

PROJECT

Πολυδιάστατες Δομές Δεδομένων



ΔΗΜΗΤΡΗΣ ΚΑΤΣΑΝΟΣ AM: 1093384

ΑΝΔΡΕΑΣ ΖΑΦΕΙΡΟΠΟΥΛΟΣ AM: 1093361

ΘΑΛΗΣ-ΧΡΥΣΟΣΤΟΜΟΣ ΤΣΙΩΝΑΣ AM: 1097467

ΙΩΑΝΝΗΣ ΓΡΗΓΟΡΟΠΟΥΛΟΣ AM: 1097444

Ασχοληθήκαμε με το Project 1 στο οποίο υλοποιήσαμε ένα σύστημα αποδοτικής αναζήτησης ταινιών, το οποίο υποστηρίζει πολυδιάστατα αριθμητικά ερωτήματα και αναζήτηση ομοιότητας σε κατηγορικά χαρακτηριστικά. Στόχος μας ήταν η μείωση του χρόνου αναζήτησης μέσω κατάλληλων δομών ευρετηρίασης (Indexes) και τεχνικών hashing, σε σύγκριση με απλή σειριακή αναζήτηση, η οποία θα ήταν πολύ αργή λόγω του μεγάλου όγκου δεδομένων. Επίσης δημιουργήσαμε και ένα GUI για την καλύτερη αλληλεπίδραση μας με το Project.

To GUI

Movies - Trees + LSH

Filters

Year min: 2000

Year max: 2020

Popularity min: 8.0

Popularity max: 12.0

Vote min: 3.0

Vote max: 8.0

Runtime min: 30

Runtime max: 180

Countries: US,GB

Language: en

Run query (pandas only)

Run index + LSH performance

Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: KD-Tree

Genre:

Top-N: 5

Run Tree + LSH search

Ready.

Movies

title	year	popularity	vote
-------	------	------------	------

Numeric index performance

Index	Build_s	Numeric_s	Candidates
-------	---------	-----------	------------

Schemes: (Index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
--------	------------	--------	------	-----	-------------	-------------

Dataset και Προεπεξεργασία

Το Dataset που κατεβάσαμε από το link της αναφοράς περιλαμβάνει έναν μεγάλο όγκο από πληροφορίες για ταινίες, όπως έτος, popularity, μέσο όρο βαθμολογίας, διάρκεια, χώρες παραγωγής, γλώσσες παραγωγής, genres κ.α. Η φόρτωση και προεπεξεργασία των δεδομένων υλοποιείται στο αρχείο main.py μέσω των συναρτήσεων load_dataset() και preprocess_dataset(). Σε αυτό το στάδιο γίνεται καθαρισμός των δεδομένων, μετατροπή τύπων και εξαγωγή δομών που διευκολύνουν τις επόμενες φάσεις (π.χ. μετατροπή genres σε σύνολα). Στην συνέχεια το βασικό dataframe δημιουργείται με την build_base_pool() το οποίο χρησιμοποιείται τόσο από το main.py όσο και από το GUI (gui_tk.py).

Δομές Ευρετηρίασης (Indexes)

Για την επιτάχυνση των numeric range queries υλοποιήθηκαν οι τέσσερις δομές ευρετηρίασης που μας ζητήθηκαν και από την εκφώνηση. Συγκεκριμένα KD-Tree (kd_tree.py), Quad_tree (quad_tree.py), Range-Tree (range_tree.py) και R-Tree (r_tree.py). Κάθε δομή χρησιμοποιείται για να φιλτράρει γρήγορα τα δεδομένα βάσει αριθμητικών κριτηρίων και να επιστρέψει ένα σύνολο υποψήφιων αποτελεσμάτων (candidates).

