ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη και Αξιολόγηση Παράλληλων Αλγορίθμων Ομαδοποίησης Δεδομένων σε Περιβάλλον Προγραμματισμού CUDA

Ασημάκης Ι. Αργυρόπουλος

Α.Μ. 711131118

Εισηγητής: Βασίλειος Μάμαλης

Καθηγητής ΠΑ.Δ.Α.

Αθήνα, Μάρτιος 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη και Αξιολόγηση Παράλληλων Αλγορίθμων Ομαδοποίησης Δεδομένων σε Περιβάλλον Προγραμματισμού CUDA

Ασημάκης Ι. Αργυρόπουλος

Α.Μ. 711131118

Εισηγητής: Βασίλειος Μάμαλης

Καθηγητής ΠΑ.Δ.Α.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 3η Μαρτίου 2023.

Vasileios Mamalis Digitally signed by Vasileios Mamalis Date: 2023.03.06 00:26:22 +02'00'	Antonios Bogris Digitally signed by Antonios Bogris Date: 2023.03.06 18:20:45 +02'00'	Panagiotis Karkazis Digitally signed by Panagiotis Karkazis Date: 2023.03.07 11:28:38 +02'00'
Βασίλειος Μάμαλης	Αντώνιος Μπόγρης	Παναγιώτης Καρκαζής
.....	.....	.....
Καθηγητής ΠΑ.Δ.Α.	Καθηγητής ΠΑ.Δ.Α.	Αναπληρωτής Καθηγητής ΠΑ.Δ.Α.

Αθήνα, Μάρτιος 2023

Ο κάτωθι υπογεγραμμένος Αργυρόπουλος Ασημάκης του Ιωάννη, με αριθμό μητρώου 711131118 φοιτητής του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

« Βεβαιώνω ότι είμαι συγγραφέας αυτής της Διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου ».

Ο Δηλών

Αργυρόπουλος Ασημάκης

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον Καθηγητή Βασίλειο Μάμαλη . Η καθοδήγησή του, οι συμβουλές και η κριτική του με βελτίωσαν και θα συνεχίσουν να με εξελίσσουν τόσο ως άνθρωπο όσο και ως ερευνητή.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου, η στήριξη των οποίων μου δίνει καθημερινά δύναμη και κίνητρο.

Περίληψη

Αντικείμενο της πτυχιακής εργασίας θα είναι η μελέτη και αξιολόγηση σε περιβάλλον προγραμματισμού CUDA, αποδοτικών παράλληλων αλγορίθμων για το πρόβλημα υπολογισμού της ομαδοποίηση δεδομένων (data clustering) . Η αξιολόγησή τους θα πραγματοποιηθεί σε πραγματικό περιβάλλον σύγχρονων καρτών γραφικών (Titan, 2080) και θα περιλαμβάνει σύγκριση (σε επίπεδο χρόνων απόκρισης και μετρήσεων επιτάχυνσης- speedup) με τις αντίστοιχες σειριακές υλοποιήσεις τους.

Abstract

The subject of this thesis will be the study and evaluation of efficient parallel algorithms for the computation problem of data clustering (data clustering) in the programming environment CUDA . Their evaluation will take place in a real contemporary environment using modern graphics cards (Titan, 2080) and will include a comparison (at the level of response times and speedup) with their respective serial implementations.

Επιστημονική Περιοχή: Παράλληλα Υπολογιστικά Συστήματα

Λέξεις Κλειδιά: CUDA, Παράλληλος Προγραμματισμός, Ομαδοποίηση δεδομένων, Συσταδοποίηση δεδομένων

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Ζούμε σε μια εποχή που χαρακτηρίζεται από την διαρκή παραγωγή και επεξεργασία δεδομένων[14]. Κάθε μέρα συναντάμε πληροφορία σε κλίμακα που είναι πρωτοφανής στην ιστορία για αυτό άλλωστε χαρακτηρίζεται από μερικούς η εποχή μας και ως εποχή της πληροφορίας (information age). Η πληροφορία αυτή αποθηκεύεται ή αναπαριστάται ως δεδομένα τα οποία στη προσπάθεια μας να τα κατανοήσουμε τα ομαδοποιούμε ή τα κατηγοριοποιούμε βάση την ομοιότητα (ή διαφορά) τους με ήδη υπάρχουσα γνώση. Η διαδικασία αυτή της κατάτμησης των δεδομένων σε σύνολα ή εναλλακτικά της αντιστοίχησης τους σε ήδη υπάρχουσες ομάδες, ήταν ανέκαθεν μείζονος σημασίας και πρωταγωνιστικός παράγοντας στην εκμάθηση και ως εκ τούτου, στην ανάπτυξη της ανθρωπότητας ως σύνολο. Ως αποτέλεσμα των παραπάνω (αυξημένο πλήθος και συχνότητα παραγωγής δεδομένων), έγινε η απόπειρα απομίμησης της μεθόδου εκμάθησης που χρησιμοποιούμε αιώνες τώρα(πολλές φορές ασυνείδητα), σε μορφή αλγορίθμου ώστε να επωφεληθούμε από την τεράστια ταχύτητα υπολογισμών που μας προσφέρουν τα σύγχρονα υπολογιστικά συστήματα. Πιο συγκεκριμένα, ως αποτέλεσμα της απόπειρας αυτής καταλήξαμε σε δύο όμοιες αλλά διακριτές οικογένειες αλγορίθμων, αυτές της κατηγοριοποίησης (classification) και της συσταδοποίησης/ομαδοποίησης (clustering).

Η κατηγοριοποίηση ή επιβλεπόμενη κατηγοριοποίηση είναι μια προσέγγιση στο προαναφερόμενο πρόβλημα και έχει ως στόχο την προσομοίωση της διαδικασίας που πραγματοποιείται με τη διδασκαλία. Διδασκαλία υπό την έννοια ότι γίνεται προσπάθεια ορθού διαμοιρασμού της πληροφορίας (στη καθημερινή ζωή αντιπροσωπεύει είτε αυτή είναι αντικείμενα είτε ιδιότητες τους) χρησιμοποιώντας πρώτα παραδείγματα για την εκμάθηση των μοτίβων που αποτυπώνουν την κάθε κατηγορία. Λίγο πιο συγκεκριμένα, με αυτή τη προσέγγιση είναι απαραίτητη η ύπαρξη ενός συνόλου σωστά κατηγοριοποιημένων δεδομένων στα οποία βασίζονται οι ερευνητές για τη δημιουργία κριτηρίων (που εκφράζονται με τη μορφή μιας μαθηματικής συνάρτησης, συνήθως, πολλών μεταβλητών). Τα κριτήρια αυτά αφού οριστούν, χρησιμοποιούνται για την κατηγοριοποίηση των υπόλοιπων αντικειμένων για τα οποία γίνεται η απόπειρα περιγραφής. Η τεχνική της κατηγοριοποίησης είναι ένας τρόπος επεξεργασίας δεδομένων που έχει δει μεγάλη ανάπτυξη και χρήση ειδικά τα τελευταία χρόνια με την ανάπτυξη των νευρωνικών δικτύων.

Η ομαδοποίηση, συσταδοποίηση ή μη επιβλεπόμενη κατηγοριοποίηση είναι η δεύτερη προσέγγιση για την επίλυση του προβλήματος και βασίζεται περισσότερο στην ανθρώπινη διαίσθηση και λογική. Την ικανότητα αυτή του ανθρώπου να εντοπίζει ομοιότητες και διαφορές σε ένα φαινομενικά χαοτικό σύνολο δεδομένων προσπαθούν να αναπαράγουν οι αλγόριθμοι ομαδοποίησης.

Σε αυτή την εργασία, παρουσιάζονται διαφορετικές οικογένειες αλγορίθμων ομαδοποίησης δεδομένων, αρχικά σε σειριακό περιβάλλον και μετά ειδικότερα σε παράλληλο με σκοπό την εξέταση των κερδών που μπορεί να εξαχθούν από την επιτάχυνση τους σε κάποιους επιλεγμένους υλοποιημένους αλγορίθμους.

Κεφάλαιο 2

Οικογένειες Αλγορίθμων Συσταδοποίηση

Όπως αναφέρθηκε προηγουμένως, η διαδικασία της ομαδοποίησης αποσκοπεί στο διαχωρισμό ενός συνόλου δεδομένων σε ομάδες. Πως ορίζεται όμως η ομάδα; Δεν υπάρχει κάποιος κοινά αποδεκτός ορισμός, παρ' όλα αυτά, η πλειοψηφία των ερευνητών, περιγράφουν μια ομάδα ή συστάδα σύμφωνα με την ομοιογένεια και τον διαχωρισμό της από τις υπόλοιπες (external separation). Δηλαδή, δεδομένα στην ίδια ομάδα θα πρέπει να είναι όμοια μεταξύ τους ενώ δεδομένα σε διαφορετικές ομάδες όσο πιο ανόμοια γίνεται [14]. Σύμφωνα με τον αυστηρή ερμηνεία αυτού του ορισμού, κατά την ομαδοποίηση των δεδομένων, το καθένα μπορεί να ανήκει σε μία και μόνο μία ομάδα. Παρ' όλα όπως θα παρουσιαστεί στη συνέχεια, υπάρχουν και ερμηνείες που ομαδοποιούν τα δεδομένα σε εμφωλευμένες ομάδες που εκφράζουν κάποια ιεραρχία στα δεδομένα (ιεραρχική συσταδοποίηση) αλλά και σε περισσότερες από μία ταυτοχρόνως με μόνη ένδειξη της ομαδοποίησης τους, ένα ποσοστό συμμετοχής στην καθεμία (ασαφής συσταδοποίηση).

2.1 Μέτρα ομοιότητας

Από την παραπάνω περιγραφή είναι εμφανές ότι πρωταρχικό ρόλο στους αλγόριθμους ομαδοποίησης έχουν οι συγκρίσεις. Συνεπώς, είναι σημαντικό να

έχει οριστεί καλά η έννοια της ομοιότητας και της διαφοράς (ή απόστασης) των αντικειμένων που επεξεργάζονται. Αρχικά, είναι σημαντικό να επισημανθεί ότι δεν υπάρχει ένα μόνο κριτήριο ομοιότητας που μπορεί να εφαρμοστεί μέσω μιας συνάρτησης ομοιότητας σε όλα τα προβλήματα. Αυτό συμβαίνει λόγω του γεγονότος ότι τα αντικείμενα που αναπαρίστανται από τα δεδομένα που είναι διαθέσιμα για κάθε πρόβλημα, περιγράφονται από ένα πλήθος χαρακτηριστικών. Τα χαρακτηριστικά αυτά με τη σειρά τους μπορεί αν είναι συνεχή, διακριτά, δυαδικά, ποσοτικά, ποιοτικά κ.α. γεγονός που καθιστά τη χρήση κοινής συνάρτησης ομοιότητας/απόστασης αδύνατη.

Για να αντιμετωπιστεί ο παραπάνω περιορισμός, ορίστηκαν κάποια πρότυπα από την ερευνητική κοινότητα. Αν μία συνάρτηση απόστασης τηρεί κάποιες συγκεκριμένες προϋποθέσεις, τότε είναι κατάλληλη για χρήση σε αλγόριθμους συσταδοποίησης. Αυτές οι προϋποθέσεις είναι:

1) Συμμετρία. $D(x_i, x_j) = D(x_j, x_i)$·

2) Θετικότητα. $D(x_i, x_j) \geq 0$ για κάθε x_i και x_j.

   Αν ισχύουν και οι:

3) Τριγωνική ανισότητα.

$$D(x_i, x_j) \leq D(x_i, x_k) + D(x_k, x_j) \text{ για κάθε } x_i,\ x_j \text{ και } x_k$$

   και

4) Ανακλαστικότητα. $D(x_i, x_j) = 0$ αν και μόνο αν $x_i = x_j$

   τότε ονομάζεται μετρική.

Αντιστοίχως, μια συνάρτηση ομοιότητας ορίζεται έτσι ώστε να ικανοποιεί τις ακόλουθες προϋποθέσεις:

1) Συμμετρία. $S(x_i, x_j) = S(x_j, x_i)$·

2) Θετικότητα. $0 \leq S(x_i, x_j) \leq 1$ για κάθε x_i και x_j.

Αν ισχύουν και οι:

3)

$S(x_i, x_j)S(x_j, x_k) \leq [S(x_i, x_j)+S(x_j, x_k)]S(x_i, x_k)$ για κάθε $x_i,\ x_j$ και x_k

και

4) $S(x_i, x_j) = 1$ αν και μόνο αν $x_i = x_j$

τότε ονομάζεται μετρική ομοιότητας.

Κάνοντας χρήση των συναρτήσεων που ορίστηκαν παραπάνω, ορίζεται για ένα σύνολο αντικειμένων ένας συμμετρικός πίνακας $N \times N$, στον οποίο το στοιχείο (i, j) (όπου $i, j = 1, ..., N$) αντιπροσωπεύει την ομοιότητα ή διαφορά (βάσει το μέτρο που χρησιμοποιήθηκε) των στοιχείων i και j. Ο πίνακας ονομάζεται πίνακας ομοιότητας ή πίνακας απόστασης αντίστοιχα.

Τυπικά, οι συναρτήσεις απόστασης χρησιμοποιούνται για τη μέτρηση συνεχών χαρακτηριστικών ενώ οι συναρτήσεις ομοιότητας είναι πιο χρήσιμες όταν εφαρμόζονται σε μη αριθμητικά χαρακτηριστικά. Η επιλογή της καταλληλότερης μεθόδου διαφέρει ανάλογα με το πρόβλημα στο οποίο εφαρμόζεται. Για δυαδικά δεδομένα, συνήθως χρησιμοποιούνται συναρτήσεις ομοιότητας (η διαφορά μπορεί να βρεθεί εύκολα μέσω της πράξης $D_{ij} = 1 - S_{ij}$). Ως παράδειγμα ενός τέτοιου μέτρου ομοιότητας, έστω ότι χρησιμοποιούνται δύο δυαδικοί δείκτες για τη μέτρηση των χαρακτηριστικών δύο αντικειμένων. Έστω επίσης ότι ορίστηκαν τα n_{00} και n_{11} έτσι ώστε να αντιπροσωπεύουν το πλήθος των χαρακτηριστικών που απουσιάζουν ή εμφανίζονται και στα δύο αντικείμενα,

αντίστοιχα, ταυτόχρονα και τα n_{01} και n_{10} ορίστηκαν έτσι ώστε να χρησιμοποιούνται για την καταμέτρηση των χαρακτηριστικών που εντοπίζονται μόνο στο ένα από τα 2 αντικείμενα. Οι δύο πιο δημοφιλείς συναρτήσεις ομοιότητας για τα στοιχεία x_i και x_j δίνονται παρακάτω:

- $$S_{ij} = \frac{n_{11} + n_{00}}{n_{11} + n_{00} + w(n_{10} + n_{01})}$$

  $w = 1,$ απλός συντελεστής ομοιότητας

  $w = 2,$ μέτρο Rogers και Tanimoto

  $w = 1/2,$ μέτρο Gower και Legendere

  Οι παραπάνω συναρτήσεις ομοιότητας υπολογίζουν απευθείας την ομοιότητα των δύο αντικειμένων. Τα χαρακτηριστικά που δεν αντιστοιχούν μεταξύ τους σταθμίζονται ανάλογα με τη συνεισφορά τους στην ομοιότητα.

- $$S_{ij} = \frac{n_{11}}{n_{11} + w(n_{10} + n_{01})}$$

  $w = 1,$ Δείκτης Jaccard

  $w = 2,$ μέτρο Sokal και Sneath

  $w = 1/2,$ μέτρο Gower και Legendere

  Οι παραπάνω συναρτήσεις εστιάζουν στη κοινή παρουσία χαρακτηριστικών ενώ αγνοούν την κοινή απουσία.

Για χαρακτηριστικά που βρίσκονται στην ονομαστική κλίμακα και έχουν περισσότερα από δύο χαρακτηριστικά (δύο χαρακτηριστικά μπορούν να εκφραστούν ως 0 ή 1), μια αποδοτική μέθοδος αξιοποιεί τη τεχνική της αντιστοίχησης.

$$S_{ij} = \frac{1}{d} \sum_{l=1}^{d} S_{ij}$$

όπου

$$S_{ij} = \begin{cases} 0, & \text{αν } i \text{ και } j \text{ δεν ταιριάζουν} \\ 1, & \text{αν } i \text{ και } j \text{ ταιριάζουν} \end{cases}$$

Γενικότερα, για τα αντικείμενα που αποτελούνται από μεικτές μεταβλητές, είναι δυνατή η αντιστοίχηση όλων των μεταβλητών στο διάστημα (0, 1) και η χρήση μέτρων σύγκρισης όπως η Ευκλείδεια μετρική. Εναλλακτικά, είναι δυνατός ο μετασχηματισμός τους σε δυαδικές μεταβλητές και η χρήση των δυαδικών συναρτήσεων σύγκρισης. Το μειονέκτημα όμως των μεθόδων που αναφέρθηκαν είναι η απώλεια της πληροφορίας.

2.2 Μέθοδοι Ομαδοποίησης

Έχοντας λοιπόν πλέον κατανοήσει τις ακρογωνιαίους λίθους που απαρτίζουν την διαδικασία της συσταδοποίησης είναι εφικτή η παρουσίαση των διαφορετικών μεθόδων που την απαρτίζουν.

2.2.1 Ιεραρχική Ομαδοποίηση

Οι ιεραρχικοί αλγόριθμοι συσταδοποίησης οργανώνουν τα δεδομένα σε μία ιεραρχική δομή σύμφωνα με τον πίνακα ομοιότητας. Τα αποτελέσματα των ιεραρχικών αλγορίθμων συνήθως αναπαρίστανται σε μορφή δυαδικού δένδρου ή δενδρογράμματος. Ο κόμβος της ρίζας αναπαριστά όλα τα δεδομένα του συνόλου που εξετάζονται και κάθε φύλλο ένα μόνο αντικείμενο. Συνεπώς, οι ενδιάμεσοι κόμβοι περιγράφουν τον ”βαθμό” ομοιότητας δύο αντικειμένων. Το ύψος του δένδρογράμματος συνήθως εκφράζει την απόσταση μεταξύ ζευγαριών

αντικειμένων και συστάδων ή ενός αντικειμένου και μιας συστάδας. Το τελικό αποτέλεσμα των ιεραρχικών αλγορίθμων δίνεται από την τομή του δενδρογράμματος σε διάφορα επίπεδα. Με αυτή την αναπαράσταση, παράγονται πολύ χρήσιμες περιγραφές και οπτικοποιήσεις από τις πιθανές δομές των δεδομένων ειδικά αν υπάρχει μια ιεραρχία που τα συνδέει.

Οι ιεραρχικοί αλγόριθμοι κατηγοριοποιούνται κυρίως ως συσσωρευτικοί ή διαιρετικοί. Η συσσωρευτική ομαδοποίηση ξεκινά με N συστάδες οι οποίες περιέχουν ακριβώς ένα αντικείμενο. Στη συνέχεια, πραγματοποιείται μια σειρά συγχωνεύσεων (merge) έτσι ώστε στο τέλος να υπάρχει μόνο μία συστάδα και όλα τα αντικείμενα να ανήκουν σε αυτή. Η διαιρετική ομαδοποίηση ακολουθεί ακριβώς την αντίθετη διαδρομή. Ξεκινά από μία συστάδα που περιέχει όλα τα αντικείμενα και με μία διαδικασία διαδοχικών υποδιαιρέσεων καταλήγει σε ένα σύνολο συστάδων (που περιέχουν μόνο ένα αντικείμενο). Από τη στιγμή όμως που το υπολογιστικό κόστος για την ομαδοποίηση N αντικειμένων είναι $2^{N-1}-1$, έτσι ώστε να εξεταστούν όλες οι πιθανές υποδιαιρέσεις, η διαιρετική ομαδοποίηση δεν χρησιμοποιείται συνήθως στη πράξη. Στη συνέχεια, θα αναλυθεί λίγο περισσότερο η συσσωρευτική ομαδοποίηση.

Το γενικό περίγραμμα ενός συσσωρευτικού αλγόριθμου ομαδοποίησης μπορεί να περιγραφεί από την ακόλουθη διαδικασία

1) Η διαδικασία ξεκινάει με N συστάδες του ενός στοιχείου και τον υπολογισμό του πίνακα ομοιότητας για αυτές τις N συστάδες.

2) Στη συνέχεια, αναζητείται η ελάχιστη απόσταση στον πίνακα ομοιότητας

$$D(C_i, C_j) = \min_{\substack{1 \le m,\, l \le N \\ m \neq l}} D(C_m, C_l)$$

όπου $D(*,*)$ είναι η συνάρτηση απόστασης που συζητήθηκε προηγουμέ-

νως. Ενώνονται οι συστάδες C_i και C_j με τη μικρότερη απόσταση για να σχηματίσουν μια νέα συστάδα.

3) Ο πίνακας ομοιότητας ενημερώνεται με τον υπολογισμό της απόστασης της νέας συστάδας από τα υπόλοιπα αντικείμενα.

4) Τέλος, τα βήματα 2) - 3) επαναλαμβάνονται μέχρι όλα τα αντικείμενα να ανήκουν στην ίδια συστάδα.

Λόγω των διαφορετικών ορισμών σχετικά με την απόσταση μεταξύ δύο συστάδων, υπάρχουν πολλοί συσσωρευτικοί αλγόριθμοι. Κάποιες από τις πιο απλές και πιο δημοφιλείς μεθόδους είναι αυτές του απλού και του πλήρους δεσμού. Στην περίπτωση του απλού δεσμού, η απόσταση δύο συστάδων καθορίζεται από την απόσταση των κοντινότερων αντικειμένων που ανήκουν σε αυτές. Αντιθέτως, στην περίπτωση του πλήρους δεσμού, η απόσταση των συστάδων καθορίζεται από την απόσταση των αντικειμένων που απέχουν περισσότερο μεταξύ τους. Τόσο ο πλήρης δεσμός όσο και ο απλός μπορούν να γενικευτούν με την συνάρτηση που προτάθηκε από τους Lance και Williams ως

$$D(C_l,(C_i,C_j)) = \alpha_i D(C_l,C_i)+\alpha_j D(C_l,C_j)+\beta D(C_i,C_j)+\gamma|D(C_l,C_i)-D(C_l,C_j)|$$

όπου $D(*,*)$ είναι η συνάρτηση απόστασης και α_i, α_j, β και γ είναι συντελεστές που παίρνουν τιμές ανάλογα τη μέθοδο που χρησιμοποιείται.

Η συνάρτηση περιγράφει την απόσταση μεταξύ μιας συστάδας l και μιας νέας συστάδας που δημιουργήθηκε από την συνένωση των i και j. Να σημειωθεί ότι όταν τα $\alpha_i = \alpha_j = 1/2$, $\beta = 0$ και $\gamma = -1/2$, τότε η συνάρτηση γίνεται

$$D(C_l,(C_i,C_j)) = min(D(C_l,C_i),D(C_l,C_j))$$

που αντιστοιχεί στον απλό δεσμό. Όταν τα $\alpha_i = \alpha_j = \gamma = 1/2$, $\beta = 0$, τότε

η συνάρτηση γίνεται

$$D(C_l, (C_i, C_j)) = max(D(C_l, C_i), D(C_l, C_j))$$

που αντιστοιχεί στον πλήρη δεσμό.

Πολλοί ακόμα πιο σύνθετοι συσσωρευτικοί αλγόριθμοι, συμπεριλαμβανομένου του μέσου όρου, κεντρώου, ενδιάμεσου και τις μεθόδου του Ward, μπορούν να υλοποιηθούν επιλέγοντας κατάλληλες τιμές για τους συντελεστές της συνάρτησης. Οι μέθοδοι του απλού, πλήρους και του δεσμού του μέσου όρου, περνάνε απ' όλα τα στοιχεία μια συστάδας όταν υπολογίζουν την απόσταση των συστάδων με αποτέλεσμα να λέγονται μέθοδοι γραφήματος. Οι υπόλοιπες, λέγονται γεωμετρικές μέθοδοι εφόσον χρησιμοποιούν γεωμετρικά κέντρα για να εκπροσωπήσουν τις συστάδες και να υπολογίσουν τις αποστάσεις.

Κάποια από τα κύρια μειονεκτήματα που παρουσιάζουν οι ιεραρχικοί αλγόριθμοι συσταδοποίησης είναι η έλλειψη ανθεκτικότητας (robustness) με αποτέλεσμα να είναι επιρρεπείς στο θόρυβο και στα ακρότατα. Επίσης, όταν ένα στοιχείο κατανεμηθεί σε κάποια ομάδα, δεν ξανα-εξετάζεται με αποτέλεσμα οι ιεραρχικοί αλγόριθμοι να μην έχουν την δυνατότητα να διορθώσουν πιθανά λάθη που έγινα κατά την ανάθεση στοιχείων σε συστάδες. Ακόμα, η υπολογιστική πολυπλοκότητα για την πλειοψηφία των ιεραρχικών αλγορίθμων είναι τουλάχιστον $O(N^2)$ με αποτέλεσμα να καθίστανται ανεπιθύμητοι για εφαρμογή σε μεγάλα σύνολα δεδομένων. Τέλος, ένα επιπλέον μειονέκτημα των ιεραρχικών αλγορίθμων είναι η τάση σχηματισμού σφαιρικών συστάδων.

2.2.2 Ομαδοποίηση βασισμένη στο τετραγωνικό λάθος (Squared-Error Based, Vector Quantization)

Σε αντίθεση με την ιεραρχική συσταδοποίηση, που παράγει διαδοχικά επίπεδα συστάδων μέσω επαναλαμβανόμενων υποδιαιρέσεων και συγχωνεύσεων, οι

διαμεριστικοί αλγόριθμοι συσταδοποίησης αναθέτουν τα αντικείμενα/δεδομένα του συνόλου που εξετάζονται σε K συστάδες χωρίς κάποια ιεραρχική δομή. Κατά κανόνα, ο βέλτιστος διαμερισμός, σύμφωνα με κάποιο κριτήριο, μπορεί να βρεθεί εξετάζοντας όλους του πιθανού συνδυασμούς. Παρ' όλα αυτά, η τακτική brute-force δεν είναι εφικτή στην πραγματικότητα, λόγω υπολογιστικού κόστους. Ακόμη και ένα πρόβλημα μικρής κλίμακας (ομαδοποίηση 30 αντικειμένων σε 3 ομάδες), θα χρειαζόταν 2×10^{14} ομαδοποιήσεις. Για αυτό το λόγο, δημιουργήθηκαν ευριστικοί αλγόριθμοι που βρίσκουν την λύση με μικρό ποσοστό απόκλισης (approximate solution).

Ένα από τα πιο σημαντικά στοιχεία των διαμεριστικών αλγόριθμων είναι η συνάρτηση του κριτηρίου. Η συνάρτηση του αθροίσματος του τετραγωνικού λάθους είναι ένα από τα πιο δημοφιλή κριτήρια και δουλεύει ως εξής. Έστω ότι ορίζεται ένα σετ αντικειμένων $x_j \in \mathbb{R}^d, j = 1, ..., N$ και πρέπει να οργανωθούν σε K υποσύνολα $C = \{C_1, ..., C_j\}$. Τότε το κριτήριο του αθροίσματος του τετραγωνικού λάθους ορίζεται ως

$$J(\mathbf{\Gamma}, \boldsymbol{M}) = \sum_{i=1}^{K} \sum_{j=1}^{N} \gamma_{ij} ||\boldsymbol{x}_j - \boldsymbol{m}_i||^2$$

όπου

$\mathbf{\Gamma} =$ ένας πίνκας διαμερισμού

$\{\gamma_{ij}\}$

$$\gamma_{ij} = \begin{cases} 1 & \text{αν } \boldsymbol{x}_j \in \text{στη συστάδα } i \\ 0 & \text{διαφορετικά} \end{cases} \qquad \text{με το } \sum_{i=1}^{K} \gamma_{ij} = 1 \forall j$$

$\boldsymbol{M} =$ προτότυπο συστάδας ή πίνακας (μέσων) κεντρώων

$[\boldsymbol{m}_i, \ldots, \boldsymbol{m}_K]$

$\boldsymbol{m}_i =$ μέσος δείγματος για την iοστή συστάδα

$(1/N_i) \sum_{j=1}^{N} \gamma_{ij} \boldsymbol{x}_j$

N_i αριθμός αντικειμένων στη iοστή συστάδα

Αξίζει να σημειωθεί η σχέση μεταξύ κριτηρίων αθροίσματος τετραγωνικού λάθους και των scatter matrices που ορίζονται στη πολυταξική διακριτή ανάλυση,

$$S_T = S_W + S_B$$

όπου

S_T συνολικά scatter matrices

$S_W =$ scatter matrix μέσα σε συστάδες

$$\sum_{i=1}^{K}\sum_{j=1}^{N}\gamma_{ij}(\boldsymbol{x}_j-\boldsymbol{m}_i)(\boldsymbol{x}_j-\boldsymbol{m}_i)^T$$

$S_B =$ scatter matrix μεταξύ συστάδων και

$$\sum_{i=1}^{K}N_i(\boldsymbol{m}_i-\boldsymbol{m})(\boldsymbol{m}_i-\boldsymbol{m})^T$$

$\boldsymbol{m} =$ μέσο διάνυσμα για όλο το σετ δεδομένων.

$$(1/N_i)\sum_{i=1}^{K}N_{ij}\boldsymbol{m}_j$$

Δεν είναι δύσκολο να δει κανείς ότι το κριτήριο βασισμένο στο ίχνος του S_W είναι ίδιο με το κριτήριο αθροίσματος τετραγωνικών λαθών. Η ελαχιστοποίηση του κριτηρίου αθροίσματος τετραγωνικών λαθών είναι ισοδύναμη με την ελαχιστοποίηση το ίχνος του S_W ή τη μεγιστοποίηση του S_B. Με αυτό το τρόπο, είναι δυνατή η απόκτηση μια πλούσιας ομάδας συναρτήσεων κριτηρίου βασισμένη στα χαρακτηριστικά των S_W και S_B.

Ο αλγόριθμος K-means είναι ο πιο γνωστός αλγόριθμος ομαδοποίησης που βασίζεται στη ρίζα του τετραγωνικού λάθους και πραγματοποιεί τα ακόλουθα βήματα.

1) Αρχικοποίηση K συστάδων τυχαία ή βάσει ήδη υπάρχουσας γνώσης και υπολογισμός ενός πίνακα πρωτότυπων συστάδων $M = [m_1, ..., m_K]$.

2) Το κάθε αντικείμενο του συνόλου ανατίθεται στην κοντινότερη συστάδα C_w δηλαδή

$x_j \in C_w$, αν $||x_j - m_w|| < ||x_j - m_i||$ για κάθε $j = 1, ..., N$, $i \neq w$ και $i = 1,$

3) Ενημέρωση του πίνακα πρωτότυπων συστάδων βάσει τον νέο διαμοιρασμό τους.

4) Επανάληψη των βημάτων 2) - 3) μέχρι να μην υπάρχει αλλαγή για όλες τις συστάδες.

Ο αλγόριθμος K-means είναι αρκετά απλός και μπορεί να υλοποιηθεί πολύ εύκολα για την επίλυση διάφορων προβλημάτων. Δουλεύει εξαιρετικά για σφαιρικές συστάδες. Η χρονική πολυπλοκότητα του K-means είναι $O(NKd)$. Εφόσον τα K και d είναι συνήθως πολύ μικρότερα του N, ο K-means μπορεί να χρησιμοποιηθεί και για μεγάλα σύνολα δεδομένων. Υπάρχουν επίσης παράλληλες τεχνικές για τον K-means που τον επιταχύνουν σημαντικά, οι οποίες θα παρουσιαστούν στο Κεφάλαιο 3. Χάρη στη δημοφιλία του, ο K-means έχει μελετηθεί σε βάθος, με αποτέλεσμα τα μειονεκτήματα του να είναι πλήρως κατανοητά και να έχουν προταθεί αλγόριθμοι που τα αντιμετωπίζουν. Ακολουθούν ορισμένα από αυτά μαζί με μερικές προτεινόμενες λύσεις.

