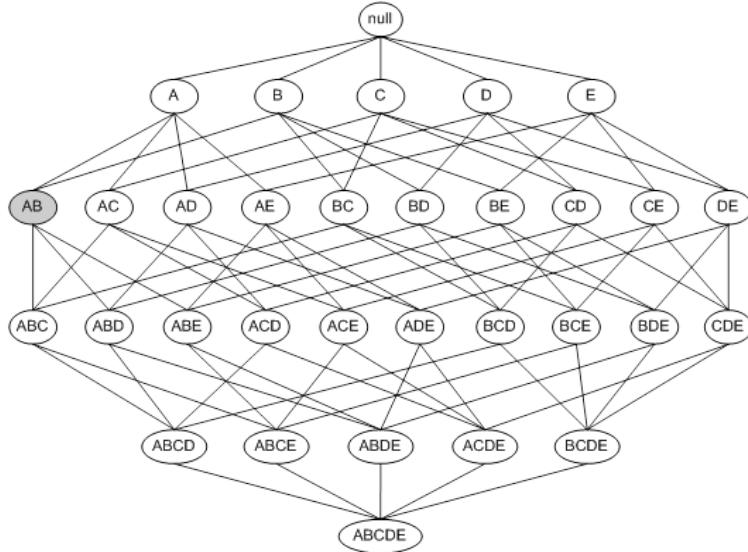




Υλοποίηση εφαρμογής που με βάση τον **A-priori** (κλασικός αλγόριθμος εξόρυξης δεδομένων), κάνει εξόρυξη συχνών στοιχειοσυνόλων και στην συνέχεια παράγει κανόνες συσχετίσεων.



Introduction to Data Mining by Tan, Steinbach, Kumar  
(<http://slideplayer.com/slide/7539139/>)[1]

Τα αρχεία της υλοποίησης βρίσκονται στο: <https://bitbucket.org/michaelgallakiis/apriori>

**Λέξεις κλειδιά:** Data mining, Apriori, Apriori-Gen, Subset, Association Rules, Itemset

## Περίληψη

Στην εργασία αυτή αρχικά, παρουσιάζονται επιγραμματικά κάποιοι κλασικοί αλγόριθμοι εξόρυξης γνώσης. Στην συνέχεια, σαν εισαγωγή, περιγράφεται αναλυτικά η λειτουργία του Apriori αλγόριθμου και παράλληλα γνωστοποιούνται κάποιες βασικές έννοιες, όπως τι θεωρούνται κανόνες συσχετίσεων (Association Rules) και στοιχειοσύνολα (itemsets). Έπειτα, παρουσιάζονται οι δυνατότητες μιας εφαρμογής που αναπτύχθηκε στα πλαίσια της εργασίας. Το λογισμικό αυτό που υλοποιήθηκε, εφαρμόζει τον αλγόριθμο A-priori, ώστε να βρίσκει συχνά στοιχειοσύνολα (ενός συγκεκριμένου δοθέντος dataset) και στην συνέχεια με βάση αυτά τα itemsets, είναι σε θέση η εφαρμογή, να παράγει κανόνες συσχετίσεων. Κατόπιν, μέσα από αυτό το κείμενο, περιγράφονται κάποια τεχνικά χαρακτηριστικά, για το πως φτιάχτηκε η εφαρμογή (πχ εργαλεία ανάπτυξης και κώδικας με σχόλια). Μετά ακολουθεί ένα παράδειγμα με αποτελέσματα της εφαρμογής και τέλος αναφέρονται κάποιες μελλοντικές επεκτάσεις, κάποια συμπεράσματα και η βιβλιογραφία που χρησιμοποιήθηκε.

# Περιεχόμενα

|   |    |
|---|----|
| Περίληψη.....   | 1  |
| Πρόλογος.....   | 2  |
| 1. Εισαγωγή.....  | 3  |
| 1.1. Κανόνες συσχετίσεων και συχνά στοιχειοσύνολα.....                | 3  |
| 1.2. Αλγόριθμος A-priori.....   | 3  |
| 1.3. Συναρτήσεις apriori-gen και subset.....                          | 5  |
| 1.4. Παραγωγή κανόνων συσχετίσεων από στοιχειοσύνολα του Apriori..... | 6  |
| 2. Σύντομη περιγραφή υλοποίησης.....                                  | 7  |
| 3. Τεχνικά χαρακτηριστικά της υλοποίησης.....                         | 9  |
| 4. Αποτελέσματα εφαρμογής-αλγορίθμου.....                             | 18 |
| 5. Μελλοντικές επεκτάσεις.....  | 20 |
| 6. Συμπεράσματα.....  | 20 |
| Βιβλιογραφία.....   | 20 |

## Πρόλογος

Στην αρχή, ψάχνοντας στην βιβλιογραφία για να βρούμε ποιον αλγόριθμο θα επιλέγαμε για να υλοποιήσουμε, γνωρίσαμε αρκετούς αλγόριθμους που χρησιμοποιούνται στην εξόρυξη δεδομένων. Οπότε ήταν σημαντικό να μάθουμε πως δουλεύουν (έστω σε γενικές γραμμές), ώστε να επιλέξουμε τον αλγόριθμο που θα μας ταίριαζε περισσότερο.

Για παράδειγμα, μέσα από την αναζήτηση μας είδαμε αλγόριθμους που σχετίζονται, με την ίσως πιο γνωστή και δημοφιλής τεχνική εξόρυξης γνώσης, που είναι η **κατηγοροποίηση (classification)**. Η οποία μπορεί πολύ συνοπτικά να περιγραφεί ως μια λειτουργία που αντιστοιχεί (κατηγοριοποιεί) ένα στοιχείο σε μία από διαφορετικές κατηγορίες που έχουν προκαθοριστεί... Η διαδικασία περιλαμβάνει δύο βήματα, την εκμάθηση (Learning) και στην συνέχεια την κατηγοροποίηση. Στην τεχνική αυτή, οι αλγόριθμοι μπορεί να βασίζονται στην στατιστική (πχ αλγόριθμος παλινδρόμησης και διάφοροι Bayesian τύποι), στην απόσταση (πχ ο K-NN), σε δένδρα αποφάσεων (όπως ο ID3, C4.5, CART, SLIQ και SPRINT), σε νευρωνικά δίκτυα, σε κανόνες ή ακόμη και σε συνδυασμό όλων των προηγούμενων για ενίσχυση και καλύτερα αποτελέσματα. Κάποια παραδείγματα εφαρμογών αυτής της τεχνικής, είναι η αναγνώριση προτύπων και εικόνας, η ιατρική διάγνωση, η έγκριση δανείων και πολλά άλλα ακόμη.

Επιπλέον, γνωρίσαμε αλγόριθμους εξόρυξης γνώσης, που σχετίζονται με την τεχνική της **συσταδοποίησης (clustering)**. Η οποία είναι παρόμοια με την κατηγοροποίηση καθώς και στις δύο περιπτώσεις τα δεδομένα οργανώνονται σε ομάδες. Ωστόσο όμως, σε αυτή τη δεύτερη τεχνική, σε αντίθεση με την κατηγοροποίηση, οι ομάδες δεν είναι προκαθορισμένες. Πιο συγκεκριμένα, η συσταδοποίηση επιτυγχάνεται βρίσκοντας ομοιότητες μεταξύ των δεδομένων βάσει των χαρακτηριστικών που υπάρχουν σε αυτά. Συνοπτικά, κάποιοι τύποι αλγορίθμων της τεχνικής αυτής είναι: ο ιεραρχικός (πχ αλγόριθμος απλού, μέσου και πλήρους συνδέσμου), ο διαμεριστικός (αλγόριθμοι όπως ο τετραγωνικού σφάλματος, K-means, πλησιέστερου γείτονα, PAM και BIRCH), ο μεικτός (πχ CURE και DBSCAN) και ο συσσωρευτικός (πχ ROCK).

Ακόμη, γνωρίσαμε άλλη μια κατηγορία αλγορίθμων για εξόρυξη δεδομένων, που χρησιμοποιούνται για την εύρεση **συχνών στοιχειοσυνόλων** και κατ' επέκταση για παραγωγή **κανόνων συσχέτισης (association rules)**. Οι κανόνες συσχετίσεων, συνοπτικά, χρησιμοποιούνται για να δείξουν τις συσχετίσεις ανάμεσα στα δεδομένα. Κάποιοι αλγόριθμοι αυτής της ομάδας είναι μονολεκτικά οι: DIC, Partition, FP-Growth, Eclat, PCY, Multistage, Multihash, SON, Toivonen και Apriori τον οποίο τελικά και επιλέξαμε για να υλοποιήσουμε και ο οποίος θα περιγραφεί αναλυτικά στην συνέχεια μέσα από αυτή την εργασία.