Παρότι το ερώτημα είναι πολυδιάστατο (έως 5 αριθμητικές διαστάσεις), κάθε δομή δεδομένων κάνει index σε διαφορετικό αριθμό διαστάσεων. Το KD-Tree υποστηρίζει queries σε περισσότερες διαστάσεις, ενώ το Quad-Tree και το R-Tree χρησιμοποιούνται σε 2 διαστάσεις και το Range-Tree σε 1 διάσταση. Τα υπόλοιπα φίλτρα εφαρμόζονται στη συνέχεια πάνω στα candidates, ώστε το τελικό αποτέλεσμα να παραμένει σωστό και η σύγκριση των σχημάτων Index + LSH να είναι δίκαιη.

Candidates και Φιλτράρισμα

Στο GUI, το φιλτράρισμα εκτελείται αρχικά με pandas, ώστε να παραχθεί το βασικό αποτέλεσμα και να υποστηριχθεί η επιλογή genres. Στην συνέχεια, τα indexes χρησιμοποιούνται για αριθμητικό φιλτράρισμα και παραγωγή candidates, με σκοπό τη σύγκριση της απόδοσης τους.

Αναζήτηση ομοιότητας με LSH

Για την αναζήτηση ομοιότητας βάση genres χρησιμοποιείται LSH, υλοποιημένο στο lsh_text.py. Η μέθοδος βασίζεται σε MinHash signatures και επιτρέπει την αποδοτική εύρεση παρόμοιων ταινιών χωρίς πλήρη σύγκριση όλων των στοιχείων.

Συνδυαστικά Σχήματα και Αξιολόγηση

Το σύστημα αξιολογείται μέσω συνδυαστικών σχημάτων της μορφής Index + LSH. Η πειραματική αξιολόγηση υλοποιείται με την συνάρτηση `evaluate_indexes()` στο `main.py` και παρουσιάζει χρόνους `build`, χρόνους `query`, πλήθος `candidates` και τελικά αποτελέσματα.

kNN

Επιπλέον υλοποιήθηκαν kNN queries στο KD-Tree στο αρχείο `kd_tree.py`. Τα kNN queries αξιολογούνται μέσω του `main.py`, συγκρίνοντας τα αποτελέσματα και τον χρόνο εκτέλεσης με μια απλή προσέγγιση που υπολογίζει τις αποστάσεις από όλα τα σημεία του dataset (brute-force)

```
# kNN
def knn_query(
    self,
    query_point: Sequence[float],
    k: int = 5,
    return_distances: bool = False,
) -> List[int] | List[Tuple[int, float]]:
    """Επιστρέφει τους k κοντινότερους γείτονες του query_point (Euclidean).
    return_distances=True επιστρέφει (index, distance), αλλιώς μόνο indices."""
    if self.root is None or self.k == 0 or k <= 0:
        return []

    if len(query_point) != self.k:
        raise ValueError("query_point must have length equal to k dimensions")

    k = min(k, self.n_points)

    # max-heap me (-dist sq, index) gia na kratame toys k kalyterous
    best: List[Tuple[float, int]] = []
    self._knn_search(self.root, query_point, k, best)

    # taksinomisi apo pio kontino se mio makrino
    best_sorted = sorted((-d2, idx) for d2, idx in best, key=lambda x: x[0])

    if return_distances:
        return [(idx, dist ** 0.5) for dist, idx in best_sorted]
    return [idx for dist, idx in best_sorted]

def _knn_search(
    self,
    node: Optional[KDNode],
    query_point: Sequence[float],
    k: int,
    best: List[Tuple[float, int]],
) -> None:
    """Αναδρομική αναζήτηση kNN με pruning."""
    if node is None:
        return

    # apostasi tou node.point apo query
    d2 = 0.0
    p = node.point
    for dim in range(self.k):
        diff = p[dim] - query_point[dim]
        d2 += diff * diff

    # enimerosi heap
    if len(best) < k:
        heapq.heappush(best, (-d2, node.index))
    else:
        # en brillame kalytero apo ton xeirotero, antikatastasi
        worst_neg_d2, _ = best[0]
        if -d2 > worst_neg_d2:
            heapq.heapreplace(best, (-d2, node.index))

    axis = node.axis
    diff_axis = query_point[axis] - p[axis]

    # prwta pame sto pio kontino ypodentro
    near = node.left if diff_axis < 0 else node.right
    far = node.right if diff_axis < 0 else node.left
    self._knn_search(near, query_point, k, best)

    # pruning
    if len(best) < k:
        self._knn_search(far, query_point, k, best)
    else:
        worst_d2 = -best[0][0]
        if diff_axis * diff_axis <= worst_d2:
            self._knn_search(far, query_point, k, best)
```

