
**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**


**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Αναγνωρίζοντας σχεδόν-Διπλότυπα Αρχεία


**Ιωάννης Χ. Ρέππας**

**Επιβλέπων:** **Δημήτριος Γουνόπουλος,** Καθηγητής



**ΑΘΗΝΑ**

**Οκτώβριος 2020**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

Αναγνωρίζοντας σχεδόν-Διπλότυπα Αρχεία

**Ιωάννης Χ. Ρέππας**
**Α.Μ.:** 1115201500137

**ΕΠΙΒΛΕΠΟΝΤΕΣ:** **Δημήτριος Γουνόπουλος,** Καθηγητής

# ΠΕΡΙΛΗΨΗ

Σκοπός της εργασίας αποτελεί η αναγνώριση των διπλότυπων αρχείων. Με τον όρο «διπλότυπα αρχεία», εννοούμε αρχεία τα οποία έχουν μεγάλο βαθμό ομοιότητας. Η μαθηματική προσέγγιση του προβλήματος περιλαμβάνει έναν αλγόριθμο για τον υπολογισμό των διπλότυπων αρχείων, ο οποίος προσεγγίζει αρκετά καλά το επιθυμητό αποτέλεσμα, σε σχέση με άλλους, πιο απλούς αλγορίθμους. Πιο συγκεκριμένα, υπολογίζεται ένα “sketch” για κάθε αρχείο. Με τον όρο sketch, εννοούμε ένα μικρό υποσύνολο δεδομένων του αρχικού αρχείου, το οποίο δεν υπερβαίνει μερικά εκατοντάδες bytes σε μέγεθος. Για τον υπολογισμό της ομοιότητας μεταξύ των δύο αρχείων, χρησιμοποιούνται τα δύο sketch τους. Η ομοιότητα (resemblance) εκφράζεται ως ένας αριθμός από το 0 μέχρι το 1. Όσο πιο κοντά στο 1 είναι το αποτέλεσμα, τόσο πιο «όμοια» είναι μεταξύ τους τα αρχεία. Με βάση τα αποτελέσματα αρκετών εκτελέσεων που έγιναν για τον παραπάνω αλγόριθμο, επιβεβαιώθηκε η ορθότητά του.

Ο αλγόριθμος της σύγκρισης και ανίχνευσης όμοιων αρχείων έχει εφαρμογές σε διάφορες περιπτώσεις, από την απλή σύγκριση δύο αρχείων μέχρι και το φιλτράρισμα όμοιων αρχείων για την προβολή μη-όμοιων ιστοσελίδων από τις μηχανές αναζήτησης, έτσι ώστε να προσφέρεται ποικιλία αποτελεσμάτων στους χρήστες. Πιο συγκεκριμένα, παρόμοιος αλγόριθμος (στον οποίο έχει βασιστεί και η πτυχιακή εργασία) έχει υλοποιηθεί και χρησιμοποιηθεί από τη μηχανή αναζήτησης AltaVista.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ**: Σύγκριση αρχείων

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ**: ομοιότητα, σύγκριση, shingling, μεταθέσεις, sketch

# ABSTRACT

The aim of this thesis was to Identify near-duplicate documents. Near-duplicate documents are documents that resemble each other. The mathematical concept of document resemblance captures well the informal notion of syntactic similarity. Specifically, a fixed size sketch per document is calculated. A fixed size sketch is a small subset of the starting document, with a size of a few hundred bytes. The sketches of two documents are used for the estimation of their resemblance. Resemblance is expressed with a number ranging from 0 to 1. The closer to 1, the more the two documents resemble each other. After performing a number of tests, it was concluded that the algorithm functions properly.

The algorithm for comparing two documents and estimating their resemblance is useful in various situations, e.g. for just comparing two documents, and for filtering duplicate results in search engines so that users get a variety of different results for more useful information. A similar algorithm to what was implemented in this thesis, has been developed and used by AltaVista search engine.



**SUBJECT AREA**: File Comparing

**KEYWORDS**: resemblance, comparison, shingling, permutations, sketch

# ΠΕΡΙΕΧΟΜΕΝΑ


**1. ΠΕΡΙΓΡΑΦΗ ΘΕΜΑΤΟΣ ..... 10**

**1.1 Εισαγωγή ..... 10**

**1.2 Στάδιο Α – Αντιστοίχιση συνόλου συμβολοσειρών ..... 11**

1.2.1 Tokens ..... 11

1.2.2 Shingling ..... 11

1.2.3 Fingerprints ..... 12

**1.3 Στάδιο Β – Επιλογή στοιχείων από τυχαία σύνολα ..... 13**

1.3.1 Δημιουργία Τυχαίων μεταθέσεων ..... 13

1.3.2 Δημιουργία sketch ..... 14

1.3.3 Απόδειξη αλγορίθμου ..... 14

1.3.4 Σχηματική αναπαράσταση αλγορίθμου ..... 15

1.3.5 Πολυπλοκότητα αλγορίθμου ..... 16

**2. ΠΕΡΙΓΡΑΦΗ ΥΛΟΠΟΙΗΣΗΣ ..... 18**

**2.1 Περιγραφή συνάρτησης main ..... 18**

**2.2 Περιγραφή συνάρτησης resemblance ..... 19**

**2.3 Περιγραφή συνάρτησης pairs ..... 20**

**3. TESTING ..... 23**

**3.1 Εισαγωγή ..... 23**

**3.2 Έλεγχος ορθότητας του αλγορίθμου ..... 23**

**3.3 Έλεγχος για διαφορετικές τιμές των αρχικών παραμέτρων ..... 32**

3.3.1 Μελέτη παραμέτρου w ..... 32

3.3.2 Μελέτη παραμέτρου t ..... 36

**4. ΣΥΜΠΕΡΑΣΜΑΤΑ .....40**

**ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ .....41**

**ΠΑΡΑΡΤΗΜΑ Ι.....42**

**ΠΑΡΑΡΤΗΜΑ ΙΙ .....43**

**ΑΝΑΦΟΡΕΣ.....44**

# ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

Σχήμα 1: ..... **Error! Bookmark not defined.**

# ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

Πίνακας 1: Pairs.....22

Πίνακας 2: Εκτέλεση 3.2α.....25

Πίνακας 3: Εκτέλεση 3.2β.....26

Πίνακας 4: Εκτέλεση 3.2γ.....27

Πίνακας 5: Εκτέλεση 3.2δ.....28

Πίνακας 6: Μέσοι όροι αποτελεσμάτων.....30

Πίνακας 7: Εκτέλεση 3.3α.....33

Πίνακας 8: Εκτέλεση 3.3β.....33

Πίνακας 9: Εκτέλεση 3.3γ.....34

Πίνακας 10: Εκτέλεση 3.3δ.....34

Πίνακας 11: Εκτέλεση 3.3ε.....35

Πίνακας 12: Εκτέλεση 3.3στ.....37

Πίνακας 13: Εκτέλεση 3.3ζ.....37

Πίνακας 14: Εκτέλεση 3.3η.....38

Πίνακας 15: Εκτέλεση 3.3θ.....39

# ΠΡΟΛΟΓΟΣ

Μετά από αρκετή προσπάθεια, έφτασε η στιγμή να γραφούν αυτές οι πιο «ανάλαφρες» γραμμές, σηματοδοτώντας την ολοκλήρωση του πρώτου κύκλου σπουδών μου στο Τμήμα Πληροφορική και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών.

Θα ήθελα να ευχαριστήσω τον Καθηγητή Δημήτρη Γουνόπουλο για την ανάθεση του θέματος, την εμπιστοσύνη στο πρόσωπό μου, την καθοδήγηση και τις συμβουλές.

Επίσης, θα ήθελα να ευχαριστήσω όλους τους δικούς μου ανθρώπους που με έχουν στηρίξει στη διάρκεια των σπουδών μου μέχρι σήμερα και κυρίως τον πατέρα μου, τη μητέρα μου και την αδελφή μου.