# 1. Εισαγωγή

## 1.1. Κανόνες συσχετίσεων και συχνά στοιχειοσύνολα

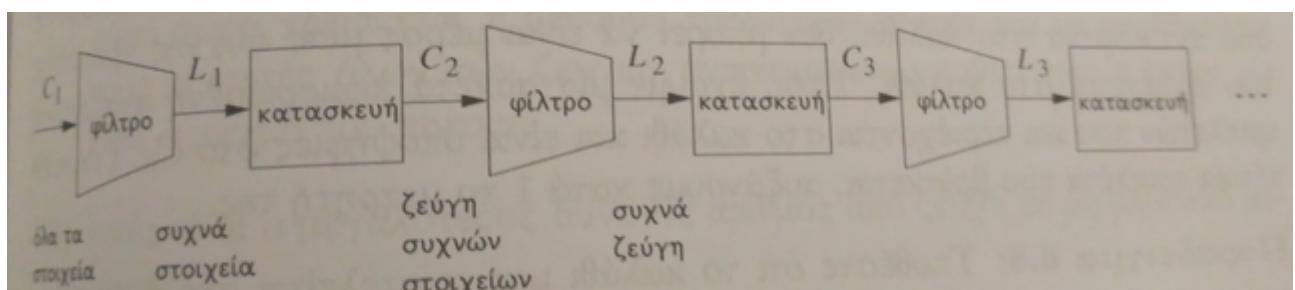
Οι κανόνες συσχετίσεων χρησιμοποιούνται για να δείξουν τις συσχετίσεις ανάμεσα στα δεδομένα. Αυτές οι συσχετίσεις που αποκαλύπτονται δεν είναι έμφυτες στα δεδομένα, όπως οι συναρτησιακές εξαρτήσεις, και δεν αντιπροσωπεύουν κανένα είδος αιτιότητας ή συσχέτισης. Αντίθετα, οι κανόνες συσχετίσεων ανιχνεύουν μία συνηθισμένη χρήση για τα στοιχεία. Για παράδειγμα, η αγορά ενός προϊόντος, όταν αγοράζεται μαζί και ένα άλλο προϊόν, αντιπροσωπεύει έναν κανόνα συσχέτισης. Οι κανόνες συσχετίσεων συχνά χρησιμοποιούνται από τα καταστήματα λιανικής πώλησης για να βιοηθήσουν στο μαρκετινγκ, στη διαφήμιση, στην ταξινόμηση των ορόφων και στον έλεγχο του καταλόγου απογραφής. Εάν και έχουν άμεση εφαρμοσιμότητα στις επιχειρήσεις λιανικής πώλησης, έχουν χρησιμοποιηθεί επίσης και για άλλους σκοπούς, συμπεριλαμβανομένης και της πρόβλεψης των λαθών στα δίκτυα τηλεπικοινωνιών. Η πιο κοινή προσέγγιση για την εύρεση κανόνων συσχέτισης, είναι η διάσπαση του προβλήματος σε δύο μέρη. Αρχικά στην εύρεση συχνών στοιχειοσυνόλων και στην συνέχεια, στην δημιουργία κανόνων από συχνά στοιχειοσύνολα. Ένα στοιχειοσύνολο είναι ένα οποιοδήποτε υποσύνολο όλων των στοιχείων ενός συνόλου. Ένα συχνό (ή μεγάλο) στοιχειοσύνολο είναι ένα στοιχειοσύνολο του οποίου ο αριθμός των εμφανίσεων είναι πάνω από ένα κατώφλι.[2] Παραδείγματος χάριν, θεωρούμε ότι αναζητάμε συχνά στοιχειοσύνολα προϊόντων ενός supermarket και έχουμε ένα δείγμα από 1000 αγορές-καλάθια. Αν έχουμε ορίσει σαν κατώφλι το 150 και δούμε ότι δύο προϊόντα (πχ καφές και ζάχαρη) εμφανίστηκαν μαζί 230 φορές στις 1000 αγορές, σημαίνει ότι έχουμε ένα μεγάλο στοιχειοσύνολο. Με βάση αυτό το στοιχειοσύνολο, μπορούμε να παράγουμε κανόνες συσχέτισης, εφόσον έχουν μεγάλη υποστήριξη (Support) και εμπιστοσύνη (Confidence) αυτά τα δύο προϊόντα, διότι για να έχουν αξία οι κανόνες συσχέτισης, πρέπει να είναι ισχυροί. Που σημαίνει πρακτικά, ότι η υποστήριξη πρέπει να είναι πάνω από 20% και η εμπιστοσύνη γύρω στο 60%. Στο παραπάνω παράδειγμα, η τιμή του support είναι πρακτικά η πιθανότητα εμφάνισης και των δύο προϊόντων μαζί, δηλαδή  $230/1000 = 23\%$ . Για το confidence στο παράδειγμα μας έχουμε δύο τιμές, μία για τον υπολογισμό της πιθανότητας ενός που πήρε καφέ να πάρει και ζάχαρη, και άλλη μία για την πιθανότητα ενός που πήρε ζάχαρη να πάρει και καφέ. Οπότε αν υποθέσουμε ότι ο καφές μόνος του εμφανίστηκε 300 φορές και η ζάχαρη 450 στις 1000 αγορές, έχουμε για τη σχέση καφές=>ζάχαρη, εμπιστοσύνη  $230/300 = 77\%$  (τύπος: support {ζάχαρη,καφές}/support{καφές}) και για τη σχέση ζάχαρη=>καφές εμπιστοσύνη  $230/450 = 51\%$ . Οπότε θα μπορούσαμε να παραγάγουμε έναν καλό κανόνα συσχέτισης, που θα έλεγε ότι υπάρχει σοβαρή πιθανότητα ένας που αγοράζει καφέ, να αγοράσει και ζάχαρη.

## 1.2. Αλγόριθμος A-priori

Ο Apriori είναι από τους πιο γνωστούς αλγόριθμους για την εύρεση κανόνων συσχέτισης και χρησιμοποιείται σε πολλά εμπορικά προϊόντα.[3] Η βασική ιδέα του αλγορίθμου είναι η δημιουργία υποψήφιων στοιχειοσυνόλων ενός συγκεκριμένου μεγέθους και στη συνέχεια η σάρωση της βάσης δεδομένων για να μετρήσουμε και να δούμε αν αυτά είναι συχνά. Μόνο εκείνοι οι υποψήφιοι που είναι συχνοί χρησιμοποιούνται για τη δημιουργία υποψήφιων για το επόμενο πέρασμα. Ένα itemset θεωρείται ως υποψήφιο μόνο όταν όλα του τα υποσύνολα είναι επίσης συχνά.[2]

Το όνομα του αλγορίθμου οφείλεται στην εξής ιδιότητα: Κάθε υποσύνολο ενός συχνού στοιχειοσυνόλου (itemset) είναι επίσης συχνό. Επίσης, ισχύει και η αντιστροφή της παρακάτω ιδιότητας: Υπάρχει τουλάχιστον ένα υποσύνολο ενός μη συχνού στοιχειοσυνόλου που να είναι επίσης μη συχνό. Δεδομένων αυτών των ιδιοτήτων μπορούμε να παράγουμε τα υποψήφια στοιχειοσύνολα από τα ήδη γνωστά itemsets και μόνο. Απορρίπτουμε έτσι ένα μεγάλο σύνολο από υποψήφια itemsets και δεν υπολογίζουμε την υποστήριξη τους, καθώς είναι γνωστό εκ των προτέρων (a priori) ότι αυτά δεν πρόκειται να είναι συχνά.[4]

Πιο συγκεκριμένα, στο πρώτο πέρασμα από το δείγμα μας (πχ βάση δεδομένων, text αρχείο κ.ά) βρίσκονται τα συχνά 1-itemsets, μετρώντας πόσες φορές εμφανίζεται το κάθε item (μετριέται η υποστήριξη) και απομακρύνοντας αυτά που εμφανίζονται λιγότερο από την ελάχιστη υποστήριξη (κατώφλι) που έχουμε ορίσει. Κάθε επόμενο πέρασμα, περιλαμβάνει 2 φάσεις. Η πρώτη αφορά τη παραγωγή των υποψήφιων στοιχειοσυνόλων (ονομάζονται υποψήφια καθώς δεν γνωρίζουμε την υποστήριξη τους και κατ' επέκταση αν είναι συχνά) από τις συνενώσεις των ήδη συχνών στοιχειοσυνόλων που βρέθηκαν στο προηγούμενο πέρασμα (Για το σκοπό αυτό χρησιμοποιείται η συνάρτηση apriori-gen που περιγράφεται στην ενότητα 1.3). Η δεύτερη φάση αφορά στον υπολογισμό του support count για τα υποψήφια itemsets. Για κάθε transaction (πχ ένα καλάθι αγορών του παραδείγματος της ενότητας 1.1) βρίσκονται τα υποψήφια itemsets που περιέχονται σε αυτό και ο μετρητής τους αυξάνεται κατά 1. Το κρίσιμο σημείο στην δεύτερη φάση είναι ο γρήγορος υπολογισμός των στοιχειοσυνόλων, των υποψήφιων δηλαδή που περιέχονται στην εκάστοτε δοσοληψία. Η συνάρτηση subset επιτυγχάνει αυτόν τον σκοπό και περιγράφεται στην ενότητα 1.3 μαζί με την apriori-gen. Στο τέλος του κάθε βήματος αποφασίζεται ποια itemsets είναι συχνά (με βάση το κατώφλι που έχει οριστεί) ώστε να χρησιμοποιηθούν για το επόμενο βήμα. Ο αλγόριθμος Apriori διαβάζει το δείγμα μας διαδοχικές φορές και το πολύ τόσες φορές όσες είναι το πλήθος των διαφορετικών items. Σταματάει η όλη διαδικασία (επαναλήψεις), όταν η συνάρτηση apriori-gen δεν μπορεί να παράξει άλλα υποψήφια στοιχειοσύνολα (με βάση το προκαθορισμένο κατώφλι). Οπότε η λύση του ζητούμενου προβλήματος είναι όλα τα συχνά στοιχειοσύνολα που έχει βρει ο apriori μετά το πέρας όλων των διαβασμάτων-περασμάτων (επαναλήψεων).[3]



Σχήμα 1. Ο αλγόριθμος Apriori εναλλάσσεται μεταξύ της κατασκευής υποψήφιων συνόλων και φιλτραρίσματος για την εύρεση των πράγματι συχνών. [4]

### 1.3. Συναρτήσεις **apriori-gen** και **subset**

Η συνάρτηση **apriori-gen** παράγει τα υποψήφια στοιχειοσύνολα από τα ήδη γνωστά στοιχειοσύνολα του προηγούμενου περάσματος-διαβάσματος του apriori. Η συνάρτηση αποτελείται από δύο βήματα, το join-step (ένωση) και το prune-step (ξεκαθάρισμα). Στο πρώτο βήμα γίνεται η ένωση δύο (k-1)-itemsets που έχουν ακριβώς (k-2) κοινά items. Έτσι το itemset που θα προκύψει από αυτήν την ένωση θα αποτελείται από τα (k-2) κοινά items συν τα μη κοινά item από τα δύο (k-1) itemsets, δηλαδή θα έχει σύνολο k items. Στο επόμενο βήμα γίνεται χρήση της βασικής αρχής του αλγόριθμου apriori. Κατά συνέπεια απορρίπτονται εκείνα τα itemsets για τα οποία υπάρχει τουλάχιστον ένα (k-1) υποσύνολο τους που να μην ανήκει στα συχνά στοιχειοσύνολα του προηγούμενου βήματος του apriori. Διότι είναι *a priori* (εκ των προτέρων) γνωστό ότι δεν είναι συχνά. Για παράδειγμα, αν έχουμε πέντε 3-itemsets:  $\{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{1,3,5\}, \{2,3,4\}$ , τότε μετά το join-step θα έχουμε τα εξής δύο στοιχειοσύνολα:  $\{\{1,2,3,4\}, \{1,3,4,5\}\}$ . Τέλος το prune-step θα διαγράψει το itemset  $\{1,3,4,5\}$  επειδή το υποσύνολο του  $\{1,4,5\}$  δεν βρίσκεται στα συχνά 3-itemsets του προηγούμενου περάσματος του apriori. Κατά συνέπεια το μόνο υποψήφιο στοιχειοσύνολο, για το επόμενο πέρασμα του apriori, θα είναι το  $\{1,2,3,4\}$ . [3]

