερωτήσεις που ο κόμβος δέχθηκε πρόσφατα. Με αυτόν τον τρόπο αν ο κόμβος δεχθεί ξανά την ίδια ερώτηση μπορεί να ξαναβρεί απαντήσεις με χαμηλό κόστος. Παρόλα αυτά η μέθοδος αυτή δε μειώνει το κόστος απάντησης όλων των ερωτήσεων γενικά.

Μία άλλη μέθοδος είναι αυτή των ευρετηρίων δρομολόγησης (*routing indices*) [3]. Στα συστήματα που ακολουθούν αυτή τη μέθοδο οι κόμβοι εκτός από τα τοπικά ευρετήρια χρησιμοποιούν και άλλα ευρετήρια μέσω των οποίων μπορούν να έχουν μία περίληψη των δεδομένων που είναι προσβάσιμα μέσω των γειτόνων τους. Χρησιμοποιώντας αυτά τα ευρετήρια ο κάθε κόμβος αποφασίζει σε ποιους γείτονες θα προωθήσει κάθε ερώτηση. Τα συγχωνευμένα ιστογράμματα bloom που περιγράψαμε στο Κεφάλαιο 4 μπορούν να χρησιμοποιηθούν ως ευρετήρια δρομολόγησης και αυτό το ζήτημα θα μας απασχολήσει στη συνέχεια του Κεφαλαίου 5.

Τέλος θα πρέπει να αναφέρουμε και μεθόδους αναζήτησης όπως οι CHORD [6] και CAN[5]. Οι μέθοδοι αυτοί δημιουργούν ένα ευρετήριο για όλα τα δεδομένα που βρίσκονται αποθηκευμένα στο σύστημα. Το ευρετήριο αυτό είναι κατανεμημένο καθώς τμήματά του διαμοιράζονται μεταξύ όλων των κόμβων. Πρόκειται για αποδοτικές μεθόδους οι οποίες όμως είναι πιο πολύπλοκες στην υλοποίησή τους και λειτουργούν καλύτερα όταν η δομή του συστήματος δε μεταβάλλεται συνεχώς (καθώς απαιτείται κάποιος χρόνος για την προσαρμογή του κατανεμημένου ευρετηρίου στη νέα τοπολογία του συστήματος).

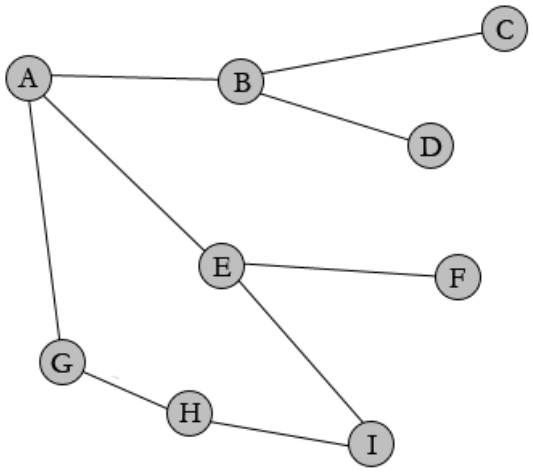
5.3 Ιστογράμματα Bloom ως ευρετήρια δρομολόγησης

Όπως είδαμε στο Κεφάλαιο 4 ένα συγχωνευμένο ιστόγραμμα bloom μπορεί να συνοψίσει την πληροφορία που ήταν αποθηκευμένη στα ιστογράμματα bloom από τα οποία προήλθε. Επίσης, για την κατασκευή του δεν είναι απαραίτητη η συμβολή του συνόλου δεδομένων από το οποίο προήλθαν τα αρχικά ιστογράμματα. Πρακτικά αυτό σημαίνει ότι συγχωνεύσεις ιστογραμμάτων bloom μπορούν να πραγματοποιηθούν σε οποιονδήποτε κόμβο ενός συστήματος ομότιμων κόμβων και όχι μόνο στον κόμβο που είναι αποθηκευμένο και το αρχικό σύνολο δεδομένων. Επίσης η κατασκευή ενός συγχωνευμένου ιστογράμματος είναι πιο απλή διαδικασία από την κατασκευή ενός βέλτιστου ιστογράμματος μέσω του Αλγορίθμου 2-2.

Τα ιστογράμματα bloom μπορούν να χρησιμοποιηθούν στη δρομολόγηση ερωτήσεων ως εξής: Κάθε κόμβος του συστήματος διατηρεί δύο ιστογράμματα. Το ένα ονομάζεται *τοπικό ιστόγραμμα bloom* και περιέχει πληροφορία για τα δεδομένα που είναι αποθηκευμένα τοπικά στον συγκεκριμένο κόμβο. Το άλλο ονομάζεται *συγχωνευμένο ιστόγραμμα bloom* και περιέχει πληροφορία για τα δεδομένα που είναι προσβάσιμα μέσω των γειτόνων του κόμβου αυτού. Το συγχωνευμένο ιστόγραμμα bloom προκύπτει από την συγχώνευση όλων των ιστογραμμάτων τα οποία οι άμεσοι γείτονες του κόμβου στέλνουν σε αυτόν. Μία προσθήκη που μπορεί να γίνει στα συγχωνευμένα ιστογράμματα bloom είναι να κρατάμε πληροφορία για το ποιοι γείτονες του κόμβου συμμετέχουν σε κάθε κάδο του τελικού συγχωνευμένου ιστογράμματος. Κάτι τέτοιο μειώνει το κόστος απάντησης μίας ερώτησης, όπως θα δούμε στη συνέχεια.

5.3.1 Χρήση ευρετηρίων δρομολόγησης

Στην ενότητα αυτή θα περιγράψουμε τη χρήση ευρετηρίων δρομολόγησης βασισμένων στα ιστογράμματα bloom. Ας υποθέσουμε πως διαθέτουμε το σύστημα ομότιμων κόμβων του Σχήματος 5-2 και πως το τοπικό και το συγχωνευμένο ιστόγραμμα του κόμβου *A* είναι αυτά που φαίνονται στο σχήμα.



Τοπικό ιστόγραμμα bloom [A]	
Φίλτρο bloom	**Συχνότητα**
BF(/a, /a/b)	10
BF(/a/f/c, /a/e)	100
BF(/c/d)	1000
BF(/a/i, /a/o)	1500

Συγχωνευμένο ιστόγραμμα bloom [A]		
Φίλτρο bloom	**Συχνότητα**	**Γείτονες**
BF(/a, /a/b)	50	B, E
BF(/a/f/c, /a/e)	400	B, G
BF(/a/z, /a/s)	900	B, E, G
BF(/a/i, /a/o)	3000	B

Σχήμα 5-2: Ευρετήρια δρομολόγησης.

Όταν ο κόμβος *A* δεχθεί μία ερώτηση της μορφής μονοπατιού πρώτα ελέγχει το τοπικό ιστόγραμμα bloom και επιστρέφει τη συχνότητα των αποτελεσμάτων που εκτιμά πως διαθέτει. Στη συνέχεια ελέγχει το συγχωνευμένο ιστόγραμμα bloom και προωθεί την ερώτηση στους γείτονες που διαθέτουν σχετικά αποτελέσματα. Η ερώτηση προωθείται μόνο εφόσον υπάρχουν γείτονες με σχετικά αποτελέσματα και μόνο σε αυτούς. Με αυτόν τον τρόπο μειώνεται το πλήθος των μηνυμάτων που ανταλλάσσονται μεταξύ των κόμβων του συστήματος.

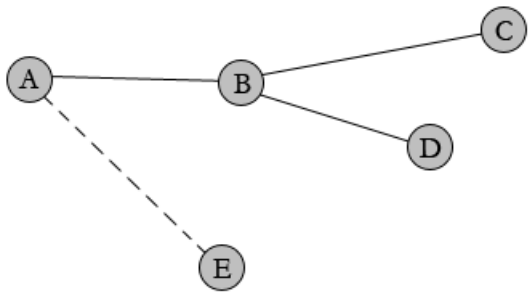
Στο παράδειγμά μας, ας υποθέσουμε πως ο κόμβος *A* δέχεται την ερώτηση */a/b*. Αρχικά ελέγχει το τοπικό ιστόγραμμα bloom και βρίσκει πως διαθέτει τοπικά 10 μονοπάτια που απαντούν την ερώτηση. Στη συνέχεια ελέγχει το συγχωνευμένο ιστόγραμμα bloom και βρίσκει πως έχει γείτονες που διαθέτουν απαντήσεις για την ερώτηση και πως αυτοί είναι οι *B* και *E*. Επομένως ο *A* προωθεί την ερώτηση μόνο στους *B* και *E* (και όχι στον G).