Αθήνα, Οκτώβριος 2020

# 1. ΠΕΡΙΓΡΑΦΗ ΘΕΜΑΤΟΣ

## 1.1 Εισαγωγή

Ο Παγκόσμιος Ιστός περιλαμβάνει τρομακτικά μεγέθη δεδομένων και όσο ζούμε αυτά τα μεγέθη συνεχίζουν και αυξάνονται. Δεν μπορούμε να αγνοήσουμε το γεγονός ότι πλέον όχι οι μόνο οι υπολογιστές και τα κινητά τηλέφωνα, αλλά και άλλες συσκευές, όπως ψυγεία και αυτοκίνητα μπορούν να συνδέονται στο διαδίκτυο και να ανταλλάσσουν μεταξύ τους πληροφορίες. Είναι φυσικό λοιπόν, μέσα σε αυτό το πολύπλοκο περιβάλλον από αρχεία και δεδομένα να συναντάμε πολλές πληροφορίες παραπάνω από μία φορά. Πολλές μηχανές αναζήτησης λοιπόν, όπως για παράδειγμα η AltaVista, θέλησαν να ασχοληθούν με το συγκεκριμένο θέμα, όχι μόνο από ερευνητικό ενδιαφέρον, αλλά και για να μπορούν να παρουσιάζουν αποτελέσματα στους χρήστες, τα οποία δεν επαναλαμβάνονται. Για να αναλογιστεί κάποιος την έκταση αυτού του προβλήματος, δεν έχει παρά να σκεφτεί ότι 30- 45% όλων των αρχείων στον Παγκόσμιο Ιστό, κατά πάσα πιθανότητα αποτελεί διπλότυπη πληροφορία ή τουλάχιστον παρόμοια [1,2]. Σκοπός της εργασίας είναι να μελετήσει τον τρόπο με τον οποίο μπορούμε να μελετήσουμε την ομοιότητα μεταξύ δύο αρχείων.

Η Πτυχιακή Εργασία βασίζεται πάνω στο άρθρο του Andrei Z. Broder με τίτλο “Identifying and Filtering Near-Duplicate Documents” [3]. Το άρθρο αυτό περιγράφει και αναλύει το πρόβλημα της σύγκρισης δύο αρχείων, αλλά και το πώς μπορούμε με έναν αποδοτικό τρόπο να αποφανθούμε κατά το πόσο δύο αρχεία είναι όμοια μεταξύ τους. Η προσέγγιση του Broder για την κατασκευή ενός τέτοιου αλγορίθμου περιλαμβάνει δύο βασικά στάδια: Το πρώτο στάδιο αποτελεί ένα πρόβλημα αντιστοίχισης ενός συνόλου συμβολοσειρών, ενώ το δεύτερο στάδιο αποτελεί ένα πρόβλημα τυχαίας επιλογής στοιχείων από τις αντιστοιχίσεις που έχουν

γίνει στο πρώτο στάδιο. Τόσο στον τρόπο επιλογής αυτών των στοιχείων, όσο και στον τρόπο με τον οποίο δημιουργούνται οι αντιστοιχίσεις, μπορεί να υπάρξει μεγάλη έρευνα και πειραματισμός.

## 1.2 Στάδιο Α – Αντιστοίχιση συνόλου συμβολοσειρών

Αρχικά λοιπόν, η μετάβαση από ένα απλό document σε ένα πρόβλημα αντιστοίχισης ενός συνόλου συμβολοσειρών, αποτελεί μία διαδικασία με όνομα shingling. Μέσω του shingling, κάθε αρχείο D μετατρέπεται σε ένα σύνολο $S_D$ από shingles. Αυτό συμβαίνει με την εξής διαδικασία:

### 1.2.1 Tokens

Αντιμετωπίζουμε κάθε αρχείο σαν ένα σύνολο από tokens. Μπορούμε να θεωρήσουμε ότι τα tokens είναι λέξεις, προτάσεις ή ακόμα και γράμματα. Θεωρούμε ότι έχουμε στη διάθεσή μας έναν parser, ο οποίος αναλαμβάνει τον «καθαρισμό δεδομένων» για το αρχείο. Με τον καθαρισμό δεδομένων, αγνοούμε πολλά δεδομένα του αρχείου όπως κεφαλαία γράμματα και σημεία στίξης. Με αυτόν τον τρόπο μειώνουμε το σύνολο των διαφορετικών tokens στο αρχείο, χωρίς όμως να χάνεται κάποια πληροφορία που είναι απαραίτητη για τη σύγκριση των 2 αρχείων.

### 1.2.2 Shingling

Μια συνεχής ακολουθία από w tokens σε ένα αρχείο D, ονομάζεται shingle. Ένα shingle μεγέθους q , ονομάζεται και q-gram (ειδικότερα όταν τα tokens είναι γράμματα του αλφάβητου). Για ένα αρχείο D, ορίζουμε το w-shingling του ως την ακολουθία όλων των w-shingles που υπάρχουν μέσα στο D. Για παράδειγμα, το 4-shingling για το σύνολο από tokens

{a,rose,is,a,rose,is,a,rose}

είναι

{(a,rose,is,a),(rose,is,a,rose),(is,a,rose,is)}

Με τη μετατροπή του απλού αρχείου σε ένα σύνολο από shingles, καταφέρνουμε να αντιμετωπίζουμε το αρχείο με τη βοήθεια κάποιας βασικής οντότητας. Επιπλέον, είμαστε πιο σίγουροι για τα επόμενα βήματα που θα κάνουμε, μιας και επί της ουσίας μπορούμε και έχουμε στην κατοχή μας ένα μέρος του αρχείου και να μπορούμε να συγκρίνουμε όχι μόνο ένα token με ένα άλλο token κάποιου άλλου αρχείου, αλλά να έχουμε πρόσβαση και στις προηγούμενες/επόμενες καταστάσεις τους (π.χ. προηγούμενες/επόμενες λέξεις). Παρ' όλα αυτά, το μέγεθος από δεδομένα που καταλαμβάνει ένα shingle στη μνήμη του υπολογιστή μπορεί να αποτελέσει πρόβλημα. Πέραν της μνήμης που καταλαμβάνεται από κάθε shingle, η διαχείρισή τους από τον υπολογιστή μπορεί να είναι εξίσου δύσκολη.

### 1.2.3 Fingerprints

Τα παραπάνω προβλήματα μπορούν να αντιμετωπιστούν χρησιμοποιώντας τα fingerprints. Τα fingerprints, όπως υποδηλώνει και η ίδια η λέξη, κωδικοποιούν το shingle σε έναν ακέραιο αριθμό, έτσι ώστε η τιμή του αριθμού αυτού να είναι (σχεδόν πάντα) ξεχωριστή για το συγκεκριμένο shingle. Εάν δύο fingerprints είναι διαφορετικά, τότε οπωσδήποτε τα δύο shingles θα είναι διαφορετικά και η πιθανότητα δύο διαφορετικά shingles να παράγουν το ίδιο fingerprint είναι εξαιρετικά μικρή. Με αυτόν τον τρόπο, η διαχείριση των shingles γίνεται πιο εύκολη, αλλά και ο χώρος που καταλαμβάνει το αρχείο είναι μικρότερος στη μνήμη.

Σε αυτό το σημείο λοιπόν θεωρητικά, μπορούμε να υπολογίσουμε την ομοιότητα μεταξύ δύο αρχείων A και B, όπου το σύνολο των shingle τους είναι αντίστοιχα $S_A$ και $S_B$, με τον ακόλουθο τύπο:

$$r(A, B) = \frac{|SA \cap SB|}{|SA \cup SB|} \qquad (1)$$

### 1.3 Στάδιο Β – Επιλογή στοιχείων από τυχαία σύνολα

Δυστυχώς όμως και εδώ αντιμετωπίζουμε ένα παρόμοιο πρόβλημα με αυτό που είδαμε και στα fingerprints. Το μέγεθος το αρχείου και συνεπώς των συνόλων $S_A$ και $S_B$ μπορεί να είναι τόσο μεγάλο που αυτό θα έχει ως αποτέλεσμα οι πράξεις της ένωσης και της τομής να είναι εξαιρετικά χρονοβόρες. Επιπλέον, πρέπει να λάβουμε υπόψη μας και το γεγονός ότι ίσως να μην έχουμε πάντα να ασχοληθούμε μόνο με 2 αρχεία, αλλά και με πολλά περισσότερα, σε περίπτωση που θέλουμε να εφαρμόσουμε αλγορίθμους ομαδοποίησης.

Συνεπώς, χρειαζόμαστε έναν γρήγορο τρόπο, ο οποίος ταυτόχρονα να μην απαιτεί πολλή μνήμη από τον υπολογιστή. Όπως αναφέρθηκε και στην Εισαγωγή, για τον υπολογισμό της ομοιότητας 2 αρχείων, αποθηκεύουμε για κάθε αρχείο από ένα sketch. Τα sketches μπορούν να υπολογιστούν σχετικά γρήγορα. Πιο συγκεκριμένα, κάθε sketch απαιτεί για να υπολογιστεί γραμμικό χρόνο ως προς το μέγεθος του αρχείου από το οποίο παράγεται. Επίσης, η ομοιότητα 2 αρχείων απαιτεί γραμμικό χρόνο ως προς το μέγεθος των 2 sketches για να υπολογιστεί.