GUI

Το GUI επιτρέπει την εκτέλεση **baseline queries**, **index-based queries** και **πειραματικής αξιολόγησης**. Χρησιμοποιείται για επίδειξη και σύγκριση απόδοσης των μεθόδων, χωρίς να επηρεάζει την βασική λογική του συστήματος.

Ακολουθεί από κάτω ένα παράδειγμα με την λειτουργικότητα του GUI:

1.Αρχική Κατάσταση GUI

The screenshot shows a GUI window titled "Movies - Trees + LSH". It contains several sections:

- Filters:** A grid of dropdown menus for "Year min" (2000), "Year max" (2020), "Popularity min" (8.0), "Popularity max" (12.0), "Vote min" (3.0), "Vote max" (8.0), "Runtime min" (30), "Runtime max" (180), "Countries" (US,GB), and "Language" (en). Below these are three buttons: "Run query (pandas only)", "Run index + LSH performance", and "Run eval_lsh.py".
- Tree-based LSH search (results shown in table):** Includes a dropdown for "Index" (KD-Tree), a dropdown for "Genre", and a "Top-N: 5" input field. A "Run Tree + LSH search" button is at the bottom.
- Ready.** A status indicator.
- Movies:** A table with columns "title", "year", "popularity", and "vote". The table is currently empty.
- Numeric index performance:** A table with columns "Index", "Build_s", "Numeric_s", and "Candidates". The table is currently empty.
- Schemes: (Index + LSH):** A table with columns "Scheme", "IdxBuild_s", "NumQ_s", "Cand", "Res", "LSH_build_s", and "LSH_query_s". The table is currently empty.

Ορίζουμε τα αριθμητικά φίλτρα του ερωτήματος (έτος, popularity, βαθμολογία, διάρκεια, χώρα και γλώσσα) πριν την εκτέλεση της αναζήτησης

2. Εκτέλεση pandas-only query

Movies - Trees + LSH

Filters

Year min: 2000

Year max: 2020

Popularity min: 8.0

Popularity max: 12.0

Vote min: 3.0

Vote max: 8.0

Runtime min: 30

Runtime max: 180

Countries: US,GB

Language: en

Run query (pandas only)

Run index + LSH performance

Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: KD-Tree

Genre:

Top-N: 5

Run Tree + LSH search

Loading dataset...

Movies

title	year	popularity	vote
-------	------	------------	------

Numeric index performance

Index	Build_s	Numeric_s	Candidates
-------	---------	-----------	------------

Schemes: (Index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
--------	------------	--------	------	-----	-------------	-------------

Εκτελούμε αναζήτηση με χρήση pandas, ώστε να εφαρμοστούν όλα τα φίλτρα και να παραχθεί το βασικό αποτέλεσμα (ground truth).

3.Αποτελέσματα pandas-only

Movies - Trees + LSH

Filters

Year min: 2000

Year max: 2020

Popularity min: 8.0

Popularity max: 12.0

Vote min: 3.0

Vote max: 8.0

Runtime min: 30

Runtime max: 180

Countries: US,GB

Language: en

Run query (pandas only)

Run index + LSH performance

Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: KD-Tree

Genre: Action

Top-N: 5

Run Tree + LSH search

Query done. Found 388 movies.