Είναι συχνό φαινόμενο οι ερωτήσεις που υποβάλλονται στα συστήματα ομότιμων κόμβων να συνοδεύονται από κάποιες *συνθήκες περιορισμού*, όπως π.χ. «επέστρεψε τα 20 πρώτα αποτελέσματα». Η χρήση των ιστογραμμάτων bloom μπορεί σε αυτές τις περιπτώσεις να μειώσει περαιτέρω το κόστος της ερώτησης. Αν μαζί με την ερώτηση προωθείται στους γειτονικούς κόμβους και ο αριθμός των αποτελεσμάτων που έχουν ήδη βρεθεί, οι κόμβοι μπορούν να σταματήσουν την προώθηση της ερώτησης όταν με τον αριθμό των αποτελεσμάτων από το τοπικό τους ιστόγραμμα συμπληρώνεται ο απαιτούμενος αριθμός αποτελεσμάτων.

5.3.2 Δημιουργία και ενημέρωση ευρετηρίων δρομολόγησης

Στην §5.3.1 είδαμε πως μπορούμε να χρησιμοποιήσουμε τα ιστογράμματα bloom ως ευρετήρια δρομολόγησης. Στη συνέχεια θα δούμε πως τα ευρετήρια αυτά δημιουργούνται καθώς νέοι κόμβοι εισέρχονται στο σύστημα.

Ας υποθέσουμε πως διαθέτουμε το σύστημα που παρουσιάζεται στο Σχήμα 5-3 με τις συμπαγείς γραμμές και πως στο σύστημα εισέρχεται ένας νέος κόμβος (η νέα σύνδεση παρουσιάζεται με διακεκομμένη γραμμή).



Σχήμα 5-3: Προσθήκη νέου κόμβου στο σύστημα.

Κατά τη σύνδεση του νέου κόμβου E, ο E στέλνει στον A το τοπικό του ιστόγραμμα bloom. Ο A συγχωνεύει το ιστόγραμμα αυτό στο συγχωνευμένο ιστόγραμμα bloom που ήδη διαθέτει και παράγει ένα νέο με το οποίο αντικαθιστά το παλιό. Επίσης, ο A στέλνει στον E το προηγούμενο συγχωνευμένο ιστόγραμμα bloom που διέθετε (το οποίο δεν περιλαμβάνει τις αλλαγές που οφείλονται στον κόμβο E) συγχωνευμένο με το δικό του

τοπικό ιστόγραμμα έτσι ώστε ο E να δημιουργήσει το δικό του συγχωνευμένο ιστόγραμμα bloom (που σε αυτό το αρχικό στάδιο περιέχει μόνο πληροφορία για τα μονοπάτια που είναι προσβάσιμα μέσω του A που είναι και ο μοναδικός γείτονας του E). Παρατηρούμε πως η ανταλλαγή ιστογραμμάτων μεταξύ των δύο κόμβων μπορεί να πραγματοποιηθεί παράλληλα.

Στην περίπτωση που ο E αποτελούσε ήδη μέρος του συστήματος (και άρα η σύνδεση $A \rightarrow E$ δημιουργεί κύκλο) ο E θα έπρεπε να στείλει το συγχωνευμένο του ιστόγραμμα στον A και αντίστοιχα να συγχωνεύσει το ιστόγραμμα που θα του έστελνε ο A σε αυτό.

Μένει ακόμα να ενημερωθούν και οι υπόλοιποι γείτονες του A για τα νέα δεδομένα που είναι προσβάσιμα μέσω αυτού. Όλοι οι κόμβοι στέλνουν περιοδικά αντίγραφα του συγχωνευμένου ιστογράμματος bloom που διαθέτουν στους γείτονες τους. Το πόσο συχνά γίνεται αυτό αποτελεί μία από τις παραμέτρους του συστήματος. Κάθε κόμβος δημιουργεί κατά διαστήματα ένα νέο συγχωνευμένο ιστόγραμμα bloom με βάση αυτά που λαμβάνει από τους γείτονές του. Επομένως, οι αλλαγές που έγιναν στο συγχωνευμένο ιστόγραμμα του A θα αρχίσουν να διαδίδονται κάποια στιγμή και στο υπόλοιπο σύστημα.

Μέσω της περιοδικής ανταλλαγής ιστογραμμάτων οι κόμβοι ενημερώνονται και για αποχωρήσεις άλλων κόμβων από το σύστημα. Αν κάποιος κόμβος δε λάβει ένα ιστόγραμμα από κάποιον γείτονά του για κάποιο συγκεκριμένο χρονικό διάστημα θεωρεί πως ο κόμβος αυτός δεν αποτελεί πλέον μέρος του συστήματος.

5.3.3 Αντιμετώπιση κύκλων

Όπως αναφέραμε είναι δυνατό σε ένα σύστημα ομότιμων κόμβων να δημιουργηθούν κύκλοι μεταξύ των κόμβων. Κάτι τέτοιο αποτελεί ένα γενικότερο πρόβλημα των συστημάτων αυτών και μπορεί να δημιουργήσει προβλήματα στη λειτουργία των ευρετηρίων δρομολόγησης, καθώς αλλαγές που οφείλονται σε έναν κόμβο μπορεί να συνεχίσουν να διαδίδονται έπ' άπειρον κατά μήκος του κύκλου. Οι τακτικές που ακολουθούνται για την αντιμετώπιση αυτού του προβλήματος χωρίζονται σε δύο κατηγορίες:

- **Αποφυγή δημιουργίας κύκλων:** Σε αυτή την περίπτωση οι κόμβοι δεν προωθούν τις αλλαγές σε διαδρομές που σχηματίζουν κύκλους. Υπάρχουν διάφορες τεχνικές που μπορούν να εφαρμοστούν για την εύρεση κύκλων που υπάρχουν στο σύστημα αλλά η χρήση τους αυξάνει την πολυπλοκότητα του συστήματος και το συνολικό κόστος.
- **Εντοπισμός κύκλων και ανάκαμψη:** Σε αυτή την περίπτωση οι κύκλοι εντοπίζονται αφού δημιουργηθούν και λαμβάνονται μέτρα για την αναίρεση των αλλαγών που έγιναν λόγω αυτών. Ένας τρόπος για να εντοπίσουμε κύκλους είναι να συμπεριλαμβάνουμε στα διάφορα μηνύματα και την ταυτότητα του αρχικού αποστολέα.

Το ζήτημα της αντιμετώπισης κύκλων είναι ένα γενικό πρόβλημα των συστημάτων ομότιμων κόμβων και δε θα μας απασχολήσει άλλο σε αυτή την εργασία.

5.4 Ομαδοποίηση κόμβων με βάση το περιεχόμενο

Ένα άλλο θέμα που αφορά τα συστήματα ομότιμων κόμβων είναι ο τρόπος με τον οποίο επιλέγονται οι γείτονες του κάθε κόμβου. Συνήθως αυτό γίνεται με τρόπο βασισμένο στο δίκτυο μέσω του οποίου επικοινωνούν οι κόμβοι ή στις υπολογιστικές ικανότητες αυτών. Μία ακόμα ενδιαφέρουσα προσέγγιση στο θέμα αυτό είναι η οργάνωση του συστήματος με βάση τα δεδομένα που βρίσκονται αποθηκευμένα σε κάθε κόμβο.

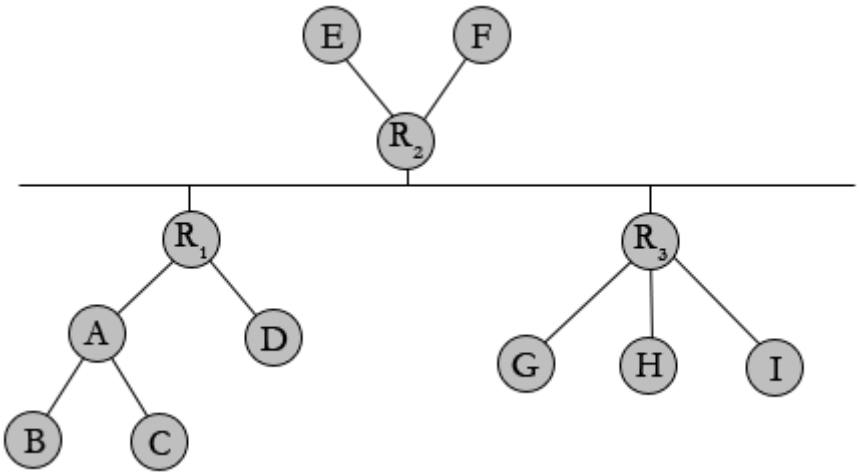
Σκοπός αυτής της οργάνωσης είναι ο εξής: κόμβοι με παρόμοια δεδομένα να βρίσκονται σε μικρή απόσταση ο ένας από τον άλλο, δηλαδή να μπορούν να επικοινωνήσουν μέσω ενός μικρού αριθμού ενδιάμεσων κόμβων. Με αυτόν τον τρόπο μειώνεται το πλήθος των κόμβων που αν και δε διαθέτουν δεδομένα που απαντούν μία ερώτηση συμμετέχουν στη διαδικασία δρομολόγησής της.