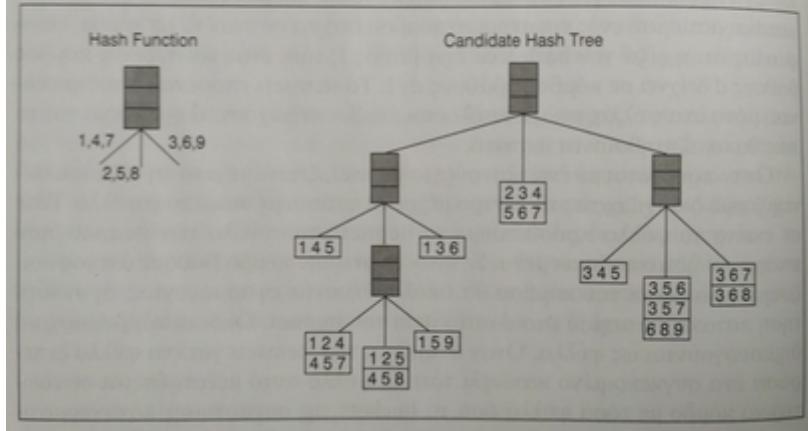
ΠΙΝΑΚΑΣ 6.7: Παράδειγμα του Apriori-Gen

| Πέρασμα | Υποψήφιο   | Συχνά στοιχειοσύνολα   |
|---------|--|--|
| 1       | $\{\text{Blouse}\}, \{\text{Jeans}\}, \{\text{Shoes}\}, \{\text{Shorts}\}, \{\text{Skirt}\}, \{\text{TShirt}\}$  | $\{\text{Jeans}\}, \{\text{Shoes}\}, \{\text{Shorts}\}, \{\text{Skirt}\}, \{\text{TShirt}\}$   |
| 2       | $\{\text{Jeans, Shoes}\}, \{\text{Jeans, Shorts}\}, \{\text{Jeans, Skirt}\}, \{\text{Jeans, TShirt}\}, \{\text{Shoes, Shorts}\}, \{\text{Shoes, Skirt}\}, \{\text{Shoes, TShirt}\}, \{\text{Shorts, Skirt}\}, \{\text{Shorts, TShirt}\}, \{\text{Skirt, TShirt}\}$ | $\{\text{Jeans, Shoes}\}, \{\text{Jeans, Shorts}\}, \{\text{Jeans, TShirt}\}, \{\text{Shoes, Shorts}\}, \{\text{Shoes, TShirt}\}, \{\text{Shorts, TShirt}\}, \{\text{Skirt, TShirt}\}$ |
| 3       | $\{\text{Jeans, Shoes, Shorts}\}, \{\text{Jeans, Shoes, TShirt}\}, \{\text{Jeans, Shorts, TShirt}\}, \{\text{Jeans, Skirt, TShirt}\}, \{\text{Shoes, Shorts, TShirt}\}, \{\text{Shoes, Skirt, TShirt}\}, \{\text{Shorts, Skirt, TShirt}\}$                         | $\{\text{Jeans, Shoes, Shorts}\}, \{\text{Jeans, Shoes, TShirt}\}, \{\text{Jeans, Shorts, TShirt}\}, \{\text{Shoes, Shorts, TShirt}\}$   |
| 4       | $\{\text{Jeans, Shoes, Shorts, TShirt}\}$  | $\{\text{Jeans, Shoes, Shorts, TShirt}\}$  |
| 5       | $\emptyset$  | $\emptyset$  |

Σχήμα 2. Ένα παράδειγμα του Apriori-Gen από συναλλαγές με είδη ρουχισμού. [2]

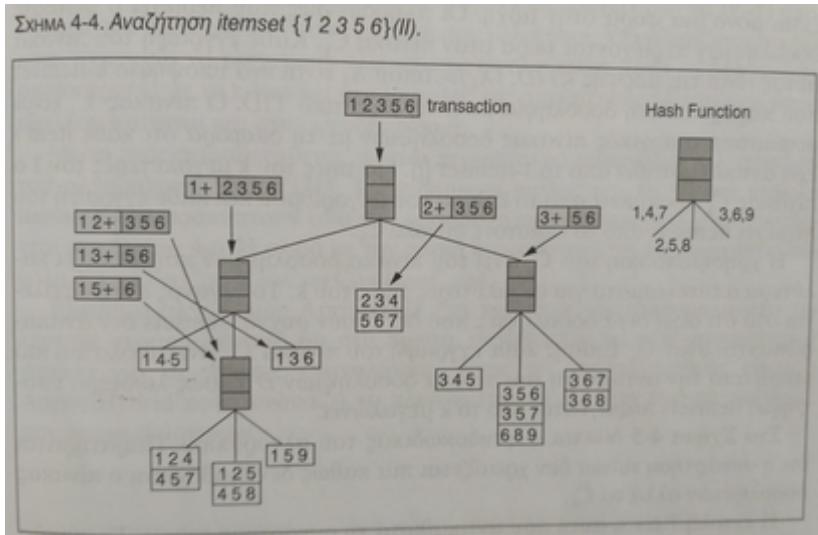
Η συνάρτηση **subset** έχει αναλάβει το πιο δύσκολο έργο του αλγορίθμου. Πρέπει να υπολογίζει ποια υποψήφια itemsets περιέχονται σε κάθε transaction, για κάθε “πέρασμα” που κάνει ο Apriori αλγόριθμος. Γίνεται αντιληπτό, ότι για το σκοπό αυτόν πρέπει τα υποψήφια itemsets να αποθηκεύονται με τέτοιον τρόπο ώστε να επιταχύνεται η όλη διαδικασία.[3] Οπότε, συνοπτικά να αναφερθεί ότι υπάρχει ανάγκη για χρήση δομών δεδομένων, όπως δέντρων και πινάκων κατακερματισμού (hash-table), για την όλη διαχείριση των itemsets, ώστε μαζί με την ευκολία αποθήκευσης και αναζήτησης, παράλληλα να πετυχαίνουμε και μεγαλύτερη απόδοση.

ΣΧΗΜΑ 4-2. Αποθήκευση itemset {1 5 9}.



Σχήμα 3. Πως γίνεται σχηματικά η αποθήκευση itemset με χρήση δέντρου [3]

ΣΧΗΜΑ 4-4. Αναζήτηση itemset {1 2 3 5 6}(II).



Σχήμα 4. Πως γίνεται σχηματικά η αναζήτηση itemset με χρήση δέντρου. [3]

#### 1.4. Παραγωγή κανόνων συσχετίσεων από στοιχειοσύνολα του Apriori

Εφόσον έχουμε βρει σε ένα δείγμα (πχ ένα πλήθος από καλάθια αγορών ενός supermarket) όλα τα συχνά στοιχειοσύνολα από τον αλγόριθμο του Apriori, είναι εφικτό να υπολογίσουμε τις σχέσεις μεταξύ των στοιχειοσυνόλων και να βρούμε εν τέλει κανόνες συσχέτισης. Για παράδειγμα, αν έχουμε βρει ότι ένα συχνό 3-itemset αποτελείται από {“βούτυρο”, “τυρί”, “ψωμί”} με support 22%, ξέρουμε επίσης ότι έχουμε όλα τα αντίστοιχα support των υπο-στοιχειοσυνόλων αυτού του στοιχειοσυνόλου. Που σημαίνει ότι μπορούμε να υπολογίσουμε τις 6 διαφορετικές σχέσεις που προκύπτουν με όλους τους δυνατούς συνδυασμούς, δηλαδή τις σχέσεις: {“βούτυρο”, “τυρί”} => {“ψωμί”}, {“βούτυρο”, “ψωμί”} => {“τυρί”}, {“ψωμί”, “τυρί”} => {“βούτυρο”}, {“ψωμί”} => {“βούτυρο”, “τυρί”}, {“τυρί”} => {“βούτυρο”, “ψωμί”} και {“βούτυρο”} => {“ψωμί”, “τυρί”}. Μέσα από τα βήματα του apriori, γνωρίζουμε τα support όλων των παραπάνω υπο-στοιχειοσυνόλων, οπότε εφαρμόζοντας τον τύπο: “ $\text{Confidence}[X \rightarrow Y] = \text{support}[XUY]/\text{support}[X]$ ” μπορούμε να βρούμε όλα τα αντίστοιχα confidence. Άρα, εφόσον αξιοποιήσουμε μόνο τις “ισχυρές” σχέσεις μπορούμε ουσιαστικά να βγάλουμε καλούς κανόνες συσχέτισης.

Υλοποίηση εφαρμογής που με βάση τον **A-priori** παράγει κανόνες συσχέτισης. (6/20)

## 2. Σύντομη περιγραφή υλοποίησης

Στα πλαίσια της εργασίας αυτής, αναπτύχθηκε μια **desktop εφαρμογή** η οποία είναι σε θέση να διαβάζει από την **mongoDB transactions** (πχ καλάθια αγορών) και να εφαρμόζει τον αλγόριθμο του **Apriori** σε αυτά, ώστε να βρίσκει **συχνά στοιχειοσύνολα** και στην συνέχεια να παράγει **κανόνες συσχέτισης**. Χρησιμοποιήσαμε σαν input της εφαρμογής την mongoDB και όχι να δέχεται πχ κάποιο text αρχείο, διότι θέλαμε να δούμε να λειτουργεί η εφαρμογή σε όσο πιο πραγματικές συνθήκες γίνεται. ‘Ωστε για παράδειγμα να είναι σε θέση η εφαρμογή να δέχεται σαν είσοδο ακόμη και **εκατομμύρια transactions** μέσα από μια **NoSQL** βάση δεδομένων, όπως θα μπορούσε να συμβαίνει και σε μια πραγματική εμπορική εφαρμογή.

Στο διάστημα ανάπτυξης του λογισμικού, φτιάχτηκε υποστηρικτικά και ένας **generator**, που με βάση ένα αρχείο κειμένου με **κανόνες συσχέτισης**, δημιουργεί ένα σύνολο από **transactions** (πχ καλάθια αγορών) και τα **καταχωρεί** σε ένα collection της **MongoDB**. Ο σκοπός ύπαρξης της “γεννήτριας” είναι το γεγονός ότι για να λειτουργήσει το κύριο μέρος της εφαρμογής (δηλαδή ο Apriori και η δημιουργία κανόνων συσχέτισης), χρειάζεται δεδομένα ώστε να μπορεί να βρεί συχνά στοιχειοσύνολα και κατ’ επέκταση κανόνες συσχέτισης.

Μια εικόνα για το πως φαίνεται η καρτέλα του Generator, μέσα από την εφαρμογή, ενώ δημιουργεί

1.000.000 transactions, με βάση ένα αρχείο με κανόνες συσχέτισης:

(Φαίνονται disable οι δυνατότητες του χρήστη, διότι είναι απασχολημένη η εφαρμογή...)

The screenshot shows the 'Generator' application window titled 'Π.Μ & Μ.Γ (Apriori + Association rules)'. The interface includes fields for 'DB Name' (AprioriSamples), 'Collection Name' (transactions), 'Drop Collection', 'Number of Samples' (1,000,000), 'Load File', and 'Apply!' buttons. On the left, a list of items with their respective support values is displayed, such as Coffee # 0.75, Sugar # 0.70, Milk # 0.60, Corn flakes # 0.52, Washing powder # 0.39, Fabric softener # 0.28, Baby wipes # 0.40, Baby nappies # 0.44, Cleanser # 0.43, Bleach Cleaner # 0.31, Butter # 0.35, ham # 0.40, Cheese # 0.40, Bread # 0.45, Paper # 0.45, Cookies # 0.45, Fish # 0.38, Oil # 0.35, Wine # 0.32, beer # 0.58, apples # 0.30, Coke # 0.48, Pasta # 0.39, Meat # 0.44, Toothpaste # 0.30, followed by a separator line and then a list of frequent itemsets like Coffee # Sugar # 0.82 # 0.82, Sugar # Coffee # 0.82 # 0.65, Milk # Corn flakes # 0.74 # 0.62, Corn flakes # Milk # 0.74 # 0.75, etc. At the bottom, log output indicates the generator file loaded without errors, 10000 samples created for MongoDB, and various MongoDB statistics: Mongo-serverUsed:127.0.0.1:27017, Mongo-ns:AprioriSamples.transactions, Mongo-Count:10020, Mongo-size:2219328, Mongo-avgObjSize:221, Mongo-storageSize:2793472.