#### 1.3.1 Δημιουργία Τυχαίων μεταθέσεων

Το παραπάνω στάδιο υλοποιείται ως εξής. Αποθηκεύουμε όλα τα shingles για κάθε αρχείο που έχουμε στη διάθεσή μας σε ένα dataset. Εάν έχουμε επιλέξει να αναπαραστήσουμε κάθε shingle σαν fingerprint, μπορούμε και να θεωρήσουμε εξ αρχής ότι το dataset αποτελείται από όλες τις τιμές που μπορεί να πάρει το κάθε fingerprint(ανάλογα με τον

αλγόριθμο που έχουμε επιλέξει να εφαρμόζουμε). Για παράδειγμα, εάν επιλέξουμε να εφαρμόσουμε τον αλγόριθμο των Rabin fingerprints [4], το dataset θα περιέχει τις τιμές {0,1,…,$2^{64}$ - 1}. Από το dataset παράγουμε ένα σύνολο από t τυχαίες μεταθέσεις (random permutations) των στοιχείων του.

### 1.3.2 Δημιουργία sketch

Έτσι λοιπόν, έχουμε στην κατοχή μας N σύνολα από shingles για N αρχεία (file shingles) και t τυχαίες μεταθέσεις όλων των shingles για όλα τα αρχεία (random permutations). Για να βρούμε το sketch ενός αρχείου, ξεκινάμε να διατρέχουμε με έναν αλγόριθμο min-max όλο το file shingle του αρχείου. Στόχος μας είναι να βρούμε το στοιχείο του file shingle το οποίο συναντάται πρώτο στην τυχαία μετάθεση την οποία εξετάζουμε. Επαναλαμβάνουμε αυτό το βήμα για όλες τις τυχαίες μεταθέσεις τις οποίες έχουμε ορίσει (t στο σύνολο) και αποθηκεύουμε το index της μικρότερης τιμής για το κάθε random permutation σε ένα νέο σύνολο, το οποίο είναι και το sketch του αρχείου.

### 1.3.3 Απόδειξη αλγορίθμου

Από τη στιγμή που έχουμε να κάνουμε με τυχαίες μεταθέσεις του αρχικού dataset, για μία τυχαία μετάθεση π και ένα σύνολο X με τα shingles ενός αρχείου, ισχύει ότι:

$$Pr(\min\{\pi(X)\} = \pi(X)) = 1 / |X|,$$

Που σημαίνει ότι κάθε στοιχείο x του συνόλου X, έχει τις ίδιες πιθανότητες να παρουσιαστεί πρώτο από τα υπόλοιπα στην τυχαία μετάθεση π.

Έστω α το στοιχείο που συναντάται πρώτο στο π από το σύνολο $S_A \cup S_B$.

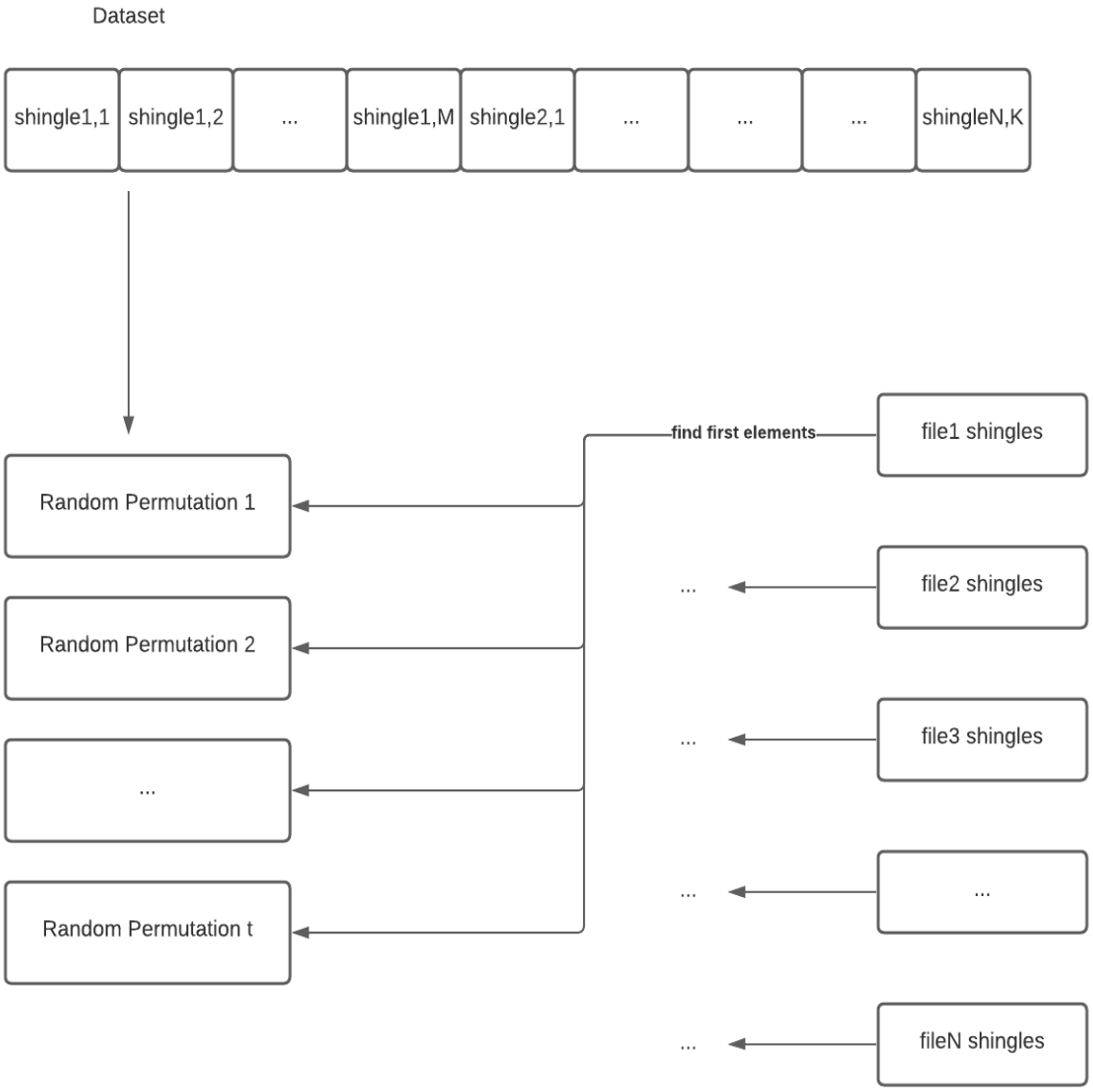
Τότε,το στοιχείο του $S_A$ που συναντάμε πρώτο στο π είναι ίδιο με το

στοιχείο του $S_B$ που συναντάμε πρώτο στο π, αν και μόνο αν το α υπάρχει στο σύνολο $S_A \cap S_B$ . Για να πάρουμε την πιθανότητα του στοιχειου α να υπάρχει στο σύνολο $S_A \cap S_B$, μπορούμε απλά να εφαρμόσουμε τον τύπο (1). Οπότε, έχουμε ότι:

$$\Pr(\min\{\pi(S_A)\} = \min\{\pi(S_B)\}) = \frac{|SA \cap SB|}{|SA \cup SB|} = \text{resemblance}(A,B) \qquad (2)$$

### 1.3.4 Σχηματική αναπαράσταση αλγορίθμου

Η διαδικασία παρουσιάζεται στο Σχήμα 1. Ως shingle 1,1 αναφερόμαστε στο πρώτο shingle του πρώτου αρχείου, ως shingle 2,1 στο πρώτο shingle του δεύτερου αρχείου κ.ο.κ. Για κάθε αρχείο παράγεται και ένα sketch από ακεραίους, μεγέθους t.




**Σχήμα 1:**

Τέλος, αφού έχουμε στην κατοχή μας τα sketches κάθε αρχείου, εφαρμόζοντας τον τύπο 1, παίρνουμε και το τελικό αποτέλεσμα του resemblance μεταξύ 2 αρχείων.