1) Δεν υπάρχει κάποιος αποδοτικός τρόπος που ταυτόχρονα να δουλεύει για κάθε πρόβλημα ώστε να καθοριστούν το πλήθος και η αρχική θέση των K συστάδων. Τα κέντρα βάρους των συστάδων μετά την σύγκλιση διαφέρουν ανάλογα με τις αρχικές θέσεις. Μια γενική στρατηγική που ακολουθείται είναι να εκτελεστεί ο αλγόριθμος πολλές φορές με τυχαίες αρχικές θέσεις συστάδων. Για την επίλυση αυτού του προβλήματος, έχουν προταθεί πολλές λύσεις η κάθε μία με διαφορετική αποτελεσματικότητα. Παρ' όλα αυτά, μία τεχνική που έχει ξεχωρίσει, είναι ο αλγόριθμος ISODATA, που αναπτύχθηκε από τους Ball και Hall.

   Ο ISODATA αποπειράται την εκτίμηση του K με τη δυναμική ρύθμιση των συστάδων που δημιουργούνται. Αυτό γίνεται με την συγχώνευση και

υποδιαίρεση συστάδων σύμφωνα με κάποιες μεταβλητές (επί της ουσίας, το πρόβλημα μετατρέπεται από εκτίμηση του K σε βελτιστοποίηση παραμέτρων). Το K στο οποίο θα καταλήξει αυτή η τεχνική, θα χρησιμοποιηθεί ως ο νέος εκτιμώμενος αριθμός συστάδων για την επόμενη επανάληψη του αλγορίθμου.

2) Η ομαλή επαναληπτική λειτουργία του K-means όπως περιγράφηκε λίγο παραπάνω δεν μπορεί να εγγυηθεί σύγκλιση στην καθολικά βέλτιστη λύση (global optimum). Οι βέλτιστες στοχαστικές τεχνικές όπως simulated annealing (SA) και οι γενετικοί αλγόριθμοι, μπορούν να βρουν την καθολικά βέλτιστη λύση με το επιπλέον κόστος όμως ακριβών υπολογιστικών πράξεων. Οι Krishna και Murty σχεδίασαν νέους τελεστές στο υβριδικό τους ”σχέδιο” που ονόμασαν GKA με στόχο την καθολική αναζήτηση και γρήγορη σύγκλιση. Ο τελεστής μετάλλαξης που ορίστηκε, βασίζεται στην Ευκλείδεια απόσταση μεταξύ ενός αντικειμένου και ενός κεντρώου με σκοπό την αποφυγή τοπικών ελάχιστων. Ένας άλλος τελεστής, ο τελεστής K-means (K-means operator, KMO), αντικαθιστά τους υπολογιστικά ακριβούς τελεστές crossover και ”καταπολεμά” κάποιες από τις επιπλοκές που συσχετίζονται με αυτούς. Μια άλλη τεχνική, ορίζει μια εκδοχή του αλγόριθμου K-means που χρησιμοποιεί adaptive learning rate (προσαρμοστικός δείκτης μάθησης) στην online εκδοχή του. Ο δείκτης εκμάθησης, βασίζεται αποκλειστικά σε μεταβολές του σετ δεδομένων χωρίς να χρειάζεται την επέμβαση του χρήστη. Τέλος, ο βελτιωμένος αλγόριθμος LBG (Enhanced LBG, ELBG), χρησιμοποιεί τον μηχανισμό της ρουλέτας, τυπικό στους γενετικούς αλγορίθμους ώστε να μην είναι ”ευαίσθητος” στις αρχικοποιήσεις.

3) Ο K-means είναι ευαίσθητος στο θόρυβο και τα ακραία σημεία. Ακόμα και

αν ένα σημείο βρίσκεται πολύ μακριά από το κέντρο βάρους της συστάδας, εντάσσεται σε μια ομάδα με αποτέλεσμα να παραμορφώνει το τελικό σχήμα της ομάδας στην οποία κατηγοριοποιήθηκε. Τόσο ο ISODATA όσο και ο PAM παίρνουν υπόψιν τους τα αποτελέσματα των ακραίων σημείων κατά τη διάρκεια της συσταδοποίησης. Ο ISODATA αφαιρεί τις συστάδες με λίγα στοιχεία ενώ με την λειτουργία της υποδιαίρεσης των συστάδων αποτρέπεται η πιθανότητα παραμόρφωσης των συστάδων (που είναι χαρακτηριστικό του K-means). Ο PAM χρησιμοποιεί στοιχεία του συνόλου δεδομένων για κέντρα βάρους των συστάδων ως προς αποφυγή των επιπτώσεων των ακραίων στοιχείων.

4) Με τον όρο means (μέσος όρος), ο αλγόριθμος περιορίζεται σε αριθμητικές τιμές. Ο αλγόριθμος K-medoids (διάμεσος) είναι μια προφανής λύση όταν ο υπολογισμός του μέσου δεν είναι δυνατός καθώς χρησιμοποιεί στοιχεία από το σύνολο δεδομένων ως κέντρα βάρους των συστάδων. Οι Huang και Gupta και λοιποί όρισαν κάποια μέτρα διαφοράς σε μια απόπειρα να επεκτείνουν τον K-means και για κατηγορικές μεταβλητές. Η μέθοδος του Huang έχει ως στόχο την ελαχιστοποίηση της συνάρτησης κόστους $J(\mathbf{\Gamma}, \boldsymbol{Q}) = \sum_{i=1}^{K} \sum_{j=1}^{N} \gamma_{ij} D(\boldsymbol{x_j}, \boldsymbol{Q_i})$, όπου

$$D(x_i, x_j) = \sum_{l=1}^{d} \delta(x_{il}, x_{jl})$$

και

$$\delta(x_{il}, x_{jl}) = \begin{cases} 1 & x_{il} \neq x_{jl} \\ 0 & x_{il} = x_{jl} \end{cases}$$

με ένα σετ διανυσμάτων d-διαστάσεων $Q = \{\boldsymbol{Q_1}, ..., \boldsymbol{Q_K}\}$, όπου $Q_j = \{Q_{j1}, ..., Q_{jd}\}$. Το κάθε διάνυσμα $\boldsymbol{Q_j}$ είναι γνωστό ως mode και ορίζεται με σκοπό την ελαχιστοποίηση του αθροίσματος των αποστάσεων $\sum_{i=1}^{N} D(\boldsymbol{x_i}, \boldsymbol{Q_j})$.

Ο προτεινόμενος αλγόριθμος K-modes λειτουργεί παρόμοια με τον K-means.

2.2.3 Ομαδοποίηση βασισμένη σε μείξη πιθανοτήτων (Mixture Densities)

Από την προοπτική των πιθανοτήτων, υπάρχει η υπόθεση ότι τα αντικείμενα παράγονται σύμφωνα με κάποια κατανομή πιθανότητας. Ότι δηλαδή τα δεδομένα σε διαφορετικές συστάδες, παράχθηκαν από διαφορετικές κατανομές. Αυτά τα δεδομένα, μπορούν να παραχθούν από διαφορετικές συναρτήσει πυκνότητας (πχ Γκαουσιανή κατανομή, κατανομή t-student), ή από την ίδια κατανομή και συνάρτηση πυκνότητας αλλά με διαφορετικές παραμέτρους. Αν οι κατανομές είναι γνωστές, τότε το να βρεθεί η συστάδα ενός υπάρχοντος συνόλου δεδομένων είναι ισοδύναμο με την εκτίμηση των παραμέτρων των υποκείμενων μοντέλων που τα περιγράφουν.

Αν υποθέσουμε ότι η "a-priori" πιθανότητα $P(C_i)$ για τις συστάδες $C_i, i = 1, \ldots, K$ (το K θεωρείται γνωστό, υπάρχουν τεχνικές εκτίμησης του K) και η δεσμευτική πιθανότητα πυκνότητας $p(x|C_i, \boldsymbol{\theta}_i)$ (όπου $\boldsymbol{\theta}_i$ είναι η άγνωστη παράμετρος) είναι γνωστά. Τότε η "μικτή" πιθανότητα πυκνότητας (mixture probality density) εκφράζεται ως

$$p(x|\boldsymbol{\theta}) = \sum_{i=1}^{K} p(x|C_i, \boldsymbol{\theta}_i) P(C_i)$$

όπου $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$, και $\sum_{i=1}^{K} P(C_i) = 1$. Η "a-posteriori" πιθανότητα για την κατανομή των δεδομένων σε συστάδες μπορεί πολύ εύκολα να υπολογιστεί χρησιμοποιώντας το θεώρημα του Bayes αρκεί να οριστεί η παράμετρος $\boldsymbol{\theta}$. Σε αυτό το σημείο κατασκευάζεται το μοντέλο. Κυρίως και πιο συχνά χρησιμοποιούνται Γκαουσιανές πυκνότητες πολλών μεταβλητών χάρη στην ολοκληρωμένη θεωρία και τη δυνατότητα ελέγχου (tractability).

Η εκτίμηση μέγιστης πιθανότητας (maximum likelihood, ML) είναι μια σημαντική στατιστική προσέγγιση για την εκτίμηση των παραμέτρων και αποφα-

σίζει ως βέλτιστη εκτίμηση αυτή που μεγιστοποιεί την πιθανότητα ”δημιουργίας” όλων των παρατηρήσεων, που δίνεται από τη συνάρτηση ”κοινών”(joint) πυκνοτήτων.

$$p(\{x_1, \ldots, x_N\} \mid \boldsymbol{\theta}) = \prod_{j=1}^{N} p(x_j \mid \boldsymbol{\theta})$$

ή, σε λογαριθμική μορφή

$$l(\theta) = \sum_{j=1}^{N} \ln p(x_j \mid \boldsymbol{\theta})$$

. Η καλύτερη εκτίμηση μπορεί να βρεθεί από την λύση της λογαριθμικής συνάρτησης πιθανότητας $(\partial l(\boldsymbol{\theta}))/(\partial \boldsymbol{\theta}) = 0$

Δυστυχώς, εφόσον είναι αδύνατο να βρεθούν αναλυτικά οι λύσεις των συναρτήσεων πιθανότητας τις περισσότερες φορές, η λύση προσεγγίζεται με κάποιους αλγόριθμους. Ο πιο δημοφιλής τέτοιος αλγόριθμος είναι ο expectation-maximization (EM). Ο EM αντιμετωπίζει το σετ δεδομένων ως ημιτελές και χωρίζει το κάθε δεδομένο x_j σε δύο μέρη $x_j = \{x_j^g, x_j^m\}$ όπου το x_j^g είναι το παρατηρηθέν feature και το $x_j^m = (x_{j1}^m, \ldots, x_{jK}^m)$ τα δεδομένα που λείπουν, με το x_j^m να παίρνει τιμές 1 ή 0 ανάλογα αν το x_j ανήκει στο αντικείμενο i ή όχι. Επομένως, η λογαριθμική πιθανότητα των ολοκληρωμένων δεδομένων είναι:

$$l(\theta) = \sum_{j=1}^{N} \sum_{i=1}^{K} x_{ji}^m \log[P(C_i) p(x_j^g \mid \boldsymbol{\theta}_i)]$$

Ο βασικός αλγόριθμος ΕΜ παράγει μια σειρά εκτιμώμενων παραμέτρων $\{\theta^0, \theta^1, \ldots, \theta^T\}$, όπου το T αναπαριστά την επίτευξη των κριτηρίων σύγκλισης ακολουθώντας τα βήματα:

1) Αρχικοποίηση του θ^0 και ορισμός του $t = 0$

2) βήμα-e: Υπολογισμός εκτίμησης της λογαριθμικής πιθανότητας των ολοκληρωμένων δεδομένων

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t) = E[\log p(\boldsymbol{x}^g, \boldsymbol{x}^m | \boldsymbol{\theta}) | \boldsymbol{x}^g, \boldsymbol{\theta}^t]$$

3) βήμα-m: Επιλογή μιας νέας εκτίμησης παραμέτρου έτσι ώστε να μεγιστοποιείται η συνάρτηση Q, $\boldsymbol{\theta}^{t+1} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^t)$

4) Αύξηση του $t = t + 1$. Επανάληψη των βημάτων 2) - 3) μέχρι να ... τα κριτήρια σύγκλισης.

Τα κύρια μειονεκτήματα του αλγόριθμου ΕΜ είναι η ευαισθησία στην επιλογή των αρχικών παραμέτρων, οι επιπτώσεις ενός πίνακα συνδιακύμανσης, η πιθανότητα σύγκλισης σε τοπικό βέλτιστο και η αργή σύγκλιση. Εναλλαγές του ΕΜ για την επίλυση αυτών των προβλημάτων, συζητούνται στα .... .

Ενδιαφέρουσα είναι επίσης, η σχέση που υπάρχει ανάμεσα στους αλγόριθμους K-means και ΕΜ. Οι Celeux και Govaert απέδειξαν ότι όταν ο αλγόριθμος συσταδοποίησης ΕΜ (clustering EM, CEM) χρησιμοποιεί σφαιρικές Gaussian mixtures τότε είναι ισοδύναμο με τον αλγόριθμο K-means.

2.2.4 Ομαδοποίηση βασισμένη στη Θεωρία Γράφων

Οι ιδέες και θεωρίες γύρω από τη θεωρία γράφων, τη θέτουν σε πολλή καλή θέση για να περιγραφούν προβλήματα συσταδοποίησης με τη χρήση γράφων. Οι κόμβοι V ενός βεβαρημένου γράφου G αντιστοιχούν στα δεδομένα του χώρου που εξετάζουμε και οι ακμές E εκφράζουν την εγγύτητα δύο δεδομένων. Αν ο πίνακας ανομοιότητας οριστεί ως

$$D_{ij} = \begin{cases} 1 & \text{αν } D(x_i, x_j) < d_0 \\ 0 & \text{διαφορετικά} \end{cases}$$

όπου το d_0 είναι η τιμή για το κατώφλι, τότε ο γράφος απλοποιείται σε έναν μη-βεβαρημένο γράφο κατωφλίου. Οι αλγόριθμοι ιεραρχικής συσταδοποίησης απλού και πλήρη δεσμού που παρουσιάστηκαν παραπάνω, μπορούν να περιγραφούν ως γράφοι κατωφλίου. Η συσταδοποίηση απλού δεσμού είναι ισοδύναμη

με την αναζήτηση του κατά μέγιστο συνδεδεμένου υπογράφου (συνεκτικές συνιστώσες) ενώ η συσταδοποίηση πλήρους δεσμού αντιστοιχεί στην αναζήτηση του κατά μέγιστο πλήρους υπογράφου (κλίκα). Οι Jain και Dubes παρουσίασαν περισσότερες εφαρμογές με τις οποίες η θεωρία γράφων εφαρμόζεται στην ιεραρχική συσταδοποίηση.

Ο αλγόριθμος Chameleon είναι ένα ακόμα παράδειγμα συσσωρευτικού αλγόριθμου ιεραρχικής συσταδοποίησης που κάνει χρήση γράφων. Βασίζεται στον γράφο του k-κοντινότερου-γείτονα, στον οποίο μια ακμή διαγράφεται αν δεν είναι και οι δύο κορυφές μέσα στα k κοντινότερα σημεία που έχουν. Στο πρώτο βήμα, ο Chameleon υποδιαιρεί τον συνδεδεμένο γράφο σε ένα σύνολο υπογράφων αφαιρώντας τις ελάχιστες ακμές (minimal edge cut). Ο κάθε υπογράφος θα πρέπει να περιέχει αρκετές κορυφές έτσι ώστε να είναι δυνατός ο υπολογισμός της ομοιότητας. Συνδυάζοντας την σχετική κεντρικότητα εγγύτητας (relative closeness) και την σχετική δια-συνδεσιμότητα (relative interconnectivity), τις οποίες συνδυάζει ο Chameleon ώστε να έχει αρκετή ευελιξία για να "εξερευνήσει" τα χαρακτηριστικά των συστάδων, συγχωνεύει τους υπο-γράφους για να φτάσει στην τελική λύση. Εδώ, η σχετική δια-συνεκτικότητα (ή εγγύτητα) παράγεται μέσω της κανονικοποίησης του αθροίσματος των βαρών (ή μέσου βάρους) των ακμών που συνδέουν τις δύο συστάδες μέσω της εσωτερικής συνεκτικότητας (ή εγγύτητας) των συστάδων.

Ο DTG είναι μια ακόμα σημαντική αναπαράσταση γράφου που χρησιμοποιείται για την ανάλυση ιεραρχικών συστάδων και έχει χρησιμοποιηθεί για αλγόριθμους όπως ο AMOEBA.

Η θεωρία γράφων μπορεί να χρησιμοποιηθεί και για μη ιεραρχική συσταδοποίηση. Ο αλγόριθμος συσταδοποίησης του Zahn ψάχνει για συστάδες κάνοντας χρήση ενός spanning tree (γεννητικού δέντρου) στο οποίο αναζητούνται

οι συνδεδεμένοι υπο-γράφοι που παράγονται από τον εντοπισμό και διαγραφή ”ασυνεπών” ακμών.

Οι Hartuv και Shamir αντιμετώπισαν τις συστάδες ως πολύ-συνδεδεμένους υπογράφους (HCS, higly connected subgraphs), εδώ το πολύ-συνδεδεμένος σημαίνει ότι η συνδεσιμότητα (το ελάχιστο πλήθος ακμών που πρέπει να διαγραφούν για να υποδιαιρέσουν τον γράφο)του υπογράφου είναι τουλάχιστον ίση με το μισό πλήθος των κορυφών του. Η διαδικασία του minimum cut, που αποσκοπεί στον διαχωρισμό ενός γράφου με την ελάχιστη διαγραφή ακμών, χρησιμοποιείται για να βρεθούν οι HCS αναδρομικά.

Ένας επιπλέον αλγόριθμος, ο CLICK, βασίζεται στον υπολογισμό της τομής βάσει το ελάχιστο βάρος για να σχηματίσει συστάδες. Εδώ, ο γράφος είναι βεβαρημένος, με τα βάρη στις ακμές να ορίζονται συνδυάζοντας πιθανότητες και θεωρία γράφων. Πιο συγκεκριμένα, το βάρος μιας ακμής που συνδέει την κορυφή i με τη κορυφή j ορίζεται από το παρακάτω

$$e_{ij} = \log \frac{\text{Prob(τα } i, j\text{να ανήκουν στην ίδια συστάδα } |S_{ij})}{\text{Prob(τα } i, j\text{να μην ανήκουν στην ίδια συστάδα } |S_{ij})}$$

όπου το S_{ij} αναπαριστά την ομοιότητα μεταξύ δύο κορυφών. Επιπλέον, ο CLICK υποθέτει ότι οι τιμές ομοιότητας μέσα στη συστάδα αλλά και μεταξύ συστάδων ακολουθούν Γκαουσιανή κατανομή με διαφορετικούς μέσους και διακυμάνσεις, αντίστοιχα. Συνεπώς, η προηγούμενη εξίσωση μπορεί να γραφεί ξανά χρησιμοποιώντας το θεώρημα του Bayes ως

$$e_{ij} = \log \frac{p_0 \sigma_B}{(1-p_0)\sigma_W} + \frac{(S_{ij}-\mu_B)^2}{2\sigma_B^2} - \frac{(S_{ij}-\mu_W)^2}{2\sigma_W^2}$$

όπου p_0 είναι η ”a-priori” πιθανότητα δύο αντικείμενα να ανήκουν στην ίδια συστάδα και $\mu_B, \sigma_B^2, \mu_W, \sigma_B^2$ είναι οι μέσοι και οι διακυμάνσεις για την ομοιότητα ανάμεσα σε συστάδες (between-cluster similarity) και εντός των συστάδων (within-cluster similarity) αντίστοιχα. Αυτές οι παράμετροι μπορούν είτε

να εκτιμηθούν χρησιμοποιώντας ήδη υπάρχουσα γνώση, είτε χρησιμοποιώντας προσεγγιστικές τεχνικές παραμέτρων. Ο CLICK ελέγχει αναδρομικά τον κάθε υπογράφο και δημιουργεί ένα πυρήνα που αποτελείται από τις συνεκτικές συνιστώσες που ικανοποιούν κάποια συνάρτηση κριτηρίων. Οι υπογράφοι που περιέχουν μόνο μια κορυφή αντιμετωπίζονται ως singletons και απομονώνονται για περαιτέρω επεξεργασία. Χρησιμοποιώντας τους πυρήνες ως τη βάση των πυρήνων, ο αλγόριθμος CLICK πραγματοποιεί μια σειρά συγχωνεύσεων με singletons και άλλες συστάδες έτσι ώστε να παραχθεί η τελικές συστάδες. Στη παραπάνω διαδικασία, μπορούν να προστεθούν και ευρετικά κριτήρια για την επιτάχυνση του αλγορίθμου.

Παρόμοια, ο αλγόριθμος CAST συμπεριλαμβάνει ένα μοντέλο πιθανοτήτων κατά τη διάρκεια της διαδικασίας σχεδιασμού του αλγορίθμου συσταδοποίησης που βασίζεται σε γράφους. Οι συστάδες σχεδιάζονται ως "αλλοιωμένες" κλίκες, οι οποίες, σε ιδανικές συνθήκες, αντιμετωπίζονται ως μη συνδεδεμένες κλίκες (disjoint cliques). Ο αλγόριθμος παίρνει υπόψιν του τον θόρυβο προσθέτοντας και αφαιρώντας ακμές από το "ιδανικό" μοντέλο με μια πιθανότητα α. Υπάρχουν δεδομένα που αποδεικνύουν ότι είναι δυνατή με μεγάλη πιθανότητα η ανάκτηση του αρχικού γράφου από τις "αλλοιωμένες" κλίκες.

Ο αλγόριθμος CAST, αποτελεί την ευρετική υλοποίηση της αρχικά θεωρητικής εκδοχής. Ο CAST δημιουργεί συστάδες σειριακά με τη κάθε συστάδα να ξεκινάει με ένα τυχαίο και μη χρησιμοποιημένο δεδομένο. Η σχέση που οδηγεί μια συστάδα C_0 να δημιουργηθεί από ένα δεδομένο i καθορίζεται από ένα μέτρο που ονομάζεται affinity και ορίζεται ως $\alpha(i) = \sum_{j \in C_0} S_{ij}$ και το κατώφλι του affinity t. Όταν $\alpha(i) \geq t|C_0|$, τότε το δεδομένο i έχει μεγάλο βαθμό συσχέτισης με την συστάδα C_0 και αντίστροφα. Ο αλγόριθμος CAST εναλλάσσεται μεταξύ προσθήκης και διαγραφή δεδομένων με μεγάλο και χαμηλό affinity αντίστοιχα

μέχρι να μην γίνονται πλέον αλλαγές.

2.2.5 Ομαδοποίηση βασισμένη σε συνδυαστική αναζήτηση

Το βασικό αντικείμενο των τεχνικών αναζήτησης είναι η εύρεση των καθολικών ή των κατά προσέγγιση καθολικών βέλτιστων για συνδυαστικά προβλήματα βελτιστοποίησης, τα οποία συνήθως έχουν βαθμό δυσκολία και πολυπλοκότητας NP και χρειάζεται να ψάξουν έναν εξαιρετικά μεγάλο χώρο λύσεων. Η συσταδοποίηση, μπορεί να θεωρηθεί ως μία κατηγορία των προβλημάτων βελτιστοποίησης. Δεδομένου ενός σετ δεδομένων $x_j \in \mathbb{R}^d, j = 1, \ldots, N$, οι αλγόριθμοι συσταδοποίησης προσπαθούν να τα ομαδοποιήσουν σε υποσύνολα $\{C_1, \ldots, C_K\}$ που βελτιστοποιούν κάποια συνάρτηση κριτηρίων. Ο πιθανός διαμοιρασμός των σημείων σε K συστάδες δίνεται από την εξίσωση

$$P(N, K) = \frac{1}{K!} \sum_{m=1}^{K} (-1)^{K-m} C_K^m m^N.$$

όπως φαίνεται παραπάνω, ακόμα και για μικρό ζευγάρι N και K, η υπολογιστική πολυπλοκότητα είναι εξαιρετικά μεγάλη, χωρίς να αναφερθούν τα προβλήματα συσταδοποίησης πολυ-διάστατων και μεγάλων σετ δεδομένων. Απλές τεχνικές αναζήτησης, όπως ο αλγόριθμος hill-climbing, μπορούν να χρησιμοποιηθούν για τον διαμερισμό των στοιχείων αλλά κολλάνε πολύ εύκολα σε τοπικά ελάχιστα οπότε δεν μπορούν να εγγυηθούν την εύρεση της βέλτιστης λύσης. Πιο περίπλοκες τεχνικές αναζήτησης όπως deterministic annealing, μέθοδοι στοχαστικής βελτιστοποίησης (stochastic optimization methods) κ.α. είναι πιο ευέλικτες και αποδοτικές.

Εμπνευσμένη από την φυσική διαδικασία της εξέλιξης, η ερευνητική κοινότητα δημιούργησε αλγόριθμους που προσπαθούν να μιμηθούν αυτή τη διαδικασία. Το αποτέλεσμα ήταν αυτό που ονομάζεται σήμερα ως εξελικτικός υπολογισμός (evolutionary computation) ο οποίος αποτελείται από τους γενετικούς

αλγόριθμους (Genetic Algorithms, GA),εξελικτικές στρατηγικές (evolutionary strategies, ESs), εξελικτικό προγραμματισμό (evolutionary programming, EG) και τον γενετικό προγραμματισμό (genetic programming, GP). Με τη χρήση των παραπάνω αλγορίθμων επιτυγχάνεται η βελτιστοποίηση του πληθυσμού (σετ δεδομένων στη περίπτωση της συσταδοποίησης) μέσω ενός συνόλου εξελικτικών τελεστών. Μια συνάρτηση βελτιστοποίησης, η οποία ονομάζεται συνάρτηση καταλληλότητας είναι το μέτρο με το οποίο καθορίζεται ο βαθμός βελτιστοποίησης του πληθυσμού, από τον οποίο κάθε μέλος έχει μια τιμή που του αντιστοιχεί. Κάποιοι από τους πιο διαδεδομένους εξελικτικούς τελεστές, είναι η επιλογή, η ανακατανομή και η μετάλλαξη (selection, recombintation, mutation). Ο τελεστής της επιλογής εξασφαλίζει τη ”διαιώνιση” του πληθυσμού ευνοώντας τα καταλληλότερα μέλη για την επόμενη γενιά. Οι τελεστές της ανακατάταξης και μετάλλαξης εξασφαλίζουν την ποικιλομορφία του πληθυσμού μέσω εναλλαγών στα μέλη.

Από τους αλγόριθμους που ανήκουν στην κατηγορία του εξελικτικού υπολογισμού, οι γενετικοί αλγόριθμοι αποδείχθηκαν οι πιο δημοφιλείς για εφαρμογές ανάλυσης συστάδων. Στους γενετικούς αλγορίθμους, το κάθε μέλος κωδικοποιείται ως μία συμβολοσειρά από bit (binary bit string) το οποίο ονομάζεται χρωμόσωμα. Στη συνέχεια, παράγεται ένας αρχικό πληθυσμός βάση κάποιου ευρετικού κριτηρίου ή ακόμα και τυχαία. Τέλος, μετά την διαδοχική εφαρμογή των τελεστών (όπως η επιλογή, η μετάλλαξη και η μίξη) στον πληθυσμό, ελέγχεται η συνθήκη παύσης, η οποία τερματίζει την διαδικασία αν ικανοποιηθεί.

Οι Hall, Özyurt, και Bezdek πρότειναν τον αλγόριθμο GGA ο οποίος αποτελεί μια παραλλαγή του γενετικού αλγορίθμου που περιγράφηκε παραπάνω αλλά με χρήση της αλγορίθμου gradient descent το οποίο μπορεί να θεωρηθεί ως ένα γενικό σχεδιάγραμμα για τα προβλήματα συσταδοποίησης που βασίζονται σε

κάποιο κέντρο (center-based). Οι συναρτήσεις καταλληλότητας αναδιαμορφώνονται από την συνάρτηση κριτηρίου αθροίσματος τετραγωνικών λαθών, έτσι ώστε να προσαρμοστεί στην αλλαγή της δομής του προβλήματος βελτιστοποίησης.

$$J(\boldsymbol{M}) = \sum_{j=1}^{N} \min(D_{1j}, D_{2j}, \ldots, D_{Kj}) \qquad \text{για αυστηρή ομαδοποίηση}$$

$$J(\boldsymbol{M}) = \sum_{j=1}^{N} \left(\sum_{i=1}^{K} D_{ij}^{1/(1-m)} \right) \qquad \text{για ασαφή (fuzzy) ομαδοποίηση}$$

όπου το $D_{ij}, i = 1, \ldots, K, j = 1, \ldots, N$, είναι η απόσταση μεταξύ της συστάδας i και του δεδομένου j και το m είναι η παράμετρος ασάφειας.

Ο GGA πραγματοποιεί τα ακόλουθα βήματα

1) Επιλογή των κατάλληλων παραμέτρων αλγορίθμου. Τυχαία αρχικοποίηση του πληθυσμού με χρήση P μελών, το κάθε μέλος αναπαριστά ένα πίνακα πρωτοτύπων $K \times d$ και είναι κωδικοποιημένο χρησιμοποιώντας τον κώδικα Gray.

2) Γίνεται χρήση του τελεστή της επιλογής (επιλογή τύπου τουρνουά) για να διαλέξει τα μέλη γονείς για την ”αναπαραγωγή”.

3) Γίνεται χρήση των τελεστών της μετάλλαξης (bit-wise mutation) και της μίξης (two-point crossover) για την παραγωγή των ”παιδιών” των μελών που επιλέχθηκαν στο προηγούμενο βήμα (2)

4) Αποφασίζεται η επόμενη γενιά κρατώντας τα μέλη που έχουν το υψηλότερο σκορ καταλληλότητας.

5) Επανάληψη των βημάτων 2)-4) μέχρι να ικανοποιηθεί η συνθήκη παύσης.

Πέρα από τον GGA, δημιουργήθηκαν και άλλοι αλγόριθμοι που βασίζονται στους γενετικούς αλγορίθμους διαφοροποιώντας κάποιο συνδυασμό της κωδικοποίησης του πληθυσμού, της έννοιας του μέλους, της συνάρτησης καταλληλότητας ή και τον εξελικτικών τελεστών.

Ο αλγόριθμος CLUSTERING, συμπεριλαμβάνει μία ευρετική διαδικασία για την εκτίμηση του αριθμού των συστάδων που υπάρχουν στα δεδομένα. Επιπλέον, χρησιμοποιεί τον αλγόριθμο του κοντινότερου γείτονα (nearest-neighbour) για να διασπάσει το σετ δεδομένων σε μικρότερα σύνολα πριν ξεκινήσει την συσταδοποίηση, έτσι ώστε να μειώσει την υπολογιστική πολυπλοκότητα.

Οι γενετικοί αλγόριθμοι χρησιμοποιούνται επίσης για την βελτίωση της επίδοσης των αλγορίθμων K-means. Οι Babu και Murty, χρησιμοποίησαν του γενετικούς αλγόριθμους για να βελτιστοποιήσουν τον αρχικό διαχωρισμό. Οι Krishna και Murty, συνδύασαν τους γενετικούς αλγορίθμους με τον K-means και δημιούργησαν τον αλγόριθμο GKA ο οποίος έχει τη δυνατότητα εύρεσης του καθολικά βέλτιστου αποτελέσματος.

Πέρα από της προηγούμενες εφαρμογές, οι γενετικοί αλγόριθμοι μπορούν να χρησιμοποιηθούν και για ιεραρχική συσταδοποίηση. Οι Loranzo και Larrañag συζήτησαν τις ιδιότητες της ultrametric απόστασης (η μετρική ultrametric αποτελεί μη Αρχιμήδεια μετρική) και επαναπροσδιόρισαν την ιεραρχική συσταδοποίηση ως ένα πρόβλημα βελτιστοποίησης με στόχο να βρει την μικρότερη ultrametric απόσταση για μια δοθέν ανομοιότητα κάνοντας χρήση της Ευκλείδειας νόρμας. Η πρόταση τους για την επίλυση του παραπάνω προβλήματος είναι ένα order-based γενετικός αλγόριθμος.