Διακρίνεται στο παραπάνω screenshot, ότι μπορούμε να συμπληρώσουμε το όνομα της βάσης και του collection. Επίσης, μπορούμε πατώντας το κουμπί “Load File”, να φορτώσουμε ένα text αρχείο

με κανόνες συσχέτισης και πατώντας το κουμπί “Apply!” να δημιουργηθούν τόσα transactions όσα έχουν συμπληρωθεί στο αντίστοιχο πεδίο που φαίνεται δίπλα στο κουμπί “Load File”. Επίσης, για τις ανάγκες των διαφόρων δοκιμών, υπάρχει και ένα κουμπί “Drop collection” που διαγράφει το collection από την βάση.

*Μια εικόνα για το πως φαίνεται η καρτέλα με το κύριο μέρος της εφαρμογής:*

```

***** RESULTS *****
Limit = 0.2
$$$$$$$$$$$$$$$$
Evo (1) We found
Description: {Support_Group} Found 553 times, Support: 0.15077117 [Less than a limit :()]
Description: {Social_Club} Found 655 times, Support: 0.18805628 [Less than a limit :()]
Description: {Professional} Found 1130 times, Support: 0.32443297 [Greater than the limit :()]
Description: {Family} Found 1758 times, Support: 0.38989377 [Greater than the limit :()]
Description: {Hobbies} Found 1045 times, Support: 0.3000207 [Greater than the limit :()]
Description: {Religious} Found 1458 times, Support: 0.41860464 [Greater than the limit :()]
Description: {Political} Found 327 times, Support: 0.09388456 [Less than a limit :()]
Evo (2) We hold
Description: {Professional} {Religious}
Description: {Family} {Religious}
Description: {Family} {Professional}
Description: {Family} {Hobbies}
Description: {Hobbies} {Professional}
Description: {Hobbies} {Religious}
Evo (3) We found
Description: {Professional} {Religious} Found 432 times, Support: 0.12403101 [Less than a limit :()]
Description: {Family} {Religious} Found 782 times, Support: 0.22451909 [Greater than the limit :()]
Description: {Family} {Professional} Found 454 times, Support: 0.13034374 [Less than a limit :()]
Description: {Family} {Hobbies} Found 651 times, Support: 0.18690784 [Less than a limit :()]
Description: {Hobbies} {Professional} Found 335 times, Support: 0.09618145 [Less than a limit :()]
Description: {Hobbies} {Religious} Found 832 times, Support: 0.23887454 [Greater than the limit :()]
Evo (4) We hold
Description: {Family} {Hobbies} {Religious}
Evo (5) We found
Description: {Family} {Hobbies} {Religious} Found 539 times, Support: 0.15475164 [Less than a limit :()]
Evo (6) We hold
$$$$$$$$$$$$$$$$
number Of samples = 3483
Classified:
$$$$$$$$$$$$$$$$
Description: {Family} {Religious} Found 782 times, Support: 0.22451909
Description: {Hobbies} {Religious} Found 832 times, Support: 0.23887454
Description: {Hobbies} Found 1045 times, Support: 0.3000207
Description: {Professional} Found 1130 times, Support: 0.32443297
Description: {Family} Found 1758 times, Support: 0.38989377
Description: {Religious} Found 1458 times, Support: 0.41860464
$$$$$$$$$$$$$$$$
A Priori finished!!!
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Colleration Rules: Support(min)= 0.2 Confidence(min)= 0.6
{Hobbies} -> {Religious} Support= 0.2389 , Confidence= 0.7962 , Lift= 1.902 , Conviction= 2.852
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Colleration Rules: Support(min)= 0.2 Confidence(min)= 0.5
{Religious} -> {Family} Support= 0.2245 , Confidence= 0.5364 , Lift= 1.3756 , Conviction= 1.3159
{Religious} -> {Hobbies} Support= 0.2389 , Confidence= 0.5706 , Lift= 1.902 , Conviction= 1.6303
{Family} -> {Religious} Support= 0.2245 , Confidence= 0.5758 , Lift= 1.3756 , Conviction= 1.3707
{Hobbies} -> {Religious} Support= 0.2389 , Confidence= 0.7962 , Lift= 1.902 , Conviction= 2.852

```

Διακρίνεται στο παραπάνω screenshot, ότι μπορούμε και εδώ να συμπληρώσουμε το όνομα της βάσης και του collection. Επίσης, μπορούμε πατώντας το κουμπί “Apply Apriori”, να εφαρμόσουμε τον Apriori στα δεδομένα του collection, όπου σαν κατώφλι για τα υποψήφια στοιχειούντα λα είναι η τιμή που έχει συμπληρωθεί στο πεδίο “limit”. Στην συνέχεια, μπορούμε πατώντας το κουμπί “Find association rules” να βρούμε κανόνες συσχέτισης, εφόσον ικανοποιούν τα όρια που έχουμε συμπληρώσει σαν ελάχιστα για support και confidence. Το κουμπί με το σήμα του TEI Αθήνας, απλά εμφανίζει ένα παράθυρο με πληροφορίες σχετικές με την εργασία. Τέλος, τα checkboxes χρησιμοποιούνται, για να επιλέγει ο χρήστης τι ακριβώς θέλει να βλέπει σαν output.

Αξίζει επίσης να σημειωθεί, ότι τα περάσματα του apriori γίνονται με παράλληλο τρόπο, αξιοποιώντας όλα τα φυσικά thread που έχει το μηχάνημα στο οποίο τρέχει η εφαρμογή. Για παράδειγμα, αν έχουμε ένα δείγμα με 1.000.000 transactions, και η εφαρμογή τρέχει σε μηχάνημα με 4 πυρήνες-threads, ο κάθε πυρήνας θα διαβάζει από 250.000 σε κάθε πέρασμα. Δόθηκε έμφαση στο σημείο αυτό, ώστε να έχουμε την καλύτερη δυνατή απόδοση, αφού η μεγαλύτερη καθυστέρηση του αλγορίθμου έγκειται στο διάβασμα από την βοηθητική μνήμη.

### 3. Τεχνικά χαρακτηριστικά της υλοποίησης

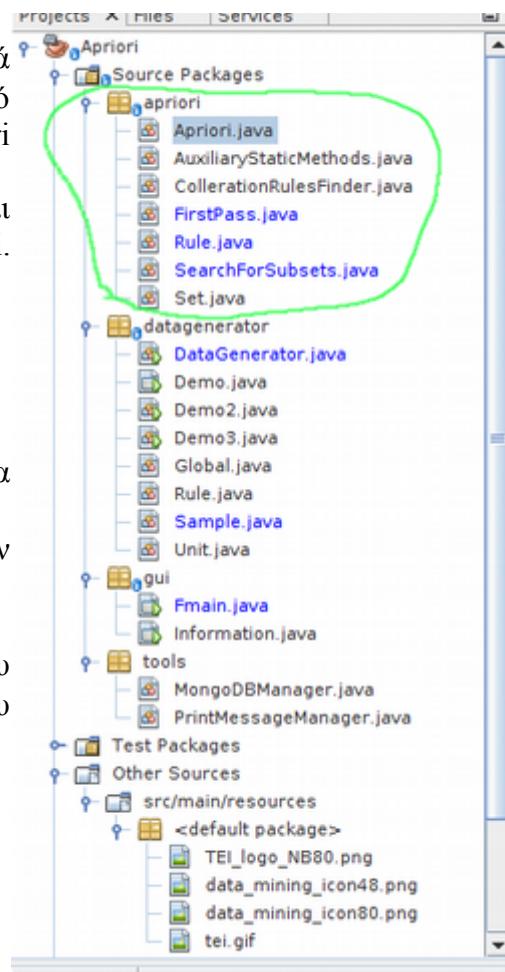
Για να υλοποιήσουμε την εφαρμογή χρησιμοποιήσαμε την γλώσσα προγραμματισμού **Java** (8) και αναπτύξαμε τον κώδικα μέσα από το περιβάλλον του **Netbeans IDE** (8.2). Επίσης, για τις ανάγκες του versioning αλλά και την παράλληλης ανάπτυξης του project από δύο άτομα, χρησιμοποιήσαμε το **git** (Ιδιωτικό project στο <https://bitbucket.org/>). Ακόμη, να αναφερθεί ότι η εφαρμογή έγινε σε **Maven Project**.

Ο λόγος επιλογής της γλώσσας Java ήταν διότι δίνει πολλές δυνατότητες στον προγραμματιστή με τα έτοιμα “εργαλεία” που έχει. Όπως για παράδειγμα, έχει έτοιμες δομές δεδομένων (HashMap, Tree, List κ.ά), βιβλιοθήκες για γρήγορη ανάπτυξη GUI και γενικά “εργαλεία” ώστε να κάνει κάποιος προγραμματιστής το οτιδήποτε (πχ διαχείριση αρχείων στο δίσκο, πρόσβαση σε βάσεις δεδομένων κ.ά), πολύ εύκολα και γρήγορα.

Φτιάχτηκαν 4 πακέτα:

- apriori: Έχει αρχεία Java που αφορούν το Apriori και την δημιουργία κανόνων συσχέτισης
- datagenerator: Έχει αρχεία Java που αφορούν το κομμάτι της γεννήτριας
- gui: Έχει αρχεία Java που αφορούν το γραφικό περιβάλλον της εφαρμογής
- tools: Έχει δύο Java αρχεία, 1 για διαχείριση της βάσης και 1 για την διαχείριση μηνυμάτων

- Μέσα από αυτό το κείμενο, θα παρουσιαστούν τα πιο σημαντικά σημεία της εφαρμογής, όπου αφορούν το καθαρά αλγορίθμικό κομμάτι, δηλαδή τους κώδικες που υλοποιούν τον apriori αλγόριθμο και δημιουργούν τους κανόνες συσχετίσεων. Ως εκ τούτου, τα κομμάτια κώδικα που ακολουθούν βρίσκονται όλα στο πακέτο apriori.



Πολύ συνοπτικά το πακέτο apriori έχει τα εξής αρχεία-κλάσεις:

- Apriori.java: Έχει static μεθόδους για τον αλγόριθμο apriori
- AuxiliaryStaticMethods: Έχει βοηθητικούς static μεθόδους
- CollerationRulesFinder: Έχει static μεθόδους για την δημιουργία των κανόνων συσχέτισης.
- FirstPass: Κλάση (έχει κληρονομήσει την Thread), που κάνει την διαδικασία του πρώτου περάσματος του Apriori
- Rule: Κλάση που χρειάζεται για την διαχείριση των κανόνων
- SearchForSubsets: Κλάση (έχει κληρονομήσει την Thread), που κάνει την διαδικασία όλων των υπόλοιπων περασμάτων του Apriori (εκτός του πρώτου). Πρακτικά υλοποιεί το subset
- Set: Κλάση που χρειάζεται για την διαχείριση των itemsets.