### 1.3.5 Πολυπλοκότητα αλγορίθμου

Με αυτόν τον τρόπο λοιπόν, υπολογίζουμε το βαθμό ομοιότητας μεταξύ 2 αρχείων. Προκειμένου να υπολογίσουμε τα sketches των αρχείων

σε γραμμικό χρόνο ως προς το μέγεθος του file shingle (όπως αναφέρθηκε και πριν) δεν έχουμε παρά να μετατρέψουμε την κάθε τυχαία μετάθεση από πίνακα ή λίστα σε ένα hash table, με κλειδί το shingle και τιμή το index του shingle στην τυχαία μετάθεση. Μετά από αυτό, η αναζήτηση ενός shingle από το σύνολο των file shingles σε αυτό ενός random permutation θα απαιτεί O(1) χρόνο. Κάπως έτσι δικαιολογούνται και το γεγονός ότι ο υπολογισμός ενός shingle απαιτεί γραμμικό χρόνο ως προς το μέγεθος του αρχείου. Πιο συγκεκριμένα:

$O(1 \cdot t \cdot N) = O(N)$, όπου N το μέγεθος του αρχείου

# 2. ΠΕΡΙΓΡΑΦΗ ΥΛΟΠΟΙΗΣΗΣ

Στόχος της υλοποίησης είναι η κατασκευή αλγορίθμου σύγκρισης αρχείων, με βάση όσα αναφέρθηκαν στην Περιγραφή Θέματος. Το πρόγραμμα δέχεται σαν είσοδο ένα φάκελο για όλα τα αρχεία που θα συμπεριληφθούν στο dataset, καθώς και τον αριθμό από τις τυχαίες μεταθέσεις και το μέγεθος του κάθε shingle. Το nearDuplicate.py (πρόγραμμα που υλοποιήθηκε) γράφτηκε σε python, έκδοση 2.7.13 . Ένα παράδειγμα της εκτέλεσης θα μπορούσε να είναι:

$ python nearDuplicate.py -w 5 -t 100 -d ./my_directory

, όπου:

w → shingle size

t → number of random permutations

d → directory with files of dataset

## 2.1 Περιγραφή συνάρτησης main

Ακολουθεί ψευδοκώδικας για τη main συνάρτηση του nearDuplicate.py:

```
w,t,filenames = commandLineCheck()
dataset, fileShingles = createDataset(w,filenames)
randomPermutations = createRandomPermutations(t, dataset)
hashTables = createHashTables(randomPermutations)
#########################################################
If userInput = '/resemblance filename1 filename2'
    resemblance(filename1, filename2, fileShingles, hashTables)
If userInput = '/pairs'
    pairs(filenames, fileShingles, hashTables)
```

Η εκτέλεση του προγράμματος ξεκινάει από τη γραμμή 226. Αρχικά αποθηκεύουμε σε 3 μεταβλητές τα στοιχεία που έδωσε ο χρήστης κατά την εκτέλεση του προγράμματος (μήκος shingle, μέγεθος τυχαίων μεταθέσεων, ονόματα αρχείων) – συνάρτηση commandLineCheck. Σε περίπτωση λάθος εισόδου, εμφανίζεται το κατάλληλο μήνυμα λάθους. Στη συνέχεια, προχωράμε στη δημιουργία των file shingles και του dataset – συνάρτηση createDataset. Αφού λοιπόν δημιουργούμε το dataset, παράγουμε τις τυχαίες μεταθέσεις του – συνάρτηση createRandomPermutations. Οι τυχαίες μεταθέσεις παράγονται με τη βοήθεια της βιβλιοθήκης numpy (συνάρτηση random.permutation()). Στη συνέχεια, για να μετατρέψουμε τις τυχαίες μεταθέσεις σε hash tables, καλούμε τη συνάρτηση createHashTables, η οποία όπως αναφέρθηκε και πριν, τοποθετεί σε ένα hash table όλα τα shingles του dataset σαν κλειδιά, με την τιμή για το κάθε κλειδί να είναι το location/index του στην τυχαία μετάθεση σε μορφή λίστας. Μετά από όλα τα παραπάνω είμαστε έτοιμοι να περάσουμε στους υπολογισμούς ομοιότητας αρχείων.

## 2.2 Περιγραφή συνάρτησης resemblance

Στο σημείο αυτό, εμφανίζεται στο χρήστη ένα menu, στο οποίο έχει τρεις επιλογές: α) υπολογισμό ομοιότητας μεταξύ δύο αρχείων, β) υπολογισμό ζευγαριών αρχείων, ανάλογα με τη μεγαλύτερη ομοιότητα που παρουσιάζει το κάθε αρχείο με κάποιο άλλο και γ) έξοδος από το πρόγραμμα.

Αρχικά, όσον αφορά τον υπολογισμό ομοιότητας μεταξύ δύο αρχείων (επιλογή Α), ο χρήστης πληκτρολογεί /resemblance filename1 filename2, όπου filename1 και filename2 τα ονόματα των δύο αρχείων που απαιτείται να υπολογιστεί η ομοιότητά τους. Καλώντας τη συνάρτηση resemblance, υπολογίζονται πρώτα τα sketches των δύο αρχείων και στη συνέχεια,

χρησιμοποιώντας τον τύπο (1), εμφανίζουμε το αποτέλεσμα της ομοιότητας των 2 αρχείων.

Ακολουθεί ψευδοκώδικας για τη συνάρτηση resemblance:

```
for i in randomPermutations:
    currentIndex = 1
    currentMin = i[ fileShingles[filename1][ 0 ] ]
    while currentIndex < size( fileShingles[filename1] )
        if currentMin > i[ fileShingles[filename1][ currentIndex ] ]
            currentMin = i[ fileShingles[filename1][ currentIndex ] ]
        currentIndex += 1
    sketch1.append(currentMin)
    ##################################################################
    currentIndex = 1
    currentMin = i[ fileShingles[filename2][ 0 ] ]
    while currentIndex < size( fileShingles[filename2] )
        if currentMin > i[ fileShingles[filename2][ currentIndex ] ]
            currentMin = i[ fileShingles[filename2][ currentIndex ] ]
        currentIndex += 1
    sketch2.append(currentMin)
resemblance(sketch1,sketch2) // τύπος (1)
```

## 2.3 Περιγραφή συνάρτησης pairs

Όσον αφορά τον υπολογισμό των ζευγαριών (επιλογή β), ο χρήστης πληκτρολογεί /pairs. Στην αρχή, υπολογίζονται τα sketches όλων των αρχείων. Αφού έχουμε στην κατοχή μας όλα τα skteches, υπολογίζουμε τα resemblances μεταξύ όλων των αρχείων. Για να πετύχουμε κάτι τέτοιο,

χρησιμοποιούμε έναν δυσδιάστατο πίνακα διαστάσεων filesNum X filesNum, όπου filesNum ο αριθμός των αρχείων που συμπεριλάβαμε στο dataset. Στη συνέχεια, για κάθε αρχείο A εφαρμόζουμε τον τύπο (1) μεταξύ του αρχείου A και οποιουδήποτε άλλου αρχείου, με εξαίρεση φυσικά τον εαυτό του (αρχείο A). Επαναλαμβάνοντας τη διαδικασία για κάθε αρχείο, καταλήγουμε να έχουμε έναν δισδιάστατο άνω τριγωνικό πίνακα, μιας και η ομοιότητα μεταξύ ενός αρχείου A και B υπολογίζεται στο δισδιάστατο πίνακα 2 φορές. Φυσικά, πριν τον υπολογισμό του resemblance, γίνεται ο απαραίτητος έλεγχος έτσι ώστε να μην γίνονται οι υπολογισμοί 2 φορές.

Ακολουθεί ψευδοκώδικας για τη συνάρτηση pairs:

```
allSketches = calculateAllSketches(filenames, fileShingles, hashTables)
pairs = zeros[filesNum][filesNum]
put_ones_in_pairs_diagonal()
for i = 0 until i == filesNum, i += 1
    for j = 0 until j = filesNum, j += 1
        if (pairs[i][j] == 0)
            resemblance(allSketches[filenames[i]], allSketches[filenames[j]])
            pairs[j][i] = pairs[i][j]
return max_of_each_row()
```

**Πίνακας 1: Pairs**

| | **File1** | **File2** | **File3** | | **FileN** |
|---|---|---|---|---|---|
| **File1** | 1 | r(1,2) | r(1,3) | … | r(1,N) |
| **File2** | r(1,2) | 1 | r(2,3) | … | r(2,N) |
| **File3** | r(1,3) | r(2,3) | 1 | … | r(3,N) |
| **…** | … | … | … | 1 | … |
| **FileN** | r(1,N) | r(2,N) | | … | 1 |

# 3. TESTING

## 3.1 Εισαγωγή

Η υλοποίηση που περιεγράφηκε παραπάνω ελέγχθηκε ως προς τη λειτουργικότητά της χρησιμοποιώντας πιθανές παραλλαγές ενός αρχείου κειμένου. Το αρχείο ονομάζεται b12.txt, πάρθηκε από την ιστοσελίδα test files [5] και η θεματολογία του έχει να κάνει με γενικές γνώσεις σχετικά με τη βιταμίνη b12 και την επιρροή της στον οργανισμό μας.

Οι παραλλαγές του αρχείου είναι τα αρχεία b12_10.txt, b12_20.txt, b12_30.txt, b12_50.txt και b12_70.txt, τα οποία περιλαμβάνουν το αρχικό αρχείο, αποκομμένο κατά 10%, 20%, 30%, 50% και 70%, αντίστοιχα.