2.2.6 Ασαφής Ομαδοποίηση

Μέχρι στιγμή έχει παρουσιαστεί μια πληθώρα αλγορίθμων συσταδοποίησης με την πλειοψηφία τους (όλοι εκτός του GGA) να χαρακτηρίζονται ως αλγόριθμοι ”αυστηρής” συσταδοποίησης (hard clustering), το οποίο σημαίνει ότι κάθε αντικείμενο ανατίθεται σε μία και μόνο μία συστάδα. Για την ασαφή συσταδοποίηση, ο παραπάνω περιορισμός χαλαρώνει, με το κάθε αντικείμενο να έχει πλέον ένα βαθμό συμμετοχής στη κάθε συστάδα. Όπως μπορεί κανείς να φανταστεί, αυτό είναι πολύ χρήσιμο ειδικά στις περιπτώσεις όπου τα σύνορα μεταξύ των συστάδων δεν είναι καλά ορισμένα. Επιπλέον, η συμμετοχή ενός αντικειμένου σε περισσότερες από μία συστάδες, προσφέρει παραπάνω πληροφορία με το πιθανό ενδεχόμενο ανακάλυψης πιο ”εκλεπτυσμένων” σχέσεων μεταξύ αντικειμένου και συστάδων.

Ο FCM είναι ένας από τους πιο δημοφιλής ασαφείς αλγόριθμους συσταδοποίησης. Προτάθηκε από τον Bezdek και μπορεί να θεωρηθεί γενίκευση του αλγόριθμου ISODATA. Ο αλγόριθμος FCM επιχειρεί να διαχωρίσει τα ένα σετ δεδομένων $x_j \in \mathbb{R}^d,\ j = 1, \ldots, N$ σε c ασαφής συστάδες ενώ ελαχιστοποιεί τη συνάρτηση κόστους

$$J(\boldsymbol{U}, \boldsymbol{M}) = \sum_{i=1}^{c} \sum_{j=1}^{N} (u_{i,j})^m D_{ij}$$

όπου

$U = [u_{i,j}]_{c \times N}$	είναι ο πίνακας ασαφών συστάδων και $u_{ij} \in [0,1]$ είναι ο συντελεστής συμμετοχής του αντικείμενου j στην συστάδα i
$\boldsymbol{M} = [\boldsymbol{m}_1, \ldots, \boldsymbol{m}_c]$	είναι ο πίνακας των πρωτοτύπων των συστάδων (μέσων ή κέντρων)
$m \in [1, \text{inf})$	είναι η παράμετρος ασάφειας (η οποία συνήθως τίθεται στο 2)
$D_{ij} = D(\boldsymbol{x}_j, \boldsymbol{m}_i)$	είναι το μέτρο της απόστασης μεταξύ του x_j και του $\boldsymbol{m}_i$

Ο κανονικός αλγόριθμος FCM συνοψίζεται ως εξής (για εξίσωση απόστασης χρησιμοποιείται η Ευκλείδεια ή L_2 νόρμα απόσταση)

1) Επιλογή κατάλληλων τιμών για τα m, c και ενός μικρού θετικού αριθμού ϵ. Αρχικοποίηση του πίνακα πρωτοτύπων $\boldsymbol{M}$ τυχαία. Η μεταβλητή t τίθεται στο 0.

2) Υπολογισμός (όταν $\alpha t = 0$) ή ενημέρωση (όταν $t > 0$) του πίνακα συμμετοχής $\boldsymbol{U}$ σύμφωνα με

$$u_{ij}^{t+1} = 1 \Bigg/ \left(\sum_{l=1}^{c} (D_{lj}/D_{ij})^{1/(1-m)} \right) \quad \text{για } i = 1, \ldots, c \text{ και } j = 1, \ldots, N.$$

3) Ενημέρωση του πίνακα πρωτοτύπων $\boldsymbol{M}$ σύμφωνα με

$$\boldsymbol{m}_i^{(t+1)} = \left(\sum_{j=1}^{N} \left(u_{ij}^{(} t+1) \right)^m \boldsymbol{x}_j \right) \Bigg/ \left(\sum_{j=1}^{N} \left(u_{ij}^{(} t+1) \right)^m \right) \quad \text{για } i = 1, \ldots, c \ .$$

4) Επανάληψη των βημάτων 2)-3) μέχρι $||\boldsymbol{M}(t+1) - \boldsymbol{M}(t)|| < \epsilon$.

Ως αποτέλεσμα της εκτεταμένης έρευνας στα μέτρα απόστασης, τις νέες προσεγγίσεις σχετικά με την βελτιστοποίηση του ασαφή διαμερισμού και τις βελ-

τιώσεις στα μειονεκτήματα του FCM, έχουν δημιουργηθεί πολλές παραλλαγές του αλγορίθμου FCM αλλά και άλλων αλγορίθμων ασαφούς συσταδοποίησης.

Ακριβώς όμως όπως και ο αλγόριθμος ISODATA, η "αυστηρή" εκδοχή του αλγόριθμου FCM, πάσχει από ευαισθησία στον θόρυβο και ακρότατα, ταυτόχρονα με την δυσκολία εύρεσης των αρχικών διαμοιρασμών. Οι Yager και Filev πρότειναν την μέθοδο MM (Mountain Method) για την εκτίμηση των κέντρων των συστάδων. Τα υποψήφια κέντρα αποτελούνται από ένα σετ διανυσμάτων που σχηματίζονται με τη κατασκευή ενός πλέγμα στο χώρο των μοτίβων (pattern space). Η συνάρτηση mountain για ένα διάνυσμα v_i ορίζεται ως

$$M(\boldsymbol{v_i}) = \sum_{j=1}^{M} e^{-\alpha D(\boldsymbol{x}_j, \boldsymbol{v}_i)}$$

όπου $D(\boldsymbol{x}_j, \boldsymbol{v}_i)$ είναι η απόσταση μεταξύ του δεδομένου j και του κόμβου i και το α είναι μια θετική σταθερά. Συνεπώς, όσο πιο κοντά είναι ένα δεδομένο σε έναν κόμβο, τόσο περισσότερο συνεισφέρει στη συνάρτηση mountain. Ο κόμβος $\boldsymbol{v}_{m1}$ με τη μέγιστη τιμή της συνάρτησης mountain $\boldsymbol{v}_{m1}$ επιλέγεται ως το πρώτο κέντρο. Μια διαδικασία, που ονομάζεται mountain destruction, πραγματοποιείται για να μην επηρεάζει το επιλεγμένο κέντρο τους υπόλοιπους υποψήφιους. Αυτό πραγματοποιείται αφαιρώντας από την τιμή της συνάρτησης mountain καθενός από τους υπόλοιπους κόμβους, μία ποσότητα που αντιστοιχεί στην τρέχουσα μέγιστη τιμή και την απόσταση μεταξύ κόμβου και κέντρου. Η διαδικασία επαναλαμβάνεται μέχρι ο λόγος μεταξύ τρέχων μέγιστου και του $\boldsymbol{v}_{m1}$ είναι μικρότερος από κάποιο κατώφλι. Οι Gath και Geva περιέγραψαν μια στρατηγική αρχικοποίησης για μη επιβλεπόμενο εντοπισμό πρωτότυπων των συστάδων συνδυάζοντας στην ουσία τον FCM και την ασαφή εκτίμηση ML σε ένα σχεδιάγραμμα συσταδοποίησης δύο επιπέδων (2 layer scheme).

Ο Kersten πρότεινε ότι η απόσταση Manhattan (L_1 νόρμα) ενδεχομένως να

βελτιώσει την ανθεκτικότητα του FCM απέναντι στα ακρότατα. Επιπλέον, οι Hathaway, Bezdek και Hu γενίκευσαν τον FCM χρησιμοποιώντας την απόσταση Minkowski (ή L_p νόρμα, $p \geq 1$) και ημινόρμα ($0 < p < 1$) για τα μοντέλα που επεξεργάζονται τα δεδομένα είτε απευθείας είτε έμμεσα μέσω των μέτρων διαφοράς/απόστασης. Σύμφωνα με τα εμπειρικά αποτελέσματα που πήραν, προτείνονται τα μοντέλα βασισμένα σε data objects που κάνουν χρήση των νορμών L_1 και L_2. Ανέφεραν επίσης την πιθανή βελτίωση των μοντέλων για άλλες L_p νόρμες με το μειονέκτημα όμως της αυξημένης πολυπλοκότητας των πράξεων βελτιστοποίησης.

Ο PCM είναι μια ακόμα προσέγγιση για την αντιμετώπιση των ακρότατων. Κάνοντας χρήση αυτού του μοντέλου, η συμμετοχές ερμηνεύονται από μία πιθανοτική οπτική γωνία, δηλαδή τη συμβατότητα των σημείων με τα πρωτότυπα των κλάσεων. Η επίδραση του θορύβου και των ακρότατων . Σε αυτή τη περίπτωση, η συνθήκη που ισχύει για τους συντελεστές συμμετοχής "χαλαρώνει" από

$$\sum_{i=1}^{c} u_{i,j} = 1, \forall j \quad \text{και} \quad \sum_{j=1}^{N} u_{i,j} < N, \forall i$$

, όπου c είναι το πλήθος των συστάδων και ο πλήθος των δεδομένων.

σε $\max_i u_{i,j} > 0, \forall j$. Αντιστοίχως, η συνάρτηση κόστους αλλάζει σε

$$J(\boldsymbol{U}, \boldsymbol{M}) = \sum_{i=1}^{c} \sum_{j=1}^{N} (u_{i,j})^m D_{ij} + \sum_{i=1}^{c} \eta_i \sum_{j=1}^{N} (1 - u_{i,j})^m$$

όπου η_i είναι μία θετική σταθερά. Ο επιπλέον όρος τείνει να δίνει μεγαλύτερο "κέρδος" σε μέλη με μεγάλες τιμές.

Ο FCM εναλλάσσεται μεταξύ υπολογισμού του πίνακα συμμετοχής και πρωτοτύπων, το οποίο είναι εξαιρετικά υπολογιστικά ακριβό για μεγάλα σετ δεδομένων/ σετ μεγάλων δεδομένων. Οι Kolen και Hutcheson επιτάχυναν τον υπολογισμό συνδυάζοντας την ενημέρωση των δύο πινάκων. Οι Hung και Yang

πρότειναν μία μέθοδο για μείωση του χρόνου υπολογισμού εντοπίζοντας πιο ακριβείς κέντρα για τις συστάδες. Δημιουργήθηκαν επίσης παραλλαγές του FCM έτσι ώστε να υπάρχει η δυνατότητα επεξεργασίας και άλλων τύπων δεδομένων.

Οι αλγόριθμοι της οικογένειας του fuzzy c-shells φαίνεται να έχουν τη δυνατότητα εντοπισμού διαφορετικών σχημάτων συστάδων, ειδικά τα περιγράμματα (γραμμές, κύκλοι, ελλείψεις, δαχτυλίδια, παραλληλόγραμμα, υπερβολές) σε ένα δισδιάστατο χώρο δεδομένων. Χρησιμοποιούν "κελύφη" (shells) (κυρτές επιφάνειες) ως το πρωτότυπο της συστάδας αντί για τα σημεία ή τις επιφάνειες όπως στου παραδοσιακούς αλγορίθμους ασαφής συσταδοποίησης. Στην περίπτωση του FCS, το προτεινόμενο πρωτότυπο για την συστάδα αναπαριστάται ως ένα υπερσφαιρικό κέλυφος d διαστάσεων $\boldsymbol{m}(\boldsymbol{v}, r)$ (όπου το $d = 2$), όπου $\boldsymbol{v} \in \mathbb{R}^d$ είναι το κέντρο και $r \in \mathbb{R}$ είναι η ακτίνα. Μια συνάρτηση απόστασης ορίζεται ως $D(\boldsymbol{x}_i, \boldsymbol{m}_i) = (||\boldsymbol{x}_i - \boldsymbol{v}_j|| - r_j)$ για να μετρήσει την απόσταση από το δεδομένο x_j στο πρωτότυπο $\boldsymbol{m}_j$. Αντίστοιχα, γίνεται να επιτευχθούν και άλλα σχήματα συστάδων με τον ορισμό κατάλληλων πρωτότυπων και συναρτήσεων απόστασης. Κάποια παραδείγματα είναι ο αλγόριθμος FCSS για ασαφή c σφαιρικά κελύφη, ο FCR για ασαφή c-rings, ο FCQS για ασαφή c-quadratic κελύφη και ο FCRS για ασαφή c-παραλληλόγραμμα κελύφη.

Οι θεωρίες ασαφούς λογικής μπορούν να χρησιμοποιηθούν και για τον ορισμό τεχνικών ιεραρχικής συσταδοποίησης. Ο Geva πρότεινε έναν αλγόριθμος μη-επιβλεπομένης ασαφούς ιεραρχικής συσταδοποίησης (HUFC), η οποία μπορεί να ψάξει αποδοτικά δομές δεδομένων σε διαφορετικά επίπεδα, όπως ο αλγόριθμος ιεραρχικής συσταδοποίησης, ενώ καθορίζει συνδέσεις μεταξύ κάθε αντικειμένου και τις συστάδες χρησιμοποιώντας το ποσοστό συμμετοχής. Αυτός ο σχεδιασμός καθιστά τον HUFC ικανό να "ξεπεράσει" τα μέγιστα μειονεκτήματα των ιεραρχικών αλγορίθμων, όπως την αδυναμία ανακατάταξης των

δεδομένων αφού έχουν μπει σε μία συστάδα.

2.2.7 Ομαδοποίηση βασισμένη σε πυρήνες (Kernels)

Οι αλγόριθμοι μάθησης βασισμένοι στους πυρήνες (kernels), είναι βασισμένοι στο θεώρημα του Cover Με τον μη-γραμμικό μετασχηματισμό ενός σετ δεδομένων, που είναι σύνθετα και μη γραμμικά διαχωρίσιμα, σε ένα χώρο υψηλότερων διαστάσεων, δημιουργείται η πιθανότητα διαχωρισμού των δεδομένων γραμμικά. Δυστυχώς όμως, με την μεταφορά των δεδομένων σε ένα πολυ-διάστατο χώρο, δημιουργείται το νέο πρόβλημα γνωστό και ως "κατάρα της διαστατικότητας". Το πρόβλημα αυτό αντιμετωπίζεται με τη χρήση του kernel trick, το οποίο προκύπτει από το θεώρημα του Mercer. Με τον κατάλληλο σχεδιασμό και τον υπολογισμό ενός πυρήνα εσωτερικού γινομένου, είναι δυνατή η παράκαμψη της επίπονης, χρονοβόρας και κάποιες φορές αδύνατης διαδικασίας υπολογισμού των συντεταγμένων των σημείων στον μετασχηματισμένο πολυδιάστατο χώρο, που απαιτείται από την απεικόνιση τους σε αυτόν.

Οι Schölkopf, Smola και Müller αποτύπωσαν μια online εκδοχή του K-means που βασίζεται σε πυρήνες (kernel-K-means). Αν υποτεθεί ότι υπάρχει ένα σετ στοιχείων $x_j \in \mathbb{R}^d, j = 1, \ldots, N$ και μία μη-γραμμική απεικόνιση $\Phi : \mathbb{R}^d \rightarrow F$. Εδώ, το F αντιπροσωπεύει ένα feature space με τυχαία μεγάλο αριθμό διαστάσεων. Ο στόχος του αλγορίθμου είναι να βρει K κέντρα έτσι ώστε να ελαχιστοποιηθεί η απόσταση μεταξύ των απεικονισμένων στοιχείων και των κοντινότερων τους κέντρων (όπως και στον παραδοσιακό K-means)

$$
\begin{aligned}
&||\Phi(\boldsymbol{x}) - m_l||^2 \\
&\quad = ||\Phi(\boldsymbol{x}) - \sum_{j=1}^{N} \tau_{lj}\Phi(\boldsymbol{x}_j)||^2 \\
&\quad = k(\boldsymbol{x},\boldsymbol{x}) - 2\sum_{j=1}^{N} \tau_{lj}k(\boldsymbol{x},\boldsymbol{x}_j) + \sum_{i,j=1}^{N} \tau_{li}\tau_{lj}k(\boldsymbol{x}_i,\boldsymbol{x}_j)
\end{aligned}
$$

όπου m_l είναι το κέντρο της l συστάδας και ανήκει $\Phi(x_1), \ldots, \Phi(x_N)$,και $k(x, x_j) = \Phi(x) \cdot \Phi(x_j)$ είναι ο πυρήνας εσωτερικού γινομένου.

Η μεταβλητή ανάθεσης σε συστάδα ορίζεται ως

$$
C_{jl} = \begin{cases} 1, & \text{αν το } \ x_j \text{ ανήκει στη συστάδα } \ l \\ 0, & \text{διαφορετικά} \end{cases}
$$

Τότε ο αλγόριθμος kernel-K-means μπορεί να αναλυθεί ως εξής

1) Αρχικοποίηση των κέντρων m_l με τα πρώτα i, $(i \geq K)$, παρατηρημένα στοιχεία

2) Για ένα νέο στοιχείο x_{i+1} υπολογίζεται το $C_{i+1}h$ όπως φαίνεται στην εξίσωση στην υποσημείωση

3) Ενημέρωση του μέσου διανύσματος m_h του οποίου το αντίστοιχο $C_{i+1}h$ είναι 1

$$
\boldsymbol{m}_h^{new} = \boldsymbol{m}_h^{old} + \xi(\Phi(\boldsymbol{x}_{i+1}) - \boldsymbol{m}_h^{old})
$$

όπου $\xi = C_{(i+1)h} / \sum_{j=1}^{i+1} C_{jh}$.

4) Προσαρμογή του συντελεστή τ_{hj} για κάθε $\phi(x_j)$ ως

$$
\tau_{hj}^{new} = \begin{cases} \tau_{hj}^{old}(1-\xi), & \text{για } j \neq i+1 \\ \xi, & \text{για } j = i+1 \end{cases}
$$

5) Επανάληψη των βημάτων 2) - 4) μέχρι να επιτευχθεί σύγκλιση.

Δύο διαφορετικές εκδοχές του kernel-K-means εμπνεύστηκαν από τα δίκτυα SOFM και ART. Αυτές οι παραλλαγές στον αρχικό αλγόριθμο kernel-K-means συνυπολογίζουν την επίδραση των "γειτονικών" σχέσεων κατά τη διαδικασία της ρύθμισης των μεταβλητών ανάθεσης σε συστάδες και κάνουν χρήση μια παραμέτρου vigilance (επαγρύπνησης) για να "ελέγξουν" την διαδικασία παραγωγής μέσου διανύσματος. Οι συγγραφείς έδωσαν παράδειγμα υλοποίησης των παραλλαγών σε case based reasoning συστήματα.

Σε μια εναλλακτική προσέγγιση στη χρήση πυρήνων για συσταδοποίηση το πρόβλημα διασκευάστηκε έτσι ώστε στόχος να είναι ο βέλτιστος διαμερισμός $\mathbf{\Gamma}$ με αποτέλεσμα την ελαχιστοποίηση του ίχνους του πίνακα within-group scatter στο feature space

$$\begin{aligned}\mathbf{\Gamma} &= arg_{\mathbf{\Gamma}} \min Tr(S_W^{\Phi}) \\ &= arg_{\mathbf{\Gamma}} \min Tr\left\{ \frac{1}{N} \sum_{i=1}^{K} \sum_{j=1}^{N} \gamma_{ij} (\Phi(x_j) - m_i) \times (\Phi(x_j) - m_i)^T \right\} \\ &= arg_{\mathbf{\Gamma}} \max \sum_{i=1}^{K} \xi_i R(x|C_i)\end{aligned}$$

όπου

$\xi_i = N_i/N, R(x|C_i) = (1/N_i^2) \sum_{l=1}^{N} \sum_{j=1}^{N} \gamma_{il}\gamma_{ij} k(\boldsymbol{x}_l, \boldsymbol{x}_j)$, και το N_i είναι ο συνολικό αριθμός μοτίβων στην iοστή συστάδα.

Να σημειωθεί ότι η συνάρτηση πυρήνα που χρησιμοποιείται στη συγκεκριμένη περίπτωση είναι η radial basis function (RBF) και το $R(x|C_i)$ μπορεί να ερμηνευτεί ως ένα μέτρο της πυκνότητας για την συστάδα i.

Οι αλγόριθμοι συσταδοποίησης που βασίζονται στους πυρήνες έχουν πολλά πλεονεκτήματα κάποια από τα οποία είναι:

1) Είναι πολύ πιο πιθανό να αποκτηθεί γραμμικά διαχωρίσιμο υπερεπίπεδο σε

χώρους με μεγάλο ή άπειρο αριθμό διαστάσεων.

2) Υπάρχει η δυνατότητα να σχηματίσουν αυθαίρετα σχήματα πέρα από υπερσφαίρες και υπερελλειψοειδή.

3) Αλγόριθμοι συσταδοποίησης που βασίζονται σε πυρήνες, όπως ο SVC, έχουν τη δυνατότητα αντιμετώπισης θορύβου και ακραίων σημείων

4) Για τον SVC, δεν χρειάζεται προηγούμενη γνώση για να καθοριστεί η τοπολογική δομή του συστήματος με τον πίνακα πυρήνα να παρέχει τα μέσα για να εκτιμηθεί ο αριθμός των συστάδων.

Παρ' όλα αυτά, όπως και σε πολλούς άλλους αλγορίθμους υπάρχουν κάποια προβλήματα που χρειάζονται περαιτέρω διερεύνηση. Ένα από αυτά, είναι η εκτίμηση των παραμέτρων, για παράδειγμα, το πλάτος του Γκαουσιανού πυρήνα, δεν είναι διόλου ασήμαντο και μπορεί να γίνει σοβαρό πρόβλημα για μεγάλα σετ δεδομένων.

Η διαδικασία κατασκευής του αλγορίθμου συσταδοποίησης αθροίσματος λαθών και του αλγορίθμου K-means είναι ένα καλό παράδειγμα ανακατασκευής πιο "ισχυρών" μη-γραμμικών εκδοχών για πολλούς ήδη υπάρχοντες γραμμικούς αλγόριθμους, με τη προϋπόθεση ότι μπορεί να παραχθεί το εσωτερικό γινόμενο. Θεωρητικά, είναι σημαντικό να ερευνηθεί αν οι μη-γραμμικές εκδοχές μπορούν επιτυχών να διατηρήσουν κάποιες από τις χρήσιμες και απαραίτητες ιδιότητες των αρχικών αλγορίθμων και αν οι πυρήνες Mercer συνεισφέρουν θετικά στην βελτίωση των αλγορίθμων αυτών.

2.2.8 Ομαδοποίηση βασισμένη στη πυκνότητα (Density Based)

Η ιδέα των αλγορίθμων συσταδοποίησης που βασίζονται στην πυκνότητα είναι ότι οι συστάδες είναι περιοχές με μεγάλη πυκνότητα δεδομένων ή σημείων, οι

οποίες διαχωρίζονται από περιοχές με σημαντικά χαμηλότερη πυκνότητα σημείων και μπορούν να τυποποιηθούν χρησιμοποιώντας δύο παραμέτρους, εξ ονόματι την $\epsilon \in \mathbb{R}^+$ και την $MinPts \in \mathbb{N}^+$ [2]. Οι αλγόριθμοι γυρίζουν γύρω από το σημείο πυρήνα ή βασικό σημείο. Ένα δεδομένο P ονομάζεται σημείο πυρήνα μίας συστάδας αν έχει τουλάχιστον $MinPts$ δεδομένα (συμπεριλαμβανομένου και του ίδιου του P) στην ϵ-γειτονιά του, η οποία αναγράφεται $N_\epsilon(P)$ και αντιστοιχεί σε μία σφαίρα ακτίνας ϵ. Τυπικά:

Σημείο Πυρήνα

Έστω D ένα σετ n δεδομένων από το $\mathbb{R}^d$, όπου d ο αριθμός των διαστάσεων, $\epsilon \in \mathbb{R}^+$ και $MinPts \in \mathbb{N}^+$. Ένα δεδομένο $P \in D$ είναι σημείο πυρήνα αν και μόνο αν

$$|N_\epsilon(P)| \geq MinPts, \quad \text{όπου } N_\epsilon(P) = \{Q \in D : ||P - Q|| \leq \epsilon\}$$

Δύο δεδομένα μπορούν να ανατεθούν σε κοινή συστάδα. Στη συσταδοποίηση που βασίζεται στη πυκνότητα αυτό τυποποιείται με τις έννοιες direct density reachability και density connectedness.

Direct Density Reachability

Έστω $P, Q \in D$. Το Q λέγεται directly density reachable από το P (συμβολικά: $P \triangleleft Q$) αν και μόνο αν

1. Το P είναι σημείο πυρήνα στο D, και
2. $Q \in N_\epsilon(P)$

Αν τα P και Q είναι και τα δύο σημεία πυρήνα τότε το $P \triangleleft Q$ είναι ισοδύναμο με το $P \triangleright Q$. Η density connectedness είναι η μεταβατική και συμμετρική

εκδοχή του direct density reachability.

Density Connectedness

Δύο δεδομένα P και Q ονομάζονται density connected (συμβολικά $P \bowtie Q$) αν και μόνο αν υπάρχει μια ακολουθία βασικών σημείων ($P_1, \ldots, P_m$) αυθαίρετου μήκους τέτοια ώστε

$$P \triangleright P_1 \triangleright \ldots triangleleft P_m \triangleleft Q$$

Στη συσταδοποίηση που βασίζεται στη πυκνότητα, μια συστάδα ορίζεται ως το μέγιστο σετ density connected δεδομένων:

Συστάδα

Ένα υποσύνολο $C \subseteq D$ ονομάζεται συστάδα αν και μόνο αν τηρεί δύο προϋποθέσεις:

1. Density connectedness: $\forall P, Q \in C : P \bowtie Q$
2. Μεγιστοποίηση: $\forall P \in C, \forall Q \in D \setminus C : \neg P \bowtie Q$

Ο αλγόριθμος DBSCAN υλοποιεί αυτή τη μεθοδολογία συσταδοποίησης με μία δομή δεδομένων που ονομάζεται seed list S και περιέχει δεδομένα για επέκταση της συστάδας. Πιο συγκεκριμένα, ο αλγόριθμος εκτελεί τα ακόλουθα βήματα:

1. Όλα τα δεδομένα σημειώνονται ως μη επεξεργασμένα.
2. Εξετάζεται τυχαίο μη επεξεργασμένο δεδομένο $P \in D$.
3. Αν το P είναι σημείο πυρήνα, ανατίθεται νέο ID συστάδας C και εκτελείται το βήμα (4) για όλα τα δεδομένα $Q \in N_\epsilon(P)$, τα οποία δεν έχουν ακόμα ID συστάδας:

4. (α΄) όλα τα δεδομένα Q σημειώνονται με ID συστάδας C και

   (β΄) τα δεδομένα Q μπαίνουν στο seed list S

5. Όσο το S δεν είναι άδειο επαναλαμβάνεται το βήμα (6) για όλα τα δεδομένα $P' \in S$:

6. Αν το P' είναι σημείο πυρήνα, εκτελείται το βήμα (7) για όλα τα δεδομένα $Q \in N_\epsilon(P')$, τα οποία δεν έχουν ακόμα ID συστάδας:

7. (α΄) όλα τα δεδομένα Q σημειώνονται με ID συστάδας C και

   (β΄) τα δεδομένα Q μπαίνουν στο seed list S

8. Αν το D περιέχει ακόμα μη επεξεργασμένα δεδομένα, τότε εκτελείται το βήμα (2).

Κεφάλαιο 3

Τεχνικές Παραλληλοποίησης Αλγορίθμων Συσταδοποίησης για Κάρτες Γραφικών

Όπως αναφέρθηκε και στην εισαγωγή, με την δημιουργία και χρήση των αλγόριθμων συσταδοποίησης έγινε η απόπειρα αξιοποίησης της υπολογιστικής ισχύος των ηλεκτρονικών υπολογιστών για την σχετικά αυτοματοποιημένη επεξεργασία ενός μεγάλου εύρους δεδομένων. Με την πάροδο των χρόνων όμως και την εξέλιξη και διάδοση της τεχνολογίας, η ποσότητα των δεδομένων τα οποία χρειάζονται επεξεργασία έχει αλλάξει κλίμακα. Πλέον το μέγεθος, η ταχύτητα παραγωγής και η ποικιλία τους, έχουν οδηγήσει την ερευνητική κοινότητα να χρησιμοποιήσει έναν καινούργιο όρο για να τα χαρακτηρίσει. Ο όρος αυτός είναι Big Data. Χάρη στα Big Data, οι αλγόριθμοι συσταδοποίησης που έχουν παρουσιαστεί μέχρι στιγμής θεωρούνται επί το πλείστον παρωχημένοι[5]. Για τον λόγο αυτό έχει επενδυθεί πολλή ενέργεια ώστε να εκσυγχρονιστούν και να παραλληλοποιηθούν με το στόχο την επιτάχυνση, κλιμακοθεσιμότητας και throuput.

Για την υλοποίηση της παραλληλοποίησης υπάρχουν πολλές προσεγγίσεις οι οποίες κατηγοριοποιούνται σε μία από δύο κατηγορίες, την οριζόντια και την κάθετη κλιμακόσιμες πλατφόρμες. Στην πρώτη κατηγορία, δηλαδή τις οριζό-

ντια κλιμακούμενες πλατφόρμες, ακολουθείται η λογική του διαμοιρασμού του φόρτου εργασίας σε πλήθος μηχανών. Αυτό επιτυγχάνεται με τη χρήση πλατφορμών όπως τα δίκτυα peer-to-peer, το MapReduce και το Spark. Οι κάθετα κλιμακούμενες πλατφόρμες από την άλλη ακολουθούν τη λογική της αύξησης των πόρων ενός συστήματος όπως μνήμη, επεξεργαστικές μονάδες και γρηγορότερο hardware. Στη κατηγορία αυτή ανήκουν πλατφόρμες που κάνουν χρήση πολυπύρηνους επεξεργαστές, επεξεργαστικές μονάδες γραφικών (κάρτες γραφικών, GPUs), Field Programmable Gate Arrays (FPGA) και High Performance Computing Clusters (HPC).

Όλες οι πλατφόρμες που αναφέρθηκαν παραπάνω παρουσιάζουν ενδιαφέρον και αξίζουν να αναλυθούν, παρ' όλα αυτά, σε αυτή την εργασία θα παρουσιαστούν παραδείγματα μόνο παράλληλων αλγορίθμων συσταδοποίησης που κάνουν χρήση της επεξεργαστικής μονάδας γραφικών.

3.1 Πλατφόρμα GPU

Με την εμφάνιση προγραμματίσιμου υλικού γραφικών στις αρχές του 21ου αιώνα, η χρήση των καρτών γραφικών ως ένα σχετικά φθηνό υψηλά παραλληλοποιήσιμο συνεπεξεργαστή (co-processor) έγινε μια πολύτιμη διαθέσιμη επιλογή για την επίλυση του προβλήματος που αναφέρθηκε παραπάνω. Τα χρόνια που ακολούθησαν την κυκλοφορία των προγραμματίσιμων υλικών γραφικών, έγιναν πολυάριθμες δημοσιεύσεις που παρουσίαζαν τα πλεονεκτήματα των καρτών γραφικών από τις κεντρικές μονάδες επεξεργασίας. Ως αποτέλεσμα, επενδύθηκε αρκετός χρόνος στην απόπειρα μεταφοράς των τεχνικών παραλληλίας, που ήδη υπήρχαν, στις κάρτες γραφικών αλλά και στη δημιουργία πλαισίων που θα διευκόλυναν τον προγραμματισμό σε αυτές. Η δυσκολία που αντιμετωπίστηκε κατά τη διαδικασία τη μετατροπής των τεχνικών σε μία μορφή φιλική ως

προς το pipeline γραφικών, ήταν ότι ήταν αναγκαία η γνώση προγραμματισμού γραφικών υπολογιστών. Το πρόβλημα αυτό έγινε σημαντικά ευκολότερο να λυθεί με την εισαγωγή της γλώσσας CUDA από την NVIDIA. Η CUDA προσφέρει εκτεταμένη τεκμηρίωση και προσέγγιση υψηλού επιπέδου (high-level approach) στην χρήση καρτών γραφικών χάρη στη εύκολη ενσωμάτωση της με τις γλώσσες C και C++[15]. Στη συνέχεια αυτού του κεφαλαίου, θα παρουσιαστούν αλγόριθμοι που εκμεταλλεύονται τη γλώσσα CUDA με συγκεκριμένες υλοποιήσεις στο επόμενο κεφάλαιο.