Τα αρχεία που είναι “μπλε” (στην φωτογραφία), είναι έτσι διότι με την βοήθεια του git, το Netbeans μας κάνει διακριτά τα αρχεία που έχουν υποστεί αλλαγές σύμφωνα με το τελευταίο commit. Κάποιο νέο αρχείο θα το εμφάνιζε με πράσινο χρώμα.

## Ακολουθούν κάποια screenshots από τα σημαντικότερα κομμάτια της εφαρμογής (με σχόλια):

Static μέθοδος που εφαρμόζει τον apriori για να βρει συχνά στοιχειοσύνολα:

```
public static HashMap<String,Set> aprioriFinderProcessParallel(MongoDBManager mdbm,float limit,boolean viewEvo, boolean viewSets,boolean viewAllSets)
{
    int processors = Runtime.getRuntime().availableProcessors(); //Πλήθος από threads που έχει το μηχάνημα
    int collerationSize = mdbm.getCollectionSize(); //Πλήθος από transactions του δείγματος μας
    int part = collerationSize/processors ; //Πλήθος από transactions που αναλογούν σε κάθε thread

    //Έκτελείτε το πρώτο βήμα του Apriori αλγόριθμου
    ArrayList<HashMap<String, Set>> alSets = aprioriFirstStep(mdbm,limit,processors,part,collerationSize);
    //Έκτελούνται όλα τα υπόλοιπα βήματα του Apriori αλγορίθμου
    alSets = aprioriEachOtherStep(alSets,mdbm,limit,processors,part,collerationSize);
    //Εμφανίζονται μηνύματα με τα αποτελέσματα του Apriori (με βάση τις επιλογές του χρήστη)
    printAprioriMessages(alSets,limit,processors,collerationSize,viewEvo, viewSets,viewAllSets) ;

    //Ο αλγόριθμος επιστρέφει ένα hashMap με όλα τα διαφορετικά itemsets που βρήκε ο Apriori.
    return AuxiliaryStaticMethods.getOnlyDifferentsHMSets(alSets);
}
```

Static μέθοδος που κάνει την διαδικασία για το πρώτο βήμα του apriori αλγόριθμου:

```
public static ArrayList<HashMap<String, Set>> aprioriFirstStep(MongoDBManager mdbm
    ,float limit,int processors,int part,int numberofsamples)
{
    //Δημιουργείτε ένας Barrier για συγχρονισμό όλων των threads.
    CyclicBarrier barrier = new CyclicBarrier(processors+1);
    //Δημιουργείτε μια λίστα που θα έχει όλα τα itemsets που θα βρεθούν
    ArrayList<HashMap<String, Set>> alSets = new ArrayList<>();

    //Δημιουργείτε μια προσωρινή λίστα για την ανάγκες της παραλληλίας
    //μεταξύ των threads. Πρέπει να λυθεί το θέμα της ταυτόχρονης προσπέλασης.
    ArrayList<HashMap<String,Set>> alTmp = new ArrayList<>() ;
    for(int i=0; i < processors; i++)// Για όσοι είναι οι επεξεργαστές
    {
        //Δημιουργείτε μια HashMap μέσα στην λίστα (μια για κάθε thread)
        alTmp.add(new HashMap<>());
        //Δημιουργείτε ένα αντικείμενο FirstPass και τρέχει σε ξεχωριστό thread
        if (i==processors-1)//Αν είναι το τελευταίο thread, ουσιαστικά παίρνει και
            //το υπόλοιπο της ακέραιας διαίρεσης "collerationSize/processors"...
            new FirstPass(mdbm.getCursor(i*part),alTmp.get(i),barrier).start();
        else
            new FirstPass(mdbm.getCursor(i*part,part),alTmp.get(i),barrier).start();
    }
    try {
        barrier.await(); //Περιμένει το main thread της εφαρμογής τα υπόλοιπα threads που έχουν ανοίξει
    } catch (InterruptedException | BrokenBarrierException ex) {
        Logger.getLogger(Apriori.class.getName()).log(Level.SEVERE, null, ex);
    }
    //Γίνεται merge όλων των HashMap που έχει η προσωρινή λίστα (υπήρχε θέμα ταυτόχρονης προσπέλασης
    //από τα threads...) και γεμίζει η λίστα με τα itemsets, με ένα hashMap με όλα τα itemsets που βρέθηκαν
    alSets.add(AuxiliaryStaticMethods.merge(alTmp));
    //Υπολογίζονται τα support, με βάση το πλήθος των transactions
    AuxiliaryStaticMethods.calcAllSupports(alSets.get(0), numberofsamples);
    int step = 1;
    //Έκτελείτε η aprioriGen ώστε να βρεθούν τα υποψήφια στοιχειοσύνολα για το
    //επόμενο βήμα και στην συνέχεια, προσθέτετε μια HashMap με τα itemsets
    //στην λίστα με τα sets (σαν δεύτερο HashMap της).
    alSets.add(aprioriGen(alSets.get(0), limit, step));

    //Επιστρέφετε η λίστα με τα Sets
    return alSets ;
}
```

Κώδικας για υπολογισμό των 1-itemset, στο πρώτο πέρασμα του apriori αλγόριθμου (σε άλλο thread):

```

private HashMap<String, Set> hmSets ;
CyclicBarrier barrier ;

public FirstPass(DBCursor cursor, HashMap<String, Set> hmSets, CyclicBarrier barrier) {
    this.cursor = cursor;
    this.hmSets = hmSets;
    this.barrier = barrier ;
}

@Override
public void run() { //Όταν ξεκινάει το ξεχωριστό thread
    int numberOfSamples = 0;
    //Με την βοήθεια ενός cursor, προσπελαύνουμε όλες τις σειρές του
    //collection που αναλογούν στο συγκεκριμένο thread.
    while (cursor.hasNext()) {
        //Δημιουργείται ένα ένα DBObject για το τρέχων transaction
        BasicDBObject transaction = (BasicDBObject) cursor.next();
        //Εάν ουσιαστικά το transaction δεν έχει στοιχεία
        if (transaction.getString("transaction").length()==0)
            continue; //Γίνεται skip η όλη διαδικασία και συνεχίζουμε στο επόμενο transaction
        //Γίνεται split των περιεχόμενου από το πεδίο basket, με βάση το χαρακτήρα #
        //και τα "κομμάτια" μπαίνουν μέσα στον πίνακα tokens (πχ 1#2#3# => 3 tokens [1,2,3])
        String[] tokens = transaction.getString("transaction").split("#");
        for (String token : tokens) { //Για κάθε token
            if (numberOfSamples<=100) //Τα πρώτα 101 transactions γίνονται print.
                PrintMessageManager.print(token + " | " ,PrintMessageManager.MPT.apriori);
            token = "{" + token + "}"; // Το token, μορφοποιείται κατάλληλα
            //Εάν η HashMap μας έχει ήδη το token
            if (hmSets.containsKey(token)) {
                hmSets.get(token).plusOneToTotal(); //Αυξάνει το συγκεκριμένο itemset κατά 1
            } else { //αλλιώς
                hmSets.put(token, new Set(token)); //Δημιουργεί και προσθέτει στην HashMap,
                //το καινούριο 1-itemset που βρέθηκε.
            }
        }
        if (numberOfSamples<=100) //Για τα πρώτα 101 transactions αλλάζουμε γραμμή για το print
            PrintMessageManager.println(PrintMessageManager.MPT.apriori);
        numberOfSamples++; //Αυξάνεται ο αριθμός των δειγμάτων κατά 1.
    }
    try {
        barrier.await(); // Περιμένει ο barrier για συγχρονισμό μεταξύ των threads.
    } catch (InterruptedException | BrokenBarrierException ex) {
        Logger.getLogger(Apriori.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

H static μέθοδος aprioriGen (των apriori αλγορίθμου):

```

public static HashMap<String, Set> aprioriGen(HashMap<String, Set> hmSets, float limit, int step) {
    //Η συνάρτηση aprioriGen κάνει join σε πρώτη φάση, για να βρει υποψήφια
    //στοιχειοσύνολα με βάση ένα όριο.
    HashMap<String, Set> hmResSets = joinStep(hmSets,limit,step) ;
    //Σε δεύτερη φάση "κλαδεύονται" τα στοιχειοσύνολα που έχουν υπό-στοιχειοσύνολα
    //με support μικρότερο από το καθορισμένο όριο.
    if (step>1) //Παραλείπεται αυτό το βήμα ουσιαστικά όταν τα στοιχειοσύνολα
        //είναι 2 στοιχείων και πρακτικά δεν υπάρχει λόγος να εκτελεστεί...
        pruneStep(hmSets,hmResSets);

    //Επιστρέφεται ένα HashMap με όλα τα νέα υποψήφια itemsets.
    return hmResSets;
}

```

Τα δύο βήματα της *aprioriGen* (*JoinStep* και *pruneStep*):

```

public static HashMap<String, Set> joinStep(HashMap<String, Set> hmSets, float limit, int step){
    //Δημιουργείται μια λίστα από την HashMap των itemsets.
    ArrayList<Set> alAllSets = new ArrayList<>(hmSets.values());
    //Γίνεται ταξινόμηση της λίστας με τα itemsets, με βάση το support.
    Global.quickSortSet(alAllSets, 0, alAllSets.size() - 1);

    //Δημιουργείται μια HashMap που θα πάρει τα όποια υποψήφια itemsets
    HashMap<String, Set> hmResSets = new HashMap<>();

    for (int i = 0; i < alAllSets.size(); i++) {
        Set set = alAllSets.get(i);
        //Εάν το itemset έχει support μικρότερο από το limit
        if (set.getSupport() < limit) {
            set.setTag("[Less than a limit :( ]"); //Το itemset χαρακτηρίζεται ανάλογα
            continue; //πάμε στο επόμενο βήμα του loop, χωρίς να συνεχιστεί η ροή του loop...
        }
        //Το itemset χαρακτηρίζεται ότι έχει support μεγαλύτερο από το limit.
        set.setTag("[Greater than the limit :) ]");

        for (int j = i+1; j <alAllSets.size(); j++) { //Από το επόμενο itemset μέχρι το τέλος της λίστας
            int foundDifferents = 0;
            if (step>1){//Εάν το βήμα του apriori είναι μεγαλύτερο από 1
                //δηλαδή τα k-itemsets που έχουμε είναι της τάξης k=3 και πάνω...
                for(String one : set.getHashDescriptions()){ //Για κάθε description του itemset
                    boolean foundOneSame = false;

                    //Για όσα είναι τα descriptions του δεύτερου itemset που πάμε να ενώσουμε
                    for(String two : alAllSets.get(j).getHashDescriptions()){
                        if (one.equals(two)){//Εάν βρούμε ίδιο description
                            foundOneSame = true; //Γίνεται true
                            break; //και βγαίνουμε από το loop
                        }
                    }
                    //Εάν δεν έχουμε το ίδιο description και στα δύο itemsets
                    if (!foundOneSame)
                        //ο μετρητής foundDifferents αυξάνεται κατά 1
                        if (foundDifferents++>1)//Εάν είναι πάνω από 1
                            break; //κάνουμε break γιατί δεν έχουν νόημα οι επαναλήψεις
                }
            }
            else
                foundDifferents = 1; //δίνουμε 1 απλά για να ταχύσει η παρακάτω
            //συνθήκη όταν το step είναι μικρότερο ή ίσο με 1.