Προκειμένου να εμπλουτίσουμε με περισσότερα δεδομένα το αρχικό μας dataset, χρησιμοποιούμε επιπλέον αρχεία, τα οποία πάρθηκαν επίσης από την ιστοσελίδα test files. Τα αρχεία δεν έχουν κάποια ιδιαίτερη ομοιότητα μεταξύ τους (σχεδόν όλα διαφέρουν περισσότερο από 50% μεταξύ τους).

Η διαδικασία εκτέλεσης του αλγορίθμου και επαλήθευσης των αποτελεσμάτων περιλαμβάνει 2 στάδια: α) Εκτέλεση του αλγορίθμου για επαλήθευση ορθότητας της σύγκρισης 2 αρχείων β) Μελέτη συμπεριφοράς αλγορίθμου για διαφορετικές τιμές των αρχικών παραμέτρων (μέγεθος shingle, αριθμός τυχαίων μεταθέσεων, μέγεθος dataset).

## 3.2 Έλεγχος ορθότητας του αλγορίθμου

Αρχικά, εκτελούμε το nearDuplicate.py με την εξής εντολή:

python nearDuplicate.py -w 5 -t 100 -d ./food2

, που σημαίνει ότι εκτελούμε τον αλγόριθμο για μήκος shingle 5

, για αριθμό τυχαίων μεταθέσεων του dataset = 100

, παίρνοντας τα αρχεία από τα οποία θα δημιουργηθεί το dataset από το φάκελο food2

Εκτελούμε τον αλγόριθμο συνολικά 4 φορές, συγκρίνοντας τις παραλλαγές του αρχείου μεταξύ τους. Αυτό που αναμένουμε, είναι δούμε σε όλες τις εκτελέσεις το αποτέλεσμα της κάθε σύγκρισης να προσεγγίζει όσο το δυνατόν καλύτερα τη διαφορά μεταξύ των δύο αρχείων που έχουμε ορίσει. Για παράδειγμα, αναμένουμε τα αρχεία b12.txt και b12b.txt να έχουν ομοιότητα περίπου 0.9 .

**Πίνακας 2: Εκτέλεση 3.2α**

| File1 | File2 | Resemblance |
|---|---|---|
| b12.txt | b12_10.txt | 0.92 |
| b12.txt | b12_20.txt | 0.814814814815 |
| b12.txt | b12_30.txt | 0.733333333333 |
| b12.txt | b12_50.txt | 0.446808510636 |
| b12.txt | b12_70.txt | 0.323076923077 |
| b12_10.txt | b12_20.txt | 0.888888888889 |
| b12_10.txt | b12_30.txt | 0.741935483871 |
| b12_10.txt | b12_50.txt | 0.489361702128 |
| b12_10.txt | b12_70.txt | 0.353846153846 |
| b12_20.txt | b12_30.txt | 0.833333333333 |
| b12_20.txt | b12_50.txt | 0.510638297872 |
| b12_20.txt | b12_70.txt | 0.369230769231 |
| b12_30.txt | b12_50.txt | 0.574468085106 |
| b12_30.txt | b12_70.txt | 0.352941176471 |
| b12_50.txt | b12_70.txt | 0.521126760563 |

**Πίνακας 3: Εκτέλεση 3.2β**

| File1 | File2 | Resemblance |
|---|---|---|
| b12.txt | b12_10.txt | 0.733333333333 |
| b12.txt | b12_20.txt | 0.647058823529 |
| b12.txt | b12_30.txt | 0.564102564103 |
| b12.txt | b12_50.txt | 0.44 |
| b12.txt | b12_70.txt | 0.205479452055 |
| b12_10.txt | b12_20.txt | 0.714285714286 |
| b12_10.txt | b12_30.txt | 0.666666666667 |
| b12_10.txt | b12_50.txt | 0.52 |
| b12_10.txt | b12_70.txt | 0.260273972603 |
| b12_20.txt | b12_30.txt | 0.815789473684 |
| b12_20.txt | b12_50.txt | 0.509433962264 |
| b12_20.txt | b12_70.txt | 0.28 |
| b12_30.txt | b12_50.txt | 0.634615384615 |
| b12_30.txt | b12_70.txt | 0.328947368421 |
| b12_50.txt | b12_70.txt | 0.454545454545 |

**Πίνακας 4: Εκτέλεση 3.2γ**

| File1 | File2 | Resemblance |
|---|---|---|
| b12.txt | b12_10.txt | 0.916666666667 |
| b12.txt | b12_20.txt | 0.666666666667 |
| b12.txt | b12_30.txt | 0.636363636364 |
| b12.txt | b12_50.txt | 0.444444444444 |
| b12.txt | b12_70.txt | 0.333333333333 |
| b12_10.txt | b12_20.txt | 0.733333333333 |
| b12_10.txt | b12_30.txt | 0.69696969697 |
| b12_10.txt | b12_50.txt | 0.488888888889 |
| b12_10.txt | b12_70.txt | 0.366666666667 |
| b12_20.txt | b12_30.txt | 0.818181818182 |
| b12_20.txt | b12_50.txt | 0.613636363636 |
| b12_20.txt | b12_70.txt | 0.457627118644 |
| b12_30.txt | b12_50.txt | 0.666666666667 |
| b12_30.txt | b12_70.txt | 0.5 |
| b12_50.txt | b12_70.txt | 0.603174603175 |

**Πίνακας 5: Εκτέλεση 3.2δ**

| File1 | File2 | Resemblance |
|---|---|---|
| b12.txt | b12_10.txt | 0.888888888889 |
| b12.txt | b12_20.txt | 0.857142857143 |
| b12.txt | b12_30.txt | 0.657142857143 |
| b12.txt | b12_50.txt | 0.575 |
| b12.txt | b12_70.txt | 0.328125 |
| b12_10.txt | b12_20.txt | 0.964285714286 |
| b12_10.txt | b12_30.txt | 0.742857142857 |
| b12_10.txt | b12_50.txt | 0.571428571429 |
| b12_10.txt | b12_70.txt | 0.313432835821 |
| b12_20.txt | b12_30.txt | 0.771428571429 |
| b12_20.txt | b12_50.txt | 0.595238095238 |
| b12_20.txt | b12_70.txt | 0.328358208955 |
| b12_30.txt | b12_50.txt | 0.738095238095 |
| b12_30.txt | b12_70.txt | 0.397058823529 |
| b12_50.txt | b12_70.txt | 0.538461538462 |

Ο λόγος για τον οποίο χρησιμοποιούμε διαφορετικές εκτελέσεις για να συγκρίνουμε τα αποτελέσματά μας, είναι διότι δεν πρέπει να ξεχνάμε το γεγονός ότι τα sketches παράγονται από τυχαίες μεταθέσεις του αρχικού dataset. Με τον τυχαίο παράγοντα που προσθέτουμε στην υλοποίησή μας, υπάρχει περίπτωση να υποπέσουμε σε λάθη, λόγω των μορφών που θα πάρουν οι τυχαίες μεταθέσεις. Για παράδειγμα, υπάρχουν περιπτώσεις που οι τυχαίες μεταθέσεις που παράγονται να μοιάζουν υπερβολικά πολύ μεταξύ τους και αυτό θα έχει ως αποτέλεσμα τα sketches των δύο αρχείων, ανεξάρτητα του πόσο μοιάζουν μεταξύ τους να έχουν το ίδιο στοιχείο να επαναλαμβάνεται. Αυτό θα έχει ως αποτέλεσμα να παράγονται νουμέρα διαφορετικά από το αναμενόμενο. Ο τρόπος που μπορούμε να σκεφτούμε για να επιλύσουμε αυτό το πρόβλημα είναι είτε να χρησιμοποιήσουμε περισσότερες μεταθέσεις (το οποίο αναλύεται στη συνέχεια), είτε να ελέγξουμε το πρόγραμμα για πολλές εκτελέσεις.

Ως εκ τούτου, έχουμε τον παρακάτω πίνακα με τους μέσους όρους των εκτελέσεων:

**Πίνακας 6: Μέσοι όροι αποτελεσμάτων**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.86472222222225 | 0.9 |
| b12.txt | b12_20.txt | 0.7464207905385 | 0.8 |
| b12.txt | b12_30.txt | 0.64773559773575 | 0.7 |
| b12.txt | b12_50.txt | 0.47656323877 | 0.5 |
| b12.txt | b12_70.txt | 0.29750367711625 | 0.3 |
| b12_10.txt | b12_20.txt | 0.8251984126985 | 0.88 |
| b12_10.txt | b12_30.txt | 0.71210724759125 | 0.77 |
| b12_10.txt | b12_50.txt | 0.5174197906115 | 0.55 |
| b12_10.txt | b12_70.txt | 0.32355490723425 | 0.33 |
| b12_20.txt | b12_30.txt | 0.809683299157 | 0.88 |
| b12_20.txt | b12_50.txt | 0.5572366797525 | 0.62 |
| b12_20.txt | b12_70.txt | 0.3588040242075 | 0.37 |
| b12_30.txt | b12_50.txt | 0.65346134362075 | 0.71 |
| b12_30.txt | b12_70.txt | 0.39473684210525 | 0.42 |
| b12_50.txt | b12_70.txt | 0.52932708918625 | 0.6 |

Όπως βλέπουμε από τις εκτελέσεις, τα αποτελέσματα σε γενικές γραμμές είναι ικανοποιητικά. Τα αναμενόμενα αποτελέσματα μεταξύ των παραλλαγών του b12.txt βρέθηκαν μετά τη δημιουργία τους, βρίσκοντας το ποσοστό των γραμμών που λείπουν από το καθένα και αφαιρώντας το από το 1 (υψηλότερη τιμή resemblance).