Movies

title	year	popularity	vote
The Emperor's New Groove	2000	10.97	7.6
Scary Movie 2	2001	9.61	5.8
Final Destination	2000	10.99	6.6
Scary Movie	2000	10.63	6.4
The Mummy Returns	2001	8.59	6.4
Mickey's Magical Christmas: Snowed in at the House of Mouse	2001	8.98	6.9
Child Star: The Shirley Temple Story	2001	9.19	6.7
Legally Blonde	2001	8.64	6.8
American Psycho	2000	8.38	7.4
What Lies Beneath	2000	9.64	6.4

Numeric index performance

Index	Build_s	Numeric_s	Candidates
-------	---------	-----------	------------

Schemes: (Index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
--------	------------	--------	------	-----	-------------	-------------

Εμφανίζουμε τα αποτελέσματα της baseline αναζήτησης χωρίς χρήση index, τα οποία χρησιμοποιούμε ως σημείο αναφοράς και για την επιλογή genre.

4. Tree + LSH search

Filters

Year min: 2000 Year max: 2020
Popularity min: 8.0 Popularity max: 12.0
Vote min: 3.0 Vote max: 8.0
Runtime min: 30 Runtime max: 180
Countries: US,GB Language: en

Run query (pandas only) Run index + LSH performance
Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: KD-Tree Genre: Action Top-N: 5
Run Tree + LSH search

Tree + LSH search done (KD-Tree), Genre='Action', returned 5 movies.

Movies

title	year	popularity	vote
Bastille Day	2016	9.23	6.3
Born to Raise Hell	2010	8.55	4.7
Fast and Fierce: Death Race	2020	11.03	5.6
Creed	2015	9.26	7.4
Never Back Down 2: The Beatdown	2011	9.85	6.5

Numeric index performance

Index	Build_s	Numeric_s	Candidates
-------	---------	-----------	------------

Schemes: (index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
--------	------------	--------	------	-----	-------------	-------------

Filters

Year min: 2000 Year max: 2020
Popularity min: 8.0 Popularity max: 12.0
Vote min: 3.0 Vote max: 8.0
Runtime min: 30 Runtime max: 180
Countries: US,GB Language: en

Run query (pandas only) Run index + LSH performance
Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: Range-Tree Genre: Action Top-N: 5
Run Tree + LSH search

Tree + LSH search done (Range-Tree), Genre='Action', returned 5 movies.

Movies

title	year	popularity	vote
Bastille Day	2016	9.23	6.3
Born to Raise Hell	2010	8.55	4.7
Creed II	2018	8.89	7.0
Never Back Down 2: The Beatdown	2011	9.85	6.5
Puncture Wounds	2014	8.40	4.5

Numeric index performance

Index	Build_s	Numeric_s	Candidates
-------	---------	-----------	------------

Schemes: (index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
--------	------------	--------	------	-----	-------------	-------------

Εκτελούμε αναζήτηση χρησιμοποιώντας διαφορετικά είδη Tree για το αριθμητικό φιλτράρισμα και LSH για αναζήτηση ομοιότητας με βάση το επιλεγμένο genre. Στα δυο αυτά screenshots δείχνουμε ότι με 2 διαφορετικά αλλά στο ίδιο genre, επιστρέφει διαφορετικά αποτελέσματα μεταξύ τους. Κάτι το οποίο είναι λογικό, καθώς κάθε index επιστρέφει διαφορετικό σύνολο υποψήφιων αποτελεσμάτων πάνω στο οποίο εφαρμόζεται το LSH.

5. Εκτέλεση Index + LSH performance

Movies - Trees + LSH

Filters

Year min: 2000

Year max: 2020

Popularity min: 8.0

Popularity max: 12.0

Vote min: 3.0

Vote max: 8.0

Runtime min: 30

Runtime max: 180

Countries: US,GB

Language: en

Run query (pandas only)

Run index + LSH performance

Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: Range-Tree

Genre: Action

Top-N: 5

Run Tree + LSH search

Running index + LSH performance...

Movies

title	year	popularity	vote
Bastille Day	2016	9.23	6.3
Born to Raise Hell	2010	8.55	4.7
Creed II	2018	8.89	7.0
Never Back Down 2: The Beatdown	2011	9.85	6.5
Puncture Wounds	2014	8.40	4.5

Numeric index performance

Index	Build_s	Numeric_s	Candidates
-------	---------	-----------	------------

Schemes: (Index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
--------	------------	--------	------	-----	-------------	-------------

Εκτελούμε την διαδικασία αξιολόγησης της απόδοσης για όλα τα indexes και τα συνδυαστικά σχήματα Index + LSH.