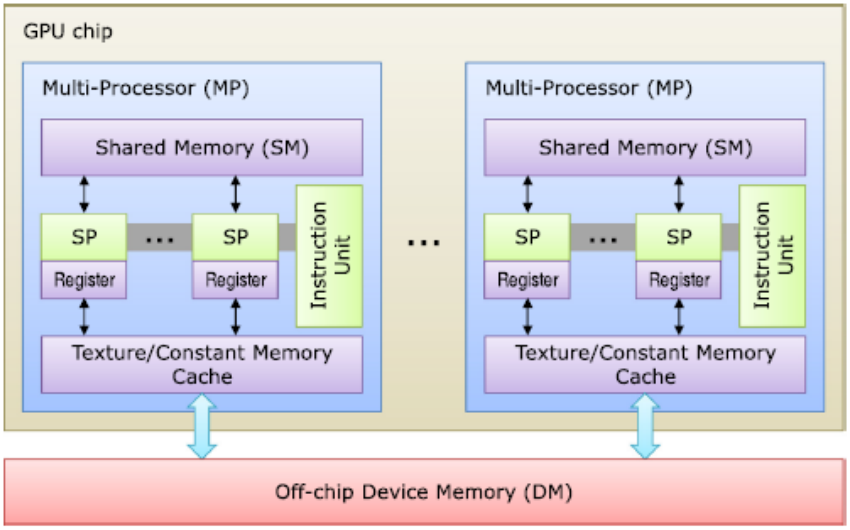
3.1.1 Λίγα λόγια για την CUDA

Το 2006 με τη σειρά 8800 και την κωδική ονομασία G80, η NVIDIA παρουσίασε την CUDA, μία πλατφόρμα παράλληλου υπολογισμού γενική χρήσης και προγραμματιστικό μοντέλο που αξιοποιεί την παράλληλη υπολογιστική μηχανή στις κάρτες γραφικών NVIDIA για την επίλυση σύνθετων υπολογιστικών προβλημάτων αποδοτικότερα από μία κεντρική μονάδα επεξεργασίας[10]. Από τότε, η NVIDIA έχει παρουσιάσει νέες, πιο προηγμένες αρχιτεκτονικές όπως Ampere, Turing, Volta και Pascal. Μία κάρτα γραφικών αποτελείται από πολλούς (streaming) multi-processors (MPs), ο καθένας από τους οποίους περιέχει,πολλούς stream processors (SPs) (κατά τη διάρκεια της συγγραφής ο αριθμός είναι 1024 SPs ανά MP). Κάθε MP είναι σχεδιασμένος να εκτελεί εκατοντάδες SPs ταυτοχρόνως και αυτό το πετυχαίνει χάρη την SIMT αρχιτεκτονική (Single-Instruction, Multiple-Thread). Τα SP είναι μία μονάδα εκτέλεσης και έχουν απλή δομή, όπως ένα ALU στους κεντρικούς επεξεργαστές. Ως γενικός κανόνας, κάθε MP εκτελεί ανεξάρτητες διαδικασίες ενώ κάθε SP που περιέχεται MP εκτελεί την ίδια πράξη/εντολή αλλά σε διαφορετικά δεδομένα. Από την αρχιτεκτονική Turing και μετά, η NVIDIA εισήγαγε στις κάρτες τις κάτι που ο-

νομάζεται GPU Processing Clusters (GPC). Το GPC είναι στην ουσία μια ομάδα από MPs. Από την έκδοση Compute Capability 9.0 (ή οποία δεν είναι διαθέσιμη ακόμα) χάρη στο GPC θα επιτρέπει η παράλληλη εκτέλεση των MPs[13].

Οι κάρτες γραφικών NVIDIA έχουν μία περίπλοκη κατανομή μνήμης που αποσκοπεί στης μεγιστοποίηση της υπολογιστικής ταχύτητας των εφαρμογών. Κάθε SP έχει τα δικά του registers και τοπική μνήμη για αποθήκευση δεδομένων και άλλα δεδομένα απαραίτητα για την εκτέλεση εντολών. Ένα SP δεν έχει πρόσβαση στα registers ή το local memory ενός άλλου SP. Ένας MP έχει διαμοιραζόμενη μνήμη και μνήμη cache, που είναι προσβάσιμη από όλα τα SP μέσα στον MP, αλλά όχι από τα SPs των άλλων MPs. Οι διαμοιραζόμενη μνήμη χρησιμοποιείται για την αποθήκευση κοινών δεδομένων και τον συγχρονισμό των SPs. Από το Compute Capability 9.x θα επιτρέπεται επίσης και η επικοινωνία μεταξύ διαφορετικών MPs μέσω κατανεμημένης μνήμης. Αντίθετα όμως με τη τοπική και διαμοιραζόμενη μνήμη που είναι ενσωματωμένη στα τσιπ, υπάρχει και μία μη-ενσωματωμένη μνήμη, η global μνήμη ή μνήμη device που είναι συνδεδεμένη με τα τσιπ και είναι προσβάσιμη από όλα τα MPs και τα SPs. Δεδομένα που είναι να χρησιμοποιηθούν από πολλά MPs, πρέπει να αποθηκευτούν στην global μνήμη, η οποία έχει πολύ μεγαλύτερη χωρητικότητα από τις ενσωματωμένες μνήμες. Το βασικό μειονέκτημα της global μνήμης είναι το ακριβό κόστος πρόσβασης συγκριτικά με τις ενσωματωμένες μνήμες (περίπου εκτατό φορές περισσότερο). Για το λόγο αυτό, προκειμένου να αυξηθεί η αποδοτικότητα της κάρτας γραφικών, η χρήση των ενσωματωμένων μνημών συνιστάται.

Όπως αναφέρθηκε παραπάνω, η CUDA δημιουργήθηκε για την αξιοποίηση των καρτώ γραφικών ως παράλληλη υπολογιστική μηχανή. Η CUDA είναι ένα σύνολο επεκτάσεων της C++ (και άλλων γλωσσών προγραμματισμού πλήρη



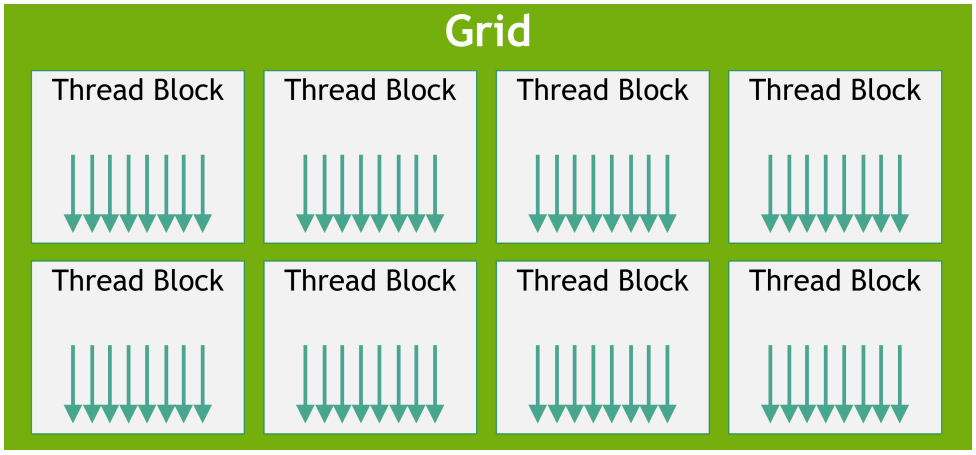
Σχήμα 3.1: Αρχιτεκτονική καρτών γραφικών NVIDIA[10]

λίστα στο [13]) και μεταγλωττίζεται σε ένα εκτελέσιμο χρησιμοποιώντας τα εργαλεία που παρέχονται στο toolkit της CUDA. Ένα πρόγραμμα CUDA περιέχει εντολές τόσο για την κάρτα γραφικών όσο και για την κεντρική μονάδα επεξεργασίας. Οι εντολές που εκτελούνται από την κάρτα γραφικών περιέχονται σε μία συνάρτηση που ονομάζεται kernel η οποία εκτελείται από τα SPs ταυτοχρόνως, ενώ οι εντολές που εκτελούνται από τον κεντρικό επεξεργαστή περιέχονται στη συνάρτηση ”host function”. Το πλήθος των SPs που εκτελούν τις εντολές του kernel είναι συγκεκριμένος και ορίζεται κατά την δημιουργία του. Ο αριθμός των thread που εκτελούν ένα πρόγραμμα ταυτοχρόνως είναι μεγαλύτερος από τα διαθέσιμα SP τότε το πρόγραμμα εκτελείται με time-sharing τρόπο. Η γενική ροή ενός προγράμματος CUDA είναι ως εξής:

(1) Τα δεδομένα που είναι να επεξεργαστούν στην κάρτα γραφικών αντιγράφονται στη global μνήμη

(2) Όταν κληθεί η συνάρτηση kernel από τον κεντρικό επεξεργαστεί, ξεκινάει ένας μεγάλος αριθμός νημάτων στην κάρτα γραφικών

(3) Κάθε νήμα επεξεργάζεται διαφορετικά δεδομένα, ενώ όλα τα νήματα εκτε-



Σχήμα 3.2: Οργάνωση Νημάτων στις πλατφόρμα CUDA[13]

λούν την ίδια συνάρτηση kernel.

(4) Όταν τερματίσουν όλα τα νήματα, τότε τα αποτελέσματα αντιγράφονται από τη μνήμη device πίσω στη κύρια μνήμη

Ένα πρόγραμμα CUDA μπορεί να επαναλάβει τα παραπάνω βήματα παραπάνω από μία φορές.

Η λογική δομή του τεράστιου αριθμού παράλληλων kernel νημάτων σε μία κάρτα γραφικών είναι ως εξής. Τα kernel threads χωρίζονται σε blocks, το καθένα από τα οποία έχει ισάριθμα (μεταξύ τους) threads. Τα blocks με την σειρά τους, οργανώνονται σε grids, αν και από το Compute Capability 9.x θα είναι δυνατόν να οργανωθούν και σε thread block clusters τα οποία είναι μία ενδιάμεση δομή (τα block clusters περιέχουν blocks και τα grids, θα περιέχουν block clusters). Κατά την κλήση ενός kernel δηλώνεται το μέγεθος των grids και των blocks. Σε ένα πρόγραμμα CUDA είναι δυνατό να εκτελεστούν συναρτήσεις kernel πολλές φορές με διαφορετικά μεγέθη για τα grids και τα blocks.

Η αντιστοίχιση από την φυσική δομή στη λογική της κάρτας γραφικών έχει ως εξής. Ένα block cluster αντιστοιχεί σε ένα GPC, ένα block αντιστοιχεί σε ένα MP και ένα thread αντιστοιχεί σε ένα SP. Ένα MP μπορεί να εκτελέσει

πολλά blocks, αλλά ένα block μπορεί να εκτελεστεί μόνο σε ένα MP. Για να αξιοποιηθεί η κάρτα γραφικών στο έπακρο, συνήθως τίθεται ο αριθμός των νημάτων ενός block ως ακέραιο πολλαπλάσιο του αριθμού των SPs. Τα νήματα σε ένα block χωρίζονται φυσικά σε ομάδες των 32 και ονομάζονται wraps. Όλα τα νήματα ενός wraps εκτελούν τον ίδιο κώδικα. Σχετικά πρόσφατα δόθηκε και η δυνατότητα σε ένα νήμα να έχει τη δυνατότητα εκτέλεσης αυτόνομου κώδικα, ανεξάρτητα από τα υπόλοιπα του wrap (με τη προσθήκη address counter και register state) αν και αυτό έρχεται με το κόστος απόδοσης.

3.1.2 Αλγόριθμος K-means

Λόγω της δημοφιλίας του, ο K-means ήταν ένας αλγόριθμος που τράβηξε αρκετό ενδιαφέρον και ενέργεια σε μια απόπειρα να παραλληλοποιηθεί. Αυτό επιτεύχθηκε με την πάροδο των χρόνων με τη χρήση κατανεμημένων συστημάτων. Με την είσοδο των προγραμματίσιμων καρτών γραφικών, η ερευνητική κοινότητα προσπάθησε να μεταφέρει αυτές τις τεχνικές από τα κατανεμημένα συστήματα στις κάρτες γραφικών σε μια προσπάθεια να επωφεληθούν το χαμηλό κόστος με τις υψηλές βελτιώσεις που φαινόταν θεωρητικά ότι ήταν εφικτές. Από την αρχή όμως, κυρίως λόγω περιορισμών από το υλικό, συναντήθηκαν αρκετά προβλήματα. Οι Zechner και Granitzer στο [15] προσπάθησαν να εντοπίσουν αυτά τα προβλήματα που παρουσιάζονταν στις πιο πρώιμες υλοποιήσεις παράλληλων K-means για κάρτες γραφικών και σύντομα κατέληξαν στο ακόλουθα συμπέρασμα

1) οι μέχρι τότε υλοποιήσεις έπασχαν από αρχιτεκτονικούς περιορισμούς, όπως τον περιορισμό του πλήθους των διαστάσεων, του πλήθους των συστάδων και του πλήθους των instances λόγω μέγιστου μεγέθους textures. Η λύση που προτάθηκε από τους συγγραφείς για αυτό τον περιορισμό ήταν

η χρήση πιο ακριβών multi-pass προσεγγίσεων.

2) Λόγω της απουσίας των συνθηκών με τις οποίες έγιναν οι δοκιμές των υλοποιήσεων από κάποιες δημοσιεύσεις, είναι αδύνατο να γίνει απευθείας σύγκριση. Παρ' όλα αυτά, τα νούμερα που παρουσιάζονται στις διάφορες δημοσιεύσεις συμφωνούν μεταξύ τους και παρουσιάζουν μια εικόνα που έχει τις υλοποιήσεις σε κάρτες γραφικών να επιταχύνουν των K-means κατά μέσο όρο τρεις με τέσσερεις φορές συγκριτικά με τη σειριακή εκδοχή του.

3) Η απόδοση των υλοποιήσεων στην κάρτα γραφικών αυξάνεται όσο αυξάνονται οι διαστάσεις, το πλήθος των κεντρώων και ο αριθμός των instances σε ένα πρόβλημα

Βασιζόμενοι στα παραπάνω, και στη δουλειά που είχε παρουσιαστεί από τον Dhillon (κατανεμημένη υλοποίηση του K-means), παρουσίασαν μία υβριδική λύση η οποία αντιμετωπίζει τα προβλήματα του 1).

Εφόσον η κάρτα γραφικών είναι μία πολυεπεξεργαστική μονάδα με διαμοιραζόμενη μνήμη, έχουν γίνει κάποιες μικρές αλλαγές συγκριτικά με τον αλγόριθμο πρότυπο. Αρχικά, οι επεξεργαστές πλέον ονομάζονται νήματα (threads) και συνδέονται με ένα μοντέλο master-slave. Το κάθε νήμα χαρακτηρίζεται από ένα νούμερο μεταξύ του 0 και του $t-1$, όπου t είναι το πλήθος των διαθέσιμων νημάτων. Το νήμα 0 θεωρείται ως το master thread με όλα τα υπόλοιπα νήματα να είναι slaves. Τα νήματα, όπως είναι γνωστό από την αρχιτεκτονική των καρτών NVIDIA, μοιράζονται μνήμη ή οποία περιέχει το σύνολο των δεδομένων X, το σύνολο των κεντρώων C καθώς και τις συστάδες C_j. Επίσης, κάθε νήμα έχει μια επιπλέον τοπική μνήμη για διάφορα δεδομένα. Αξίζει να σημειωθεί ότι θεωρείται δεδομένη η ύπαρξη μηχανισμών κλειδώματος για ταυτόχρονη πρόσβαση στη μνήμη. Έχοντας λοιπόν ορίσει τα παραπάνω η λύση

των συγγραφέων πραγματοποιεί τα ακόλουθα βήματα. Η

Το master thread αρχικοποιεί τα κεντρώα όπως ακριβώς γίνεται και στη σειριακή εκδοχή του αλγορίθμου K-means. Στη συνέχεια, το X διαιρείται σε υποσύνολα $X_i, i = 0, \ldots, t$. Στην πραγματικότητα, η διαίρεση είναι ένας υπολογισμός offset και εμβέλειας (range) που πραγματοποιεί το κάθε νήμα έτσι ώστε να έχει τα x_i που του αντιστοιχούν για το κομμάτι της αντιστοίχησης των δεδομένων σε συστάδες (labelling). Όλα τα νήματα αντιστοιχούν τα δεδομένα τους σε συστάδες. Η συστάδα στην οποία ανήκει το κάθε δεδομένο (label), αποθηκεύεται σε μία συνιστώσα l_i ενός διανύσματος n διαστάσεων. Αυτό διευκολύνει και απλοποιεί αρκετά την διαδικασία της ενημέρωσης των συστάδων καθώς πλέον δεν χρειάζονται ταυτόχρονες εγγραφές στην ίδια διεύθυνση μνήμης. Μετά το τμήμα της ανάθεσης, τα νήματα συγχρονίζονται έτσι ώστε να είναι βέβαιο πως είναι διαθέσιμα όλα τα δεδομένα για τη διαδικασία ενημέρωσης των κεντρώων. Η διαδικασία της ενημέρωσης των κεντρώων, θα μπορούσε να πραγματοποιηθεί με τη λειτουργία reduction, παρ' όλα αυτά, οι συγγραφείς στην υλοποίηση που παρουσίασαν, για λόγους απλοποίησης επέλεξαν να γίνεται από το master thread σειριακά. Αντί να υπολογίζει επαναληπτικά όλα τα κεντρώα, το master thread "διασχίζει" επαναληπτικά όλες τις αναθέσεις υπολογίζοντας εν μέρη τα καινούργια κεντρώα. Ένα διάνυσμα $\boldsymbol{m}$ k διαστάσεων ενημερώνεται με κάθε επανάληψη όπου η κάθε συνιστώσα m_j περιέχει τον αριθμό των δεδομένων που έχουν ανατεθεί στη συστάδα C_j. Στη τελευταία επανάληψη κάθε κεντρώο c_j σταθμίζεται κατά $\frac{1}{m_j}$ παράγοντας τα τελικά κεντρώα. Η σύγκλιση καθορίζεται και αυτή με το master thread να ελέγχει αν έχουν υπάρξει αλλαγές στη τελευταία διαδικασία ανάθεσης. Όταν ο αλγόριθμος συγκλίνει, ο master thread στέλνει ένα σήμα στα slave threads με αποτέλεσμα αυτά να σταματήσουν και ο αλγόριθμος να τερματίσει.

Στη συγκεκριμένη περίπτωση, τον ρόλο του master thread τον αναλαμβάνει η κεντρική μονάδα επεξεργασίας η οποία προετοιμάζει τα δεδομένα και τα στέλνει στη κάρτα γραφικών. Με τη σειρά της, η κάρτα γραφικών είναι υπεύθυνη για το στάδιο της ανάθεσης των δεδομένων σε συστάδες και στη συνέχεια την αποστολή του αποτελέσματος στον επεξεργαστή για την ενημέρωση των κεντρώων.

Με την υλοποίηση που προτάθηκε, πραγματοποιήθηκαν πειράματα σε σετ δεδομένων μεγέθους 500, 5000, 50000 και 500000 με το καθένα από αυτά να έχει τρεις εκδοχές 2 διαστάσεων, 20 και 200. Η επιτάχυνση που παρατηρήθηκε ήταν του μεγέθους των 4 με 43 με τη αποδόσεις καλύτερες όσο το μεγάλωνε το σετ δεδομένων και ο αριθμός των διαστάσεων.

Οι Cuomo και λοιποί (et al.)[4], βασιζόμενοι στη δουλειά των Zechner και Granitzer[15] πρότειναν τρεις παράλληλες υλοποιήσεις του αλγόριθμου K-means με σκοπό τη μείωση του χρόνου εκτέλεσης. Οι υλοποιήσεις αυτές ακολουθούν και αυτές μία υβριδική προσέγγιση, στην οποία ο αλγόριθμος K-means υλοποιείται εν μέρη στον κεντρικό επεξεργαστή (host) και εν μέρη στη κάρτα γραφικών (device). Η μεταφορά των δεδομένων μεταξύ host και device γίνεται με αποδοτικό τρόπο συνενώνοντας τις προσβάσεις στη μνήμη με τις αιτήσεις ανάθεσης μνήμης για πίνακες (matrices). Η προτεινόμενη προσέγγιση κάνει κλήσεις στο device σε κάθε επανάληψη. Αυτό, απαιτεί τη ρύθμιση ορισμένων παραμέτρων και τον έλεγχο των διαστάσεων για τους πινάκες από δεδομένα. Κάποιες διαστάσεις αναγκάζουν την επεξεργασία των δεδομένων κομμάτι-κομμάτι. Στη συνέχεια, ο host υπολογίζει τα νέα κέντρα των συστάδων χρησιμοποιώντας την πληροφορία από τους πίνακες που παράχθηκαν από το device και ελέγχει αν ο αλγόριθμος έχει συγκλίνει. Οι συγγραφείς, παρατήρησαν ότι ο αλγόριθμος K-means χρειάζεται τρεις δομές δεδομένων για να αποθηκεύσει το σετ δεδομένων,

τα κεντρώα και τους δείκτες που υποδεικνύουν την συστάδα στην οποία ανήκει το κάθε δεδομένο. Με το σκοπό να απευθυνθούν στην παρατήρησή τους, οι συγγραφείς πρότειναν μια ”ελαφρύτερη” δομή δεδομένων ώστε να μειωθεί ο χρόνο μεταφοράς των δεδομένων. Στην πρώτη υλοποίηση, προτάθηκε η χρήση ενός πίνακα που θα περιέχει τα δεδομένα και θα ενημερώνεται σχετικά με την ”ιδιοκτησία” στις συστάδες στην κάρτα γραφικών, ενώ στη δεύτερη υλοποίηση, χρησιμοποιήθηκαν δύο διαφορετικές δομές για τον ίδιο σκοπό. Η τρίτη υλοποίηση, που θεωρείται και η πιο ”κομψή” λύση, υιοθετεί μια παράλληλη τεχνική για να υπολογίσει τη τετραγωνική Ευκλείδεια απόσταση. Η τελευταία υλοποίηση, επιτυγχάνει μια επιτάχυνση $88\times$ συγκριτικά με την εκδοχή που εκτελείται στην κεντρική μονάδα επεξεργασίας.

Οι Li, Zhao και λοιποί στη δημοσίευση τους [8] επικεντρώθηκαν στην μελέτη των διαστάσεων των δεδομένων. Πως δηλαδή μπορεί να παραλληλοποιηθεί ο αλγόριθμος K-means έτσι ώστε να επεξεργάζεται πιο αποδοτικά σετ δεδομένων με πολύ μεγάλο πλήθος δεδομένων ή μεγάλο αριθμό διαστάσεων στα δεδομένα τους. Η μελέτη αυτή ξεκίνησε με μία ανάλυση των αλγορίθμων GPUMiner, UV_k-Means, HP_k-Means τρεις αλγόριθμους (τους πιο αποδοτικούς την στιγμή συγγραφής της δημοσίευσης) που υλοποιούν τον K-means σε κάρτες γραφικών κάνοντας χρήση της γλώσσας CUDA. Γρήγορα παρατηρήθηκε ότι η κύρια διαφορά μεταξύ τους ήταν αυτή της χρήσης της μνήμης της κάρτας γραφικών, με τους αλγόριθμους που έκαναν την πιο αποδοτική χρήση, εκμεταλλευόμενοι την αρχιτεκτονική των καρτών γραφικών NVIDIA να έχουν μεγαλύτερα ποσοστά επιτάχυνσης συγκριτικά με τους υπόλοιπους. Μία επιπλέον παρατήρηση τους ήταν ότι η προσέγγιση που πρέπει να υιοθετήσει κάποιος αναλυτής ως προς την υλοποίηση ενός παράλληλου K-means θα πρέπει να παίρνει υπόψιν της τη διάσταση του σετ δεδομένων.

Λόγω των παραπάνω συμπερασμάτων, οι συγγραφείς υλοποίησαν δύο διαφορετικές προσεγγίσεις που εξαρτώνται από τον αριθμό των διαστάσεων του σετ δεδομένων. Η πρώτη προσέγγιση που υλοποιείται για χαμηλό αριθμό διαστάσεων, μοιάζει αρκετά με την μέθοδο που παρουσιάστηκε παραπάνω. Ανατίθεται ένα δεδομένο ανά νήμα το οποίο υπολογίζει την ελάχιστη απόσταση μεταξύ του δεδομένου που έχει και των κεντρώων των συστάδων. Η κύρια συνεισφορά αυτής της μεθόδου είναι η αποθήκευση των δεδομένων στους καταχωρητές των νημάτων για έτσι ώστε η ανάγνωση από την global μνήμη να γίνει μόνο μία φορά με αποτέλεσμα την σημαντική μείωση το latency της ανάγνωσης. Το κύριο πρόβλημα με αυτή τη προσέγγιση, όπως μπορεί να φανταστεί κανείς, είναι το μικρό μέγεθος των καταχωρητών. Λόγω του μεγέθους τους, περιορίζουν αρκετά τα σενάρια αξιοποίησης τους ανεξάρτητα από τα πλεονεκτήματα που προσφέρουν όσο έχει να κάνει με την ταχύτητα ανάγνωση. Αναγκαστικά λοιπόν, για δεδομένα με πολλές διαστάσεις, οι συγγραφείς πρότειναν τη δεύτερη προσέγγιση.

Στη περίπτωση που το σετ δεδομένων έχει μεγάλο αριθμό διαστάσεων, που συνεπάγεται μεγαλύτερο όγκο δεδομένων, οι συγγραφείς επέλεξαν την επόμενη καλύτερη λύση. Τα δεδομένα στη δεύτερη προσέγγιση που υιοθετήθηκε από τους συγγραφείς φορτώνονται στην local memory γεγονός που θυσιάζει ταχύτητα συγκριτικά με τους καταχωρητές αλλά είναι και πάλι καλύτερο από την global μνήμη. Παρ' όλα αυτά, ακόμα χρειάζεται να βρεθεί μια λύση για την αποδοτική πρόσβαση στα δεδομένα. Η λύση σε αυτό δόθηκε με την αναπαράσταση των δεδομένων σε μορφή πίνακα. Ένα σετ n δεδομένων d διαστάσεων ερμηνεύεται ως ένας πίνακας $data[n][d]$, τα κεντρώα k συστάδων ερμηνεύονται ως ένας πίνακας $centroid[d][k]$ και τέλος το αποτέλεσμα της συνάρτησης απόστασης ως ένας πίνακας $Result[n][k]$. Η λογική είναι απλή, όπως ακριβώς και με τον πολ-

λαπλασιασμό των πινάκων, έτσι ακριβώς και με τις παραπάνω αναπαραστάσεις των δεδομένων είναι δυνατός ο υπολογισμός των αθροισμάτων των τετραγώνων (στην περίπτωση της Ευκλείδειας απόσταση) μεταξύ κάθε δεδομένου και όλων των κεντρώων των συστάδων. Με τον υπολογισμό του $Result[n][k]$, το επόμενο βήμα είναι η εύρεση της μικρότερης απόστασης. Αυτό μπορεί να γίνει είτε αναθέτοντας σε ένα νήμα να βρει το κοντινότερο κεντρώο ενός δεδομένου από μία γραμμή του $Result$. Με τον δεύτερο περισσότερα από ένα νήματα μοιράζονται τη σειρά για να βρουν το κοντινότερο κεντρώο. Η πρώτη μέθοδος είναι πιο εύκολα υλοποιήσιμη ενώ η δεύτερη πιο αποδοτική για μεγάλα k.

Έχοντας πλέον τα αποτελέσματα της διαδικασίας ανάθεσης σε έναν πίνακα $index[n]$, το επόμενο βήμα είναι η ενημέρωση των κεντρώων η παραλληλοποίηση της οποίας είναι λίγο πιο περίπλοκη λόγω πιθανών conflicts κατά τη διάρκεια υπολογισμού. Η λύση που πρότειναν οι συγγραφείς ήταν μια διαδικασία ”διαίρει και βασίλευε” στην οποία ο πίνακας χωρίζεται σε ομάδες των n' στοιχείων όπου $n' = n/M$ με M να είναι ένα πολλαπλάσιο των νημάτων. Η υποδιαίρεση σταματάει όταν $n' < M$ και στη συνέχεια υπολογίζονται μερικά αθροίσματα μέχρι να ολοκληρωθεί η διαδικασία και να βρεθούν τα τελικά κεντρώα. Με την υποδιαίρεση των δεδομένων σε ομάδες μειώνεται σημαντικά το conflict στη πρόσβαση της μνήμης κατά τη διαδικασία του υπολογισμού.

Οι παραπάνω μέθοδοι, υποθέτουν ότι όλα τα δεδομένα μπορούν να χωρέσουν στην μνήμη της κάρτας γραφικών. Αν αυτό δεν είναι εφικτό, τότε μπορούν να χρησιμοποιηθούν τεχνικές ”divide-and-merge”. Με αυτές τις τεχνικές, ο αλγόριθμος χωρίζει το σετ δεδομένων σε M ομάδες όπου η κάθε ομάδα έχει n' δεδομένα τα οποία χωράνε στη μνήμη της κάρτας γραφικών. Η κάθε ομάδα αντιμετωπίζεται ανεξάρτητα σύμφωνα με τις παραπάνω διαδικασίες και στο τέλος της επανάληψης υπολογίζεται ένα μερικό άθροισμα (όχι ο μέσος όρος

των δεδομένων). Αφού εκτελεστεί η επανάληψη του αλγορίθμου για όλες τις ομάδες, τότε υπολογίζονται τα τελικά κεντρώα.

Στα πειράματα που πραγματοποιήθηκαν για σετ δεδομένων με μικρό αριθμό διαστάσεων, η αντίστοιχη εκδοχή του αλγόριθμου είναι τρεις με οχτώ φορές πιο γρήγορη από τον HP_k-Means, δέκα με είκοσι φορές γρηγορότερη από τον UV_k-Means και εκατό με τριακόσιες φορές γρηγορότερη από τον GPUMiner. Για δεδομένα με μεγάλο αριθμό διαστάσεων, η αντίστοιχη εκδοχή του αλγόριθμου είναι τέσσερεις με οχτώ φορές γρηγορότερη από τον UV_k-Means και δέκα με σαράντα φορές γρηγορότερη από τον GPUMiner. Ο HP_k-Means δεν παρουσιάζει αποτελέσματα για πειράματα με δεδομένα πολλών διαστάσεων οπότε δεν πραγματοποιήθηκε κάποια σύγκριση με τη δεύτερη εκδοχή του αλγόριθμου.

3.1.3 Αλγόριθμος PaStream

Οι Huang και λοιποί πρότειναν τον αλγόριθμο PaStream για τη συσταδοποίηση ροές δεδομένων. Αυτός ο αλγόριθμος, περνάει μία φορά από τα δεδομένα ”ανακαλύπτοντας” συστάδες με αυθαίρετα σχήματα και εντοπίζοντας ακραία σημεία. Ο αλγόριθμος PaStream υλοποιείται σύμφωνα με ένα πλαίσιο (framework) συσταδοποίησης γρήγορα μεταβαλλόμενων ροών δεδομένων που προτάθηκε από τους Aggrawal και λοιποί και ονόμασαν CluStream.

Ο PaStream χρειάζεται ένα βήμα αρχικοποίησης στο οποίο δημιουργεί μερικές μικρο-συστάδες χρησιμοποιώντας τον αλγόριθμο K-means. Στη συνέχεια, ακολουθούν δύο φάσεις. Στην on-line φάση, ο αλγόριθμος μαζεύει τα δεδομένα και τα εντάσσει σε μικρο-συστάδες. Στην off-line φάση, οι μικρο-συστάδες συνενώνονται σε μακρο-συστάδες (macro-clusters). Στην on-line φάση, για κάθε δεδομένο, πρέπει να υπολογιστούν τρεις περιπτώσεις. Στην πρώτη περίπτωση, μία ήδη υπάρχουσα μικρο-συστάδα απορροφά το δεδομένο. Στη δεύτερη πε-

ρίπτωση, το δεδομένο που εξετάζεται σχηματίζει μια νέα μικρο-συστάδα, ενώ στην τρίτη περίπτωση, οι δύο κοντινότερες μικρο-συστάδες ενώνονται σε μία. Αυτή η απόφαση εξαρτάται από μια παράμετρο που ονομάστηκε ”ρυθμός απορρόφησης”. Επιπλέον, οι συγγραφείς έχουν ορίσει έναν συντελεστή συγχώνευσης, ο οποίος καθορίζει πότε οι γειτονικές μικρο-συστάδες που βρίσκονται κοντά μεταξύ τους πρέπει να συγχωνευτούν έτσι ώστε να βελτιστοποιηθεί η χρήση χώρου αλλά και ο χρόνος που χρειάζεται για την εκτέλεση του αλγορίθμου. Στην off-line φάση, υπολογίζεται η Ευκλείδεια απόσταση μεταξύ όλων των κέντρων. Στη συνέχεια, δημιουργείται ο γράφος αποφάσεων (decision graph) ανάλογα την τοπική απόσταση και πυκνότητα για κάθε μικρο-συστάδα.