Ωστόσο, παρά το γεγονός ότι τα νούμερα προσεγγίζουν σωστά την ομοιότητα μεταξύ των αρχείων, παρατηρούμε μία απόκλιση περίπου στο 0.05 από την αναμενόμενη τιμή. Αυτό μπορεί να ερμηνευτεί με δύο τρόπους. Πρώτον, κατά τη δημιουργία των παραλλαγών του b12.txt, δεν αποκόβουμε το 10% του κάθε αρχείου ολόκληρο, αλλά σε μέρη. Για παράδειγμα, αν το σύνολο των γραμμών που πρέπει να διαγραφτούν είναι 60, τότε αποκόβουμε όχι 60 γραμμές, αλλά 6 διαφορετικές 10δες από γραμμές. Αυτό έχει ως αποτέλεσμα να δημιουργηθούν καινούρια shingles για κάθε παραλλαγή, με αποτέλεσμα, προσμετρούνται κατά τη δημιουργία του κάθε sketch μόνο για το ένα αρχείο και όχι για τους «προγόνους» του. Ωστόσο, ακόμα και έτσι, είναι υπερβολικό  να παίρνουμε τόσο μεγάλες αποκλίσεις, μιας και ο αριθμός των καινούριων shingles είναι πολύ μικρότερος σε σχέση με όλα τα shingles που παράγονται. Για αυτό το λόγο, καταφεύγουμε στην δεύτερη εξήγηση και αυτή είναι ότι οι παράμετροι που δώσαμε στο πρόγραμμα στην αρχή της εκτέλεσης δεν είναι οι βέλτιστες που μπορούμε να δώσουμε.

## 3.3 Έλεγχος για διαφορετικές τιμές των αρχικών παραμέτρων

### 3.3.1 Μελέτη παραμέτρου w

Αρχικά, θα μελετήσουμε τη συμπεριφορά του προγράμματος για διαφορετικές τιμές της παραμέτρου w. Η παράμετρος w περιγράφει το μήκος που θα έχει το κάθε shingle, τόσο στο dataset και στις τυχαίες μεταθέσεις, όσο και στο κάθε σύνολο από file shingles. Όπως αναφέρθηκε και στην παράγραφο της περιγραφής του θέματος της εργασίας, ο λόγος για τον οποίο επιτρέπουμε στον αλγόριθμο να λειτουργεί με διαφορετικό μέγεθος από shingles, είναι επειδή επιτρέπουμε συγκρίσεις μεταξύ λέξεων, μελετώντας όμως και προηγούμενες «καταστάσεις» της κάθε λέξης/token. Με αυτόν τον τρόπο, μπορούμε και καταλαβαίνουμε ότι δύο αρχεία έχουν κοινά στοιχεία μεταξύ τους με μεγαλύτερη σιγουριά, μιας και εξετάζουμε «κομμάτια» των δύο κειμένων και όχι ξεχωριστές λέξεις, οι οποίες είναι πιο πιθανό να βρίσκονται στα δύο αρχεία απλά τυχαία, χωρίς να έχουν κάτι κοινό μεταξύ τους. Από την άλλη πλευρά βέβαια, το μέγεθος των shingles δεν πρέπει να είναι αρκετά μεγάλο, σε σχέση με το μέγεθος του αρχείου τους, επειδή για μεγάλο μέγεθος shingle επικαλύπτονται μέρη του αρχείου που μπορεί να είναι ίδια με αυτά άλλου αρχείου. Για παράδειγμα, σε δύο αρχεία για μέγεθος shingle = 20, ενδεχομένως να χαθούν αρκετά μέρη των δυο αρχείων τα οποία είναι ίδια για 19 ή λιγότερες λέξεις. Σαν συμπέρασμα, αυτό που θεωρούμε καταλληλότερο, αλλά και αυτό που περιμένουμε να δούμε από τις εκτελέσεις, είναι ότι το μέγεθος των shingles πρέπει να είναι στο κατάλληλο σημείο, σε σχέση με το μέγεθος του αρχείου, έτσι ώστε να μην εμποδίζει τη σύγκριση των 2 αρχείων για μικρές τιμές, αλλά και ούτε να επικαλύπτει ομοιότητες για μεγάλες τιμές. Στις παραπάνω εκτελέσεις, το μέγεθος των αρχείων που συγκρίνουμε αλλά και που συμπεριλαμβάνουμε στο dataset είναι συνήθως γύρω στις 500 γραμμές.

Για εκτέλεση: python -w 1 -t 100 -d ./food2

**Πίνακας 7: Εκτέλεση 3.3α**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.842105263158 | 0.9 |
| b12.txt | b12_70.txt | 0.457142857143 | 0.3 |
| b12_10.txt | b12_50.txt | 0.692307692308 | 0.55 |
| b12_50.txt | b12_70.txt | 0.764705882353 | 0.6 |

Για εκτέλεση: python -w 5 -t 100 -d ./food2

**Πίνακας 8: Εκτέλεση 3.3β**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.86472222222225 | 0.9 |
| b12.txt | b12_70.txt | 0.29750367711625 | 0.3 |
| b12_10.txt | b12_50.txt | 0.5174197906115 | 0.55 |
| b12_50.txt | b12_70.txt | 0.52932708918625 | 0.6 |

Για εκτέλεση: python -w 10 -t 100 -d ./food2

**Πίνακας 9: Εκτέλεση 3.3γ**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.785714285714 | 0.9 |
| b12.txt | b12_70.txt | 0.257142857143 | 0.3 |
| b12_10.txt | b12_50.txt | 0.5 | 0.55 |
| b12_50.txt | b12_70.txt | 0.486842105263 | 0.6 |

Για εκτέλεση: python -w 20 -t 100 -d ./food2

**Πίνακας 10: Εκτέλεση 3.3δ**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.774193548387 | 0.9 |
| b12.txt | b12_70.txt | 0.393442622951 | 0.3 |
| b12_10.txt | b12_50.txt | 0.613636363636 | 0.55 |
| b12_50.txt | b12_70.txt | 0.442857142857 | 0.6 |

Για εκτέλεση: python -w 50 -t 100 -d ./food2

**Πίνακας 11: Εκτέλεση 3.3ε**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.766666666667 | 0.9 |
| b12.txt | b12_70.txt | 0.289855072464 | 0.3 |
| b12_10.txt | b12_50.txt | 0.581395348837 | 0.55 |
| b12_50.txt | b12_70.txt | 0.424657534247 | 0.6 |

Από τα αποτελέσματα, βλέπουμε ότι η εκτέλεση που προσεγγίζει καλύτερα τα αναμενόμενα αποτελέσματα, είναι αυτή που έχει μέγεθος shingle w = 5, δηλαδή η εκτέλεση που είχαμε και στον έλεγχο ορθότητας του αλγορίθμου. Το γεγονός αυτό δε θα πρέπει να μας προβληματίζει, καθώς όπως αναφέραμε, το μέγεθος των shingles δεν πρέπει να είναι ούτε αρκετά μικρό, αλλά ούτε και αρκετά μεγάλο, σε σχέση με το μέγεθος του αρχείου. Ενδεχομένως, για μεγαλύτερα μεγέθη αρχείων, να χρειαζόμασταν μεγαλύτερο μέγεθος από shingles. Από τις παραπάνω εκτελέσεις, θα προτείναμε ένα μέγεθος από shingles που τηρεί την αναλογία shinglesSize - filesRows : 1-100

Επίσης, παρατηρούμε ότι για τις εκτελέσεις με w = 1 και w = 50 παίρνουμε τις μεγαλύτερες αποκλίσεις με βάση τα αναμενόμενα αποτελέσματα. Πιο συγκεκριμένα αποτελούν τις μόνες εκτελέσεις όπου συναντάμε περισσότερες από μία αποκλίσεις της τάξης του 0.15 . Οι παρατηρήσεις, παρουσιάζονται και σε μορφή διαγράμματος στο Διάγραμμα του Παραρτήματος 1.