            //Εφόσον τα δύο itemsets έχουν όλα τα descriptions του κοινά εκτός από ένα μόνο
            //διαφορετικό (πχ αποδεκτό: {1,2,3} & {1,2,4}, αλλά όχι πχ {1,2,3} & {1,4,5})
            if (foundDifferents==1)
            {
                //Δημιουργείται ένα TreeSet
                TreeSet<String> hmTmp = new TreeSet<>();
                //Προσθέτουμε όλα τα descriptions του πρώτου itemset
                hmTmp.addAll(set.getHashDescriptions());
                //Προσθέτουμε όλα τα descriptions του δεύτερου itemset
                hmTmp.addAll(alAllSets.get(j).getHashDescriptions());
                //Το TreeSet, δεν παίρνει διπλότυπα, όποτε πχ αν προσθέσουμε
                //{1,2,3} και {1,2,4}, το TreeSet θα έχει {1,2,3,4}.

                //Δημιουργείται ένα αντικείμενο τύπου Set, με τα παραπάνω descriptions
                Set newSet = new Set(hmTmp);
                //Εάν η HashMap μας δεν έχει ήδη το συγκεκριμένο description
                if (!hmResSets.containsKey(newSet.getDescription()))
                    //προσθέτουμε στην HashMap το νέο υποψήφιο itemset
                    //με "key" το Description και "value" το αντικείμενο newSet
                    hmResSets.put(newSet.getDescription(), newSet);
            }
        }
    }
    //Επιστρέφεται το HashMap με όλα τα νέα υποψήφια itemsets.
    return hmResSets;
}

```

```

public static void pruneStep(HashMap<String, Set> hmPreviousSets,HashMap<String, Set> hmSets){
    //Για κάθε υποψήφιο k-itemset ελέγχεται αν έχει έστω και ένα (k-1)-itemset μη συχνό, ώστε να "κλαδευτεί"
    //Υπάρχει η ανάγκη για iterator, για να μπορούμε να διαγράφουμε στοιχεία μέσα σε loop...
    for(Iterator<HashMap.Entry<String, Set>> it = hmSets.entrySet().iterator(); it.hasNext(); ) {
        HashMap.Entry<String, Set> entry = it.next();
        if (entry.getValue().toPrune(hmPreviousSets)) {//Ελεγχός για το αν πρέπει να "κλαδευτεί"
            it.remove(); //Διαγράφεται από την hashMap
        }
    }
}

```

Υλοποίηση εφαρμογής που με βάση τον **A-priori** παράγει κανόνες συσχέτισης. (12/20)

Η μέθοδος `toPrune` της κλάσης `Set`, η οποία καλείται από την `pruneStep`:

```
/*
  Η toPrune, δημιουργεί όλα τα (k-1)-itemsets ενός k-itemset
  και εξετάζει αν βρίσκονται μέσα σε μια HashMap (Η οποία έχει όλα
  τα συχνά (k-1)-itemsets).
  Αν κάποιο (k-1)-itemset του k-itemset δεν είναι συχνό, τότε γίνεται
  break και επιστρέφεται true, αλλιώς αν όλα τα (k-1)-itemsets είναι
  συχνά, επιστρέφεται false.
*/
public boolean toPrune(HashMap<String, Set> hmSets)
{
    boolean result = false ;
    int index ;

    for (int i = 0; i < descriptions.size();i++){
        Set tmpSet = new Set() ;
        index= 0 ;
        for (String desc : descriptions){
            if (index!=i)
                tmpSet.addDescription(desc) ;
            index++ ;
        }
        if (!hmSets.containsKey(tmpSet.getDescriptions())){
            result = true ;
            break ;
        }
    }
    return result ;
}
```

Ενδεικτικά, οι μεταβλητές που έχουν οι κλάσεις `Rule` και `Set`, και δύο από τους μεθόδους τους:

|  |  |
|--|--|
| <pre>public class Set{     private TreeSet&lt;String&gt; descriptions ;     private String description ;     private int total ;     private float support ;     private int number_of_samples ;     private String tag ;      public Set() {         descriptions = new TreeSet&lt;&gt;();     } }</pre>  | <pre>public class Rule {     private Set treatySet ;     private Set effectSet ;     private float confidence ;     private int total ;     private float support ;     private float lift ;     private float conviction ;     private static DecimalFormat df = new DecimalFormat("#.#####");      public Rule() {     }      //Υπολογίζεται το Support     public void calcSupport(int number_of_samples)     {         this.number_of_samples = number_of_samples ;         if (number_of_samples!=0)             support = (float)total / number_of_samples ;         else             support = 0.0F ;     } }</pre> |
| <pre>//Υπολογίζονται για κάθε κανόνα τα confidence, lift και conviction με βάση τους τύπους public void calcAll() {     confidence= (float)total/treatySet.getTotal() ; //The same: support/treatySet.getSupport() ;     lift= support/ (treatySet.getSupport() * effectSet.getSupport()); //confidence/effectSet.getSupport();     conviction = (1-effectSet.getSupport())/(1-confidence) ; }</pre> |  |

*Static μέθοδος που κάνει την διαδικασία που γίνεται σε όλα τα βήματα του apriori αλγόριθμου (εκτός των πρώτου), δηλαδή εκτελεί τις δύο φάσεις subset και aprioriGen:*

```

public static ArrayList<HashMap<String, Set>> aprioriEachOtherStep(ArrayList<HashMap<String, Set>> alSets
        ,MongoDBManager mdbm,float limit,int processors,int part,int numberOfSamples)
{
    //Δημιουργείτε ένας Barrier για συγχρονισμό όλων των threads.
    CyclicBarrier barrier = new CyclicBarrier(processors+1);
    int indexOfSubsetsWeHold = 1; //Άρχικοποιείται ο δείκτης για το HashMap των υποψήφιων itemsets
    int indexOfHMSets = 2; //Άρχικοποιείται ο δείκτης για το HashMap των sets που θα βρεθούν
    int step = 1 ; // Το βήμα ορίζεται στην αρχή 1
    ArrayList<HashMap<String,Set>> alTmp ; //Δηλώνεται μια προσωρινή λίστα για την ανάγκες της παραλληλίας
    //μεταξύ των threads. Πρέπει να λυθεί το θέμα της ταυτόχρονης προσπέλασης.
    while (!alSets.get(indexOfSubsetsWeHold).isEmpty()) { //Εφόσον το "τρέχων" HashMap με τα υποψήφια
        //itemsets δεν είναι άδειο
        step += 2; //Το βήμα γίνεται +2

        alTmp = new ArrayList<>(); //Δημιουργείται στην μνήμη η προσωρινή λίστα
        for(int i=0; i < processors; i++) //Για όσα είναι τα threads
        {
            //Δημιουργείτε μια HashMap μέσα στην λίστα (μια για κάθε thread)
            alTmp.add(new HashMap<>());
            //Δημιουργείτε ένα αντικείμενο SearchForSubsets και τρέχει σε ξεχωριστό thread
            if (i==processors-1)//Αν είναι το τελευταίο thread, ουσιαστικά παίρνει και
                //το υπόλοιπο της ακέραιας διαίρεσης "collerationSize/processors"...
            new SearchForSubsets(mdbm.getCursor(i*part),alSets.get(indexOfSubsetsWeHold), alTmp.get(i),barrier).start();
            else
                new SearchForSubsets(mdbm.getCursor(i*part,part),alSets.get(indexOfSubsetsWeHold), alTmp.get(i),barrier).start();
        }
        try {
            barrier.await(); //Περιμένει το main thread της εφαρμογής τα υπόλοιπα threads που έχουν ανοίξει
        } catch (InterruptedException | BrokenBarrierException ex) {
            Logger.getLogger(Apriori.class.getName()).log(Level.SEVERE, null, ex);
        }
        //Γίνεται merge όλων των HashMap που έχει η προσωρινή λίστα (υπήρχε θέμα ταυτόχρονης προσπέλασης
        //από τα threads...) και γεμίζει η λίστα με τα itemsets, με ένα hashMap με όλα τα itemsets που βρέθηκαν
        alSets.add(AuxiliaryStaticMethods.merge(alTmp));
        //Υπολογίζονται τα support, με βάση το πλήθος των transactions
        AuxiliaryStaticMethods.calcAllSupports(alSets.get(indexOfHMSets), numberOfSamples);
        //Εκτελείτε η aprioriGen ώστε να βρεθούν τα υποψήφια στοιχειούντα για το
        //επόμενο βήμα και στην συνέχεια, προσθέτετε μια HashMap με τα itemsets
        //στην λίστα με τα sets.
        alSets.add(aprioriGen(alSets.get(indexOfHMSets), limit, step));
        //Αλλάζουν οι δείκτες της λίστας με τα itemsets
        indexOfSubsetsWeHold += 2;
        indexOfHMSets += 2;
    }
    //Εφόσον δεν έχουμε άλλα υποψήφια itemsets και έχουν τελειώσει οι
    //επαναλήψεις, επιστρέφεται μια λίστα με HashMap, που έχουν όλη την
    //εξέλιξη του αλγορίθμου και παράλληλα όλα τα συχνά στοιχειούντα...
    return alSets ;
}

```

*Κώδικας για την υλοποίηση του subset του apriori αλγόριθμου (σε ξεχωριστό thread):  
[Διακρίνεται η χρήση TreeSet και HashMap...]*