5.4.1 Ιεραρχική οργάνωση συστημάτων ομότιμων κόμβων

Για να μπορέσει να επιτευχθεί μία οργάνωση του συστήματος όπως αυτή που περιγράφηκε, οι νέοι κόμβοι που εισάγονται σε αυτό δεν μπορούν να συνδέονται σε κάποια τυχαία θέση. Θα πρέπει να εντοπίζουν μία ομάδα κόμβων με περιεχόμενο παρόμοιο με το δικό τους και να συνδέονται με αυτή. Αυτός ο εντοπισμός διευκολύνεται αν το σύστημά μας είναι

δομημένο ιεραρχικά και όχι με τυχαίο τρόπο. Ένας τρόπος ιεραρχικής οργάνωσης που θα περιγράψουμε συνοπτικά στη συνέχεια αναφέρεται στο [4].

Διακρίνουμε δύο είδη κόμβων, τους *κεντρικούς* και τους *απλούς*. Οι κεντρικοί κόμβοι συνδέονται όλοι μεταξύ τους μέσω ενός κεντρικού καναλιού επικοινωνίας. Πρόκειται συνήθως για κόμβους με αυξημένες υπολογιστικές δυνατότητες που βρίσκονται για μεγάλα χρονικά διαστήματα συνδεδεμένοι στο σύστημα. Σε κάθε κεντρικό κόμβο συνδέεται ένα πλήθος απλών κόμβων με τοπολογία δένδρου. Μία τέτοια οργάνωση φαίνεται στο Σχήμα 5-4 (όπου με R_i συμβολίζουμε τους κεντρικούς κόμβους):



Σχήμα 5-4: Ιεραρχική οργάνωση P2P.

Κάθε κόμβος διατηρεί δύο ευρετήρια, ένα τοπικό και ένα συγχωνευμένο όπως περιγράψαμε και στην §5.3. Σε αυτήν την περίπτωση το συγχωνευμένο ευρετήριο περιέχει πληροφορία όχι για όλους τους γείτονες του κόμβου αλλά μόνο για αυτούς που ανήκουν στο υποδένδρο του. Για το λόγο αυτό οι κόμβοι που βρίσκονται στα άκρα του δικτύου δε διαθέτουν συγχωνευμένο ευρετήριο. Επιπλέον, οι κεντρικοί κόμβοι διατηρούν εκτός από το δικό τους συγχωνευμένο ευρετήριο και από ένα αντίγραφο των συγχωνευμένων ευρετηρίων των υπόλοιπων κεντρικών κόμβων.

5.4.2 Εισαγωγή νέων κόμβων

Όταν ένας νέος κόμβος θέλει να εισέλθει στο σύστημα επικοινωνεί αρχικά με κάποιον κεντρικό κόμβο αποστέλλοντάς του το τοπικό του ευρετήριο. Ο κεντρικός κόμβος

υπολογίζει τον κεντρικό κόμβο που διαθέτει το πλησιέστερο σε αυτό συγχωνευμένο ευρετήριο (ευρετήρια που απέχουν λιγότερο το ένα από το άλλο αντιστοιχούν σε πιο παρόμοια δεδομένα. Το πώς γίνεται ο υπολογισμός της απόστασης αυτής στην περίπτωση που ως ευρετήρια χρησιμοποιούνται ιστογράμματα bloom θα μας απασχολήσει στην §5.5) και επιστρέφει το αποτέλεσμα στο νέο κόμβο. Στη συνέχεια ο νέος κόμβος επικοινωνεί με αυτόν τον κόμβο και με παρόμοιες συγκρίσεις μεταξύ του τοπικού του ευρετηρίου και των συγχωνευμένων ευρετηρίων των κόμβων του αντίστοιχου υποδένδρου εντοπίζει το κατάλληλο σημείο του συστήματος στο οποίο πρέπει να συνδεθεί.

5.5 Ομοιότητα ιστογραμμάτων bloom

Στην ενότητα αυτή προτείνουμε κάποιους τρόπους για τον υπολογισμό της απόστασης μεταξύ δύο ιστογραμμάτων bloom. Θα υποθέσουμε ότι τα ιστογράμματα διαθέτουν ίσο μήκος φίλτρων bloom και ίδιο αριθμό κάδων αν και μερικοί από τους τρόπους που θα δούμε μπορούν να γενικευθούν και στην περίπτωση που αυτό δεν ισχύει.

Δεν είναι προφανές το πώς μπορούμε να ορίσουμε την ομοιότητα μεταξύ δύο ιστογραμμάτων bloom. Διαισθητικά, θα μπορούσαμε να πούμε ότι δύο ιστογράμματα είναι περισσότερο όμοια όταν διαθέτουν παρόμοιο αριθμό κάδων με παρόμοια φίλτρα bloom και συχνότητες. Ο τρόπος όμως με τον οποίο δημιουργούνται τα ιστογράμματα bloom δεν καθιστά εύκολες αυτές τις συγκρίσεις. Μία από τις δυσκολίες που συναντούμε είναι ότι τα αρχικά δεδομένα βρίσκονται κωδικοποιημένα στα φίλτρα bloom του ιστογράμματος με τέτοιον τρόπο έτσι ώστε να μη μπορούν να εξαχθούν από αυτά. Δύο διαφορετικά σύνολα δεδομένων θα μπορούσαν να δημιουργήσουν παρόμοια φίλτρα bloom αν οι συναρτήσεις κατακερματισμού του ιστογράμματος το επέτρεπαν. Επίσης, καθώς τα διάφορα μονοπάτια των αρχικών δεδομένων μοιράζονται στους κάδους σύμφωνα με τη συχνότητά τους αλλά και τον αριθμό των κάδων του ιστογράμματος, δύο ιστογράμματα που προέρχονται από το ίδιο σύνολο δεδομένων αλλά διαθέτουν διαφορετικό αριθμό κάδων μπορεί να διαφέρουν αρκετά.

Στην ενότητα αυτή επιχειρούμε μια πρώτη προσέγγιση στην εύρεση παρόμοιων ιστογραμμάτων bloom. Επειδή το περιεχόμενο του αρχικού συνόλου δεδομένων κωδικοποιείται μέσω των φίλτρων bloom των ιστογραμμάτων, μία ιδέα είναι να ορίσουμε ένα μέτρο σύγκρισης βασισμένο στην απόσταση μεταξύ δύο φίλτρων bloom B_1 και B_2 (την

οποία ορίσαμε ως $dist(B_1, B_2) = |B_1[1] - B_2[1]| + |B_1[2] - B_2[2]| + ... + |B_1[m] - B_2[m]|$ στην §4.2.2, όπου *m* το μήκος των φίλτρων). Προς το παρόν θα αγνοήσουμε τις συχνότητες των κάδων καθώς θεωρούμε πιο σημαντικό για την ομοιότητα των ιστογραμμάτων την ομοιότητα των φίλτρων bloom (και κατ' επέκταση και την ομοιότητα των δεδομένων που έχουν εισαχθεί σε αυτά). Πιο συγκεκριμένα, προσπαθούμε να εντοπίσουμε ζεύγη φίλτρων bloom (ένα από κάθε ιστόγραμμα) τα οποία να απέχουν όσο το δυνατόν λιγότερο μεταξύ τους. Στη συνέχεια αθροίζουμε τις αποστάσεις μεταξύ των ζευγών αυτών και ορίζουμε το αποτέλεσμα να είναι η απόσταση μεταξύ των δύο ιστογραμμάτων bloom.

Μία προσέγγιση παρουσιάζεται στον Αλγόριθμο 5-1:

Αλγόριθμος 5-1: `BloomHistogram_dist_1(BH1, BH2)`

```
1: total_dist = 0;
2: for i = 1 to b2 do
3:  BH2.used[i] = false;
4: end for
5: for i = 1 to b1 do
6:  dist = +∞
7:  for j = 1 to b2 do
8:   if dist(BH1.BF[i], BH2.BF[j]) < dist and BH2.used[j] = false then
9:    dist = dist(BH1.BF[i], BH2.BF[j]);
10:    BH2.used[j] = true;
11:   end if
12:  end for
13:  total_dist = total_dist + dist;
14: end for
15: return total_dist;
```

Αλγόριθμος 5-1: Υπολογισμός απόστασης ιστογραμμάτων bloom (1).

Πιο αναλυτικά, ο Αλγόριθμος 5-1 υπολογίζει την απόσταση (`total_dist`) του ιστογράμματος BH_1 από το ιστόγραμμα BH_2. Σε κάθε κάδο του ιστογράμματος BH_2 αντιστοιχούμε ένα πεδίο `used` που παίρνει την τιμή `true` ή `false`, ανάλογα με το αν ο αντίστοιχος κάδος έχει ήδη συνδυαστεί με κάποιον κάδο του BH_1. Στη συνέχεια, για κάθε κάδο του BH_1 εντοπίζεται ο κάδος εκείνος του BH_2, που δεν έχει συνδυαστεί ακόμα με κανέναν άλλο κάδο, έτσι ώστε τα δύο φίλτρα bloom να απέχουν το λιγότερο δυνατό.