### 3.3.2 Μελέτη παραμέτρου t

Στη συνέχεια, γίνονται δοκιμές ως προς τις διαφορετικές τιμές της παραμέτρου t. Η παράμετρος t αποτελεί τον αριθμό των τυχαίων μεταθέσεων του αρχικού dataset, τις οποίες χρησιμοποιούμε για τον υπλογισμό των sketches. Σε αυτό το σημείο η πρόβλεψη των αποτελεσμάτων είναι πιο εύκολη. Αυτό που περιμένουμε είναι όσο αυξάνεται ο αριθμός των τυχαίων μεταθέσεων, να μειώνεται και το σφάλμα μεταξύ των τιμών που παίρνουμε και των αναμενόμενων τιμών. Είναι λογικό όσο λιγότερες τυχαίες μεταθέσεις έχουμε, να υποπίπτουμε και σε πιο πολλά και σημαντικά λάθη. Για παράδειγμα, σε περίπτωση που έχουμε μόνο μία τυχαία μετάθεση στη διάθεσή μας, τότε αν πάμε να συγκρίνουμε δύο αρχεία με βάση τα sketches τους, αυτό που θα παίρναμε θα ήταν μόνο 2 διαφορετικά αποτελέσματα: 0 και 1. Αυτό συμβαίνει επειδή τα sketches έχουν μέγεθος ίσο με το t, οπότε θα καταλήγαμε με δύο sketches μεγέθους 1 το καθένα. Αντίθετα, εάν είχαμε περισσότερα sketches στη διάθεσή μας, θα μπορούσαμε να εφαρμόσουμε τον τύπο (2) από την απόδειξη του αλγορίθμου περισσότερες φορές, και έτσι να παίρναμε ένα ακριβέστερο και πιο σίγουρο αποτέλεσμα για την ομοιότητα των 2 αρχείων. Από την άλλη πλευρά βέβαια, πρέπει να έχουμε στο μυαλό μας ότι ο αριθμός των τυχαίων μεταθέσεων προσθέτει επιπλέον υπολογισμούς σε όλα τα σημεία του αλγορίθμου, μιας και οι τυχαίες μεταθέσεις συμμετέχουν επί της ουσίας σε όλο τον υπολογισμό των sketches. Επομένως, πρέπει να έχουμε στο μυαλό μας ότι θα πρέπει να χρησιμοποιήσουμε αρκετές μεν, αλλά όχι και υπερβολικά μεγάλο αριθμό, έτσι ώστε να μην καταναλώσουμε όλους τους πόρους του υπολογιστή στον οποίο εργαζόμαστε.

Ακολουθούν εκτελέσεις για διαφορετικό αριθμό από τυχαίες μεταθέσεις, με μήκος shingle = 5:

Για εκτέλεση: python -w 5 -t 1 -d ./food2

**Πίνακας 12: Εκτέλεση 3.3στ**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 1.0 | 0.9 |
| b12.txt | b12_70.txt | 0.0 | 0.3 |
| b12_10.txt | b12_50.txt | 1.0 | 0.55 |
| b12_50.txt | b12_70.txt | 0.0 | 0.6 |

**Χρόνος εκτέλεσης:** 25.6569998264 seconds

Για εκτέλεση: python -w 5 -t 20 -d ./food2

**Πίνακας 13: Εκτέλεση 3.3ζ**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.818181818182 | 0.9 |
| b12.txt | b12_70.txt | 0.173913043478 | 0.3 |
| b12_10.txt | b12_50.txt | 0.473684210526 | 0.55 |
| b12_50.txt | b12_70.txt | 0.296296296296 | 0.6 |

**Χρόνος εκτέλεσης:** 42.4769999981 seconds

Για εκτέλεση: python -w 5 -t 100 -d ./food2

**Πίνακας 14: Εκτέλεση 3.3η**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.86472222222225 | 0.9 |
| b12.txt | b12_70.txt | 0.29750367711625 | 0.3 |
| b12_10.txt | b12_50.txt | 0.5174197906115 | 0.55 |
| b12_50.txt | b12_70.txt | 0.52932708918625 | 0.6 |

**Χρόνος εκτέλεσης:** 54.3309998512 seconds

Για εκτέλεση: python -w 5 -t 200 -d ./food2

**Πίνακας 15: Εκτέλεση 3.3θ**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.882352941176 | 0.9 |
| b12.txt | b12_70.txt | 0.337209302326 | 0.3 |
| b12_10.txt | b12_50.txt | 0.571428571429 | 0.55 |
| b12_50.txt | b12_70.txt | 0.561797752809 | 0.6 |

**Χρόνος εκτέλεσης:** 58.257999897 seconds

Για εκτέλεση: python -w 5 -t 1000 -d ./food2

**Πίνακας 16: Εκτέλεση 3.3ι**

| File1 | File2 | Resemblance | Expected |
|---|---|---|---|
| b12.txt | b12_10.txt | 0.923076923077 | 0.9 |
| b12.txt | b12_70.txt | 0.322147651007 | 0.3 |
| b12_10.txt | b12_50.txt | 0.573033707865 | 0.55 |
| b12_50.txt | b12_70.txt | 0.590604026846 | 0.6 |

**Χρόνος εκτέλεσης:** 984.805000067 seconds

Όπως βλέπουμε και από τις εκτελέσεις, το σφάλμα μειώνεται όσο αυξάνεται ο αριθμός των μεταθέσεων. Χαρακτηριστικά βλέπουμε το σφάλμα για 1000 τυχαίες μεταθέσεις να έχει φτάσει στο 0.02. Παρ' όλα αυτά, παρατηρούμε ότι ο χρόνος εκτέλεσης αυξάνεται δραματικά. Γι' αυτό το λόγο, πρέπει να επιλέγουμε τον αριθμό των μεταθέσεων ανάλογα με την ακρίβεια που επιθυμούμε. Αν για παράδειγμα, μηχανές αναζήτησης θέλουν να χρησιμοποιήσουν τον αλγόριθμο για την ανίχνευση ίδιων αρχείων έτσι ώστε να μην παρουσιάζουν ίδια αποτελέσματα σε κάθε αναζήτηση, αρκούν 100 μεταθέσεις, μιας και δεν τους απασχολεί η ακριβής ομοιότητα, αλλά απλά αν δύο αρχεία είναι όμοια ή σχεδόν όμοια, για παράδειγμα αν η ομοιότητά τους υπερβαίνει η όχι το 0.5 . Οι παρατηρήσεις, παρουσιάζονται και σε μορφή διαγράμματος στο Διάγραμμα του Παραρτήματος 2.