Αξίζει να σημειωθεί ότι ο αλγόριθμος εκτελείται σε ομάδες δεδομένων όχι μεμονωμένα δεδομένα όπως αυτά λαμβάνονται (εφόσον μιλάμε για ροές δεδομένων). Αυτό βοηθάει στη μείωση του χρόνου που απαιτείται για τη μεταφορά των δεδομένων από τη κεντρική μονάδα επεξεργασίας στη κάρτα γραφικών. Τα πειράματα που εκτελέστηκαν χρησιμοποιώντας τόσο πραγματικά όσο και πλασματικά σετ δεδομένων, δείχνουν ότι ο αλγόριθμος PaStream αποδίδει καλύτερα από την εκδοχή που τρέχει στη CPU όσο αφορά χρόνο εκτέλεσης και ποιότητα παραγόμενων αποτελεσμάτων.

3.1.4 Αλγόριθμος Ομαδοποίησης Εγγράφων

Οι Zang και λοιποί, πρότειναν μια παράλληλη εκδοχή ενός αλγορίθμου ομαδοποίησης εγγράφων βασισμένο από τη φύση. Η πρωτότυπη εκδοχή του ονομάζεται αλγόριθμος ομαδοποίησης εγγράφων βασισμένος σε σμήνη. Με σκοπό την μείωση του χρόνου εκτέλεσης και της πολυπλοκότητας, οι συγγραφείς, εκμεταλλεύτηκαν την υπολογιστική ισχύ των υπολογιστικών σμηνών (computer clusters) Beowulf τα οποία ήταν εξοπλισμένα με κάρτες γραφικών. Σε ένα τυπι-

κό μοντέλο σμήνους, η συμπεριφορά ενός μέλους, βασίζεται εξ ολοκλήρου από τα γειτονικά μέλη μέσα σε μια συγκεκριμένη απόσταση. Αυτή η συμπεριφορά, περιγράφεται από τρεις κανόνες: διαχωρισμός, ευθυγράμμιση και συνοχή. Για να εφαρμοστεί το μοντέλο αυτό στο πρόβλημα της ομαδοποίησης εγγράφων, το έγγραφο θεωρείται ως ένα μέλος και συμμετέχει στο σχηματισμό του σμήνους. Στη συνέχεια, οι τρεις κανόνες εφαρμόζονται για τα όμοια γειτονικά έγγραφα, ενώ μόνο ο κανόνας του διαχωρισμού εφαρμόζεται στα ανόμοια γειτονικά έγγραφα. Οι συγγραφείς, ανέπτυξαν ένα ειδικά διαμορφωμένο μοντέλο καρτών γραφικών για να υλοποιήσουν τη προτεινόμενη μέθοδο. Αποτελείται από μία κατανεμημένη διεπαφή αντικειμένων (distributed object interface) για να ενοποιήσει την διαχείριση μνήμης της CUDA και τις ρουτίνες μεταφοράς μηνυμάτων, ένα μηχανισμό για τη δημιουργία (spawn) ενός ευέλικτου αριθμού από host νήματα με σκοπό την παραλληλοποίηση (μπορεί να ξεπεράσει τον αριθμό καρτών γραφικών που είναι διαθέσιμες στο σύστημα) και τέλος μια διεπαφή που επιτρέπει σε εξειδικευμένους χρήστες τον έλεγχο του προγραμματισμού (scheduling) των νημάτων στα υπολογιστικά σμήνη (clusters).

Τα πειράματα που έγιναν πάνω σε αυτό το μοντέλο δείχνουν ότι έχει την ικανότητα χειρισμού για περισσότερα από ένα εκατομμύριο έγγραφα όταν αυτά επεξεργάζονται από ένα σμήνος καρτών γραφικών με 16 κόμβους. Επιπλέον, η προτεινόμενη μέθοδος μπορεί να φτάσει μέχρι και πενήντα (50) φορές επιτάχυνση συγκριτικά με την υλοποίηση στη κεντρική μονάδα επεξεργασίας.

3.1.5 Αλγόριθμος Async-EM

Ο αλγόριθμος Async-EM προτάθηκε από τους Altinigneli και λοιποί και είναι μια παράλληλη εκδοχή του αλγόριθμου Προσδοκίας-Μεγιστοποίησης (Expectation-Maximization, EM), βασισμένος σε κάρτες γραφικών. Η κύρια συνεισφορά

τους αφορά τον συγχρονισμό των πυρήνων και την οργάνωση της πρόσβασης στη μνήμης.

Οι συγγραφείς παρατήρησαν ότι οι πολλές ανανεώσεις των αντιπροσώπων των καθολικών συστάδων έχουν ως αποτέλεσμα την μη αποδοτική χρήση του εύρους της μνήμης (memory bandwidth) και ένα σημαντικό overhead στη συγχρονισμό. Συνεπώς, για την αποφυγή αυτών των προβλημάτων, πρότειναν οι αντιπρόσωποι των καθολικών συστάδων να ενημερώνονται μόνο όταν έχει αλλάξει ένα συγκεκριμένο πλήθος μελών. Αυτό επιτυγχάνεται μέσα από την ιδέα της ασύγχρονης ενημέρωσης του μοντέλου σε συνδυασμό με μία αποδοτική τεχνική που ονομάζεται μοντέλο ενοποίησης (model of consolidation).

Το μοντέλο ενοποίησης είναι υπεύθυνο για την συγχώνευση των διαφορετικών σετ αντιπροσώπων συστάδων όταν οι ενημερώσεις των τοπικών μοντέλων ανταλλαχθούν. Για να το πετύχει αυτό, εκμεταλλεύεται τα χαρακτηριστικά της ιεραρχίας μνήμης των μοντέρνων καρτών γραφικών.

Τα πειράματα που έγιναν στο μοντέλο αυτό χρησιμοποιώντας πραγματικά και συνθετικά σετ δεδομένων, δείχνουν ότι ο αλγόριθμος Async-EM αποδίδει καλύτερα από τους αλγόριθμους incremental-EM και batch-EM στους τομείς τις σύγκλισης, λάθη κατά τη διάρκεια συσταδοποίησης (modeling error) και τον χρόνο εκτέλεσης. Η σύγκριση υλοποιήσεων μεταξύ κάρτας γραφικών και κεντρικής επεξεργαστική μονάδας έχει επίσης δείξει καλύτερη επίδοση για την εκδοχή της κάρτας γραφικών.

3.1.6 Αλγόριθμος OPTICS

Το μοντέλο που προτάθηκε από τους Melo και λοιποί είναι μέλος των αλγορίθμων ομαδοποίησης που βασίζονται στην πυκνότητα. Εστιάζει κυρίως στην παραλληλοποίηση του αλγόριθμου OPTICS χρησιμοποιώντας την κάρτα γρα-

φικών και βασιζόμενο σε στρατηγική ευρετηρίασης δεδομένων. Ο αλγόριθμος OPTICS δημιουργεί μια επαυξημένη ταξινόμηση του σετ δεδομένων έτσι ώστε να αναπαριστάται η δομή της συσταδοποίησης βασισμένης στη πυκνότητα. Η προτεινόμενη παράλληλη εκδοχή του OPTICS αποτελείται από δύο φάσεις: την δημιουργία των γράφων και την εφαρμογή του OPTICS. Η αναπαράσταση των δεδομένων του μοντέλου που προτάθηκε γίνεται σε μορφή γράφου που ακολουθεί την δομή δεδομένων METIS, η οποία προτάθηκε από τους Karypis και Kumar. Αυτό το πετυχαίνει με τη χρήση τριών διανυσμάτων για την αποθήκευση των κόμβων, τους κόμβους στη λίστα γειτνίασης και την απόσταση. Η διαδικασία αποτελείται από τέσσερα βασικά βήματα, δηλαδή, τον υπολογισμό του βαθμού των κόμβων, τον υπολογισμό των δεικτών γειτνίασης, την δημιουργία και την ταξινόμηση της λίστας γειτνίασης. Αυτά τα βήματα σε συνδυασμό τη κατασκευή των δομών δεδομένων και την αποθήκευση τους στο τέλος της λίστας γειτνίασης των κόμβων, γίνονται όλα παράλληλα με τη χρήση της κάρτας γραφικών.

Τα πειράματα που πραγματοποιήθηκαν έδειξαν ότι η προτεινόμενη προσέγγιση, ρίχνει σημαντικά την πολυπλοκότητα του αλγορίθμου OPTICS με αποτέλεσμα να είναι γρηγορότερη από τη σειριακή εκδοχή του που τρέχει στη κεντρική μονάδα επεξεργασίας.

Μια άλλη πρόταση βελτίωσης του αλγόριθμου OPTICS έγινε από τους Deng και λοιποί. Η πρόταση τους αποτελεί μία νέα προσέγγιση στην ομαδοποίηση τροχιών και βασίζεται στον POPTICS (Parallel Optics). Για να προσαρμοστεί ο αλγόριθμος POPTICS στα δεδομένα των τροχιών, οι συγγραφείς εφάρμοσαν στον προτεινόμενο τους αλγόριθμο, χωροχρονική (spatio-temporal) απόσταση (space-temporal distance) για να μετρήσουν την ομοιότητα μεταξύ τροχιών και μία τεχνική ευρετηρίασης βασισμένη στα STR δέντρα. Αυτός ο αλγόριθμος

ονομάστηκε Tra-POPTICS και βασίζεται στην κοινή μνήμη και περιλαμβάνει τρία βασικά βήματα. Στο πρώτο βήμα, το κάθε νήμα της κάρτας γραφικών επεξεργάζεται ένα τοπικά ανεξάρτητο υποσύνολο του σετ δεδομένων. Βρίσκει τους ”γείτονες” για την κάθε τροχιά. Στη συνέχεια, υπολογίζει την απόσταση core για την κάθε τροχιά και μετά το τοπικό(local) Prim’s minimum spanning tree (MST). Το επόμενο βήμα δημιουργεί ένα καθολικό MST και τέλος το τελευταίο βήμα εξάγει τις συστάδες από το καθολικό MST. Ο αλγόριθμος αυτός, έχει πολλά κοινά με τον HDBSCAN που θα αναλυθεί στο κεφάλαιο των υλοποιήσεων.

3.1.7 Αλγόριθμος DBSCAN

Σύμφωνα με τους Böhm, Plant και λοιποί [2], ο αλγόριθμος OPTICS είναι στην ουσία η ιεραρχική επέκταση του αλγόριθμου DBSCAN, ενός από τους πιο διδαδεδομένους αλγορίθμους όσο αφορά την πρακτική του εφαρμογή. Στην ίδια δημοσίευση, οι συγγραφείς τονίζοντας τις δυσκολίες παραλληλοποίησης που φέρουν εγγενώς οι αλγόριθμοι συσταδοποίησης που βασίζονται στην πυκνότητα των δεδομένων, κυρίως λόγω των περίπλοκων σχέσεων μεταξύ δεδομένων και του άγνωστου αριθμού συστάδων, πρότειναν έναν αλγόριθμο με την ονομασία CUDA-DCLUST. Για τους παραπάνω λόγους, δεν θεώρησαν εφικτές τις παραδοσιακές τακτικές παραλληλοποίησης όπως για παράδειγμα την ανάθεση ενός δεδομένου ή μίας διάστασης στο κάθε νήμα. Η λύση που δόθηκε ήταν η δημιουργία μίας νέας δομή δεδομένων που ονόμασαν αλυσίδα (chain). Η αλυσίδα, είναι ένα σύνολο από δεδομένα που ανήκουν σε μια κοινή συστάδα. Αξίζει να σημειωθεί ότι μια συστάδα μπορεί να αποτελείται από πολλές αλυσίδες αλλά μία αλυσίδα μπορεί να ανήκει μόνο σε μία συστάδα. Αποτελεί στην ουσία μία ”χαλαρότερη” μορφή της κατά ορισμού συστάδας για αυτού του τύ-

που αλγόριθμους συσταδοποίησης. Κάθε αλυσίδα ανατίθεται τον δικό της αριθμό που την χαρακτηρίζει και αποτελεί τον ακρογωνιαίο λίθο γύρω από τον οποίο χτίστηκε ο CUDA-DCLUST. Η βασική ιδέα του αλγορίθμου είναι ότι εκμεταλλευόμενοι αυτή τη νέα δομή δεδομένων, αντί να εκτελείται μία επέκταση συστάδας(cluster expansion) ένα από τα βασικά βήματα των αλγορίθμων που βασίζονται στην πυκνότητα τη φορά, οι συγγραφείς εκτέλεσαν την διαδικασία της επέκτασης των συστάδων παράλληλα για διαφορετικά σημεία αφετηρίας. Προφανώς, ο CUDA-DCLUST θα πρέπει να εντοπίσει και να καταγράψει την κάθε "σύγκρουση" (collision) μεταξύ αλυσίδων. Με τον όρο "σύγκρουση" εκφράζεται το γεγονός ότι δύο αλυσίδες συμμετέχουν στην ίδια συστάδα.

Ο εντοπισμός και η καταγραφή των "συγκρούσεων" γίνεται μέσω ενός πίνακα συγκρούσεων (collision matrix). Ένας πίνακας συγκρούσεων C είναι ένα άνω τριγωνικός πίνακας μεγέθους $(p \times p)$ όπου p είναι το πλήθος των αλυσίδων που επεκτείνονται ταυτόχρονα. Αν εντοπιστεί μια "σύγκρουση" μεταξύ των αλυσίδων i και j, τότε τίθεται $C_{i,j} := \boldsymbol{true}$ αν $i < j$ διαφορετικά, τίθεται $C_{j,i} := \boldsymbol{true}$. Εφόσον ο πίνακας C είναι σχετικά μικρός, ο CUDA-DCLUST τρέχει στο τέλος του ένα σειριακό αλγόριθμο για να βρεθούν οι συστάδες. Τέλος, ένα cluster-ID ανατίθεται στην κάθε αλυσίδα που ανήκει στην ίδια συστάδα το οποίο στη συνέχει διανέμεται στα δεδομένα, παράλληλα.

Πέρα από την έννοια των αλυσίδων, μια ιδέα που σε γενικές γραμμές υιοθετήθηκε και από μεταγενέστερες υλοποιήσεις του DBSCAN, οι Böhm και λοιποί πρότειναν μια επιπλέον ιδέα για την βελτίωση της απόδοσης των αλγορίθμων συσταδοποίησης βασισμένους στη πυκνότητα. Όταν ένα δεδομένο S (seed) εξετάζεται για το αν είναι βασικό σημείο ή σημείο πυρήνα και όταν πρέπει να σημειωθούν οι γείτονες του ως υποψήφιοι σημείων πυρήνα, τότε χρειάζεται να καθοριστούν όλα τα δεδομένα στη γειτονιά του $S, N_\epsilon(S)$. Επομένως, καλείται

ένας αριθμός νημάτων για να υπολογίσουν τα παραπάνω ταυτόχρονα. Ο κάθε πιθανός γείτονας του δεδομένου που εξετάζεται, που πρέπει με την σειρά τους να εξεταστούν, επεξεργάζεται από διαφορετικό block.

Η διαδικασία επέκτασης της συστάδας είναι η κύρια διαδικασία του αλγόριθμου CUDA-DCLUST. Για την εκτέλεση της διαδικασίας αυτή, δημιουργούνται παραπάνω από ένα νήματα. Η εκτέλεση ξεκινά με ένα δεδομένο P για το οποίο πρέπει να καθοριστεί αν είναι σημείο πυρήνα ή όχι. Οι κύριες λειτουργίες που θα εκτελεστούν είναι ο καθορισμός των σημείων-γειτόνων του P, ο καθορισμός του αν είναι σημείο πυρήνα ή όχι, η σημείωση των σημείων-γειτόνων ως μέλη της αλυσίδας και η καταγραφή πιθανών "συγκρούσεων" στον πίνακα C. Αρχικά, οι συντεταγμένες του σημείου P διαβάζονται από μία ομάδα νημάτων, τα οποία καθορίζουν το minimum bounding rectangle $MBR_\epsilon(P)$ της γειτονιάς του $P, N_\epsilon(P)$ επιτρέποντας με αυτό τον τρόπο d-fold παραλληλοποίηση. Στη συνέχεια, αφού τα νήματα συγχρονιστούν, επεξεργάζονται όλα δεδομένα Q που είναι υποψήφιοι γείτονες του P. Αυτό γίνεται από όλα τα νήματα της ομάδας, ως προς μεγιστοποίηση της παραλληλοποίησης. Οι υποψήφιοι γείτονες επιστρέφονται με batches όπου το μέγεθος του batch είναι το πλήθος των νημάτων στην ομάδα. Η περαιτέρω επεξεργασία των Q εξαρτάται από το αν το σημείο P είναι βασικό σημείο ή όχι καθώς οι συστάδες επεκτείνονται μόνο από βασικά σημεία. Παρ' όλα αυτά, καθώς η διαδικασία γίνεται παράλληλα, δεν είναι πάντα δυνατόν να είναι γνωστό το status του P. Ανεξαρτήτως, η απόσταση δ μεταξύ Q και P, πρέπει να βρεθεί. Αν το δ απόσταση είναι μεγαλύτερη από το ϵ τότε δεν χρειάζονται επιπλέον ενέργειες. Διαφορετικά, αν το P έχει καθοριστεί ως σημείο πυρήνα τότε αμέσως το Q σημειώνεται ως μέλος της αλυσίδας. Αν από την άλλη, δεν είναι ακόμα γνωστό το status του P, τότε αυξάνεται ο αριθμός γειτόνων του και το Q μπαίνει σε μία προσωρινή λίστα $neighborBuer$

μεγέθους όσο και το MinPts. Όποτε ένα δεδομένο/σημείο σημειώνεται ως μέλος της αλυσίδας, πρέπει να ελεγχτεί αν είναι ήδη μέλος της ίδιας ή κάποιας άλλης αλυσίδας. Στην πρώτη περίπτωση, δεν χρειάζεται να γίνει τίποτα. Στη δεύτερη περίπτωση, υπάρχει μια ”σύγκρουση” αλυσίδων. Αυτό σημαίνει ότι θα πρέπει να σημειωθούν στον C τα ids των αλυσίδων που στην ουσία ανήκουν στην ίδια συστάδα.

Για την βελτίωση της απόδοσης του αλγορίθμου CUDA-DCLUST προτάθηκε και ένα multidimensional index structure το οποίο υποστηρίζει την αναζήτηση για τα σημεία Q της ϵ-γειτονιάς του σημείου $P, N_\epsilon(P)$. Ο index αυτός είναι σχεδιασμένος για να διαβάζεται παράλληλα. Για αυτό τον λόγο, αλλά και για το επιπλέον κριτήριο του μικρού μεγέθους, προτάθηκε ως ένας σταθερός αριθμός directory επιπέδων όπου το κάθε directory είναι μία διάσταση. Ο αλγόριθμος CUDA-DCLUST που κάνει χρήση αυτής της δομής ονομάστηκε CUDA-DCLUST*.

Τα πειράματα που πραγματοποιήθηκαν, έδειξαν ότι ο αλγόριθμος CUDA-DClust συγκριτικά με τον αλγόριθμος DBSCAN τρέχει 15 φορές πιο γρήγορα, ενώ ο CUDA-Dclust* συγκριτικά με τον αλγόριθμο DBSCAN που έχει index support επίσης παρουσιάζει επιτάχυνση κοντά στις 15 φορές.

Οι Loh και Yu αναγνώρισαν το γεγονός ότι αν και ο CUDA-DClust επιτυγχάνει τον στόχο του, να επιταχύνει τον αλγόριθμος DBSCAN, θα μπορούσε να είναι ακόμα πιο γρήγορος αν ήταν πιο αποδοτικός στον υπολογισμό των αποστάσεων για την εύρεση της γειτονιάς[10]. Θεώρησαν ότι είναι σπατάλη υπολογιστικής ισχύος ο υπολογισμός της απόστασης για όλα τα δεδομένα. Για αυτό το λόγο, πρότειναν τον αλγόριθμο CudaSCAN μια παράλληλη εκδοχή του DBSCAN. Η κύρια ιδέα είναι σχετικά απλή. Ο χώρος των δεδομένων χωρίζεται σε μικρότερα κομμάτια στα οποία εκτελείται η διαδικασία της συσταδοποίησης

και τέλος τα αποτελέσματα συγχωνεύονται. Από τη στιγμή που η συσταδοποίηση γίνεται για υποσύνολα του σετ δεδομένων, δεν είναι πλέον απαραίτητος ο υπολογισμός των αποστάσεων για όλα τα σημεία, εξοικονομώντας έτσι πόρους. Επιγραμματικά, ο αλγόριθμος έχει τρία στάδια: το στάδιο της υποδιαίρεσης, το στάδιο της υπο-συσταδοποίησης και το στάδιο της συνένωσης των αποτελεσμάτων.

Στο πρώτο στάδιο, αυτό της υποδιαίρεσης, ο αλγόριθμος CudaSCAN χωρίζει όλα τα δεδομένα d διαστάσεων σε υπο-περιοχές. Οι συνθήκες που πρέπει να τηρούνται από τις υπο-περιοχές είναι: (α) το μέγεθος κάθε υπο-περιοχής (αριθμός στοιχείων $\times$ μέγεθος δεδομένων) δεν πρέπει να υπερβαίνει το προκαθορισμένο μέγεθος S και (β) κατά την υποδιαίρεση, τα στοιχεία θα πρέπει να είναι όσο το δυνατόν ισομοιρασμένα έτσι ώστε η κάθε υπο-περιοχή να έχει παρόμοιο αριθμό στοιχείων με τις υπόλοιπες. Ο αλγόριθμος υποδιαίρεσης που προτάθηκε, χωρίζει την περιοχή R_D σε r υπο-περιοχές $R_i(0 \leq i < r)$, όπου η περιοχή ονομάζεται grid cell ή g-cell. $R_i \cap R_j = \varnothing$ για οποιαδήποτε δύο διαφορετικά g-cells R_i και $R_j(i \neq j)$ και $\cup_i R_i = R_D$ για όλα τα g-cells $R_i(0 \leq i < r)$. Μια περιοχή E_i $(0 \leq i < r)$, που είναι η επέκταση ενός g-cell κατά ϵ σε κάθε διάσταση, ονομάζεται expanded cell ή ϵ-cell. Αξίζει να σημειωθεί ότι ένα ϵ-cell πάντα συμπεριλαμβάνει το αντίστοιχο του g-cell και ότι τα ϵ-cells μπορούν να επικαλύπτονται ενώ τα g-cells οχι. Η περιοχή σε ένα ϵ-cell, χωρίς το g-cell $(E_i - R_i)$, ονομάζεται ϵ-region.

Αρχικά, ο αλγόριθμος χωρίζει το χώρο των δεδομένων σε υπο-περιοχές σταθερού μεγέθους, με κάθε υπο-περιοχή να είναι ένα αρχικό g-cell. Τα αρχικά g-cells είναι πανομοιότυπα σε διαστάσεις. Στη συνέχεια, υπολογίζεται το αρχικό ϵ-cell για κάθε g-cell. Αν το μέγεθος του ϵ-cell (αριθμός στοιχείων στο ϵ-cell $\times$ μέγεθος δεδομένων) ξεπερνά το S, τότε το αντίστοιχο g-cell πρέπει να υποδι-

πλασιαστεί. Πιο συγκεκριμένα, το μέγεθος δίνεται από τον τύπο $n*(d*f)$ όπου n ο αριθμός των δεδομένων στο ϵ-cell, d ο αριθμός των διαστάσεων του δεδομένου και f είναι το μέγεθος μιας συνιστώσας σε byte. Εφόσον στη CudaSCAN όλα τα δεδομένα είναι τύπου float, το f έχει σταθερό μέγεθος 4. Μετά την υποδιαίρεση, υπολογίζονται τα νέα ϵ-cells και η διαδικασία επαναλαμβάνεται μέχρει το μέγεθος να είναι μικρότερο του S ή $n = \lfloor \frac{S}{d*f} \rfloor$ στοιχεία. Η καταμέτρηση των στοιχείων γίνεται παράλληλα.

Στο στάδιο της υπο-συσταδοποίησης, ο αλγόριθμος CudaSCAN αναθέτει μία υπο-περιοχή ανά thread block και κάθε block εκτελεί την υπο-συσταδοποίηση ανεξάρτητα. Τα στοιχεία μέσα σε μία υπο-περιοχή επίσης επεξεργάζονται παράλληλα σε μία διαδικασία που είναι η παράλληλη εκδοχή του DBSCAN. Με άλλα λόγια, από μακροσκοπική προοπτική, τα blocks τρέχουν παράλληλα και από μικροσκοπική προοπτική, τα στοιχεία επεξεργάζονται παράλληλα μέσα σε ένα block. Αν ο αριθμός των υπο-περιοχών είναι μεγαλύτερο από τα διαθέσιμα blocks της κάρτας γραφικών, τότε οι υπο-περιοχές θα επεξεργαστούν με στυλ round-robin.

Αξίζει να σημειωθεί ότι τα στοιχεία και/ή τα ενδιάμεσα αποτελέσματα της διαδικασίας της υπο-συσταδοποίησης για κάθε υπο-περιοχή, επειδή χρειάζονται μόνο το block που εκτελεί τη λειτουργία αυτή, αποθηκεύονται στη κοινή μνήμη της κάρτας γραφικών. Σε αντίθεση, εφόσον τα ενδιάμεσα αποτελέσματα ήταν απαραίτητα για όλα τα thread blocks, ο CUDA-DClust τα αποθήκευε στη κύρια μνήμη με αποτέλεσμα αυξημένο χρόνο προσπέλασης τους.

Στο τελευταίο στάδιο, αυτό της συνένωσης των αποτελεσμάτων, ο αλγόριθμος CudaSCAN συνενώνει τα αποτελέσματα των υπο-συσταδοποιήσεων με τέτοιο τρόπο ώστε το αποτέλεσμα να είναι ίδιο με το αν ο αλγόριθμος DBSCAN έτρεχε σε όλο το σετ δεδομένων. Όπως αναφέρθηκε παραπάνω, τα στοιχεία

μέσα σε ϵ-region ενός ϵ-cell, μπορεί να ανήκουν σε παραπάνω από μία περιοχές (λόγω υπερκάλυψης) με αποτέλεσμα να ανήκουν σε διαφορετικές συστάδες σε διαφορετικές υπο-περιοχές. Στο στάδιο της συγχώνευσης, οι συστάδες που επικαλύπτουν την ϵ-region, θα πρέπει να ελεγχθούν για συνένωση. Για να είναι εφικτή η συνένωση, οι συγγραφείς αποφάσισαν ότι χρειάζονται τρεις πίνακες:

- Ένας πίνακας για την καταγραφή των καθολικών ID των συστάδων. Ο πίνακας αυτό είναι N θέσεων και περιέχει τον αριθμό της συστάδας του κάθε στοιχείου του συνόλου των δεδομένων.
- Ένα δισδιάστατο πίνακα για τη καταγραφή των "συγκρούσεων", αν δηλαδή χρειάζεται η συνένωση δύο συστάδων.
- Ένα πίνακα για τη καταγραφή της αντιπροσώπευσης των τοπικών συστάδων. Με λίγα λόγια το ποιο καθολικό ID αντιστοιχεί στο ID των τοπικών συστάδων.

Κατά τη διαδικασία της συγχώνευσης, πρώτα αξιολογείται ποιες συστάδες πρέπει να συγχωνευτούν. Αν ένα ζευγάρι τοπικών συστάδων C_i και C_j βρίσκονται σε διαφορετικές υπο-περιοχές που πρέπει να συγχωνευτούν, τότε αυτό καταγράφεται στον αντίστοιχο πίνακα που περιγράφηκε. Όταν πλέον είναι γνωστό ποιες συστάδες συγχωνεύονται, σημειώνεται στον τρίτο πίνακα το νέο καθολικό ID συστάδας. Τέλος η πληροφορία του τρίτου πίνακα μεταφέρεται στον πίνακα που περιέχει το καθολικό ID των συστάδων και ο αλγόριθμος τερματίζει.

Στα πειράματα που εκτελέστηκαν, ο αλγόριθμος CudaSCAN συγκρίθηκε με τον αλγόριθμο CUDA-DClust. Με μεταβλητό το πλήθος του συνόλου δεδομένων, ο CudaSCAN ήταν γρηγορότερος με ratio από 12,6 μέχρι 24,3 με την επιτάχυνση να μειώνεται για μεγαλύτερο πλήθος δεδομένων. Για μεταβλητό αριθμό διαστάσεων η εικόνα ήταν παρόμοια με τον CudaSCAN γρηγορότερο 17,8

φορές. Για μεταβλητό αριθμό S μέγιστου μεγέθους το ίδιο, με τον CudaSCAN γρηγορότερο 15,9 φορές. Για μεταβλητό αριθμό $MinPts$ και ϵ 16.2 και 163,6 με 15,9 αντίστοιχα. Τα αποτελέσματα αυτά δικαιολογούνται λόγω του γεγονότος ότι η απόδοση του CudaSCAN εξαρτάται σε μεγάλο βαθμό από τον αριθμό των υπο-περιοχών που δημιουργεί για αυτό και στα αποτελέσματα που παρουσιάζουν οι συγγραφείς είναι εμφανής η μείωση της επιτάχυνσης όσο τα μεγέθη που επηρεάζουν τον αριθμό υπο-περιοχών αυξάνονται, αν και παραμένει σημαντικά μεγάλη.

Ένας ακόμα παράλληλος αλγόριθμος συσταδοποίησης που βασίζεται στη πυκνότητα προτάθηκε από τους Andrade και λοιποί[1]. Είναι μια εκδοχή του DBSCAN που βασίζεται σε κάρτες γραφικών με χρήση απλών τεχνικών ευρετηρίασης των δεδομένων που βασίζονται σε γράφους και λέγεται G-DBSCAN. Η προτεινόμενη προσέγγιση αποτελείται από δύο βασικά βήματα, την δημιουργία του γράφου και τον εντοπισμό των συστάδων χρησιμοποιώντας τον αλγόριθμο αναζήτηση κατά πλάτος. Το πρώτο βήμα έχει ως στόχο την κατασκευή γράφου για την αναπαράσταση των δεδομένων. Πράγματι, κάθε δεδομένο αναπαριστάται ως κόμβος στο γράφο και όταν το μέτρο ομοιότητας μεταξύ δύο αντικειμένων είναι μικρότερο από κάποιο κατώφλι, το οποίο δίνεται ως παράμετρος εισόδου, μια ακμή σχηματίζεται ανάμεσα τους. Συνεπώς, το πρώτο βήμα υπολογίζει τον βαθμό των κόμβων, τον υπολογισμό δεικτών της λίστας γειτνίασης και την συναρμολόγηση των λιστών γειτνίασης. Το δεύτερο βήμα, έχει ως σκοπό την εύρεση των συστάδων διασχίζοντας τον γράφο που δημιουργήθηκε στο προηγούμενο βήμα χρησιμοποιώντας τον αλγόριθμο αναζήτηση ”αναζήτηση κατά πλάτος”. Τα πειράματα που πραγματοποιήθηκαν, έδειξαν ότι ο αλγόριθμος G-DBSCAN είναι εκατό φορές πιο γρήγορος από την σειριακό υλοποίηση του αλγόριθμου DBSCAN.