Ο Αλγόριθμος 5-1 έχει το μειονέκτημα να δίνει αποστάσεις που δεν είναι πάντα συμμετρικές. Υπάρχει δηλαδή περίπτωση να ισχύει `BloomHistogram_dist_1(BH1, BH2) ≠ BloomHistogram_dist_1(BH2, BH1)`. Παρόλα αυτά είναι εύκολος στην υλοποίηση του και εξυπηρετεί στην ομαδοποίηση κόμβων με βάση το περιεχόμενο αφού παρέχει έναν τρόπο εύρεσης του ιστογράμματος που υπάρχει ήδη στο σύστημα και απέχει το λιγότερο από το νέο ιστόγραμμα που εισάγεται στο σύστημα. Η πολυπλοκότητα του Αλγορίθμου 5-1 είναι $O(b^2)$, όπου b ο αριθμός των κάδων του κάθε ιστογράμματος.

Επεκτείνοντας την ιδέα αυτή, μπορούμε να χρησιμοποιήσουμε μία παραλλαγή του Αλγορίθμου 5-1 συνδυάζοντας σε κάθε βήμα εκείνο το ζευγάρι των κάδων το οποίο έχει τη μικρότερη απόσταση από όλα τα άλλα. Κάτι τέτοιο αυξάνει την πολυπλοκότητα του αλγορίθμου σε $O(b^3)$ (πιθανόν λιγότερο αν χρησιμοποιήσουμε τεχνικές δυναμικού προγραμματισμού) αλλά οδηγεί σε συμμετρικές αποστάσεις. Η παραλλαγή αυτή φαίνεται στον Αλγόριθμο 5-2:

Αλγόριθμος 5-2: `BloomHistogram_dist_2(BH1, BH2)`

```
1: total_dist = 0;
2: for i = 1 to b do
3:  BH1.used[i] = false;
4:  BH2.used[i] = false;
5: end for
6: for i = 1 to b do
7:  dist = +∞
8:  for j = 1 to b do
9:   for k = 1 to b do
10:    if dist(BH1.BF[j], BH2.BF[k]) < dist
11:     and BH1.used[j] = false
12:     and BH2.used[k] = false then
13:      dist = dist(BH1.BF[j], BH2.BF[k]);
14:      BH1.used[j] = true;
15:      BH2.used[k] = true;
16:    end if
17:   end for
18:  end for
19:  total_dist = total_dist + dist;
20: end for
21: return total_dist;
```

Αλγόριθμος 5-2: ***Υπολογισμός απόστασης ιστογραμμάτων bloom (2).***

Αρχικά, για κάθε κάδο και των δύο ιστογραμμάτων δίνουμε στο πεδίο `used` την τιμή `false`. Στη συνέχεια επιλέγουμε το ζεύγος των κάδων (έναν από κάθε ιστόγραμμα) που έχει τη μικρότερη απόσταση από όλα τα άλλα ζεύγη. Συνδυάζουμε τους κάδους αυτούς και δίνουμε στα πεδία `used` αυτών την τιμή `true`. Η διαδικασία επαναλαμβάνεται έως ότου συνδυαστούν όλοι οι κάδοι των ιστογραμμάτων.

Μία άλλη ιδέα είναι να συνδυάσουμε όλα τα φίλτρα bloom ενός εκ των δύο ιστογραμμάτων σε ένα νέο φίλτρο *BF* μέσω της πράξης του δυαδικού OR και στη συνέχεια να αθροίσουμε τις αποστάσεις όλων των φίλτρων του άλλου ιστογράμματος από το *BF*. Ο αλγόριθμος αυτός έχει χαμηλή πολυπλοκότητα $O(b)$ αλλά δεν είναι ιδιαίτερα αποτελεσματικός όταν τα φίλτρα bloom έχουν πολλά bit με τιμή 1 έτσι ώστε όταν δημιουργείται το *BF* να αποτελείται σχεδόν αποκλειστικά από bit με τιμή 1. Παρόλα αυτά όταν τα δεδομένα που εισάγονται στα ιστογράμματα είναι μικρά σε όγκο, ο αλγόριθμος αυτός μπορεί να εφαρμοστεί. Τα αντίστοιχα βήματα φαίνονται στον Αλγόριθμο 5-3:

Αλγόριθμος 5-3: `BloomHistogram_dist_3(BH1, BH2)`

```
1: total_dist = 0;
2: BF = 0;
3: for i = 1 to b2 do
4:  BF = BF BOR BH2.BF[i];
5: end for
6: for i = 1 to b1 do
7:  total_dist = total_dist + dist(BH1.BF[i], BF);
8: end for
9: return total_dist;
```

Αλγόριθμος 5-3: Υπολογισμός απόστασης ιστογραμμάτων bloom (3).

Και οι τρεις αλγόριθμοι μπορούν να χρησιμοποιηθούν για την ομαδοποίηση των κόμβων ενός συστήματος ομότιμων κόμβων. Καθένας από αυτούς έχει πλεονεκτήματα και μειονεκτήματα. Ο Αλγόριθμος 5-1 δε δίνει συμμετρικές αποστάσεις αλλά έχει ικανοποιητική πολυπλοκότητα και μπορεί να χρησιμοποιηθεί και με ιστογράμματα που διαθέτουν διαφορετικό αριθμό κάδων. Ο Αλγόριθμος 5-2 έχει αυξημένη πολυπλοκότητα αλλά δίνει συμμετρικές αποστάσεις και διαισθητικά φαίνεται πως παρέχει έναν καλύτερο ορισμό της απόστασης δύο ιστογραμμάτων bloom. Μπορεί όμως να χρησιμοποιηθεί μόνο με ιστογράμματα που διαθέτουν τον ίδιο αριθμό κάδων. Ο Αλγόριθμος 5-3 είναι ένας απλός αλγόριθμος με χαμηλή πολυπλοκότητα ο οποίος όμως μπορεί να χρησιμοποιηθεί

μόνο σε ιδιαίτερες περιπτώσεις όπου τα φίλτρα bloom των ιστογραμμάτων είναι «αραιά», δηλαδή έχουν λίγα bit με την τιμή 1.

Κεφάλαιο 6: Μελλοντική εργασία

Τα ιστογράμματα bloom αποτελούν μία αρκετά ενδιαφέρουσα νέα δομή. Αρχικά προτάθηκαν απλά για να συμβάλλουν στην εκτίμηση της συχνότητας διάφορων μονοπατιών XML δεδομένων [7], αλλά όπως διαπιστώσαμε μπορούν να εφαρμοστούν σε διάφορες πτυχές της λειτουργίας ενός συστήματος ομότιμων κόμβων.

Στην εργασία αυτή εξερευνήσαμε δύο τέτοιες πτυχές. Πρώτον, τη συγχώνευση των ιστογραμμάτων σε ιστογράμματα που περιέχουν συγκεντρωτική πληροφορία για τα δεδομένα που περιέχονται στο σύστημα. Δεύτερον, την χρήση αυτών των ιστογραμμάτων στη δρομολόγηση ερωτήσεων μέσα στο σύστημα.

Υπάρχουν διάφορα ζητήματα που θεωρούμε πως αξίζει να μελετηθούν περισσότερο στο μέλλον. Ένα από αυτά είναι η απόδοση των αλγορίθμων συγχώνευσης που προτάθηκαν στο Κεφάλαιο 4 και η προσπάθεια εύρεσης κάποιων κανόνων για το πότε είναι πιο αποδοτικός καθένας από αυτούς. Ένα άλλο ενδιαφέρον ζήτημα είναι ο ορισμός της απόστασης μεταξύ δύο ιστογραμμάτων bloom. Οι αλγόριθμοι που προτάθηκαν στην §5.5 είναι ικανοί να χρησιμοποιηθούν για την ομαδοποίηση κόμβων αλλά παρόλα αυτά ακόμα δεν έχει προταθεί ένας τρόπος ορισμού της απόστασης μεταξύ δύο ιστογραμμάτων bloom ο οποίος να λαμβάνει υπόψη τόσο τα δεδομένα που έχουν κωδικοποιηθεί μέσω των φίλτρων bloom όσο και τη συχνότητα εμφάνισής τους. Εκτιμούμε πως ένας τέτοιος ορισμός δεν είναι προφανής. Καθώς όμως η απόσταση δύο ιστογραμμάτων bloom μπορεί να βρει διάφορες εφαρμογές, θα ήταν ενδιαφέρον να μελετηθεί περισσότερο.