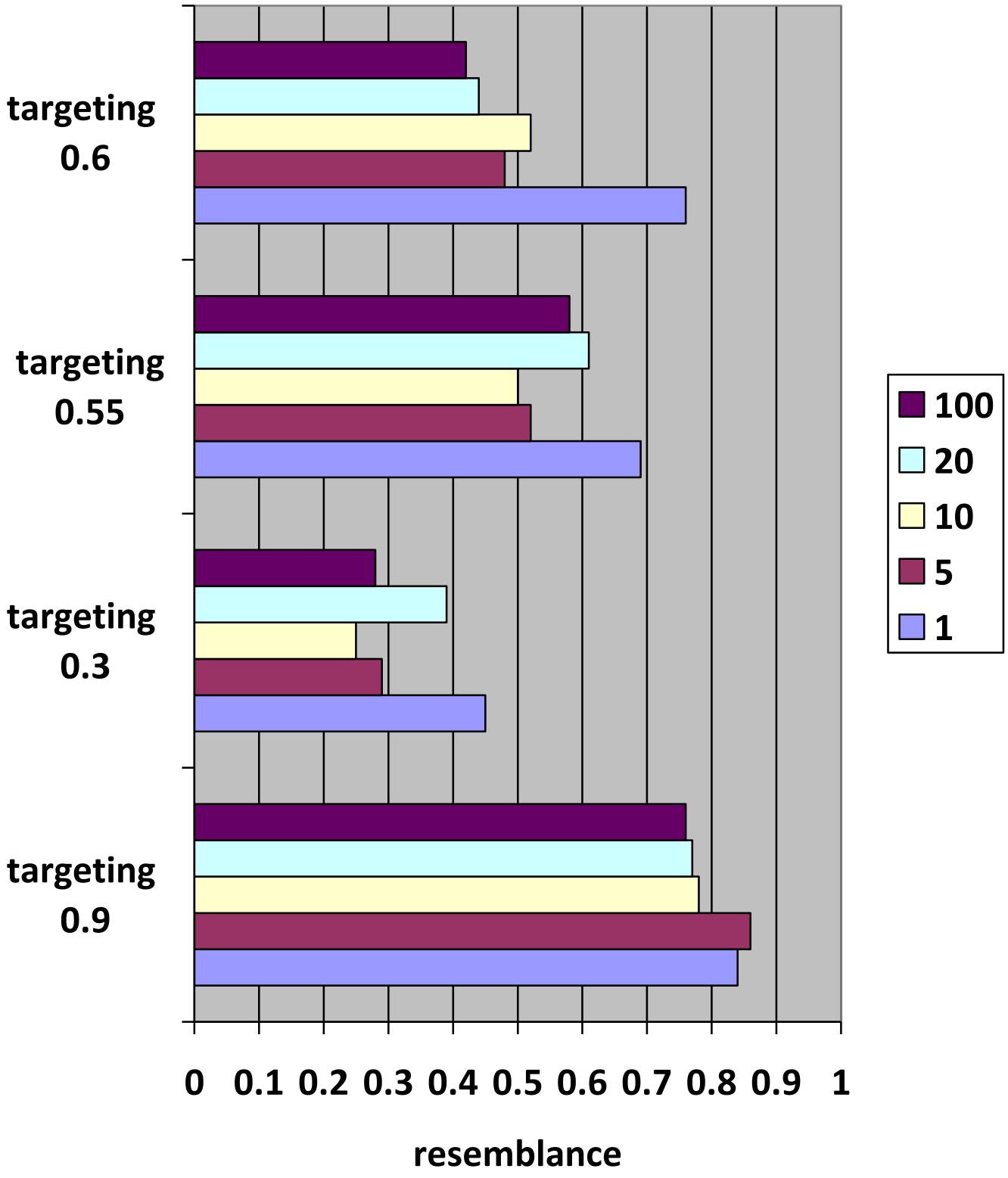
# 4. ΣΥΜΠΕΡΑΣΜΑΤΑ

Με βάση όλα τα παραπάνω μπορούμε να καταλήξουμε σε κάποια βασικά συμπεράσματα. Πρώτα απ' όλα, ο αλγόριθμος για τη σύγκριση αρχείων που περιεγράφηκε είναι σωστός και παράγει σωστά αποτελέσματα. Ο χρόνος εκτέλεσής του μεταβάλλεται ανάλογα με το μέγεθος της εισόδου και με τις παραμέτρους που δίνουμε σαν είσοδο. Παρ' όλα αυτά, τα δύο στάδια του αλγορίθμου απαιτούν χρόνο γραμμικό ως προς το μέγεθος του συνόλου αρχείων που δίνουμε σαν είσοδο και χρόνο γραμμικό ως προς το μέγεθος των sketch των 2 αρχείων που δίνουμε σαν είσοδο αντίστοιχα, προκειμένου να εκτελεστεί. Όσον αφορά την καταλληλότερη τιμή της παραμέτρου w, πρέπει να βρούμε τη σωστή αναλογία, προκειμένου να παράγουμε τα καταλληλότερα shingles για τη σύγκριση των sketches. Όσον αφορά την τιμή της παραμέτρου t, το σφάλμα των αποτελεσμάτων αυξάνεται, όσο αυξάνεται και η τιμή της, σε αντίθεση βέβαια με το χρόνο εκτέλεσης του προγράμματος, ο οποίος αυξάνεται. Για αυτό το λόγο, πρέπει να επιλέγουμε την τιμή της, ανάλογα με την προτεραιότητά μας ανάμεσα σε ορθότητα αποτελεσμάτων και χρόνο εκτέλεσης.

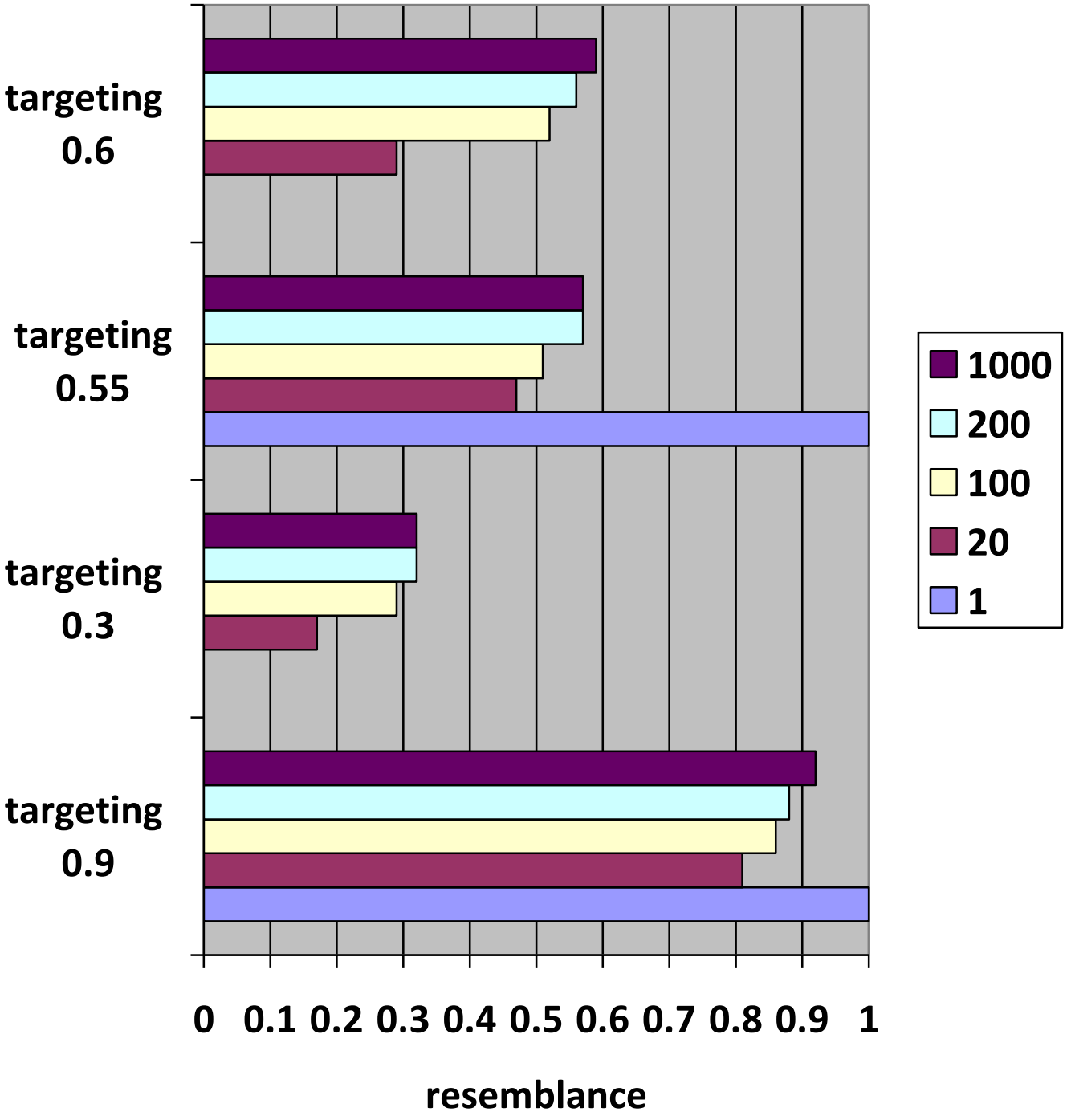
# ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

| **Ξενόγλωσσος όρος** | **Ελληνικός Όρος** |
|---|---|
| Resemblance | Βαθμός ομοιότητας μεταξύ δύο αρχείων (δεκαδικές τιμές από 0 έως 1) |
| Token | Βασικό στοιχείο ενός shingle, γράμμα/λέξη/γραμμή ενός αρχείου |
| Shingle | Σύνολο σταθερού μήκους από tokens ενός αρχείου |
| Fingerprint | Ακέραιος αριθμός ο οποίος χαρακτηρίζει ένα shingle |
| Sketch | Υποσύνολο δεδομένων ενός αρχείου |

## ΠΑΡΑΡΤΗΜΑ Ι


targeting 0.6
targeting 0.55
targeting 0.3
targeting 0.9
100
20
10
5
1
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
resemblance

# ΠΑΡΑΡΤΗΜΑ ΙΙ



# ΑΝΑΦΟΡΕΣ

1) A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proceedings of the Sixth International World Wide Web Conference*, pages 391-404, 1997.

2) N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents on the web. In *Proceedings of Workshop on Web Databases (WebDB'98)*, March 1998.

3) A. Z. Broder, Identifying and filtering near-Duplicate Documents, R. Giancarlo and D. Sankoff (Eds.): CPM 2000, LNCS 1848, 2000

4) M. O. Rabin. Fingerprinting by random polynomials. Center for Research in Computing Technology, Harvard University, Report TR-15-81, 1981.

5) http://textfiles.com/food/