```

@Override
public void run() { //Όταν ξεκινάει το ξεχωριστό thread
    //Με την βοήθεια ενός cursor, προσπελαύνουμε όλες τις σειρές του
    //collection που αναλογούν στο συγκεκριμένο thread.
    while (cursor.hasNext()) {
        //Δημιουργείται ένα ένα DBObject για το τρέχων transaction
        BasicDBObject transaction = (BasicDBObject) cursor.next();
        //Γίνεται split του περιεχόμενου από το πεδίο basket, με βάση το χαρακτήρα #
        //και τα "κομμάτια" μπαίνουν μέσα στον πίνακα tokens (πχ 1#2#3# => 3 tokens [1,2,3])
        String[] tokens = transaction.getString("transaction").split("#");
        //Δημιοργείται ένα TreeSet
        TreeSet<String> hsTokens = new TreeSet<>();
        //Προσθέτουμε στο TreeSet όλα τα token του transaction
        for (int i = 0; i < tokens.length; i++) {
            hsTokens.add("{ " + tokens[i] + " }");
        }

        for (String key : hmSubsetsWeHold.keySet()) { //Για όλα τα υποψήφια itemsets
            boolean isThere = true;
            for (String desc : hmSubsetsWeHold.get(key).getHashDescriptions()) {
                //Ψάχνουμε να ζούμε αν το transaction έχει όλα τα description του υποψήφιου itemset
                if (hsTokens.contains(desc)) {
                    continue;
                }
                //Αν δεν έχει κάποιο description, σημαίνει ότι το υποψήφιο itemset δεν
                //υπάρχει μέσα στο συγκεκριμένο transaction, οπότε κάνουμε το isThere=false
                isThere = false;
                //και break από το for γιατί δεν υπάρχει λόγος να συνεχιστούν οι επαναλήψεις.
                break;
            }
            //Εάν μετά τις επαναλήψεις του προηγούμενου for έχουμε βρει ότι υπάρχει το
            //υποψήφιο itemset μέσα στο transaction
            if (isThere) {
                //Κοιτάμε αν έχουμε ήδη το υποψήφιο itemset μέσα στην HashMap μας
                if (hmSets.containsKey(hmSubsetsWeHold.get(key).getDescriptions())) {
                    //Και αν υπάρχει, αυξάνουμε κατά 1 τον "μετρητή" του συγκεκριμένου itemset
                    hmSets.get(hmSubsetsWeHold.get(key).getDescriptions()).plusOneToTotal();
                } else {
                    //αλλιώς δημιουργούμε και προσθέτουμε στην HashMap, το καινούριο itemset που βρέθηκε για πρώτη φορά
                    hmSets.put(hmSubsetsWeHold.get(key).getDescriptions(), new Set (hmSubsetsWeHold.get(key).getHashDescriptions()));
                }
            }
        }
        try {
            barrier.await(); // Περιμένει ο barrier για συγχρονισμό μεταξύ των threads.
        } catch (InterruptedException | BrokenBarrierException ex) {
            Logger.getLogger(SearchForSubsets.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

---



---

Για την δημιουργία association rules, υπήρχε η ανάγκη να αναπτυχθούν κάποιες static μέθοδοι, οι οποίοι αναλαμβάνουν να δημιουργούν όλους τους δυνατούς συνδυασμούς, ώστε σε επόμενη φάση να είναι σε θέση η εφαρμογή να παράγει κανόνες συσχετίσεων.

Για την καλύτερη κατανόηση της λειτουργίας αυτών των μεθόδων, ακολουθούν δύο απλά παραδείγματα εκτέλεσης, όπου εμφανίζονται και οι έξοδοι που έχουν.

Παράδειγμα χρήσης της static μεθόδου *clock*:

```
public static void main(String args[]) {
    ArrayList<Integer> alist = new ArrayList<>();
    alist.add(1);
    alist.add(2);
    alist.add(3);
    while(true)
    {
        alist = clock(alist,5);
        if (alist==null)
            break;
        for (Integer item : alist)
            System.out.print(item + " | ");
        System.out.println();
    }
}
```

/\* Output:  

|   |   |   |    |
|---|---|---|----|
| 1 | 2 | 4 |    |
| 1 | 2 | 5 |    |
| 1 | 3 | 4 |    |
| 1 | 3 | 5 |    |
| 1 | 4 | 5 |    |
| 2 | 3 | 4 |    |
| 2 | 3 | 5 |    |
| 2 | 4 | 5 |    |
| 3 | 4 | 5 | */ |

Παράδειγμα χρήσης της static αναδρομικής μεθόδου *findCombinations*:

```
public static void main(String args[]) {
    ArrayList<ArrayList<Integer>> alist = new ArrayList<>();

    //Βρίσκει όλους τους συνδιασμούς για τα υποσύνολα
    findCombinations(alist,4,0);
    for (ArrayList<Integer> list : alist){
        for (Integer item : list)
            System.out.print(item + " | ");
        System.out.println();
    }
}
```

|   |   |   |   |  |
|---|---|---|---|--|
| 0 | 1 | 2 | 3 |  |
| 0 | 1 | 2 | 4 |  |
| 0 | 1 | 3 | 4 |  |
| 0 | 2 | 3 | 4 |  |
| 1 | 2 | 3 | 4 |  |
| 0 | 1 | 2 |   |  |
| 0 | 1 | 3 |   |  |
| 0 | 1 | 4 |   |  |
| 0 | 2 | 3 |   |  |
| 0 | 2 | 4 |   |  |
| 0 | 3 | 4 |   |  |
| 1 | 2 | 3 |   |  |
| 1 | 2 | 4 |   |  |
| 1 | 3 | 4 |   |  |
| 2 | 3 | 4 |   |  |
| 2 | 3 | 5 |   |  |
| 2 | 4 | 5 |   |  |
| 3 | 4 | 5 |   |  |
|   |   |   | * |  |
| 0 | 1 |   |   |  |
| 0 | 2 |   |   |  |
| 0 | 3 |   |   |  |
| 0 | 4 |   |   |  |
| 1 | 2 |   |   |  |
| 1 | 3 |   |   |  |
| 1 | 4 |   |   |  |
| 2 | 3 |   |   |  |
| 2 | 4 |   |   |  |
| 3 | 4 |   |   |  |
| 0 |   |   |   |  |
| 1 |   |   |   |  |
| 2 |   |   |   |  |
| 3 |   |   |   |  |
| 4 |   |   |   |  |

Οι static μέθοδοι *findCombinations* και *clock*:

```
public static void findCombinations(ArrayList<ArrayList<Integer>> alAllCombinations,int crowd, int limit)
{
    if (limit<crowd)
        findCombinations(alAllCombinations,crowd,limit+1);
    else
        return ;
    ArrayList<Integer> alIndexes = new ArrayList<>();
    for (int i=0;i<limit;i++)
        alIndexes.add(i);
    do
    {
        alAllCombinations.add(alIndexes);
        alIndexes = clock(alIndexes,crowd);
    }while(alIndexes!=null);
}

public static ArrayList<Integer> clock(ArrayList<Integer> alInd, int crowd)
{
    ArrayList<Integer> alIndexes = new ArrayList<>(alInd);
    int lastIndex = alIndexes.size()-1;
    boolean Notfinished = false;
    for(int i=lastIndex, limit = crowd ;i>=0;i--, limit--){
        if (alIndexes.get(i)<limit)
        {
            alIndexes.set(i, alIndexes.get(i)+1);
            for(int j=i+1;j<=lastIndex;j++)
                alIndexes.set(j, alIndexes.get(j-1)+1);
            Notfinished = true;
            break;
        }
    }
    if (Notfinished)
        return alIndexes;
    else
        return null;
}
```

*H static μέθοδος findAllCombinations:*

```
public static ArrayList<ArrayList<ArrayList<Integer>>> findAllCombinations(int crowd)
{
    ArrayList<ArrayList<ArrayList<Integer>>> alAllCombinations = new ArrayList<>();
    for(int i=0; i<crowd;i++)
    {
        alAllCombinations.add(new ArrayList<>());
        findCombinations(alAllCombinations.get(i),i,0) ; //size-1, 0 ;
    }
    return alAllCombinations ;
}
```

*Oι static μέθοδος findRules, η οποία βρίσκει τους κανόνες συσχετίσεων:*

```
public static ArrayList<Rule> findRules(HashMap<String,Set> hmSets,int depth)
{
    return findRules(hmSets, depth,0.0F,0.0F) ;
}

public static ArrayList<Rule> findRules(HashMap<String,Set> hmSets,int depth, float minSupport, float minConfidence)
{
    //Βρίσκουμε μια λίστα, με λίστες όλων των δυνατών συνδυασμών με βάση ένα αριθμό βάθους (depth)
    //μου καθορίζεται με βάση το σύνολο των δημιουργήσανται του apriori
    ArrayList<ArrayList<Integer>> alAllCombinations = findAllCombinations(depth) ;
    //Δημιουργίζεται μια λίστα, όπου θα μπουν οι κανόνες που θα δημιουργήθουν
    ArrayList<Rule> alRules = new ArrayList<>();

    int counter=0 ;

    for(Set set : hmSets.values()) // Για όλα τα στοιχειοσύνολα που έχουμε βρει από τον Apriori
    {
        //Εάν έχουμε support μικρότερο από αυτό που έχουμε θέσει σαν minimum
        //συνεχίζουμε στο επόμενο βήμα του loop, διότι δεν έχει νόημα να
        //φτιάξουμε κανόνες με βάση αυτό το στοιχειοσύνολο.
        if (set.getSupport()<minSupport)
            continue;

        //Βρίσκουμε τον δείκτη για την λίστα με όλους τους δυνατούς συνδυασμούς
        //με βάση τα item που έχει το συγκεκριμένο itemset.
        int combinationIndex = set.getHashDescriptions().size()-1 ;

        if (combinationIndex<1) //Αν το itemset έχει ένα στοιχείο, δεν έχει νόημα να
            continue;           //ψάξουμε για κανόνες

        //Για όλους τους δυνατούς συνδυασμούς
        for(ArrayList<Integer> alIndexes : alAllCombinations.get(combinationIndex))
        {
            Set tmpTreaty = new Set(); //Δημιουργείται το αριστερό itemset του κανόνα
            Set tmpEffect = new Set(); //Δημιουργείται το δεξιό itemset του κανόνα

            //Με βάση τους συνδυασμούς, γεμίζουν με descriptions το αριστερό και
            //δεξιό μέρος του "υποψήφιου" κανόνα.
            int indexDesc = 0;
            for(String desc : set.getHashDescriptions())
            {
                boolean broken = false;
                for(Integer index : alIndexes)
                {
                    if (index==indexDesc)
                    {
                        broken = true;
                        tmpTreaty.addDescription(desc);
                        break;
                    }
                }
                if (!broken)
                    tmpEffect.addDescription(desc);
                indexDesc++;
            }
            //Έφασαν το HashMap με όλα τα συγκά itemsets έχουν τα δύο itemsets(αριστερό και δεξιό) του κανόνα
            if (hmSets.containsKey(tmpTreaty.getDescription()) && hmSets.containsKey(tmpEffect.getDescription())){
                //Δημιουργείται ο κανόνας
                Rule rule = new Rule(hmSets.get(tmpTreaty.getDescription()),hmSets.get(tmpEffect.getDescription()),set.getTotal(),set.getSupport());
                rule.calcAll();
                //Εάν ο κανόνας έχει μεγαλύτερο ή ίσο Confidence και support με αυτά που έχουν οριστεί σαν minimum
                if (rule.getConfidence()>minConfidence && rule.getSupport()>minSupport)
                    alRules.add(rule); //Προοθέτεται ο κανόνας στην λίστα με τους κανόνες
            }
        }
        //Επιστρέφεται η λίστα με τους κανόνες
        return alRules ;
    }
}
```

Υλοποίηση εφαρμογής που με βάση τον **A-priori** παράγει κανόνες συσχέτισης. (17/20)

## 4. Αποτελέσματα εφαρμογής-αλγορίθμου

Για λόγους μέτρησης της αξιοπιστίας της εφαρμογής, το dataset όπου εφαρμόστηκε ο Apriori στο συγκεκριμένο ενδεικτικό παράδειγμα παρουσίασης των αποτελεσμάτων της εφαρμογής, δεν είναι από τον generator, αλλά από το αρχείο “Association\_Rules\_DataSet.csv” που βρίσκεται στο eclass. Τα περιεχόμενα του csv αρχείου μπήκαν προγραμματιστικά στην MongoDB και στα groups, αντί για “1” εκχωρούνταν η αντίστοιχη λέξη με βάση την στήλη (Αν ήταν 0, δεν γινόταν εκχώρηση).