Αναφορές

[1] S. Abiteboul, P. Buneman, D. Suciu, *Data on the Web From Relations to Data and XML*, Morgan Kaufmann Publishers, 1999

[2] B. H. Bloom, *Space/time trade-offs in hash coding with allowable errors*, Communications of the ACM, vol. 13(7) pages 422-426, 1970

[3] A. Crespo, H. Garcia-Molina, *Routing Indices For Peer-to-Peer Systems*, International Conference on Distributed Computing Systems, 2002

[4] G. Koloniari, E. Pitoura, *Content-Based Routing of Path Queries in Peer-to-Peer Systems*, EDBT 2004: 29-47

[5] S. Ratnamasy, P. Francis, M. Handley, R. Karp, S. Shenker, *A Scalable Content-Addressable Network*, ACM SIGCOMM, 2001

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H.Balakrishnan, *CHORD: A Scalable Peer-to-Peer Lookup Service for Internet Applications*, ACM SIGCOMM, 2001

[7] Wei Wang, Haifeng Jiang, Hongjun Lu, Jeffrey Xu Yu, *Bloom Histogram: Path Selectivity Estimation for XML Data with Updates*, Proceedings of the 30th VLDB Conference, 2004

[8] Freenet, *http://freenet.sourceforge.com*

[9] General Hash Functions, *http://www.partow.net/programming/hashfunctions/*

[10] Gnutella, *http://gnutella.wego.com*

[11] Napster, *http://www.napster.com*

[12] Xerces, *http://xml.apache.org/xerces-c/*

[13] XMark, *http://monetdb.cwi.nl/xml/*

[14] XML, *http://www.w3.org/XML/*

[15] XML Data Repository, *http://www.cs.washington.edu/research/xmldatasets/*

[16] XPath, *http://www.w3.org/TR/xpath20/*

Παράρτημα Α

Κώδικας υλοποίησης

1. Path_Count

Είσοδος:	Το αρχικό σύνολο XML δεδομένων <source file>.
Λειτουργία:	Parsing του συνόλου δεδομένων με τη βοήθεια της βιβλιοθήκης Xerces[5]. Δημιουργία δέντρου DOM και χρήση αυτού για την εύρεση όλων των μονοπατιών που συναντούνται στο σύνολο δεδομένων, ανεξαρτήτως μήκους και θέσης αυτών στο δέντρο.
Έξοδος:	Όλα τα μονοπάτια που βρέθηκαν γράφονται στο <target file>.
Τρόπος κλήσης:	path_count <source file> <target file>

```
#include <xercesc/util/PlatformUtils.hpp>
#include <xercesc/parsers/XercesDOMParser.hpp>
#include <xercesc/dom/DOM.hpp>
#include <xercesc/sax/HandlerBase.hpp>
#include <xercesc/util/XMLString.hpp>
#include <xercesc/framework/StdOutFormatTarget.hpp>

#define MAX_PATH_LENGTH 500

using namespace std;
XERCES_CPP_NAMESPACE_USE

// We are going to find all paths under the node <node>. The path //
// that leads to <node> is <path>.
void findPaths(DOMNode *node, char *path, FILE *outFile)
```

```
{
  DOMNodeList *children = NULL;
  DOMNode *current = NULL;
  char mypath[MAX_PATH_LENGTH],
     subpath[MAX_PATH_LENGTH],
     tmppath[MAX_PATH_LENGTH];
  int k;

  // The path up to now
  strcpy(mypath, path);

  // Get the current node's name
  char *strValue = XMLString::transcode(node->getNodeName());
  sprintf(subpath, "%s/", strValue);
  XMLString::release(&strValue);

  // Add the current node's name to the path
  strcat(mypath, subpath);
  if(strcmp(mypath, subpath) != 0)
  {
    fprintf(outFile, "%s\n", mypath);
  }

  // Call findPaths for all the children of <node>
  if(node->hasChildNodes())
  {
    children = node->getChildNodes();
    k = 0;
    current = children->item(k);

    while(current != NULL)
    {
      if(current->getNodeType() == DOMNode::ELEMENT_NODE)
      {
        findPaths(current, mypath, outFile);
        if(strcmp(path, "\0") == 0)
                        findPaths(current, "\0", outFile);
      }

      current = children->item(++k);
    }
```

```
  }
}

int main(int argc, char* argv[])
{
  if(argc < 3)
  {
    cout << "Usage: path_count.exe <source file> <target file>.
         Try again.\n";
    return 1;
  }

  try
  {
    XMLPlatformUtils::Initialize();
  }
  catch (const XMLException& toCatch)
  {
    char* message = XMLString::transcode(toCatch.getMessage());
    cout << "Error during initialization! :\n"
       << message << "\n";
    XMLString::release(&message);
    return 1;
  }

  // Parse the XML document and create the DOM tree
  XercesDOMParser* parser = new XercesDOMParser();

  ErrorHandler* errHandler = (ErrorHandler*) new HandlerBase();
  parser->setErrorHandler(errHandler);

  char* xmlFile = argv[1];

  try
  {
    parser->parse(xmlFile);
  }
  catch (const XMLException& toCatch)
  {
    char* message = XMLString::transcode(toCatch.getMessage());
    cout << "Exception message is: \n" << message << "\n";
```

```
  XMLString::release(&message);
  return -1;
}
catch (const DOMException& toCatch)
{
  char* message = XMLString::transcode(toCatch.msg);
  cout << "Exception message is: \n" << message << "\n";
  XMLString::release(&message);
  return -1;
}
catch (...)
{
  cout << "Unexpected Exception \n" ;
  return -1;
}

// Get the DOM tree and its root node
DOMDocument *pDoc = parser->getDocument();
DOMElement *root = pDoc->getDocumentElement();

// Initialize path
char path[MAX_PATH_LENGTH] = "\0";

// Find all paths in the tree
FILE *outFile;
if((outFile = fopen(argv[2], "w")) == NULL)
{
  printf("Unable to open target file %s. Exiting...\n",
      argv[2]);
  delete parser;
  delete errHandler;
  exit(1);
}

findPaths(root, path, outFile);

fclose(outFile);
delete parser;
delete errHandler;

return 0;
```

```
}
```