3.1.8 Αλγόριθμος MCL

Ο CUDA-MCL είναι ένα αλγόριθμος που προτάθηκε για εφαρμογή σε δίκτυα αλληλεπίδρασης πρωτεϊνών στο Bustamam και λοιποί και είναι μια παράλληλη εκδοχή του αλγόριθμου συσταδοποίησης Markov (Markov clustering algorithm, MCL). Ο αλγόριθμος MCL ”ανακαλύπτει” συστάδες σε γράφους αξιοποιώντας την τεχνική των τυχαίων περιπάτων. Ο προτεινόμενος αλγόριθμος είναι βασισμένος σε προγραμματισμό καρτών γραφικών χρησιμοποιώντας τη γλώσσα CUDA. Ο MCL αλγόριθμος είναι βασισμένος σε δύο αλγεβρικές πράξεις πάνω στους πίνακες Markov, την επέκταση και την εμφύσηση (inflation). Συνεπώς, η επίδοση του αλγορίθμου βασίζεται στο μέγεθος του πίνακα Markov.

Η κύρια συνεισφορά των συγγραφέων συμπεριλαμβάνει την βελτίωση της επίδοσης του αρχικού MCL κάνοντας χρήση παράλληλων διαδικασιών για την εκτέλεση των πράξεων της επέκτασης και της εμφύσησης. Επιπλέον, βελτιστοποιήθηκε η χρήση του αποθηκευτικού χώρου με τη αξιοποίηση αραιών πινάκων. Ο προτεινόμενος αλγόριθμος κάνει χρήση τριών παράλληλων CUDA πυρήνων (kernels). Έναν πυρήνα για να υπολογιστούν παράλληλα οι διαδικασίες επέκτασης του MCL, έναν δεύτερο πυρήνα για να υπολογιστούν παράλληλα οι διαδικασίες εμφύσησης του MCL και ένας τελευταίος πυρήνας για να υπολογίσει παράλληλα τη τοπικό και καθολικό χάος.

Συγκρίνοντας τον αρχικό αλγόριθμο MCL, που είναι βασισμένος στη κεντρική μονάδα επεξεργασίας, με την προτεινόμενη εκδοχή του, αποφάνθηκε ότι το δεύτερο είναι γρηγορότερο σε μεγαλύτερα σετ δεδομένων.

3.1.9 Αλγόριθμος CAST

Ο αλγόριθμος Calculation-On-Demand CAST με χρήση κάρτας γραφικών (COD-CAST-GPU) είναι ο παράλληλος σχεδιασμός του Clustering Affinity Search

Technique (CAST). Όπως αναφέρεται στην ονομασία του, βασίζεται στη πλατφόρμα της κάρτας γραφικών και στη μεμονωμένη μνήμη των καρτών γραφικών. Ο αλγόριθμος CAST χρειάζεται ως είσοδο τον πίνακα ομοιότητας και το κατώφλι για το affinity. Συνεπώς, το μέγεθος του πίνακα είναι ένα κρίσιμο σημείο το οποίο δημιουργεί προβλήματα αποθήκευσης. Η ιδέα πίσω από τον προτεινόμενο αλγόριθμο είναι να υπολογιστεί η ομοιότητα των κόμβων μόνο όταν αυτή χρειάζεται, αποφεύγοντας με αυτόν το τρόπο τον υπολογισμό του πίνακα ομοιότητας. Επιπλέον, για να επιταχυνθεί ο αλγόριθμος CAST και να υπάρξει κέρδος σχετικά με την επίδοση, οι συγγραφείς εκμεταλλεύθηκαν τις δυνατότητες των καρτών γραφικών. Ο προτεινόμενος αλγόριθμος ξεκινάει επιλέγοντας έναν τυχαίο κόμβο ως μία νέα συστάδα. Στη συνέχεια, χρησιμοποιεί δύο πράξεις: ADD και REMOVE για να σχηματίσει τη νέα συστάδα. Μετά από κάθε αλλαγή, οι τιμές affinity ενημερώνονται παράλληλα κάνοντας χρήση της κάρτας γραφικών. Η τεχνική που χρησιμοποιήθηκε από τους συγγραφείς έδειξε εντυπωσιακές βελτιώσεις σχετικά με την απόδοση συγκριτικά με τον αρχικό αλγόριθμο.

Κεφάλαιο 4

Υλοποιήσεις Αλγορίθμων

Έχοντας παρουσιάσει αλγορίθμους συσταδοποίησης τόσο από σειριακή όσο και παράλληλη οπτική, στο παρόν κεφάλαιο θα παρουσιαστούν κάποιες από τις παράλληλες εκδοχές που περιγράφηκαν δίνοντας περισσότερο έμφαση στην πρακτική υλοποίηση τους στο περιβάλλον CUDA. Οι αλγόριθμοι που επιλέχθηκαν προς παρουσίαση ήταν ο αλγόριθμος K-means, HDBSCAN (Hierarchical DBSCAN) και ο αλγόριθμος GPIC (GPU Power Iteration Clustering) που ανήκει στην οικογένεια των αλγορίθμων που βασίζονται σε πυρήνα (kernel based).

4.1 HDBSCAN

Ο αλγόριθμος HDBSCAN είναι μια επέκταση του αλγόριθμου DBSCAN που παρουσιάστηκε παραπάνω και επιτρέπει, με τη χρήση ιεραρχικής συσταδοποίησης την εύρεση συστάδων που αποτελούνται από δεδομένα διαφορετικών πυκνοτήτων. Καθώς ο αλγόριθμος DBSCAN έχει την ικανότητα εύρεσης συστάδων μόνο μίας πυκνότητας, ο HDBSCAN λύνει είναι μια ευπρόσδεκτη προσθήκη στις παραλλαγές του DBSCAN [12].

Ο HDBSCAN προτάθηκε από τους Campello, Moulavi, and Sander με την γενική ιδέα να είναι η χρήση των αλγορίθμων που βασίζονται στη πυκνότητα για την εξαγωγή ιεραρχημένων συστάδων από ένα σετ δεδομένων με τη δυνα-

τότητα μετατροπής της ιεραρχίας σε ”επίπεδη” (”flat”) συσταδοποίηση [3]. Η ακολουθία βημάτων που το επιτυγχάνει αυτό, μπορεί να γενικευτεί στα ακόλουθα βήματα:

1. Μετασχηματισμός του χώρου ανάλογα με τη πυκνότητα
2. Δημιουργία του ελάχιστου γεννητικού δέντρου (minimum spanning tree) για τον βεβαρημένο γράφο αποστάσεων.
3. Δημιουργία ιεραρχίας συστάδων από συνδεδεμένες συνιστώσες.
4. Συμπύκνωση της ιεραρχίας συστάδων βάσει το ελάχιστο μέγεθος συστάδας.
5. Εξαγωγή σταθερών (stable) συστάδων από το συμπυκνωμένο δέντρο.

Βάση τα παραπάνω βήματα, δεν θα ήταν παράλογο αν ο HDBSCAN ερμηνευόταν ως ένας αλγόριθμος που εφαρμόζει αρχικά DBSCAN σε ένα σύνολο δεδομένων για να βρει την γενικές κατανομή των συστάδων και στη συνέχεια ιεραρχική συσταδοποίηση βασισμένη στη πυκνότητα για να βρει την ιδανική κατανομή των δεδομένων.

4.1.1 Μετασχηματισμός χώρου

Το πρώτο βήμα του αλγορίθμου HDBSCAN είναι ο μετασχηματισμός του χώρου ανάλογα με τη πυκνότητα ή αραιότητα του σετ δεδομένων. Αυτό γίνεται γιατί όπως είναι γνωστό, σε ένα σύνολο δεδομένων δεν μπορούν να αποφευχθούν ο θόρυβος, τα ακρότατα καθώς και τα αλλοιωμένα δεδομένα. Από τη στιγμή λοιπόν που η ύπαρξη τους είναι γνωστή και ο αλγόριθμος θα χρειαστεί να δημιουργήσει μία ιεραρχία, διαδικασία η οποία είναι ευάλωτη στον θόρυβο, τα ακρότατα και τα αλλοιωμένα δεδομένα, είναι επόμενο η προσπάθεια αφαίρεσης

τους. Εδώ έρχονται οι αλγόριθμοι που βασίζονται στη πυκνότητα, που είναι αποδεδειγμένα ανθεκτική σε τέτοιες παρεμβολές με τη χρήση. Αλγόριθμοι όπως ο DBSCAN φιλτράρουν τα αλλοιωμένα ή ανεπιθύμητα δεδομένα κάνοντας χρήση του density reachability (και βασιζόμενοι σε αυτό το denisty connectedness) το οποίο όπως έχει παρουσιαστεί και στο Κεφάλαιο 2, εξαρτάται από το πόσα δεδομένα (MinPts) βρίσκονται σε μία δεδομένη απόσταση (ϵ) από το σημείο που εξετάζεται. Ο συνδυασμός τους όμως δεσμεύει τον HDBSCAN σε μία συγκεκριμένη πυκνότητα για την εύρεση συστάδων, κάτι που οι συγγραφείς θέλουν να αποφύγουν. Επομένως, μετέτρεψαν τα κριτήρια θορύβου/μέλους ενός δεδομένου έτσι ώστε να εξαρτώνται μόνο από τη μεταβλητή MinPts.

Ως προ επεξεργασία των δεδομένων, σκοπός του μετασχηματισμού χώρου δεν είναι η εύρεση των συστάδων βάσει πυκνότητας αλλά ο διαχωρισμός των ανεπιθύμητων σημείων από τις υποψήφιες συστάδες. Αυτό μπορεί να μεταφραστεί στην απομάκρυνση των δεδομένων χαμηλής πυκνότητας από τα δεδομένα υψηλής πυκνότητας. Για την επίτευξη του παραπάνω χρειάζεται η δημιουργία ενός νέου μέτρου απόστασης. Αυτό το μέτρο είναι το mutual reachability distance και ορίζεται ως εξής:

$$d_{mreach-k}(\alpha, \beta) = \text{max}\{core_k(\alpha), core_k(\beta), d(\alpha, \beta)\}$$

όπου στη παραπάνω εξίσωση το $d(\alpha, \beta)$ είναι το αρχικό μέτρο απόστασης μεταξύ α και β (συνήθως η Ευκλείδεια απόσταση) και το $core_k(\alpha)$ είναι η εκτίμηση της πυκνότητας του δεδομένου/σημείου α. Η απόσταση $core_k(\alpha)$ εκφράζει την απόσταση του κ-αστού δεδομένου από το α με το κ να ισούται με το MinPts. Με τη χρήση αυτού του μέτρου, τα πυκνά σημεία (με χαμηλό μέτρο απόστασης $core_k$) παραμένουν στην ίδια απόσταση μεταξύ τους αλλά τα πιο αραιά σημεία απομακρύνονται σε μία απόσταση που ισούται με το $core_k$ από τα υπόλοιπα δεδομένα. Αξίζει να σημειωθεί ότι όσο μεγαλώνει το κ, δηλαδή το

MinPts, τόσο περισσότερα δεδομένα θεωρούνται ανεπιθύμητα.

4.1.2 Δημιουργία Ελάχιστου Γεννητικού Δέντρου

Από τη στιγμή που πλέον υπάρχει ένα μέτρο απόστασης, ήρθε η ώρα να ξεχωρίσουν οι συστάδες μέσα στο χώρο το δεδομένων. Παρ' όλα αυτά, η πυκνότητα των δεδομένων είναι σχετική με διαφορετικές συστάδες να έχουν διαφορετική πυκνότητα, με αποτέλεσμα να μην είναι τόσο απλός ο διαχωρισμός τους από το θόρυβο αλλά και μεταξύ τους. Προς αυτό το στόχο, οι συγγραφεί νόησαν τα δεδομένα ως ένα βεβαρημένο γράφο όπου οι κορυφές είναι τα δεδομένα στο χώρο των δεδομένων και οι ακμές να είναι οι αποστάσεις τους σύμφωνα με το mutual reachability distance. Επιπλέον νοείται και ένα κατώφλι το οποίο ξεκινάει με υψηλή τιμή και μειώνεται σταθερά. Οι ακμές που έχουν βάρος μεγαλύτερο από το κατώφλι διαγράφονται και έτσι ο γράφος ξεκινάει να "αποσυνδέεται" σε συνιστώσες. Με αυτό το τρόπο δημιουργείται μια ιεραρχία στις συστάδες/συνιστώσες για τις διάφορες τιμές του κατωφλίου. Ο υπολογισμός όμως είναι αρκετά δαπανηρός, έτσι ως στόχος τίθεται η εύρεση των ελάχιστων δυνατών ακμών έτσι ώστε η αφαίρεση οποιασδήποτε ακμής να έχει ως αποτέλεσμα την "διάσπαση" του γράφου σε συνιστώσες, με το επιπλέον όρο ότι δεν θα πρέπει να υπάρχει άλλη ακμή με μικρότερο βάρος που θα μπορούσε να τις συνδέσει. Το ελάχιστο γεννητικό δέντρο ικανοποιεί τα παραπάνω κριτήρια και με τη χρήση του αλγόριθμου του Prim γίνεται και αποδοτικά.

4.1.3 Δημιουργία Ιεραρχίας Συστάδων

Το επόμενο βήμα είναι αυτό της δημιουργίας ιεραρχίας συστάδων. Κάνοντας χρήση του ελάχιστου γεννητικού δέντρου αυτή η διαδικασία είναι αρκετά απλή. Το μόνο που χρειάζεται να γίνει είναι η αύξουσα ταξινόμηση των ακμών

του δέντρου και στη συνέχεια με μία επαναληπτική διαδικασία η διαγραφή των ακμών και η συνένωση των κορυφών που συνέδεαν σε μία διαδικασία η οποία είναι πανομοιότυπη με το αντίστοιχο βήμα της ιεραρχική συσταδοποίησης. Παρεμπιπτόντως, η εύρεση των κορυφών που θα πρέπει να συνενωθούν είναι το δυσκολότερο κομμάτι αυτής της διαδικασίας αλλά και αυτό καθίσταται απλό με τη χρήση δομών δεδομένων union-find.

4.1.4 Συμπύκνωση της Ιεραρχίας Συστάδων

Έχοντας τελειώσει με τη διαδικασία της ιεραρχικής συσταδοποίησης, το αποτέλεσμα που έχει μέχρι στιγμής παραχθεί από την επεξεργασία των δεδομένων είναι μια κλασική ιεραρχία συστάδων. Όπως αναφέρθηκε όμως και στα περί σχεδιασμού του αλγορίθμου, είναι θεμιτή η δυνατότητα εξαγωγής επίπεδων διαμοιρασμών του χώρου. Προς αυτό το σκοπό, μία πιθανή λύση θα ήταν η επιλογή ενός επιπέδου από την ιεραρχία αλλά αυτό περιορίζει τα αποτελέσματα σε μία μόνο πυκνότητα, γεγονός που οι συγγραφείς ήθελαν να αποφύγουν. Η λύση δόθηκε με τα επόμενα δύο βήματα του αλγορίθμου HDBSCAN το πρώτο εκ των οποίων είναι η συμπύκνωση της ιεραρχίας των συστάδων που δημιουργήθηκε στο προηγούμενο βήμα.

Το βήμα της συμπύκνωσης αποσκοπεί στη εξαγωγή μόνο των ”σημαντικών” συστάδων οπότε η ιεραρχία που δημιούργησε ο αλγόριθμος HDBSCAN πρέπει να απλοποιηθεί. Αυτό υλοποιείται με τον ορισμό ενός μεγέθους που εκφράζει τον ελάχιστο δυνατό αριθμό δεδομένων που πρέπει να ανήκουν σε μία συστάδα έτσι ώστε αυτή να υφίσταται (MinPts), και την διάσχιση της ιεραρχίας από την κορυφή προς τα κάτω. Κατά τη διάρκεια της διάσχισης, εξετάζεται η κάθε διάσπαση με τρία δυνατά ενδεχόμενα:

1. Αν μία από τις δύο συνιστώσες που θα δημιουργηθούν με τη διάσπαση έχει

πλήθος δεδομένων μικρότερο από τον ελάχιστο δυνατό αριθμό δεδομένων, τότε η διάσπαση δεν θεωρείται ”πραγματική” και τα στοιχεία που ανήκουν στην εν λόγο συστάδας θεωρούνται ως δεδομένα που ”έπεσαν έξω” από αυτήν. Η συνιστώσα που έχει επαρκή δεδομένα διατηρεί το id της συστάδας γονέα, η οποία αντιμετωπίζεται σαν να έχει συρρικνωθεί.

2. Αν και οι δύο συνιστώσες έχουν επαρκή δεδομένα, τότε η διάσπαση θεωρείται ”πραγματική” και σημειώνεται στην ιεραρχία.

3. Τέλος, αν καμία από τις συνιστώσες δεν έχει επαρκές αριθμό δεδομένων, τότε και οι δύο διαγράφονται από την ιεραρχία με την συστάδα γονέα να αντιμετωπίζεται σαν να έχει ”εξαφανιστεί”.

Με αυτή τη μέθοδο, η ιεραρχία έχει επιτυχώς συμπυκνωθεί ή αλλιώς, απλοποιηθεί.

4.1.5 Εξαγωγή Συστάδων

Έχοντα πλέον απλοποιήσει την ιεραρχία ήρθε η ώρα του δεύτερου βήματος, αυτό της εξαγωγής των συστάδων. Αλλά ποιες συστάδες από την ιεραρχία πρέπει να χρησιμοποιηθούν ως η κατάλληλη αναπαράσταση του χώρου δεδομένων; Αν η ιεραρχία συστάδων αναπαρασταθεί ως ένα δενδρόγραμμα, μήπως είναι οι συστάδες που αναπαριστούν τα φύλα; ή οι ενδιάμεσοι κόμβοι; Η απάντηση στην οποία κατέληξαν οι συγγραφείς ήταν ότι οι αντιπροσωπευτικές συστάδες ήταν αυτές που ”επέζησαν” περισσότερο κατά το βήμα της σταδιακής μείωσης πυκνότητας (διάσπαση σε συνιστώσες) στη διαδικασία της συμπύκνωσης της ιεραρχίας.

Τυπικά, αυτή η έννοια ορίζεται ως σταθερότητα (stability) και για να μετρηθεί, ορίστηκε μία μεταβλητή $\lambda = \frac{1}{d_{mreach-k}}$. Η απόσταση $d_{mreach-k}$ είναι

διαθέσιμη από το βήμα δημιουργίας της ιεραρχίας για κάθε δεδομένο και καταγράφεται όταν αυτό ”πέφτει έξω” από τη συστάδα κατά τη διαδικασία της συμπύκνωσης. Το λ ορίζεται και για τις συστάδες. Πιο συγκεκριμένα, οι συστάδες έχουν τα λ_{birth} και λ_{death} που αντιστοιχούν στη τιμή λ όταν η συστάδα δημιουργήθηκε (λόγω διάσπασης του γονέα) και όταν η συστάδα διασπάστηκε ή ”εξαφανίστηκε” αντίστοιχα. Όπως μπορεί κάποιος να υποθέσει, η τιμές λ των δεδομένων που ανήκουν σε μία συστάδα, έχουν τιμές ανάμεσα στα λ_{birth} και λ_{death} καθώς το δεδομένο είτε αποσπάτε από την συστάδα είτε η συστάδα διασπάτε. Έχοντας ορίσει το λ, η σταθερότητα κάθε συστάδας ορίζεται ως

$$S(C_i) = \sum_{p \in cluster} (\lambda_p - \lambda_{birth})$$

όπου το p αντιπροσωπεύει τα δεδομένα και $S(C_i)$ την σταθερότητα της συστάδας C_i.

Έχοντας ορίσει το μέτρο με βάση το οποίο θα αξιολογηθούν οι πιο κατάλληλες συστάδες, οι συγγραφείς όρισαν τον επιπλέον περιορισμό ότι αν επιλεγεί κάποια συστάδα, τότε τα παιδιά της δεν θα μπορούσαν να επιλεγούν. Συνεπώς, θέλοντας να βρεθεί ο καλύτερος δυνατόν συνδυασμός συστάδων, αυτή η διαδικασία ορίστηκε ως ένα πρόβλημα βελτιστοποίησης όπου ο στόχος ήταν η μεγιστοποίηση του αθροίσματος

$$J = \sum_{i=2} \delta_i S(C_i)$$

με το δ_i να δείχνει αν η συστάδα C_i συμμετέχει στη τελική ”επίπεδη” λύση ($\delta_i = 1$) ή όχι ($\delta_i = 0$).

Οπότε, ο κάθε κόμβος επεξεργάζεται ξεκινώντας από τα φύλλα και αποφασίζεται αν θα χρησιμοποιηθεί ο πατέρας ή τα παιδιά συγκρίνοντας τις τιμές σταθερότητας τους (πατέρας εναντίον σύνολου παιδιών που συμμετέχουν). Αυτή

η σχέση φαίνεται καθαρότερα στη παρακάτω εξίσωση.

$$\hat{S}(C_i) = \begin{cases} S(C_i), & \text{αν το } C_i \text{ είναι κόμβος φύλλο} \\ \max\{S(C_i), \hat{S}(C_{il}) + \hat{S}(C_{ir})\} & \text{αν το } C_i \text{ είναι εσωτερικός κόμβος} \end{cases}$$

όπου C_{ir} και C_{il} είναι δεξιά και αριστερά παιδιά του C_i (για χάριν απλότητας αναφέρονται δυαδικά δέντρα, αντίστοιχη λογική ισχύει και για n-ary δέντρα).

4.1.6 Επιτάχυνση HDBSCAN

Έχοντας αναλύσει την λογική πίσω από τον αλγόριθμο HDBSCAN και αναγνωρίζοντας τα προτερήματα του έναντι του κλασικού αλγόριθμου DBSCAN, αυτή η εργασία επικεντρώθηκε στην υλοποίηση ενός παράλληλου HDBSCAN.

Η υλοποίηση που παρουσιάζεται παρακάτω είναι γραμμένη στη γλώσσα Python χρησιμοποιώντας τη βιβλιοθήκη RAPIDS cuML και είναι σχεδιασμένη έτσι ώστε να δουλεύει σε αρκετά υψηλό επίπεδο, με τον χρήστη να μην χρειάζεται να ασχοληθεί καθόλου με τις παραμέτρους που απαιτεί η CUDA για να τρέξει η συνάρτηση kernel. Η υλοποίηση καθαυτή γίνεται χρησιμοποιώντας την γλώσσα C++ αλλά η μετάβαση από εντολές Python στις κατάλληλες της C++ για την εκτέλεση της συνάρτησης kenrel δεν είναι κάτι το οποίο ο χρήστης θα χρειαστεί να μάθει.

```
from sklearn import datasets
from cuml.cluster import HDBSCAN

X, y = datasets.make_moons(n_samples=50, noise=0.05)

model = HDBSCAN(min_samples=5, gen_min_span_tree=True)
y_hat = model.fit_predict(X)
```

Στο παραπάνω παράδειγμα, φαίνεται η εκτέλεση συσταδοποίησης χρησιμο-

ποιώντας τον αλγόριθμο HDBSCAN για ένα σετ δεδομένων 50 στοιχείων με 5% θόρυβο. Όπως αναφέρθηκε παραπάνω, η εκτέλεση είναι υπερβολικά απλή.

Στη συνέχεια, παρουσιάζεται η παράλληλη υλοποίηση με περισσότερη λεπτομέρεια

```
void _fit_hdbscan(const raft::handle_t& handle,
                  const value_t* X,
                  size_t m,
                  size_t n,
                  raft::distance::DistanceType metric,
                  Common::HDBSCANParams& params,
                  value_idx* labels,
                  value_t* core_dists,
                  Common::hdbscan_output<value_idx, value_t>& out)
{
  auto stream    = handle.get_stream();
  auto exec_policy = handle.get_thrust_policy();

  int min_cluster_size = params.min_cluster_size;

  build_linkage(handle, X, m, n, metric, params, core_dists, out);

  /**
   * Condense branches of tree according to min cluster size
   */
  detail::Condense::build_condensed_hierarchy(handle,
                                              out.get_children(),
                                              out.get_deltas(),
                                              out.get_sizes(),
                                              min_cluster_size,
                                              m,
                                              out.get_condensed_tree());

  /**
   * Extract labels from stability
   */
```

```
rmm::device_uvector<value_t>
    tree_stabilities(out.get_condensed_tree().get_n_clusters(),
                                        handle.get_stream());

rmm::device_uvector<value_idx> label_map(out.get_condensed_tree().get_n_clusters(),
                                     handle.get_stream());
value_idx n_selected_clusters =
  detail::Extract::extract_clusters(handle,
                                out.get_condensed_tree(),
                                m,
                                labels,
                                tree_stabilities.data(),
                                out.get_probabilities(),
                                label_map.data(),
                                params.cluster_selection_method,
                                out._get_inverse_label_map(),
                                params.allow_single_cluster,
                                params.max_cluster_size,
                                params.cluster_selection_epsilon);

out.set_n_clusters(n_selected_clusters);

auto lambdas_ptr =
    thrust::device_pointer_cast(out.get_condensed_tree().get_lambdas());
value_t max_lambda = *(thrust::max_element(
  exec_policy, lambdas_ptr, lambdas_ptr + out.get_condensed_tree().get_n_edges()));

detail::Stability::get_stability_scores(handle,
                                    labels,
                                    tree_stabilities.data(),
                                    out.get_condensed_tree().get_n_clusters(),
                                    max_lambda,
                                    m,
                                    out.get_stabilities(),
                                    label_map.data());

/**
 * Normalize labels so they are drawn from a monotonically increasing set
 * starting at 0 even in the presence of noise (-1)
```

```
   */

  thrust::transform(exec_policy,
                    labels,
                    labels + m,
                    out.get_labels(),
                    [label_map = label_map.data()] __device__(value_idx label) {
                      if (label != -1) return label_map[label];
                      return -1;
                    });
}
```

Στον κώδικα που δίνεται παραπάνω, φαίνεται ο βασικός πυρήνας που υλοποιεί τον αλγόριθμο HDBSCAN στη βιβλιοθήκη RAPIDS cuML. Όπως μπορεί κάποιος να δει, η υλοποίηση είναι χωρισμένη σε κομμάτια με τα βήματα που αναλύθηκαν παραπάνω να υλοποιούνται από διαφορετικές συναρτήσεις.

Πιο συγκεκριμένα, η συνάρτηση _fit_hdbscan δέχεται ως όρισμα το σετ δεδομένων σε ένα πίνακα X, το μέγεθος του πίνακα X,τον μέτρο που χρησιμοποιείται για τον υπολογισμό της απόστασης, τις παραμέτρους που έδωσε ο χρήστης για την εκτέλεση του προγράμματος,έναν πίνακα που περιέχει τα labels, έναν πίνακα για την αντιστοίχηση δεδομένων σε συστάδες (out) και ένα πίνακα με τις ήδη υπολογισμένες αποστάσεις core.

Όπως αναφέρθηκε προηγουμένως, ο υπολογισμός της απόστασης $core_k(\alpha)$ είναι απαραίτητος για το mutual reachability distance. Για τον υπολογισμό του, πρέπει να βρεθούν οι k κοντινότεροι γείτονες του δεδομένου που εξετάζεται και στη συνέχεια να υπολογιστεί η απόσταση μεταξύ του στοιχείου α και του πιο απομακρυσμένου γείτονα. Για την επιτάχυνση αυτής της χρονοβόρας διαδικασίας, η βιβλιοθήκη RAPIDS cuML κάνει χρήση του brute-force kNN αλγορίθμου από την βιβλιοθήκη FAISS της Meta. Το τελικό αποτέλεσμα δίνεται ως όρισμα στον βασικό πυρήνα.

Το πρώτο σημαντικό βήμα του πυρήνα είναι η κλήση της συνάρτησης build_linkage που υλοποιεί τα βήματα του μετασχηματισμού του χώρου, της δημιουργίας του ελάχιστου γεννητικού δέντρου και της δημιουργίας της ιεραρχίας. Αξίζει να σημειωθεί ότι οι βιβλιοθήκες RAPIDS περιλαμβάνει ένα νέο primitive για τον υπολογισμό του ελάχιστου γεννητικού δέντρου βελτιστοποιώντας την αξιοποίηση της μνήμης της κάρτας γραφικών.

Στη συνέχεια, καλείται η συνάρτηση build_condensed_hierarchy η οποία σύμφωνα με την ονομασία της πραγματοποιεί συμπύκνωση της ιεραρχίας. Για την επιτάχυνση της συμπύκνωσης της ιεραρχίας, η βιβλιοθήκη cuML εκμεταλλεύεται το έργο των McInnes και Healy[11] έτσι ώστε να μεγιστοποιηθεί η αξιοποίηση της κάρτας γραφικών.

Τέλος, καλείται η συνάρτηση extract_clusters που εξάγει τις συστάδες που δημιουργήθηκαν και ο πυρήνας ολοκληρώνει την εκτέλεση του κανονικοποιώντας τα labels των συστάδων και καταχωρίζοντας τα αποτελέσματα στους κατάλληλους πίνακες.

4.2 K-means

Ο αλγόριθμος K-means έχει αναλυθεί τόσο στη σειριακή όσο και σε διάφορες παράλληλες υλοποιήσεις του. Στην ανάλυση που έγινε στα πλαίσια αυτής της εργασίας, έγινε γνωστό ότι η κύρια διαφορά ανάμεσα στις υλοποιήσεις είναι η αξιοποίηση της μνήμης της κάρτας γραφικών, με τις υλοποιήσεις που κάνουν την πιο αποδοτική διαχείριση να είναι και οι πιο γρήγορες.

Οι Kruliš και Kratochvíl στο [6] παρουσίασαν μια λεπτομερή ανάλυση του αλγόριθμου k-means με σκοπό την εύρεση της βέλτιστης υλοποίησης του k-means στην αρχιτεκτονική των καρτών γραφικών η οποία περιγράφεται συνοπτικά στη συνέχεια.

Για την ευκολότερη ανάλυση του αλγορίθμου, χωρίζεται σε δύο κομμάτια αυτά της: α) ανάθεσης και β) ενημέρωσης. Το κομμάτι της ανάθεσης είναι το μέρος του αλγορίθμου στο οποίο το κάθε δεδομένο ανατίθεται στην κοντινότερη συστάδα και το κομμάτι της ενημέρωσης είναι αυτό στο οποίο υπολογίζονται τα νέα κέντρα των συστάδων. Για το κομμάτι της ανάθεσης δοκιμάστηκαν διαφορετική τρόποι χρήσης της cache μνήμης βελτιώνοντας την απόδοση σημαντικά, ενώ για το κομμάτι της ενημέρωσης αν και κρίθηκε ότι οι βελτιώσεις που ήταν εφικτές ήταν ελάχιστες, παρουσιάστηκαν διάφοροι τρόποι συγχρονισμού εγγραφής από το οποίο εξαρτάται κυρίως η απόδοση της ενημέρωσης.

Στα πλαίσια της ανάλυσης τους, οι Kruliš και Kratochvíl χρησιμοποίησαν δύο διαφορετικές διατάξεις μνήμης:

- AoS (Array-of-Structures), Πίνακας δομών, όπου δομή είναι ένα δεδομένο. Με λίγα λόγια το κάθε διάνυσμα που αντιπροσωπεύει ένα δεδομένο θα αποθηκεύεται σε ένα block μνήμης.
- SoA (Structure-of-Arrays), μία αντεστραμμένη διάταξη όπου οι τιμές των διαστάσεων αποθηκεύονται σε ένα block μνήμης.

Για τις παραπάνω διατάξεις υπάρχουν και οι AoS32 και SoA32 όπου τα δεδομένα παίρνουν padding στο κοντινότερο πολλαπλάσιο του 32 για να αξιοποιηθούν καλύτερα τα warps της κάρτας γραφικών.

4.2.1 Ανάθεση

Όπως αναφέρθηκε προηγουμένως, για το κομμάτι της ανάθεσης, θα εξεταστούν κυρίως διαφορετικοί τρόποι αξιοποίησης της μνήμης cache. Αυτό γίνεται διότι κατά τη διάρκεια της συγγραφής, ένας δημοφιλής τρόπος εκτέλεσης της διαδικασίας της ανάθεσης ήταν ο ανεξάρτητος διαμοιρασμός δεδομένων με αποτέλεσμα την κακή χρήση της μνήμης cache που οδηγούσε σε περιορισμούς της

υπολογιστικής απόδοσης λόγω περιορισμών στο bandwidth της μνήμης και το αυξημένο latency. Οπότε, ως στόχος της χρήσης μνήμης cache τέθηκε η βελτίωση του λόγου μεταξύ υπολογιστικών πράξεων προς μεταφορών δεδομένων λόγω αστοχίας στη μνήμη cache. Με λίγα λόγια, όσο το περισσότερο δυνατόν πράξεις για τα δεδομένα που είναι ήδη φορτωμένα στη μνήμη.