Δηλαδή, τα δεδομένα του παραδείγματος μετατράπηκαν όπως φαίνονται στην τρίτη στήλη:

| Family, Hobbies, Social_Club, Political, Professional, Religious, Support_Group |                  |                                       |
|---|------------------|---------------------------------------|
| 8.71,Short,M,No,53,1,0,0,0,0,0,0  | 1,0,0,0,0,0,0 => | Family#                               |
| 5.24,Medium,F,No,31,0,0,0,0,0,1,1   | 0,0,0,0,0,1,1 => | Religious#Support_Group#              |
| 4.54,Medium,M,Yes,27,1,1,1,0,0,1,0  | 1,1,1,0,0,1,0 => | Family#Hobbies#Social_Club#Religious# |

Τα βήματα που έκανε ο Apriori (μέχρις ότου δεν υπήρχαν άλλα υποψήφια στοιχειοσύνολα)  
[με ελάχιστο όριο support 20%]:

```
***** RESULTS *****  
Limit = 0.2  
$$$$$$$$$$$$$  
Evo (1) We found  
Description: {Support_Group} Found 553 times, Support: 0.15877117 [Less than a limit :()]  
Description: {Social_Club} Found 655 times, Support: 0.18805628 [Less than a limit :()]  
Description: {Professional} Found 1130 times, Support: 0.32443297 [Greater than the limit :)]  
Description: {Family} Found 1358 times, Support: 0.38989377 [Greater than the limit :)]  
Description: {Hobbies} Found 1045 times, Support: 0.3000287 [Greater than the limit :)]  
Description: {Religious} Found 1458 times, Support: 0.41860464 [Greater than the limit :)]  
Description: {Political} Found 327 times, Support: 0.09388458 [Less than a limit :()]  
Evo (2) We hold  
Description: {Professional}{Religious}  
Description: {Family}{Religious}  
Description: {Family}{Professional}  
Description: {Family}{Hobbies}  
Description: {Hobbies}{Professional}  
Description: {Hobbies}{Religious}  
Evo (3) We found  
Description: {Professional}{Religious} Found 432 times, Support: 0.12403101 [Less than a limit :()]  
Description: {Family}{Religious} Found 782 times, Support: 0.22451909 [Greater than the limit :)]  
Description: {Family}{Professional} Found 454 times, Support: 0.1303474 [Less than a limit :()]  
Description: {Family}{Hobbies} Found 651 times, Support: 0.18690784 [Less than a limit :()]  
Description: {Hobbies}{Professional} Found 335 times, Support: 0.09618145 [Less than a limit :()]  
Description: {Hobbies}{Religious} Found 832 times, Support: 0.23887454 [Greater than the limit :)]  
Evo (4) We hold  
Description: {Family}{Hobbies}{Religious}  
Evo (5) We found  
Description: {Family}{Hobbies}{Religious} Found 539 times, Support: 0.15475164 [Less than a limit :()]  
Evo (6) We hold  
$$$$$$$$$$$$$
```

Τα συγνά στοιχειοσύνολα που βρέθηκαν με βάση τον A-priori (με limit support 20%):

```

number Of samples = 3483
Classified::
$$$$$$$$$$$$$$$$
Description: {Family}{Religious} Found 782 times, Support: 0.22451909
Description: {Hobbies}{Religious} Found 832 times, Support: 0.23887454
Description: {Hobbies} Found 1045 times, Support: 0.3000287
Description: {Professional} Found 1130 times, Support: 0.32443297
Description: {Family} Found 1358 times, Support: 0.38989377
Description: {Religious} Found 1458 times, Support: 0.41860464
$$$$$$$$$$$$$$$$
A-Priori finished!!

```

Τα οποία είναι ίδια με αυτά που βρήκε το Rapidminer (Αλγόριθμος FP-Growth) στο ίδιο dataset:

| Result Overview    |      |         |              |         |
|--------------------|------|---------|--------------|---------|
| No. of Sets: 6     | Size | Support | Item 1       | Item 2  |
| Total Max. Size: 2 | 1    | 0.419   | Religious    |         |
|                    | 1    | 0.390   | Family       |         |
| Min. Size: 1       | 1    | 0.324   | Professional |         |
|                    | 1    | 0.300   | Hobbies      |         |
| Max. Size: 2       | 2    | 0.225   | Religious    | Family  |
| Contains Item:     | 2    | 0.239   | Religious    | Hobbies |

Και κανόνες συσχέτισης που βγήκαν με βάση το δείγμα  
(Εκτελέστηκε η εύρεση δύο φορές, μία για min Confidence 0.6 και μία για 0.5):

```

$$$$$$$$$$$$$$$$$$$$$$$$
Colleration Rules: Support(min)= 0.2 Confidence(min)= 0.6
{Hobbies} => {Religious} Support= 0.2389 , Confidence= 0.7962 , Lift= 1.902 , Conviction= 2.8524
$$$$$$$$$$$$$$$$$$$$$$$$
Colleration Rules: Support(min)= 0.2 Confidence(min)= 0.5
{Religious} => {Family} Support= 0.2245 , Confidence= 0.5364 , Lift= 1.3756 , Conviction= 1.3159
{Religious} => {Hobbies} Support= 0.2389 , Confidence= 0.5706 , Lift= 1.902 , Conviction= 1.6303
{Family} => {Religious} Support= 0.2245 , Confidence= 0.5758 , Lift= 1.3756 , Conviction= 1.3707
{Hobbies} => {Religious} Support= 0.2389 , Confidence= 0.7962 , Lift= 1.902 , Conviction= 2.8524
$$$$$$$$$$$$$$$$$$$$$$$$

```

Οι οποίοι είναι ίδιοι με αυτούς που βρήκε το Rapidminer στο ίδιο dataset και με τα ίδια support, Confidence, Lift και Conviction:

| No. | Premises  | Conclusion | Support | Confid... | LaPl... | Gain  | p-s   | Lift  | Conv... |
|-----|-----------|------------|---------|-----------|---------|-------|-------|-------|---------|
| 1   | Religious | Family     | 0.225   | 0.536     | 0.863   | -0.61 | 0.061 | 1.376 | 1.316   |
| 2   | Religious | Hobbies    | 0.239   | 0.571     | 0.873   | -0.59 | 0.113 | 1.902 | 1.630   |
| 3   | Family    | Religious  | 0.225   | 0.576     | 0.881   | -0.55 | 0.061 | 1.376 | 1.371   |
| 4   | Hobbies   | Religious  | 0.239   | 0.796     | 0.953   | -0.36 | 0.113 | 1.902 | 2.852   |

Να σημειωθεί, ότι η εφαρμογή εξετάστηκε και φάνηκε να λειτουργεί αποδοτικά και στις περιπτώσεις που είχαμε εκατομμύρια transactions, με εύρεση (από τον apriori) μεγαλύτερων (σε πλήθος k) συγνών στοιχειοσυνόλων και με παραγωγή πιο σύνθετων κανόνων συσχέτισης.

Υλοποίηση εφαρμογής που με βάση τον **A-priori** παράγει κανόνες συσχέτισης. (19/20)

## 5. Μελλοντικές επεκτάσεις

Ενδεικτικά, κάποιες μελλοντικές προσθήκες στην εφαρμογή, θα μπορούσαν να ήταν οι ακόλουθες:

- ✓ Η εφαρμογή να διαβάζει datasets και με άλλους τρόπους πέραν της MongoDB. Δηλαδή, να είναι σε θέση να διαβάζει text αρχεία και να καθορίζεται μέσα από την εφαρμογή ποιος χαρακτήρας θα είναι ο διαχωριστικός (πχ „,“, “:”, ”#” κ.ά). Επίσης, θα μπορούσε η εφαρμογή να διαβάζει και από παραδοσιακές SQL βάσεις δεδομένων, όπως πχ databases από το DBMS της MySQL.
- ✓ Ο χρήστης της εφαρμογής, να επιλέγει να βλέπει Meta-Data, με βάση το συγκεκριμένο dataset που έχει επιλέξει.
- ✓ Να έχει περισσότερες επιλογές παραμετρικά ο χρήστης της εφαρμογής. Δηλαδή για παράδειγμα, να μπορεί να επιλέξει να κάνει “save as” την έξοδο της εφαρμογής σε κάποιο αρχείο (με όνομα και path ορισμένο από τον ίδιο).

## 6. Συμπεράσματα

Μέσα από αυτήν την εργασία έγιναν γνωστοί (έστω συνοπτικά) οι βασικότεροι αλγόριθμοι και τεχνικές που εφαρμόζονται για την εξόρυξη γνώσης. Επιπλέον, μελετήθηκε σε βάθος ένας κλασσικός αλγόριθμος εξόρυξης δεδομένων ο A-priori και παράλληλα, υπήρξε ενασχόληση με έννοιες του data mining όπως τα “στοιχειοσύνολα” και τους κανόνες συσχετίσεων. Ακόμη, μέσα από την ανάπτυξη λογισμικού που έγινε, όπου αρχικά εφαρμόζεται ο Apriori και στην συνέχεια παράγονται κανόνες συσχετίσεων, φάνηκε έμπρακτα ένα παράδειγμα για το πως λειτουργεί η εξόρυξη δεδομένων. Συγχρόνως, θα μπορούσε κάποιος να πει, ότι αντιμετωπίστηκε στην πράξη και μια περίπτωση (μικρογραφία) διαχείρισης δεδομένων μεγάλης κλίμακας. Διότι, η εφαρμογή που υλοποιήθηκε φτιάχτηκε έτσι ώστε να μπορεί να διαβάζει μεγάλο όγκο δεδομένων (από NoSQL database) και ταυτόχρονα έγινε χρήση τεχνικών (πχ παραλληλία-multithreading) και δομών δεδομένων τέτοιων (πχ δέντρα, hashMap, λίστες, πίνακες) για την γρηγορότερη και καλύτερη δυνατή αποθήκευση και αναζήτηση των δεδομένων.

## Βιβλιογραφία

- [1] Pang - Ning Tan, Michael Steinbach and Vipin Kumar (2009), “Εισαγωγή στην εξόρυξη δεδομένων”, Αθήνα, Εκδόσεις Τζιόλα.
- [2] Margaret H. Dunham (2004), “Data Mining – Εισαγωγικά και Προηγμένα Θέματα Εξόρυξης Γνώσης από Δεδομένα”, Αθήνα, Εκδόσεις Νέων Τεχνολογιών.
- [3] M. Χαλκιδική – M. Βαζιργιάννης (2005), “Εξόρυξη γνώσης από βάσεις δεδομένων και τον παγκόσμιο ιστό”, Αθήνα, Εκδόσεις Τυπωθήτω.
- [4] Anand Rajaraman and Jeffrey David Ullman (2013), “Εξόρυξη από Μεγάλα Σύνολα Δεδομένων”, Αθήνα, Εκδόσεις Νέων Τεχνολογιών.