2. Bloom_Histogram

Είσοδος:	Το αρχείο <source file> με τον πίνακα συχνοτήτων, ο επιθυμητός αριθμός κάδων <buckets> του ιστογράμματος και το μήκος <bloom filter size> των φίλτρων bloom του ιστογράμματος.
Λειτουργία:	Εύρεση και κατασκευή του βέλτιστου ιστογράμματος bloom για τα συγκεκριμένα δεδομένα βάσει του αλγορίθμου που προτείνεται στο [3]. Αν η παράμετρος <mode> έχει την τιμή «on» σε κάθε κάδο προστίθεται και ένας αριθμός που δείχνει πόσα μονοπάτια έχουν εισαχθεί στον κάδο.
Έξοδος:	Το ιστόγραμμα bloom γράφεται στο <target file>.
Τρόπος κλήσης:	bloom_histogram <source file> <buckets> <target file> <mode> <bloom filter size>

```
#include "stdafx.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string.h>
#include <math.h>

// Hash functions library
#include "GeneralHashFunctions.h"

using namespace std;

// Hash functions number
#define HASH_NUM 4

// Max parameters
#define MAX_BF_SIZE 2048
#define MAX_PATHS 10000
#define MAX_BUCKETS 2000
#define MAX_PATH_LENGTH 500
```

```
#define INF 10000
#define min(X, Y) ((X) < (Y) ? (X) : (Y))

// Arrays
int PSUM[MAX_PATHS];
int COUNT[MAX_PATHS];
int OPT[MAX_PATHS][MAX_BUCKETS][2];  //0: value 1: bucket boundary
int BOUNDARY[MAX_BUCKETS];

// Bloom Histogram
struct BH_node
{
  int bf[MAX_BF_SIZE];
  int freq_value;
  int how_many;
};

BH_node BH[MAX_BUCKETS];

// Hash functions holder
vector < HashFunction > hash_function;

// 3rd column mode on
bool plus = false;

// BF size
int bf_size;


// ******************************************
// *     BUILD BLOOM HISTOGRAM       *
// ******************************************

// countPaths opens file <filename> and counts its lines
int countPaths(char *filename)
{
  string dummy;
  int counter = 0;

  ifstream infile (filename);
```

```
  while(getline(infile, dummy))
    counter++;

  infile.close();

  return counter;
}

// The following two functions are based on the pseudo-code
// of the Bloom Histogram paper ("Bloom Histogram: Path Selectivity
// Estimation for XML with Data Updates" - Wang, Jiang, Lu, Yu)
int f(int x, int y)
{
  int t = (x + y) / 2;

  return PSUM[y] + PSUM[x-1] - 2*PSUM[t] +
      (2*t - x - y + 1)*COUNT[t];
}

int BuildHistogram(char *filename, int n, int b)
{
  FILE *infile;
  char dummy[MAX_PATH_LENGTH];
  int frequency;
  int i = 1, j, k;
  int tmpOPT, q, r, tmp = 0;

  // Open Path-Count table file
  if((infile = fopen(filename, "r")) == NULL)
  {
    printf("Error opening frequency table file.\n");
    exit(1);
  }

  // Initialize COUNT and PSUM arrays
  fscanf(infile, "%s %d", &dummy, &frequency);
  COUNT[i] = frequency;
  PSUM[i++] = frequency;

  while(fscanf(infile, "%s %d", &dummy, &frequency) != EOF)
  {
```

```
  COUNT[i] = frequency;
  PSUM[i++] = PSUM[i-1] + frequency;
}

fclose(infile);

// Initialize OPT array
for(i=1; i<=n; i++)
  OPT[i][1][0] = f(1, i);

// Find OPT values
for(k=2; k<=b; k++)
{
  for (j=1; j<=n; j++)
  {
    OPT[j][k][0] = INF;
    OPT[j][k][1] = tmp;

    for(i=k-1; i<=j-1; i++)
    {
      tmpOPT = OPT[j][k][0];

                OPT[j][k][0]  =  min(OPT[j][k][0],  OPT[i][k-1][0]  +
                                f(i+1, j));

      if(OPT[j][k][0] < tmpOPT) //has changed
       tmp = OPT[j][k][1] = i+1;
    }
  }
}

// Define boundaries and store them in BOUNDARY[]
q = n;
if(b > n)  r = n;
else r = b;
i = b-1;

while(r > 1)
{
  BOUNDARY[i--] = OPT[q][r][1];
  q = OPT[q][r][1] - 1;
```

```
    r--;
  }

  // Print array
  for(i=0; i<=b; i++)
    printf("BOUNDARY[%d] = %d\n", i, BOUNDARY[i]);
  printf("\n");

  return OPT[n][b][0];
}

// InsertBF adds a path to a certain bloom filter
void insertBF(char *path, int *bf)
{
  unsigned int result, i;

  for(i=0; i<hash_function.size(); i++)
  {
    result = hash_function[i](path) % bf_size;

    bf[result] = 1;
  }
}

// Insert Paths adds the paths of <filename> into the global array //
BLOOM[] based on the boundaries stored in BOUNDARY[]
void insertPaths(char *filename)
{
  FILE *infile;
  char path[MAX_PATH_LENGTH];
  int freq, bucket, boundary, index, bindex, tmp;

  int value, howmany;

  double total_error, bucket_error;
  int counter, prev_freq;

  // Open Path-Count table file
  if((infile = fopen(filename, "r")) == NULL)
  {
    printf("Error opening frequency table file.\n");
```

```
  exit(1);
}

index = 1;
bindex = 1;
bucket = 0;
boundary = BOUNDARY[bindex];   //start index of the first bucket
                                 //with boundary != 0
while(boundary == 0)
{
  bindex++;
  boundary = BOUNDARY[bindex];
}

  // Also calculate the value (frequency) of each bucket. To do
  // this we add all the frequecies of the paths in the bucket and
  // divide by their number
value = howmany = 0;

total_error = 0.0;
prev_freq = 0;
counter = 0;

while(fscanf(infile, "%s %d", &path, &freq) != EOF)
{
  insertBF(path, BH[bucket].bf);

  (BH[bucket].how_many)++;

  value += freq;
  howmany++;

  counter++;

  index++;

  if(index == boundary)
  {
    BH[bucket].freq_value = value / howmany;

    bucket++;
```

```
      bindex++;
      boundary = BOUNDARY[bindex];



      value = howmany = 0;

      // Calculate mean bloom filter error for the histogram
      bucket_error = pow((1.0 - exp((double)
              (-HASH_NUM*counter)/bf_size)), HASH_NUM);
      total_error += bucket_error;
      counter = 0;
    }


    prev_freq = freq;

  }

  // The last bucket cannot be updated inside the while loop
  // because of the EOF
  BH[bucket].freq_value = value / howmany;
  fclose(infile);

  bucket_error = pow((1.0 - exp((double)
          (-HASH_NUM*counter)/bf_size)), HASH_NUM);
  total_error += bucket_error;

  printf("BH_error = %E %15.13f\n", (double)
     (total_error/bucket),  (double)(total_error/bucket));
}

// printBH prints the global BLOOM[] array
void printBH(int b, char *filename)
{
  int i, j;
  FILE *outfile;

  if((outfile = fopen(filename, "w")) == NULL)
  {
    printf("Connot write to target file. Exiting...\n");
```

```
    exit(1);
  }

  for(i=0;i<b;i++)
  {
    if(BH[i].freq_value != 0)
    {
      for(j=0;j<bf_size;j++)
        fprintf(outfile, "%d", BH[i].bf[j]);

      if (plus == true)
        fprintf(outfile, "\t%d %d\n", BH[i].freq_value,
            BH[i].how_many);
      else
        fprintf(outfile, "\t%d\n", BH[i].freq_value);
    }
  }
}


// ********************************************
// *          MAIN          *
// ********************************************

int _tmain(int argc, _TCHAR* argv[])
{
  int num_of_buckets, num_of_paths, i, j;

  if(argc != 6)
  {
    printf("Usage: bloom_histogram <source file> <number of
        buckets> <target file> <mode on/off> <bf size>.
        Try again.\n");
    exit(1);
  }

  // Using the following four hash functions
  hash_function.push_back(RSHash );
  hash_function.push_back(JSHash );
  hash_function.push_back(PJWHash );
  hash_function.push_back(BKDRHash);
```

```
  //Defining the boundaries of the buckets
  num_of_paths = countPaths(argv[1]);
  num_of_buckets = atoi(argv[2]);
  bf_size = atoi(argv[5]);

  for(i=0;i<MAX_BUCKETS;i++)
  {
    for(j=0;j<bf_size;j++)
      BH[i].bf[j] = 0;
    BH[i].how_many = 0;
  }

  if(strcmp(argv[4], "on") == 0)
    plus = true;

  BuildHistogram(argv[1], num_of_paths, num_of_buckets);

  //Inserting the paths into the buckets
  insertPaths(argv[1]);

  //Print the Bloon Histogram
  printBH(num_of_buckets, argv[3]);


  return 0;
}
```

3. Bloom_Query

Είσοδος:	Το αρχείο <bh file> που περιέχει το ιστόγραμμα bloom και ένα μονοπάτι <path>.
Λειτουργία:	Εύρεση της συχνότητας του <path> σύμφωνα με το <bh file>.
Έξοδος:	Η συχνότητα που βρέθηκε.
Τρόπος κλήσης:	bloom_query <bh file> <path>

```
#include <iostream>
#include <fstream>
```

```
#include <vector>
#include <string.h>

// Hash functions library
#include "GeneralHashFunctions.h"

using namespace std;

// Max parameters
#define MAX_PATHS 10000
#define MAX_BUCKETS 100
#define MAX_PATH_LENGTH 500
#define MAX_BF_LENGTH 1024

typedef int bloomfilter;
typedef string path;

// Hash functions holder
vector < HashFunction > hash_function;


// ******************************************
// *      QUERY BLOOM HISTOGRAM       *
// ******************************************
float QueryBloomHistogram(char *filename, path p)
{
  FILE *inFile;
  unsigned int count = 0, k = 0;
  char bf[MAX_BF_LENGTH + 1];    // +1 for '\0'
  int frequency, result;
  bool match;

  if((inFile = fopen(filename, "r")) == NULL)
  {
    printf("Unable to open bloom histogram file. Exiting...\n");
    exit(1);
  }

  while(fscanf(inFile, "%s%d", &bf, &frequency) != EOF)
  {
    match = true;
```

```
    for(unsigned int i = 0; i < hash_function.size(); i++)
    {
      result = hash_function[i](p) % (strlen(bf));
      if(bf[result] != '1')
        match = false;
     }

    if(match)
    {
      count += frequency;
      k++;
    }
  }

  if (k > 0)
    return (float) count/k;
  else
    return 0;
}

// ******************************************
// *          MAIN          *
// ******************************************

int _tmain(int argc, _TCHAR* argv[])
{
  float result;

  if(argc != 3)
  {
    printf("Usage: bloom_histogram <bloom histogram file> <path>.
        Try again.\n");
    exit(1);
  }

  hash_function.push_back(RSHash );
  hash_function.push_back(JSHash );
  hash_function.push_back(PJWHash );
  hash_function.push_back(BKDRHash);
```

```
  result = QueryBloomHistogram(argv[1], argv[2]);

  cout << result << endl;

  return result;
}
```