Οπότε, το πρώτο βήμα πριν ξεκινήσει η ανάλυση για τη βέλτιστη στρατηγική χρήσης της μνήμης cache είναι η εύρεση του βέλτιστου διαμοιρασμού του προβλήματος στους διαθέσιμους πυρήνες. Αναφορικά με το διαμοιρασμό λοιπόν, η διαδικασία της ανάθεσης εκτιμάται ότι εκτελεί O(nkd) πράξεις σε κάθε επανάληψη του αλγορίθμου, όπου n ο αριθμός των δεδομένων, k ο αριθμός των κέντρων των συστάδων και d ο αριθμός των διαστάσεων. Συνεπώς, οι πιθανοί διαχωρισμοί θα γίνου γύρω από:

- Τα n σημεία, διαμοιρασμός δηλαδή των n δεδομένων στους διαθέσιμους πυρήνες.
- Τους k μέσους, διαμοιρασμός των k μέσων στους διαθέσιμους πυρήνες όπου ο καθένας υπολογίζει την απόσταση του απ' όλα τα δεδομένα. Στη συνέχεια πραγματοποιείται ένα βήμα reduction για την εύρεση της ελάχιστης απόστασης (του κάθε δεδομένου).
- Τις d διαστάσεις, διαμοιρασμός των διαστάσεων στους διαθέσιμους πυρήνες με τα επικείμενα $n \cdot k$ βήματα reduction για τον υπολογισμό των αθροισμάτων των διανυσμάτων το οποίο ακολουθείται από ένα τελευταίο βήμα reduction για την εύρεση του ελάχιστου (για κάθε δεδομένο).

Σύμφωνα με εμπειρικά δεδομένα, ο διαμοιρασμός γύρω από τις διαστάσεις έχει μεγάλο overhead όταν το μέτρο σύγκρισης που χρησιμοποιείται είναι η Ευκλείδεια απόσταση, με αποτέλεσμα η περαιτέρω ανάλυση να επικεντρώνεται

στους διαμοιρασμούς γύρω από τους μέσους και τα δεδομένα. Επίσης, λόγω του αρκετά μεγαλύτερου αριθμού δεδομένων από τους διαθέσιμους πυρήνες, το κέρδος του συγκεκριμένου διαμοιρασμού δεν είναι η καλύτερη αξιοποίηση των πυρήνων (core occupancy) αλλά η καλύτερη αξιοποίηση της μνήμης cache όπως θα παρουσιαστεί στη συνέχεια (κυρίως λόγω της πολλαπλής χρήσης των ήδη φορτωμένων κομματιών των πινάκων X και W, όπου X ένα $n \times d$ πίνακας που περιέχει τα δεδομένα και W ένας $k \times d$ πίνακας που περιέχει τους μέσους/κέντρα των συστάδων.

Έχοντας λοιπόν αποφασίσει για τον διαμοιρασμό των δεδομένων, στη συνέχεια παρουσιάζονται οι στρατηγικές χρήσης της μνήμης cache που παρουσίασαν τα καλύτερα αποτελέσματα στην έρευνα των συγγραφέων.

- Means, στη συγκεκριμένη στρατηγική, καταχωρείται ολόκληρος ο πίνακας W στη τοπική μνήμη.
- Regs, στη συγκεκριμένη στρατηγική, τα δεδομένα του πίνακα X μοιράζονται στα διαθέσιμα thread blocks και τα δεδομένα του πίνακα W φορτώνονται σταδιακά κάνοντας χρήση ενός ολισθαίνων παραθύρου το οποίο καλύπτει τις διαστάσεις (d) και ολισθαίνει στο k.
- Fixed, σε αυτή τη στρατηγική, χρησιμοποιείται ένα ολισθαίνων παράθυρο σταθερού μεγέθους το οποίο δεν καλύπτει απαραίτητα όλες τις διαστάσεις, δηλαδή ολισθαίνει γύρω από το d και το k.

Η στρατηγική Caching all Means, ή εν συντομία Means είναι αυτή που προσφέρει την μικρότερη βελτίωση από όλες που έχουν αναφερθεί μέχρι στιγμής. Αποτελεί μία μικρή βελτίωση πάνω στη λογική φόρτωσης των δεδομένων που χρησιμοποιούνται πολλές φορές στη μνήμη cache (πίνακας W) με τα αποτελέσματα των πράξεων να φορτώνονται στους καταχωρητές. Δυστυχώς όμως,

αν και χάρη σε μία μικρή βελτίωση γύρω από τον υπολογισμό της Ευκλείδειας απόστασης, προσφέρει μια βελτίωση στην απόδοση, το bottleneck παραμένει χάρη στον περιορισμένο αριθμό καταχωρητών.

Η στρατηγική Caching in Registers, ή εν συντομία Regs, είναι ελαφρώς πιο περίπλοκη στην υλοποίηση της καθώς αποθηκεύονται στη μνήμη cache τόσο δεδομένα του πίνακα X όσο και οι μέσοι (πίνακας W). Τα δεδομένα των σημείων (πίνακας X) αποθηκεύονται καθ' όλη τη διάρκεια των υπολογισμών, ενώ οι μέσοι εναλλάσσονται σύμφωνα με ένα ολισθαίνων παράθυρο το οποίο περιέχει σταθερό πλήθος ολόκληρων διανυσμάτων του πίνακα W.

Κάθε block νημάτων αποτελείται από $p \cdot q$ νήματα τα οποία αποθηκεύουν στους καταχωρητές τους r_p δεδομένα από τον πίνακα και r_q δεδομένα από τον πίνακα W. Συνεπώς, επεξεργάζονται ταυτόχρονα $p \cdot r_p$ σημεία στα οποία, q νήματα συνεργάζονται για τον υπολογισμό των κοντινότερων μέσων r_p στοιχείων. Με αυτή τη διάταξη, το κάθε νήμα υπολογίζει συνολικά $r_p \cdot r_q$ αποστάσεις.

Ο υπολογισμός των αποστάσεων γίνεται ως εξής. Το κάθε νήμα μπαίνει σε μία επαναληπτική διαδικασία όπου υπολογίζεται η απόσταση της κάθε διάστασης ξεχωριστά. Το αποτέλεσμα αποθηκεύεται σε ένα μικρό πίνακα μερικών αποστάσεων (και αυτός βρίσκεται στη διαμοιραζόμενη μνήμη). Με τον υπολογισμό των αποστάσεων όλων των διαστάσεων, με ένα βήμα reduction υπολογίζονται τα r_p ελάχιστα του πίνακα αποστάσεων για κάθε r_q. Στη συνέχεια, τα μερικά ελάχιστα αποθηκεύονται σε ένα πίνακα $r \cdot q$ διαστάσεων στη κοινή μνήμη όπου επεξεργάζονται με ένα ακόμα βήμα reduction για την εύρεση της ελάχιστης απόστασης.

Με τη χρήση της διαμοιραζόμενης μνήμης έναντι της καθολικής, το bottleneck μετακινείται από τη global μνήμη στη διαμοιραζόμενη. Αυτό, έχει ως αποτέλεσμα την προσπάθεια μείωσης φόρτωσης δεδομένων. Αυτό που πετυχαίνει η

παραπάνω στρατηγική είναι η μείωση αυτών των φορτώσεων κατά $r_p \cdot r_q$ φορές. Αξίζει να σημειωθεί επίσης ότι εφόσον οι τιμές r, p, r_p και r_q είναι μεταβλητές πρέπει να οριστούν με έναν τρόπο που ισορροπεί τη μείωση των φορτώσεων με την απαιτούμενη μνήμη των καταχωρητών.

Η στρατηγική Fixed Caching Window, ή εν συντομία Fixed, χρησιμοποιεί ένα ολισθαίνων παράθυρο σταθερού μεγέθους και βελτιστοποιεί τον υπόλοιπο κώδικα για το συγκεκριμένο μέγεθος παραθύρου. Κάποια από τα προφανή προβλήματα που προκύπτουν από αυτή τη στρατηγική είναι ότι δεν είναι πάντα δυνατόν να συμπεριλαμβάνονται όλες οι διαστάσεις στο παράθυρο με αποτέλεσμα τα ενδιάμεσα αποτελέσματα των αθροίσεων να πρέπει να αποθηκευτούν στην μνήμη. Από την άλλη, η ίδια ιδιότητα επιτρέπει σε αυτή τη στρατηγική να μπορεί να κλιμακωθεί πολύ καλύτερα από τη στρατηγική Regs χωρίς να φτάσει το όριο της μνήμης. Επιπλέον παράγεται καλύτερος κώδικας λόγω αυτόματων βελτιστοποιήσεων χάρη στο σταθερό μέγεθος παραθύρου.

Κάθε νήμα στο block νημάτων ανατίθεται ένα στοιχείο του πίνακα X (ένα διάνυσμα) και υπολογίζει την απόσταση με r μέσους. Τα νήματα του block φορτώνουν ένα μέρος των r μέσων το οποίο καθορίζεται από το μέγεθος τους παραθύρου και στη συνέχεια το κάθε νήμα υπολογίζει ή ενημερώνει το μερικό του άθροισμα. Όταν υπολογιστούν όλες οι διαστάσεις των r μέσων, τότε το κάθε νήμα πραγματοποιεί ένα τοπικό βήμα reduction ώστε να βρει το τοπικό ελάχιστο και συνεχίζει με τους επόμενους r μέσους. Προσοχή πρέπει να δοθεί στον καθορισμό του r καθώς είναι άμεσα συνδεδεμένο με το μέγεθος και το πλήθος των καταχωρητών που απαιτούνται.

Συγκρίνοντας τις δύο αυτές στρατηγικές χρήσης μνήμης cache (Regs και Fixed) οι συγγραφείς κατέληξαν στο συμπέρασμα ότι για έναν κοινό αριθμό διαστάσεων ($d < 100$) η στρατηγική Regs παρέχει τα πιο γρήγορα αποτελέσματα

ενώ για μεγάλο αριθμό διαστάσεων ($d > 100$) η στρατηγική Fixed παρέχει τα πιο γρήγορα αποτελέσματα. Αξίζει επίσης να σημειωθεί ότι η διάταξη μνήμης που χρησιμοποιήθηκε για την ανάλυση αυτών των στρατηγικών ήταν η διάταξη SoA.

4.2.2 Ενημέρωση

Έχοντας ολοκληρώσει την ανάλυση της διαδικασίας της Ανάθεσης, ακολουθεί η διαδικασία της ενημέρωσης η οποία πραγματοποιεί reduction για τις συστάδες.

Αντίστοιχα με την διαδικασία της ανάθεσης, στη διαδικασία της ενημέρωσης ο διαμοιρασμός μπορεί να γίνει είτε γύρω από τα δεδομένα είτε γύρω από τους μέσους. Συγκριτικά, ο διαμερισμός γύρω από τα δεδομένα παρέχει καλύτερο παραλληλισμό εκμεταλλευόμενος τους πυρήνες της κάρτας γραφικών με το κόστος όμως overhead λόγω συγχρονισμού καθώς πολλά νήματα θα πρέπει να ενημερώσουν τις ίδιες τιμές ταυτοχρόνως. Ευτυχώς, οι ενημερώσεις είναι αρκετά απλές έτσι ώστε να υποστηρίζονται οι ατομικές πράξεις. Ο διαμερισμός γύρω από του μέσους από την άλλη δεν έχει προβλήματα συγχρονισμού αλλά ο παραλληλισμός περιορίζεται από το k το οποίο είναι σημαντικά μικρότερο από το πλήθος των πυρήνων της κάρτας γραφικών. Επιπλέον περιορισμό αποτελεί η ανάγκη διαχωρισμού των σχετικών δεδομένων για την κάθε συστάδα.

Για τον καθορισμό της βέλτιστης τακτικής, οι συγγραφείς υλοποίησαν τέσσερεις διαφορετικές προσεγγίσεις όσο αφορά τον συγχρονισμό. Αυτές οι προσεγγίσεις, είναι οι παρακάτω:

- Nosync, ο διαμοιρασμός των δεδομένων γίνεται γύρω από τους μέσους, έτσι δεν χρειάζεται συγχρονισμός για την ενημέρωση των τιμών. Ένα στοιχείο του πίνακα W υπολογίζεται από ένα νήμα.
- AtomicPoint, ο διαμοιρασμός των δεδομένων γίνεται γύρω από τα δεδο-

μένα του πίνακα X με τις ενημερώσεις να γίνονται χρησιμοποιώντας την πράξη atomic add. Το κάθε νήμα ανατίθεται ένα στοιχείο του πίνακα X (n νήματα επεξεργάζονται d τιμές επαναληπτικά).

- AtomicWarp Παρόμοιο με το AtomicPoint με μόνη διαφορά ότι το κάθε στοιχείο του πίνακα X ανατίθεται σε ένα warp.
- AtomicFine Παρόμοιο με το AtomicPoint με μόνη διαφορά ότι η κάθε τιμή του πίνακα X ανατίθεται σε ένα νήμα ($n \cdot d$ νήματα θα επεξεργαστούν τα στοιχεία).

Στα πειράματα που εκτελέστηκαν για τον καθορισμό της βέλτιστης προσέγγισης, αποφάνθηκε ότι ο NoSync ήταν ο πιο αργός κατά αρκετές κλίμακες μεγέθους. Στην επόμενη θέση από άποψη χειρότερης επίδοσης ήταν οι AtomicWrap και AtomicPoint με τον πρώτο να είναι ελαφρώς πιο αργός για χαμηλό αριθμό διαστάσεων και ο δεύτερος πιο αργός για μεγαλύτερο πλήθος διαστάσεων. Τέλος, το βέλτιστο αποτέλεσμα παρουσίασε ο AtomicFine.

Όσο αφορά τη διάταξη μνήμης, οι μέθοδοι AtomicFine και AtomicWarp αποδίδουν καλύτερα με τη διάταξη AoS ενώ ο AtomicPoint με τη διάταξη SoA. Επιπλέον, αξίζει να σημειωθεί ότι αν και οι μέθοδοι με το συγχρονισμό αποδείχθηκαν καλύτερες από την NoSync, παραμένει το κόστος συγχρονισμού ειδικά όταν πολλά νήματα γράφουν ταυτόχρονα στην ίδια θέση μνήμης. Για την επίλυση αυτού, βρέθηκε ότι όσο περισσότερα δεδομένα τόσο λιγότερες πιθανές συγκρούσεις θα υπάρχουν, άρα για μεγάλα k και d μειώνονται και άλλο οι χρόνοι εκτέλεσης. Τέλος, δοκιμάστηκε και μια επιπλέον τεχνική AtomicShm σε μια απόπειρα αξιοποίησης της κοινή μνήμης η οποία φάνηκε να παρουσιάζει θετικά αποτελέσματα μόνο όταν ένα νήμα είχε αρκετά δεδομένα προς επεξεργασία ώστε να αξιοποιεί τα οφέλη της (το AtomicShm αποτελεί μια παραλλαγή

του AtomicFine όπου κάθε νήμα επεξεργάζεται p τιμές του πίνακα X).

4.2.3 Εκτελέσιμος κώδικας

Έχοντας δει την ανάλυση του καθενός κομματιού που αποτελεί τον αλγόριθμο K-means, θα ήταν απλό να σκεφτεί κάποιος ότι ενώνοντας τις καλύτερες προσεγγίσεις από το κάθε κομμάτι θα δώσει τον πιο γρήγορο συνολικό αλγόριθμο K-means. Δυστυχώς όμως, αυτό δεν ισχύει κυρίως λόγω του γεγονότος ότι η βέλτιστες προσεγγίσεις χρειάζονται διαφορετικές διατάξεις μνήμης για να λειτουργήσουν τέλεια. Οι συγγραφείς λοιπόν, κατέληξαν στις συγκεκριμένες ενώσεις ως τις καλύτερες

- FusedFixed, το οποίο ενώνει τη Fixed στρατηγική χρήσης μνήμης cache με την AtomicPoint προσέγγιση στο συγχρονισμό βασιζόμενο στη λογική ότι χρειάζονται την ίδια διάταξη μνήμης καθώς και ότι χρησιμοποιούν ένα νήμα ανά στοιχείο για την επεξεργασία.
- FusedRegs που ενώνει τη Regs στρατηγική χρήσης μνήμης cache με μία προσέγγιση στο συγχρονισμό το οποίο είναι συνδυασμός των AtomicPoint και AtomicFine. Η μέθοδος συγχρονισμού που προκύπτει, χρησιμοποιεί την κατανομή νημάτων από τη Regs για την ενημέρωση του πίνακα W.

Η μέθοδος AtomicShm δεν χρησιμοποιήθηκε διότι η κοινή μνήμη έχει ήδη ανατεθεί στο βήμα της ανάθεσης.

Παρακάτω παρουσιάζεται η υλοποίηση του αλγορίθμου FusedFixed

```
template<typename F = float, typename IDX_T = std::uint32_t, class LAYOUT =
    AoSLayoutPolicy<>, class LAYOUT_MEANS, int shmK, int shmDim>
__global__ void fusedCachedFixedKernel(const F* __restrict__ data, const F*
    __restrict__ meansIn, F* __restrict__ meansOut, IDX_T *clusterSizes,
  IDX_T dim, IDX_T k, IDX_T n, std::uint32_t privCopies)
```

```
{
   volatile __shared__ F shmMeans[shmDim][shmK];

   IDX_T idx = threadIdx.x + blockIdx.x * blockDim.x;
   auto precomputedData = LAYOUT::precomputeConstants(n, dim);
   auto precomputedMeans = LAYOUT_MEANS::precomputeConstants(k, dim);

   F dists[shmK];
   F minDist = (F)100000000;
   IDX_T nearestIdx = 0;

   for (IDX_T mOffset = 0; mOffset < k; mOffset += shmK) {
      // Reset variables for new set of means.
      #pragma unroll
      for (IDX_T i = 0; i < shmK; ++i) {
         dists[i] = (F)0;
      }

      for (IDX_T dimOffset = 0; dimOffset < dim; dimOffset += shmDim) {
         // Load means to shm
         for (IDX_T i = threadIdx.x; i < shmK * shmDim; i += blockDim.x) {
            IDX_T d = i / shmK;
            IDX_T m = i % shmK;
            shmMeans[d][m] = LAYOUT_MEANS::at(meansIn, m + mOffset, d + dimOffset,
                precomputedMeans);
         }

         __syncthreads();

         // Accumulate distance values
         #pragma unroll
         for (IDX_T d = 0; d < shmDim; ++d) {
            F x = LAYOUT::at(data, idx, d + dimOffset, precomputedData);
            #pragma unroll
            for (IDX_T m = 0; m < shmK; ++m) {
               F dx = x - shmMeans[d][m];
               dists[m] += dx * dx;
            }
         }
```

```
        __syncthreads();
      }

      #pragma unroll
      for (IDX_T i = 0; i < shmK; ++i) {
        F dist = sqrtf(dists[i]);
        if (minDist > dist) {
          minDist = dist;
          nearestIdx = i + mOffset;
        }
      }
    }

    atomicInc(&clusterSizes[nearestIdx], ~(IDX_T)0);

    meansOut = privatizeResultPointer<F, IDX_T, LAYOUT_MEANS>(meansOut, dim, k,
        privCopies);
    for (std::uint32_t d = 0; d < dim; ++d) {
      F x = LAYOUT::at(data, idx, d, precomputedData);
      F* res = &LAYOUT_MEANS::at(meansOut, nearestIdx, d, precomputedMeans);
      AtomicPolicy<F>::add(res, x);
    }
}
```

Όπως φαίνεται, ο κώδικας είναι αρκετά απλός και παρουσιάζει πάρα πολλές ομοιότητες με την σειριακή του εκδοχή. Τα templates στην αρχή κατά τη δήλωση της συνάρτησης πυρήνα, χρησιμοποιούνται για τον ορισμό των διατάξεων της μνήμης αλλά και για το μέγεθος που είναι διαθέσιμο στη κάρτα γραφικών που χρησιμοποιείται. Στη συνέχεια, μετά τη δημιουργία των απαραίτητων μεταβλητών, του παραθύρου shmMeans αλλά και των πινάκων X και W με τις μεταβλητές precomputedData και precomputedMeans αντίστοιχα, ο αλγόριθμος συνεχίζει με τον υπολογισμό των αποστάσεων μέσα σε μία εμφωλευμένη for. Με την ολοκλήρωση των εμφωλευμένων for, υπολογίζεται η ελάχιστη από-

σταση και στη συνέχεια πραγματοποιείται η διαδικασία της ενημέρωσης βάση τη πολιτική συγχρονισμού που έχει δοθεί ως όρισμα από τα Templates. Αξίζει επίσης να σχολιαστεί και η εντολή privatizeResultPointer που εκτελείται αμέσως πριν από τη διαδικασία της ενημέρωσης. Η εντολή αυτή αποσκοπεί στη βελτίωση της απόδοσης της ενημέρωσης για χαμηλότερο συνδυασμό k και d χρησιμοποιώντας μια τεχνική ιδιωτικοποίησης (privatization) για την αποφυγή των συγκρούσεων. Από τη στιγμή όμως που έχει αρνητικά αποτελέσματα για μεγάλους συνδυασμούς k και d, πιο συγκεκριμένα όταν ξεπεραστεί το όριο της μνήμης cache L2, δεν εφαρμόζεται σε κάθε περίπτωση. Με λίγα λόγια, αγνοώντας τις ατομικές εντολές, η παράλληλη υλοποίηση του΄ K-means είναι στην ουσία ο σειριακή εκδοχή με διαφορετικό βήμα στο βρόχο for.

4.3 Power Iteration Clustering

Οι Silva και λοιποί στο [7] παρουσιάζουν μία παράλληλη υλοποίηση του αλγορίθμου Power Iteration Clustering (PIC), την GPU PIC (GPIC). Ο αλγόριθμος PIC είναι ένας αλγόριθμος που πραγματοποιεί τη συσταδοποίηση των δεδομένων κάνοντας χρήση των ιδιοτιμών του πίνακα ομοιότητας, παρόμοια με τον Spectral Clustering[9].

Ο αλγόριθμος PIC ξεκινάει τη διαδικασία της συσταδοποίησης με τον υπολογισμό τον πίνακα ομοιότητας του σετ δεδομένων που επεξεργάζεται. Όπως έχει ορισθεί και παραπάνω, ο πίνακας ομοιότητας είναι ένας τετραγωνικός, συμμετρικός πίνακας διαστάσεων $n \times n$, όπου n είναι το πλήθος των δεδομένων στο σετ, όπου αναγράφει τα αποτελέσματα που επιστρέφει η συνάρτηση ομοιότητας για όλα τα ζευγάρια δεδομένων. Κάνοντας χρήση του πίνακα ομοιότητας (affinity matrix) που υπολογίστηκε, υπολογίζονται με τη σειρά τους ο degree matrix που συσχετίζεται με τον πίνακα ομοιότητας και ο κανονικοποιημένος

πίνακας ομοιότητας. Ο degree matrix είναι ένας διαγώνιος πίνακας $n \times n$ όπου τα στοιχεία του είναι το άθροισμα των στοιχείων των αντίστοιχων γραμμών του πίνακα ομοιότητας. Ο κανονικοποιημένος πίνακας ομοιότητας ορίζεται ως το γινόμενο του degree matrix και του πίνακα ομοιότητας. Ο κανονικοποιημένος πίνακας ομοιότητας συσχετίζεται με τον random-walk normalized Laplacian matrix ο οποίος έχει κάποιες πολύ ενδιαφέρουσες ιδιότητας. Η πιο σημαντική ιδιότητα στα πλαίσια αυτού του αλγορίθμου είναι το γεγονός ότι το k μικρότερο ιδιοδιάνυσμα του διαμερίζει τα στοιχεία του σε k μέρη έτσι ώστε να μεγιστοποιείται το κριτήριο Normalized Cut. Τα k μικρότερα ιδιοδιανύσματα του Laplacian matrix είναι τα k μεγαλύτερα του κανονικοποιημένου πίνακα ομοιότητας. Για την προσέγγιση τους, χρησιμοποιείται μία τεχνική που ονομάζεται Power Iteration η οποία συγκλίνει στο ιδιοδιάνυσμα με τη μεγαλύτερη τιμή. Η τεχνική Power Iteration είναι μία επαναληπτική διαδικασία που ξεκινάει από ένα μη μηδενικό διάνυσμα και υπολογίζει το εξής γινόμενο

$$v^{t+1} = cWv^t$$

, όπου c είναι μια σταθερά κανονικοποίησης του γινομένου, ώστε να μην γίνει πολύ μεγάλο, v το διάνυσμα που υπολογίζεται και W ο κανονικοποιημένος πίνακας ομοιότητας. Αν και το αποτέλεσμα της σύγκλισης δεν είναι χρήσιμο όσο αφορά αυτόν τον αλγόριθμο, παρατηρήθηκε ότι με τον συνεχή υπολογισμό των v^t, ο αλγόριθμος καταλήγει σε διάφορες εκδοχές του χώρου των δεδομένων στις οποίες τα δεδομένα έχουν διαμεριστεί σε διάφορες συστάδες. Έχοντας ήδη διαχωρίσει τα δεδομένα σε ξεκάθαρα διαχωρισμένες ομάδες, μπορεί να εφαρμοστεί ένας κλασικός, γρήγορος αλγόριθμος συσταδοποίησης (όπως ο K-means) για την τελική εξαγωγή των αποτελεσμάτων.

Για την επιτάχυνση λοιπόν αυτού του αλγορίθμου, οι Silva και λοιποί ξεκίνησαν με την ανάλυση του. Μετά την εκτέλεση δοκιμών χρησιμοποιώντας

διαφορετικούς συνδυασμούς σετ δεδομένων, το συμπέρασμα στο οποίο κατέληξαν ήταν ότι από όλες τα βήματα του αλγορίθμου PIC, το πιο χρονοβόρο κατά μακράν, ήταν αυτό του υπολογισμού του πίνακα ομοιότητας. Ο υπολογισμός του πίνακα ομοιότητας αποτελεί κατά μέσο όρο το $88,61\%$ του χρόνου εκτέλεσης της συσταδοποίησης. Οπότε η επιτάχυνση αυτού του βήματος τέθηκε ως προτεραιότητα. Η παραλληλοποίηση του υπολογισμού του πίνακα ομοιότητας έγινε με παρόμοιο τρόπο όπως είχε περιγραφεί παραπάνω σε αντίστοιχα βήματα υπολογισμού ομοιότητας/απόστασης σε άλλους αλγορίθμους. Το σετ δεδομένων χωρίζεται σε μικρότερα κομμάτια που μπορούν να μεταφερθούν στη κάρτα γραφικών με κάθε νήμα στη κάρτα γραφικών να είναι υπεύθυνο για ένα πλήθος στοιχείων του σετ δεδομένων n/p όπου n είναι ο αριθμός των στοιχείων και p το πλήθος των νημάτων. Στη συνέχεια υπολογίζεται το άθροισμα των γραμμών του πίνακα ομοιότητας για να βρεθεί ο degree matrix με το κάθε νήμα επίσης να είναι υπεύθυνο για την άθροιση ενός πλήθους στοιχείων του n/p. Το τρίτο βήμα είναι ο υπολογισμός του κανονικοποιημένου πίνακα ομοιότητας όπου είναι στην ουσία γινόμενο διανύσματος με πίνακα και είναι γνωστή διαδικασία παραλληλοποίησης. Τέλος, υπολογίζεται το γινόμενο όπως έχει περιγραφεί στη σειριακή εκδοχή μέχρι η συνθήκη τέλους να γίνει αληθής. Στη συγκεκριμένη εκδοχή του αλγορίθμου, η συνθήκη τέλους είναι η μεταβολή του ιδιοδιανύσματος να είναι μικρότερη από κάποιο κατώφλι που ορίζει ο χρήστης. Η μεταβολή ιδιοδιανύσματος ορίζεται ως η διαφορά δύο ιδιοδιανύσματων $\delta^{t+1} = v^{t+1}, v^t$. Αν και οι βελτιώσεις που παρουσιάστηκαν παραπάνω απέδωσαν μια επιτάχυνση της τάξης των 188.17 φορών έναντι του αλγορίθμου PIC, η επιτάχυνση αυτή κρίθηκε ανεπαρκής καθώς αποφάνθηκε πως η αξιοποίηση της μνήμης δεν ήταν η βέλτιστη δυνατή. Αυτό λύθηκε κάνοντας χρήση της διαμοιραζόμενης μνήμης της κάρτας γραφικών έναντι της global. Αυτό είχε ως αποτέλεσμα την επιπλέον

επιτάχυνση του αλγορίθμου GPIC έναντι του αλγορίθμου PIC. Μετά από έναν αριθμό πειραμάτων χρησιμοποιώντας διάφορους συνδυασμούς σετ δεδομένων, η επιτάχυνση μετρήθηκε σε κατά μέσο όρο 685,82 φορές πιο γρήγορη εκτέλεση του προγράμματος. Οι συγγραφείς επίσης αναφέρουν ότι τα αποτελέσματα του αλγορίθμου GPIC έχουν την ίδια ακρίβεια με αυτά του αλγορίθμου PIC.