6. Numeric Index Performance Table και Schemes: (Index + LSH) Table

Movies - Trees + LSH

Filters

Year min: 2000

Year max: 2020

Popularity min: 8.0

Popularity max: 12.0

Vote min: 3.0

Vote max: 8.0

Runtime min: 30

Runtime max: 180

Countries: US,GB

Language: en

Run query (pandas only)

Run index + LSH performance

Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: Range-Tree

Genre: Action

Top-N: 5

Run Tree + LSH search

Index + LSH performance done.

Common LSH on GT (|GT|=388): build=0.0296s, query=0.000164s

Movies

	title	year	popularity	vote
Bastille Day		2016	9.23	6.3
Born to Raise Hell		2010	8.55	4.7
Creed II		2018	8.89	7.0
Never Back Down 2: The Beatdown		2011	9.85	6.5
Puncture Wounds		2014	8.40	4.5

Numeric index performance

Index	Build_s	Numeric_s	Candidates
KD-Tree	0.2991	0.0036	396
Quad-Tree	0.4292	0.0020	738
Range-Tree	0.0550	0.0110	22524
R-Tree	1.8849	0.0025	666

Schemes: (Index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
KD-Tree + LSH	0.2991	0.0036	396	388	0.0282	0.000090
Quad-Tree + LSH	0.4292	0.0020	738	388	0.0281	0.000170
Range-Tree + LSH	0.0550	0.0110	22524	364	0.0205	0.000079
R-Tree + LSH	1.8849	0.0025	666	376	0.0207	0.000071

Παρουσιάζουμε συγκριτικά την απόδοση των Indexes και των συνδυαστικών σχημάτων Index + LSH, ως προς τον χρόνο κατασκευής, τον χρόνο εκτέλεσης των αριθμητικών queries, τον αριθμό των υποψήφιων αποτελεσμάτων και τους χρόνους του LSH.

7. Εκτέλεση αξιολόγησης LSH (eval_lsh.py)

Filters

Year min: 2000 Year max: 2020

Popularity min: 8.0 Popularity max: 12.0

Vote min: 3.0 Vote max: 8.0

Runtime min: 30 Runtime max: 180

Countries: US,GB Language: en

Run query (pandas only) Run index + LSH performance

Run eval_lsh.py

Tree-based LSH search (results shown in table)

Index: Range-Tree

Genre: Action Top-N: 5

Run Tree + LSH search

Index + LSH performance done. Common LSH on GT [GT]=388; build=0.0296s, query=0.000164s

Movies

title	popularity	vote
Bastille Day	9.23	6.3
Born to Raise Hell	8.55	4.7
Creed II	8.89	7.0
Never Back Down 2: The Beatdown	9.85	6.5
Puncture Wounds	8.40	4.5

Done

eval_lsh.py finished successfully.

OK

Numeric index performance

Index	Build_s	Numeric_s	Candidates
KD-Tree	0.2991	0.0036	396
Quad-Tree	0.4292	0.0020	738
Range-Tree	0.0550	0.0110	22524
R-Tree	1.8849	0.0025	666

Schemes: (Index + LSH)

Scheme	IdxBuild_s	NumQ_s	Cand	Res	LSH_build_s	LSH_query_s
KD-Tree + LSH	0.2991	0.0036	396	388	0.0282	0.000090
Quad-Tree + LSH	0.4292	0.0020	738	388	0.0281	0.000170
Range-Tree + LSH	0.0550	0.0110	22524	364	0.0205	0.000079
R-Tree + LSH	1.8849	0.0025	666	376	0.0207	0.000071

Τέλος με το κουμπί "Run eval_lsh.py" εκτελούμε το συγκεκριμένο αρχείο, το οποίο εκτελεί ανεξάρτητη αξιολόγηση της λειτουργικότητάς του LSH και επιβεβαιώνει την σωστή εκτέλεση του.