4. Negative_Workload

Είσοδος:	Το αρχείο <dataset> που περιέχει το αρχικό σύνολο δεδομένων και ο αριθμός <number of paths> των μονοπατιών που επιθυμούμε να κατασκευάσουμε.
Λειτουργία:	Κατασκευή τυχαίων μονοπατιών από τις XML ετικέτες του <dataset>.
Έξοδος:	Το αρχείο <target file> που περιέχει τα μονοπάτια που δημιουργήθηκαν.
Τρόπος κλήσης:	negative_workload <dataset> <target file> <number of paths>

```
#include <time.h>
#include <xercesc/util/PlatformUtils.hpp>
#include <xercesc/parsers/XercesDOMParser.hpp>
#include <xercesc/dom/DOM.hpp>
#include <xercesc/sax/HandlerBase.hpp>
#include <xercesc/util/XMLString.hpp>
#include <xercesc/framework/StdOutFormatTarget.hpp>

#define MAX_PATH_LENGTH 500

#define PATH_DEPTH 6

using namespace std;
XERCES_CPP_NAMESPACE_USE

int main(int argc, char* argv[])
{
  int i, p, q, r, num;

  if(argc != 4)
  {
    printf("Usage: negative_workload <dataset> <target file>
```

```
      <number_of_paths>. Try again.\n");
  exit(1);
}

num = atoi(argv[3]);

try
{
  XMLPlatformUtils::Initialize();
}
catch (const XMLException& toCatch)
{
  char* message = XMLString::transcode(toCatch.getMessage());
  cout << "Error during initialization! :\n"
     << message << "\n";
  XMLString::release(&message);
  return 1;
}


// Parse the XML document and create the DOM tree
XercesDOMParser* parser = new XercesDOMParser();

ErrorHandler* errHandler = (ErrorHandler*) new HandlerBase();
parser->setErrorHandler(errHandler);

char* xmlFile = argv[1];

try
{
  parser->parse(xmlFile);
}
catch (const XMLException& toCatch)
{
  char* message = XMLString::transcode(toCatch.getMessage());
  cout << "Exception message is: \n" << message << "\n";
  XMLString::release(&message);
  return -1;
}
catch (const DOMException& toCatch)
{
```

```
  char* message = XMLString::transcode(toCatch.msg);
  cout << "Exception message is: \n" << message << "\n";
  XMLString::release(&message);
  return -1;
}
catch (...)
{
  cout << "Unexpected Exception \n" ;
  return -1;
}

// Get the DOM tree and its root node
DOMDocument *pDoc = parser->getDocument();
DOMElement *root = pDoc->getDocumentElement();

// Initialize paths
char path[MAX_PATH_LENGTH] = "\0";
char subpath[MAX_PATH_LENGTH] = "\0";

FILE *outFile;
if((outFile = fopen(argv[2], "w")) == NULL)
{
  printf("Unable to open target file %s. Exiting...\n",
      argv[2]);
  delete parser;
  delete errHandler;
  exit(1);
}

// create a walker to visit all text nodes.
DOMTreeWalker *walker = pDoc->createTreeWalker(root,
        DOMNodeFilter::SHOW_ELEMENT, NULL, true);
DOMNode *current = walker->nextNode();

srand((unsigned)(time(NULL)));

for(int j=0; j<num; j++)
{
  // Random path length
  p = ((rand() * PATH_DEPTH) / RAND_MAX) + 1;    // not 0
```

```
    for(i=0; i<p; i++)
    {
      // Randomly keep node
      q = (rand() * 3) / RAND_MAX;
      if(q <= 1)
      {
        char *strValue = XMLString::transcode
                 (current->getNodeName());
        sprintf(subpath, "%s/", strValue);
           XMLString::release(&strValue);

        // Add the current node's name to the path
        strcat(path, subpath);
      }

      // Randomly move down the walker
      r = (rand() * 3) / RAND_MAX;
      for(int k=0; k<r && current != 0; k++)
        current = walker->nextNode();
    }

    if(path[0] != '\0') fprintf(outFile, "%s\n", path);

    path[0] = '\0';
  }
  fclose(outFile);
  delete parser;
  delete errHandler;

  return 0;
}
```

5. Merge_bh_buckets

Είσοδος:	Το αρχείο <bh1> και <bh2> με τα δύο ιστογράμματα bloom που θα συγχωνευτούν και η παράμετρος <mode> (με τιμή «on» ή «off») με τη οποία ενεργοποιούμε τις βελτιώσεις που αναφέρθηκαν στο Κεφάλαιο 4.
Λειτουργία:	Συγχώνευση δύο ιστογραμμάτων bloom με βάση τις τιμές των κάδων.

Έξοδος:	Το αρχείο <new bh> που περιέχει το συγχωνευμένο ιστόγραμμα bloom.
Τρόπος κλήσης:	merge_bh_buckets <bh1> <bh2> <new bh> <mode>

```
#include "stdafx.h"
#include <time.h>

#define MAX_BF_SIZE 2048
#define MAX_BUCKETS 2000
#define INF 1000000

bool plus = false;

// Bloom Histogram
struct BH_node
{
  char bf[MAX_BF_SIZE];
  int freq_value;
  int how_many;
};

BH_node BH[MAX_BUCKETS];

// Global variables
int bf_size;
int index = 0;

// dist measures the distance between bf1 and bf2
int dist(char *bf1, char *bf2)
{
  int i, result = 0;

  for(i=0; i<(int)strlen(bf1); i++)
    if(bf1[i] != bf2[i])
      result ++;

  return result;
}
```

```
// loafBH loads the bloom histogram of file <bhfile> into the BH
// array
void loadBH(char *bhfile)
{
  FILE *infile;
  int pos = 0, freq, howmany;
  char bf[MAX_BF_SIZE];

  // Open bloom histogram
  if((infile = fopen(bhfile, "r")) == NULL)
  {
    printf("Error opening first bloom histogram file.
        Exiting...\n");
    exit(1);
  }

  // BH file in the form of <bloom filter, value,
  // how_many_paths_in_bucket>
  while(fscanf(infile, "%s %d %d", &bf, &freq, &howmany) != EOF)
  {
    if(pos == 0)
      bf_size = (int)strlen(bf);

    strcpy(BH[pos].bf, bf);

    BH[pos].freq_value = freq;
    BH[pos].how_many = howmany;

    pos++;
  }

  index = pos;
  fclose(infile);
}

// mergeBH merges the BH in the <bhfile> with the one stored in the
// BH array
void mergeBH(char *bhfile)
{
  FILE *infile;
  int pos = 0, freq, howmany, i;
```

```
int min, target;
char bf[MAX_BF_SIZE];
float num;

// Open bloom histogram
if((infile = fopen(bhfile, "r")) == NULL)
{
  printf("Error opening second bloom histogram file.
      Exiting...\n");
  exit(1);
}

srand((unsigned)(time(NULL)));

while(fscanf(infile, "%s %d %d", &bf, &freq, &howmany) != EOF)
{
  min = INF;
  target = 0;

  for(i=0; i<index; i++)
    if(abs(BH[i].freq_value - freq) < min)
    {
      min = abs(BH[i].freq_value - freq);
      target = i;
    }
    // Randomly choose one of the buckets with the same value
    else if(abs(BH[i].freq_value - freq) == min)
    {
      num = (float)rand() / (float)RAND_MAX;
      if(num < 0.5)
        target = i;
    }

  // Implementing MergeBuckets()
  for(i=0; i<bf_size; i++)
  {
    if(bf[i] == '0' && BH[target].bf[i] == '0')
      BH[target].bf[i] = '0';
    else
      BH[target].bf[i] = '1';
  }
```

```
    if(plus == true)
      BH[target].freq_value = (BH[target].how_many *
      BH[target].freq_value + howmany * freq) /
      (BH[target].how_many + howmany);
    else
      BH[target].freq_value = (BH[target].freq_value + freq)/2;

    BH[target].how_many += howmany;
  }

  fclose(infile);
}

// combine merges buckets which have the same bloom filter
void combine()
{
  int i, j, d;

  for(i=0; i<index; i++)
    for(j=i+1; j<index; j++)
    {
      d = dist(BH[i].bf, BH[j].bf);

      if(d == 0 && BH[i].freq_value != 0)
      {
        BH[i].freq_value += BH[j].freq_value;
        BH[i].how_many += BH[j].how_many;
        BH[j].freq_value = 0;
      }
    }
}

// printBH prints the BH array into file <filename>
void printBH(char *filename)
{
  FILE *outfile;
  int i;

  // Create bloom histogram file
  if((outfile = fopen(filename, "w")) == NULL)
```

```
  {
    printf("Error creating new bloom histogram file.
        Exiting...\n");
    exit(1);
  }

  for(i=0; i<index; i++)
    if(BH[i].freq_value != 0)
      fprintf(outfile, "%s\t%d %d\n", BH[i].bf,
          BH[i].freq_value, BH[i].how_many);

  fclose(outfile);
}

int main(int argc, char **argv)
{

  if(argc != 5)
  {
    printf("Usage: merge_bh <bh1> <bh2> <new bh> <plus mode
        on/off>. Try again.\n");
    exit(1);
  }

  if(strcmp(argv[4], "on") == 0)
    plus = true;

  // Load <bh1>
  loadBH(argv[1]);

  // Merge <bh2> into what we have in memory (<bh1>)
  mergeBH(argv[2]);

  // if(plus == true)
  //  combine();

  // Print the new bloom histogram
  printBH(argv[3]);

  return 0;
}
```