Παρακάτω, παρουσιάζεται η υλοποίηση του αλγορίθμου GPIC

```
float* cuda_pic(float **objects, int numCoords, int numObjs) {

  float *d_objects;
  float *d_line;
  float *d_reductionResult;
  float *d_D;

  float *h_reductionResult;
  float *v;
  float **h_W;
  float *h_D;

  float volume;

  int b, t, smemSize;

  // Definig sizes to use on allocations and memory copies
  unsigned long long int s1 = ((unsigned long long)numObjs)*((unsigned long
      long)numCoords)*(sizeof(float));
  unsigned long long int s2 = ((unsigned long long)numObjs)*(sizeof(float));
  unsigned long long int s3 = ((unsigned long long)numObjs)*(sizeof(float *));

  // Allocate device memory
  checkCuda(cudaMalloc(&d_objects, s1));
  checkCuda(cudaMalloc(&d_line, s2));
  checkCuda(cudaMalloc(&d_reductionResult,s2));
  checkCuda(cudaMalloc(&d_D, s2));

  // Allocate host memory
```

```
h_reductionResult = (float *)malloc(s2);
if (h_reductionResult == NULL) {
   err("\nMemory allocation failed in h_reductionResult\n\n");
}
v = (float *)malloc(s2);
if (v == NULL) {
   err("\nMemory allocation failed in v\n\n");
}
h_W = (float **)malloc(s3);
if (h_W == NULL) {
   err("\nMemory allocation failed in h_W\n\n");
}
for (int i = 0; i < numObjs; i++) {
   h_W[i] = (float *)malloc(s2);
   if (h_W[i] == NULL) {
      err("\nMemory allocation failed in h_W[%d]\n\n", i);
   }
}
h_D = (float *)malloc(s2);
if (h_D == NULL) {
   err("\nMemory allocation failed in h_D\n\n");
}

// copy host memory to device
checkCuda(cudaMemcpy(d_objects, objects[0], s1, cudaMemcpyHostToDevice));

// Setting Kernels Configuration
if (numObjs <= 1024) {
   b = 1;
   t = numObjs;
}
else {
   b = ceil(numObjs / 1024.00);
   t = 1024;
}
dim3 blocks(b);
dim3 threads(t);

// Setting the shared memory size
```

```
smemSize = (threads.x <= 32) ? 2 * threads.x * sizeof(float) : threads.x *
    sizeof(float);


for (int i = 0; i < numObjs; i++) {

   affinity_matrix_CUDA << <blocks, threads >> > (d_objects, d_line, numObjs,
       numCoords, i);
   cudaDeviceSynchronize();
   checkLastCudaError();

   reduction_CUDA << < blocks, threads, smemSize >> >(d_line, d_reductionResult,
       numObjs);
   cudaDeviceSynchronize();
   checkLastCudaError();

   checkCuda(cudaMemcpy(h_reductionResult, d_reductionResult, s2,
       cudaMemcpyDeviceToHost));

   volume = 0.0;
   for (int j = 0; j < blocks.x; j++) {
      volume = volume + h_reductionResult[j];
   }

   h_D[i] = volume;

   normalize_v_CUDA << < blocks, threads >> >(d_line, volume, numObjs);
   cudaDeviceSynchronize();
   checkLastCudaError();

   checkCuda(cudaMemcpy(h_W[i], d_line, s2, cudaMemcpyDeviceToHost));
}

checkCuda(cudaMemcpy(d_D, h_D, s2, cudaMemcpyHostToDevice));


reduction_CUDA << < blocks, threads, smemSize >> >(d_D, d_reductionResult, numObjs);
cudaDeviceSynchronize();
checkLastCudaError();
```

```
checkCuda(cudaMemcpy(h_reductionResult, d_reductionResult, s2,
    cudaMemcpyDeviceToHost));

volume = 0.0;
for (int i = 0; i < blocks.x; i++) {
   volume = volume + h_reductionResult[i];
}


normalize_v_CUDA << < blocks, threads >> >(d_D, volume, numObjs);
cudaDeviceSynchronize();
checkLastCudaError();


for (int i = 0; i < 3; i++) {

   for (int j = 0; j < numObjs; j++) {

      checkCuda(cudaMemcpy(d_line, h_W[j], s2, cudaMemcpyHostToDevice));

      mult_CUDA << <blocks, threads >> > (d_line, d_D, numObjs);
      cudaDeviceSynchronize();
      checkLastCudaError();

      reduction_CUDA << < blocks, threads, smemSize >> >(d_line, d_reductionResult,
          numObjs);
      cudaDeviceSynchronize();
      checkLastCudaError();

      checkCuda(cudaMemcpy(h_reductionResult, d_reductionResult, s2,
          cudaMemcpyDeviceToHost));

      volume = 0.0;
      for (int k = 0; k < blocks.x; k++) {
         volume = volume + h_reductionResult[k];
      }

      h_D[j] = volume;
```

```
   }

   checkCuda(cudaMemcpy(d_D, h_D, s2, cudaMemcpyHostToDevice));

   reduction_CUDA << < blocks, threads, smemSize >> >(d_D, d_reductionResult,
       numObjs);
   cudaDeviceSynchronize();
   checkLastCudaError();

   checkCuda(cudaMemcpy(h_reductionResult, d_reductionResult, s2,
       cudaMemcpyDeviceToHost));

   volume = 0.0;
   for (int j = 0; j < blocks.x; j++) {
      volume = volume + h_reductionResult[j];
   }

   normalize_v_CUDA << < blocks, threads >> >(d_D, volume, numObjs);
   cudaDeviceSynchronize();
   checkLastCudaError();
}

checkCuda(cudaMemcpy(v, d_D, s2, cudaMemcpyDeviceToHost));

// Freeing GPU data
checkCuda(cudaFree(d_objects));
checkCuda(cudaFree(d_reductionResult));
checkCuda(cudaFree(d_line));
checkCuda(cudaFree(d_D));

// Freeing Host data
free(h_reductionResult);
for (int i = 0; i < 2; i++) {
   free(h_W[i]);
}
free(h_D);

checkCuda(cudaDeviceReset());
```

```
  return v;
}
```

Η εκτέλεση του παραπάνω κώδικα ξεκινάει με τα βήματα προετοιμασίας των δεδομένων και της μνήμης που θα χρησιμοποιηθούν για την εκτέλεση του αλγόριθμου GPIC. Η προετοιμασία ξεκινάει με τη δήλωση των απαραίτητων μεταβλητών τόσο στη μεριά του host (κεντρική μονάδα επεξεργασίας) όσο και στη μεριά του device (κάρτα γραφικών) αλλά και τη δέσμευση του απαιτούμενου χώρου στη μνήμη. Με την επιτυχή δέσμευση μνήμης, τα δεδομένα αντιγράφονται από τη μνήμη του host στη μνήμη του device και ορίζονται τα μεγέθη των grids και των blocks ανάλογα με το μέγεθος των δεδομένων. Η διαδικασία προετοιμασία τελειώνει με τη δήλωση της διαμοιραζόμενης μνήμης. Το επόμενο βήμα είναι αυτό του υπολογισμού του πίνακα ομοιότητας, του degree matrix και του κανονικοποιημένου πίνακα ομοιότητας το οποίο γίνεται με την επαναληπτική κλήση των affinity_matrix_CUDA, reduction_CUDA, normalize_v_CUDA αντίστοιχα μέχρι να επεξεργαστούν όλα τα δεδομένα. Με το πέρας του υπολογισμού τους, ο κώδικας μπαίνει σε ένα ακόμα βρόχο όπου υπολογίζεται παράλληλα το γινόμενο διανύσματος με πίνακα όπως έχει εξηγηθεί παραπάνω κάνοντας χρήση του αλγορίθμου mult_CUDA. Με την ολοκλήρωση του βρόχου αποδεσμεύεται η μνήμης που είχε χρησιμοποιηθεί.

Κεφάλαιο 5

Δοκιμαστικές Εκτελέσεις και Αποτελέσματα

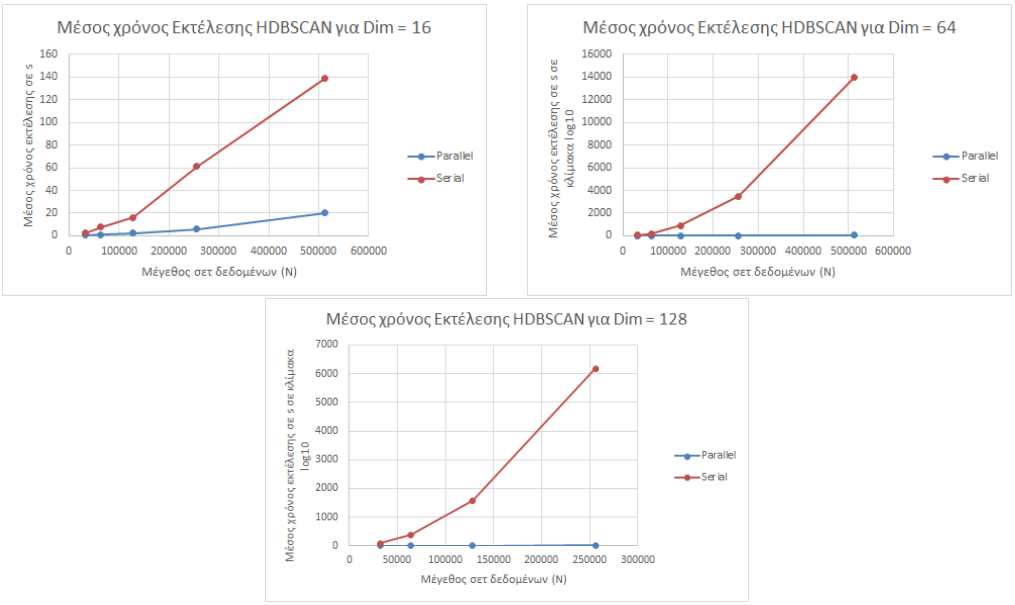
Έχοντας παρουσιάσει των πηγαίο κώδικα στο προηγούμενο κεφάλαιο, στο παρόν, παρουσιάζονται τα αποτελέσματα των δοκιμαστικών εκτελέσεων που πραγματοποιήθηκαν για του αλγορίθμους HDBSCAN και K-means.

Οι εκτελέσεις του κώδικα τα αποτελέσματα των οποίων θα παρουσιαστούν στη συνέχεια, πραγματοποιήθηκαν σε ένα σύστημα με κεντρικό επεξεργαστή AMD Ryzen 3700X (8 πυρήνες/16 νήματα) και κάρτα γραφικών NVIDIA Titan Rtx. Η κάρτα γραφικών Titan Rtx έχει 4608 πυρήνες CUDA και 576 πυρήνες Tensor.

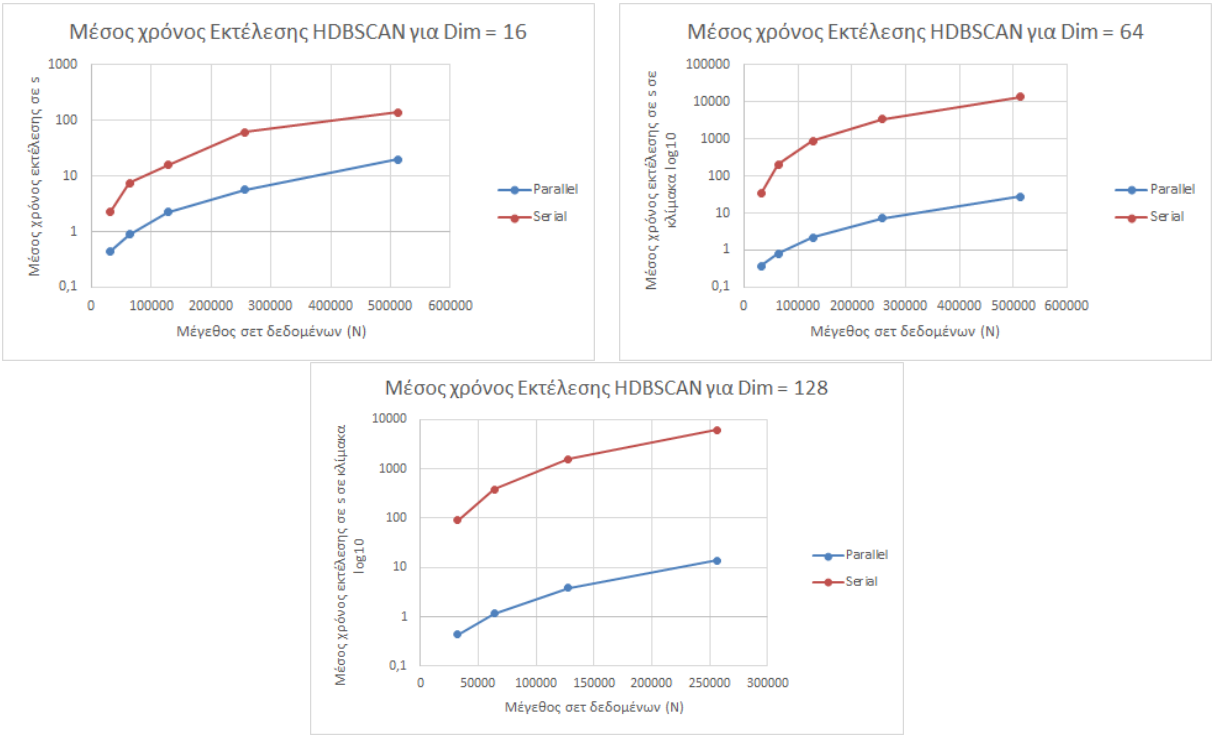
Για τον αλγόριθμο HDBSCAN εκτελέστηκαν δοκιμές σε ένα τεχνητό σετ δεδομένων. Το σετ δεδομένων που δημιουργήθηκε αποτελούταν από 32000, 64000, 128000, 256000, 512000 μέλη, τα οποία είχαν 16, 64, 128 χαρακτηριστικά. Συνολικά εκτελέστηκαν 3 δοκιμές για κάθε συνδυασμό δεδομένων-χαρακτηριστικών (45) και ως χρόνος εκτέλεσης στα παρακάτω αποτελέσματα, χρησιμοποιείται ο μέσος όρος των εκτελέσεων. Αξίζει να σημειωθεί ότι στα αποτελέσματα που ακολουθούν, δεν συμπεριλαμβάνεται τα αποτελέσματα για τον συνδυασμό N (μέλη) = 512000 και χαρακτηριστικά (Dim) = 128 καθώς ο

χρόνος εκτέλεση για τη σειριακή εκδοχή ξεπέρασε τις 5 ώρες και δεν ήταν δυνατή η καταγραφή ακριβής τιμής.

5.1 Αποτελέσματα HDBSCAN

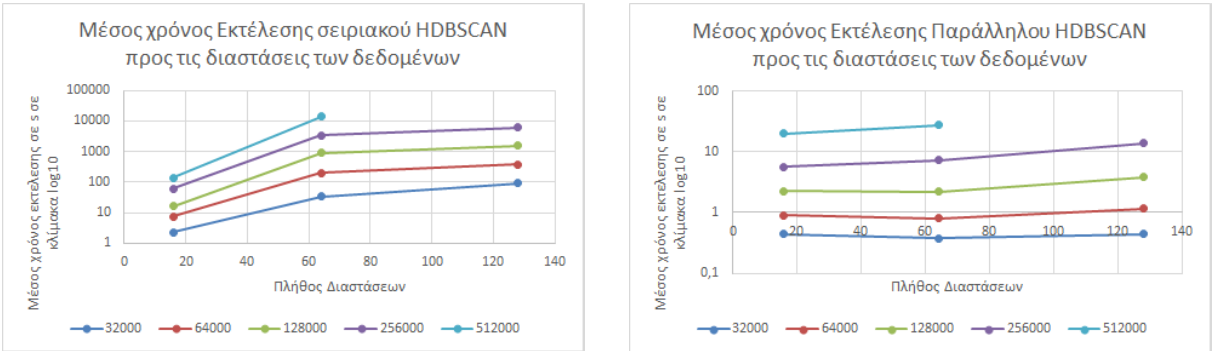


Σχήμα 5.1: Μέσος χρόνος Εκτέλεσης HDBSCAN για σταθερό αριθμό χαρακτηριστικών και μεταβαλλόμενο πλήθος δεδομένων



Σχήμα 5.2: Μέσος χρόνος Εκτέλεσης (κλίμακα $\log_{10}$) HDBSCAN για σταθερό αριθμό χαρακτηριστικών και μεταβαλλόμενο πλήθος δεδομένων

Ο αλγόριθμος HDBSCAN, όπως φαίνεται από τα γραφήματα που δίνονται παραπάνω, παρουσιάζει μεγάλη και απότομη αύξηση του μέσου χρόνου εκτέλεσης όσο αυξάνεται το πλήθος των δεδομένων. Η αύξηση αυτή μετριάζεται αρκετά στην παράλληλη υλοποίηση του αλγορίθμου



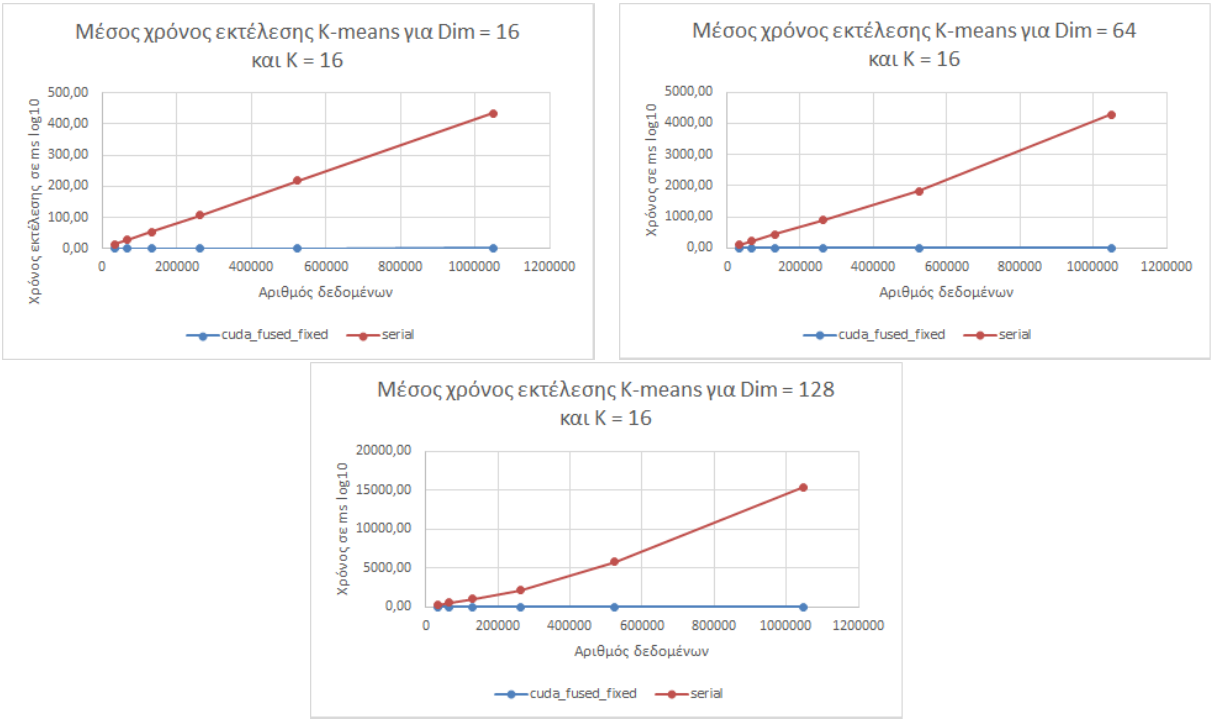
Σχήμα 5.3: Μέσος χρόνος Εκτέλεσης (κλίμακα $\log_{10}$) HDBSCAN για σταθερό αριθμό δεδομένων και μεταβαλλόμενο πλήθος χαρακτηριστικών

Η ίδια συμπεριφορά αλλά αρκετά πιο εμφανή παρουσιάζεται στο διάγραμμα

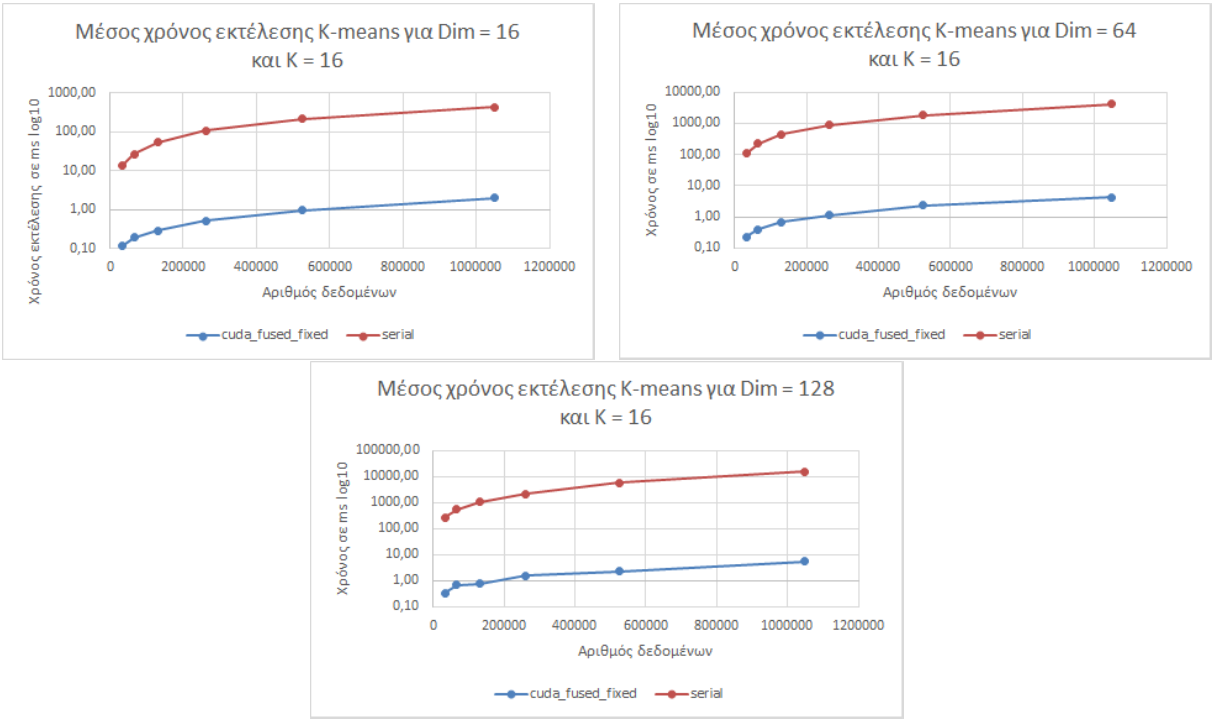
χρόνου προς αριθμός χαρακτηριστικών (διαστάσεων) όπου είναι προφανές ότι η παράλληλη εκδοχή του δεν επηρεάζεται σχεδόν καθόλου από την κλιμάκωση του αριθμού των χαρακτηριστικών ενώ η σειριακή επηρεάζεται σημαντικά.

5.2 Αποτελέσματα K-means

Για τον αλγόριθμο K-means εκτελέστηκαν δοκιμές σε ένα τεχνητό σετ δεδομένων. Το σετ δεδομένων που δημιουργήθηκε αποτελούταν από 32000, 64000, 128000, 256000, 512000, 1000000 μέλη, τα οποία είχαν 16, 64, 128 χαρακτηριστικά. Επίσης για κάθε συνδυασμό δεδομένων-χαρακτηριστικών εκτελέστηκαν δοκιμές με 5 διαφορετικές τιμές του K 16, 64, 256, 1024, 4096. Συνολικά εκτελέστηκαν 3 δοκιμές για κάθε συνδυασμό δεδομένων-χαρακτηριστικών(270) και ως χρόνος εκτέλεσης στα παρακάτω αποτελέσματα, χρησιμοποιείται ο μέσος όρος των εκτελέσεων. Αξίζει να σημειωθεί ότι ο μέσος χρόνος εκτέλεσης που αναγράφεται είναι για μία επανάληψη του αλγορίθμου.

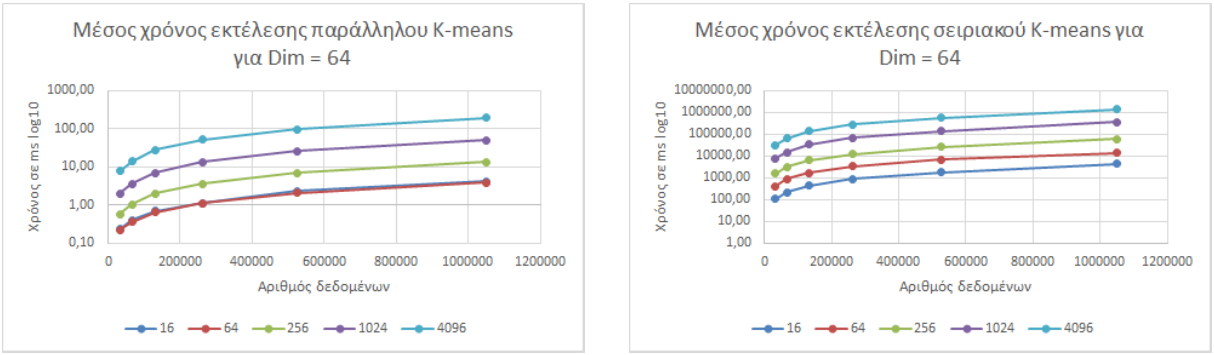


Σχήμα 5.4: Μέσος χρόνος Εκτέλεσης K-means για σταθερό αριθμό χαρακτηριστικών, σταθερό K και μεταβαλλόμενο πλήθος δεδομένων



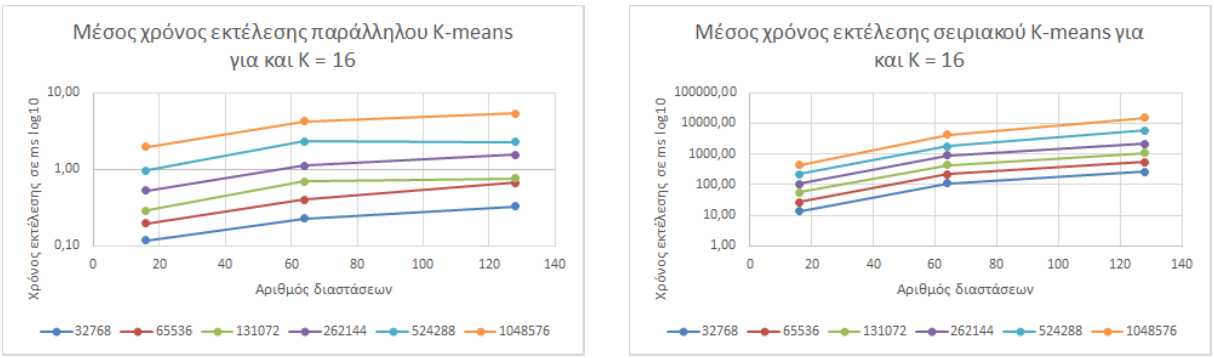
Σχήμα 5.5: Μέσος χρόνος Εκτέλεσης (κλίμακα $\log_{10}$) K-means για σταθερό αριθμό χαρακτηριστικών, σταθερό K και μεταβαλλόμενο πλήθος δεδομένων

Η συμπεριφορά του K-means όσο έχει να κάνει την αύξηση του μέσου χρόνου εκτέλεσης όσο αυξάνονται τα δεδομένα αλλά και ο αριθμός των συστάδων που δημιουργούνται, παραμένει ίδια. Με την παράλληλη εκδοχή που δοκιμάστηκε να επιτυγχάνει στην επιτάχυνση διατηρώντας τα χαρακτηριστικά του αλγορίθμου όσο έχει να κάνει με την κλιμάκωση σχετικά ίδια. Για αυτό το λόγο, τα γραφήματα που παρουσιάζονται είναι μόνο για σταθερό αριθμό συστάδων 16. Η παραπάνω παρατήρηση μπορεί να επαληθευθεί στο παρακάτω γράφημα.



Σχήμα 5.6: Μέσος χρόνος Εκτέλεσης (κλίμακα $\log_{10}$) K-means για σταθερό αριθμό χαρακτηριστικών και μεταβαλλόμενο πλήθος δεδομένων και K

Η μόνη διαφορά που παρουσιάζεται, έχει να κάνει με την αύξηση του μέσου χρόνου εκτέλεσης όσο αυξάνονται τα χαρακτηριστικά του σετ δεδομένων. Η παράλληλη υλοποίηση παρουσιάζει μικρότερη ”αύξηση” στο μέσο χρόνο εκτέλεσης συγκριτικά με τη σειριακή υλοποίηση.



Σχήμα 5.7: Μέσος χρόνος Εκτέλεσης (κλίμακα $\log_{10}$) K-means για σταθερό αριθμό χαρακτηριστικών και μεταβαλλόμενο πλήθος δεδομένων

5.3 Αποτελέσματα GPIC

Καθώς το μέγεθος της μνήμης που απαιτείται για την κατασκευή του πίνακα ομοιότητας όταν το σετ δεδομένων αποτελείται από 30000 στοιχεία είναι 16 GB και για 45000 είναι 36.5 GB δεν ήταν δυνατή η εκτέλεση δοκιμαστικών εκτελέσεων στο αυτό για τον αντίστοιχο αριθμό στοιχείων με τους άλλου αλγορίθμους.

Παρ’ όλα αυτά, έγινα δοκιμαστικές εκτελέσεις για ένα τεχνητό σετ δεδομένων. Το σετ δεδομένων που δημιουργήθηκε , αποτελούταν από 15000, 32000 και 45000 μέλη. Οι χρόνοι που χρειάστηκε ο GPIC για να ομαδοποιήσει τα δεδομένα, ήταν 2.9496, 9.8615, 18.3013 δευτερόλεπτα αντίστοιχα. Οι μετρήσεις αυτές συμφωνούν με τα αποτελέσματα των συγγραφέων του και το συμπέρασμα στο οποίο καταλήγουν είναι ότι ο GPIC επιτυγχάνει στη εκμετάλλευση των δυνατοτήτων της κάρτας γραφικών για την σημαντική επιτάχυνση του σειριακού αλγορίθμου, χωρίς να θυσιάσει την ακρίβεια των αποτελεσμάτων στην πορεία.

Κεφάλαιο 6

Συμπεράσματα

Βάσει την έρευνα που πραγματοποιήθηκε αλλά και βάσει τις εκτελέσεις του κώδικα, μπορούμε να καταλήξουμε στο συμπέρασμα ότι η επιτάχυνση των αλγορίθμων συσταδοποίησης μέσω παραλληλοποίησης τους καθιστά τόσο πιο ανθεκτικούς στο όλο αυξανόμενο όγκο δεδομένων όσο και στην αύξηση των διαστάσεων τους.

Επιπλέον σύμφωνα με την έρευνα που πραγματοποιήθηκε, συμπεραίνουμε ότι είναι σημαντική για την πλήρη εκμετάλλευση των δυνατοτήτων των αλγορίθμων, πέρα από την κατανόηση των χαρακτηριστικών τους, η σε βάθος κατανόηση της πλατφόρμας στην οποία γίνεται η παραλληλοποίηση (στα πλαίσια της εργασία, η κάρτα γραφικών με χρήση της CUDA).

Βιβλιογραφία

[1] Guilherme Andrade κ.ά. “G-DBSCAN: A GPU Accelerated Algorithm for Density-based Clustering”. Στο: *Procedia Computer Science* 18 (2013). 2013 International Conference on Computational Science, σσ. 369–378. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2013.05.200. URL: https://www.sciencedirect.com/science/article/pii/S1877050913003438.

[2] Christian Böhm κ.ά. “Density-Based Clustering Using Graphics Processors”. Στο: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. CIKM ’09. Hong Kong, China: Association for Computing Machinery, 2009, σσ. 661–670. ISBN: 9781605585123. DOI: 10.1145/1645953.1646038. URL: https://doi.org/10.1145/1645953.1646038.

[3] Ricardo J. G. B. Campello, Davoud Moulavi και Joerg Sander. “Density-Based Clustering Based on Hierarchical Density Estimates”. Στο: *Advances in Knowledge Discovery and Data Mining*. Επιμέλεια υπό Jian Pei κ.ά. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, σσ. 160–172. ISBN: 978-3-642-37456-2.

[4] S. Cuomo κ.ά. “A GPU-accelerated parallel K-means algorithm”. Στο: *Computers & Electrical Engineering* 75 (2019), σσ. 262–274. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2017.12.002. URL: https://www.sciencedirect.com/science/article/pii/S0045790617327994.

[5] Zineb Dafir, Yasmine Lamari και Said Slaoui. “A survey on parallel clustering algorithms for Big Data”. Στο: *Artificial Intelligence Review* 54 (Απρ. 2021). DOI: 10.1007/s10462-020-09918-2.

[6] Martin Kruliš και Miroslav Kratochvíl. “Detailed Analysis and Optimization of CUDA K-Means Algorithm”. Στο: *49th International Conference on Parallel Processing - ICPP*. ICPP ’20. Edmonton, AB, Canada: Association for Computing Machinery, 2020. ISBN: 9781450388160. DOI: 10.1145/3404397.3404426. URL: https://doi.org/10.1145/3404397.3404426.

[7] Gustavo Rodrigues Lacerda Silva κ.ά. “CUDA-Based Parallelization of Power Iteration Clustering for Large Datasets”. Στο: *IEEE Access* 5 (2017), σσ. 27263–27271. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2765380.

[8] You Li κ.ά. “Speeding up k-Means algorithm by GPUs”. Στο: *Journal of Computer and System Sciences* 79.2 (2013). 10th IEEE International Conference on Computer and Information Technology, 2010, σσ. 216–229. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2012.05.004. URL: https://www.sciencedirect.com/science/article/pii/S0022000012000992.

[9] Frank Lin και William W. Cohen. “Power Iteration Clustering”. Στο: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, σσ. 655–662. ISBN: 9781605589077.

[10] Woong-Kee Loh και Hwanjo Yu. “Fast density-based clustering through dataset partition using graphics processing units”. Στο: *Information Sciences* 308 (2015), σσ. 94–112. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2014.10.023. URL: https://www.sciencedirect.com/science/article/pii/S0020025514010056.

[11] Leland McInnes και John Healy. “Accelerated Hierarchical Density Based Clustering”. Στο: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, Νοέ. 2017. DOI: 10.1109/icdmw.2017.12. URL: https://doi.org/10.1109%2Ficdmw.2017.12.

[12] Leland McInnes, John Healy και Steve Astels. *How HDBSCAN Works*. https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html. 2016.

[13] NVIDIA. *CUDA C++ Programming Guide*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. 2023.

[14] Rui Xu και D. Wunsch. “Survey of clustering algorithms”. Στο: *IEEE Transactions on Neural Networks* 16.3 (2005), σσ. 645–678. DOI: 10.1109/TNN.2005.845141.

[15] Mario Zechner και Michael Granitzer. “Accelerating K-Means on the Graphics Processor via CUDA”. Στο: *2009 First International Conference on Intensive Applications and Services*. 2009, σσ. 7–15. DOI: 10.1109/INTENSIVE.2009.19.