6. Merge_bh_filters

Είσοδος:	Το αρχείο <bh1> και <bh2> με τα δύο ιστογράμματα bloom που θα συγχωνευτούν και η παράμετρος <mode> (με τιμή «on» ή «off») με τη οποία ενεργοποιούμε τις βελτιώσεις που αναφέρθηκαν στο Κεφάλαιο 4.
Λειτουργία:	Συγχώνευση δύο ιστογραμμάτων bloom με βάση τα φίλτρα bloom.
Έξοδος:	Το αρχείο <new bh> που περιέχει το συγχωνευμένο ιστόγραμμα bloom.
Τρόπος κλήσης:	merge_bh_buckets <bh1> <bh2> <new bh> <mode>

```
#include "stdafx.h"

#define MAX_BF_SIZE 2048
#define MAX_BUCKETS 2000
#define INF 1000000

bool plus = false;

// Bloom Histogram
struct BH_node
{
  char bf[MAX_BF_SIZE];
  int freq_value;
  int how_many;
};

BH_node BH[MAX_BUCKETS];

// Global variables
int bf_size;
int index = 0;

// dist measures the distance between bf1 and bf2
int dist(char *bf1, char *bf2)
{
  int i, result = 0;

  for(i=0; i<(int)strlen(bf1); i++)
```

```
    if(bf1[i] != bf2[i])
      result ++;

  return result;
}

// loafBH loads the bloom histogram of file <bhfile> into the BH
array
void loadBH(char *bhfile)
{
  FILE *infile;
  int pos = 0, freq, howmany;
  char bf[MAX_BF_SIZE];

  // Open bloom histogram
  if((infile = fopen(bhfile, "r")) == NULL)
  {
    printf("Error opening first bloom histogram file.
        Exiting...\n");
    exit(1);
  }

  // BH file in the form of <bloom filter, value,
  // how_many_paths_in_bucket>
  while(fscanf(infile, "%s %d %d", &bf, &freq, &howmany) != EOF)
  {
    if(pos == 0)
      bf_size = (int)strlen(bf);

    strcpy(BH[pos].bf, bf);

    BH[pos].freq_value = freq;
    BH[pos].how_many = howmany;

    pos++;
  }

  index = pos;
  fclose(infile);
}
```

```
// mergeBH merges the BH in the <bhfile> with the one stored in the
// BH array
void mergeBH(char *bhfile)
{
  FILE *infile;
  int pos = 0, freq, howmany, i;
  int max, target, sim;
  char bf[MAX_BF_SIZE];

  // Open bloom histogram
  if((infile = fopen(bhfile, "r")) == NULL)
  {
    printf("Error opening second bloom histogram file.
        Exiting...\n");
    exit(1);
  }

  while(fscanf(infile, "%s %d %d", &bf, &freq, &howmany) != EOF)
  {
    max = -INF;
    target = 0;

    for(i=0; i<index; i++)
    {
      sim = bf_size - dist(BH[i].bf, bf);

      if(sim > max)
      {
        max = sim;
        target = i;
      }
    }

    // Implementing MergeBuckets()
    for(i=0; i<bf_size; i++)
    {
      if(bf[i] == '0' && BH[target].bf[i] == '0')
        BH[target].bf[i] = '0';
      else
        BH[target].bf[i] = '1';
    }
```

```
    if(plus == true)
      BH[target].freq_value = (BH[target].how_many *
      BH[target].freq_value + howmany * freq) /
      (BH[target].how_many + howmany);
    else
      BH[target].freq_value = (BH[target].freq_value + freq)/2;

    BH[target].how_many += howmany;
  }

  fclose(infile);
}

// combine merges buckets which have the same bloom filter
void combine()
{
  int i, j, d;

  for(i=0; i<index; i++)
    for(j=i+1; j<index; j++)
    {
      d = dist(BH[i].bf, BH[j].bf);

      if(d == 0 && BH[i].freq_value != 0)
      {
        BH[i].freq_value += BH[j].freq_value;
        BH[i].how_many += BH[j].how_many;
        BH[j].freq_value = 0;
      }
    }
}

// printBH prints the BH array into file <filename>
void printBH(char *filename)
{
  FILE *outfile;
  int i;

  // Create bloom histogram file
  if((outfile = fopen(filename, "w")) == NULL)
```

```
  {
    printf("Error creating new bloom histogram file.
        Exiting...\n");
    exit(1);
  }

  for(i=0; i<index; i++)
    if(BH[i].freq_value != 0)
      fprintf(outfile, "%s\t%d %d\n", BH[i].bf,
          BH[i].freq_value, BH[i].how_many);
}

int main(int argc, char **argv)
{
  if(argc != 5)
  {
    printf("Usage: merge_bh <bh1> <bh2> <new bh> <plus mode
        on/off>. Try again.\n");
    exit(1);
  }

  if(strcmp(argv[4], "on") == 0)
    plus = true;

  // Load <bh1>
  loadBH(argv[1]);

  // Merge <bh2> into what we have in memory (<bh1>)
  mergeBH(argv[2]);

  // if(plus == true)
  //  combine();

  // Print the new bloom histogram
  printBH(argv[3]);

  return 0;
}
```

7. bh_dist

Είσοδος:	Τα αρχεία <bh1> και <bh2> με τα δύο ιστογράμματα bloom των οποίων την ομοιότητα θέλουμε να υπολογίσουμε.
Λειτουργία:	Υπολογισμός της ομοιότητας δύο ιστογραμμάτων bloom.
Έξοδος:	Η απόσταση των δύο ιστογραμμάτων bloom.
Τρόπος κλήσης:	Bh_dist <bh1> <bh2>

```
#include "stdafx.h"

#define MAX_BF_SIZE 2048
#define MAX_BUCKETS 2000
#define INF 1000000

// Bloom Histogram
struct BH_node
{
  char bf[MAX_BF_SIZE];
  int freq_value;
  bool used;
};

BH_node BH2[MAX_BUCKETS];

// Global variables
int bf_size;
int index = 0;

// dist measures the distance between bf1 and bf2
int dist(char *bf1, char *bf2)
{
  int i, result = 0;

  for(i=0; i<(int)strlen(bf1); i++)
    if(bf1[i] != bf2[i])
      result ++;

  return result;
```

```
}

// loafBH loads the bloom histogram of file <bhfile> into the BH
// array
void loadBH(char *bhfile)
{
  FILE *infile;
  int pos = 0, freq;
  char bf[MAX_BF_SIZE];

  // Open bloom histogram
  if((infile = fopen(bhfile, "r")) == NULL)
  {
    printf("Error opening second bloom histogram file.
        Exiting...\n");
    exit(1);
  }

  // BH file in the form of <bloom filter, value>
  while(fscanf(infile, "%s %d", &bf, &freq) != EOF)
  {
    if(pos == 0)
      bf_size = (int)strlen(bf);

    strcpy(BH2[pos].bf, bf);

    BH2[pos].freq_value = freq;
    BH2[pos].used = false;

    pos++;
  }

  index = pos;
  fclose(infile);
}

// distBH measures the distance between the BH in the <bhfile> with
// the one stored in the BH2 array
int distBH(char *bhfile)
{
  FILE *infile;
```

```
  int pos = 0, freq, i;
  int min, distance, total_dist = 0;
  char bf[MAX_BF_SIZE];

  // Open bloom histogram
  if((infile = fopen(bhfile, "r")) == NULL)
  {
    printf("Error opening first bloom histogram file.
        Exiting...\n");
    exit(1);
  }

  while(fscanf(infile, "%s %d", &bf, &freq) != EOF)
  {
    min = INF;

    for(i=0; i<index; i++)
    {
      distance = dist(BH2[i].bf, bf);

      if(distance < min && BH2[i].used == false)
      {
        min = distance;
        BH2[i].used = true;
      }
    }

    total_dist += min;
  }

  fclose(infile);
  return total_dist;
}

int main(int argc, char **argv)
{
  int result;

  if(argc != 3)
  {
    printf("Usage: dist_bh <bh1> <bh2>. Try again.\n");
```

```
    exit(1);
  }

  // Load <bh2>
  loadBH(argv[2]);

  // Measure the distance between <bh1> and what we have in memory
  // (<bh2>)
  result = distBH(argv[1]);

  // Print the result
  printf("Distance = %d\n", result);

  return 0;
}
```

Παράρτημα Β

Απόδειξη Σχέσης 2.5

$$PSUM[y]+PSUM[x-1]-2PSUM[t]+(2t-x-y+1)count[t]=$$

$$\begin{aligned}&=count[1]+count[2]+...+count[x-1]+...+count[y])\\&\quad+(count[1]+count[2]+...+count[x-1])\\&\quad-2(count[1]+count[2]+...+count[x-1]+...+count[t])+(2t-x-y+1)count[t]=\end{aligned}$$

$$\begin{aligned}&=(count[x]+...+count[t]+...+count[y])\\&\quad-2(count[x]+...+count[t])+(2t-x-y+1)count[t]=\end{aligned}$$

$$=\{-count[x]-...-count[t-1]-count[t]\}+\{count[t+1]+...+count[y])\}+(2t-x-y+1)count[t]$$

Άρα έχουμε $(t-x+1)$ όρους της μορφής $-count[i]$ και $(y-t)$ όρους της μορφής $count[i]$.

Ακόμα είναι: $(2t-x-y+1)count[t]=(t-x+1)count[t]+(y-t)(-count[t])$

Άρα σε κάθε $-count[i]$ με $x<i<t$ αντιστοιχεί ένα $count[t]$ και σε κάθε $count[i]$ με $t<i<y$ αντιστοιχεί ένα $-count[t]$.

Επομένως η αρχική παράσταση είναι ίση με $\sum_{i=x}^{y}\left|count[i]-count[t]\right|$ που είναι και το σφάλμα του κάδου.