УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Јаков Петровић

Београд 2020.

УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ИНФОРМАЦИОНИ СИСТЕМИ И ТЕХНОЛОГИЈЕ

ПОСЛОВНА ИНТЕЛИГЕНЦИЈА

ЗАВРШНИ РАД

АПРОКСИМАТИВНИ АЛОГОРИТАМ НАЈБЛИЖИХ СУСЕДА ЗА ВЕЛИКЕ ПОДАТКЕ

Јаков Петровић 2017/3029

Београд 2020.

**Сагласност чланова комисије за одбрану**

*Попуњавају чланови Комисје за одбрану:*

Комисија која је прегледала рад

кандидата Петровић (ГОРАН) ЈАКОВ

под насловом АПРОКСИМАТИВНИ АЛГОРИТАМ НАЈБЛИЖИХ СУСЕДА ЗА ВЕЛИКЕ ПОДАТКЕ и одобрила одбрану:

Ментор: др Милош Јовановић,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члан: др Милан Вукићевић,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члан: др Гордана Савић,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

БИОГРАФИЈА

Јаков Петровић је студент завршне године мастер студија на факултету органзационих наука. Као

Основне студије на истом факултету уписао је 2013. и завршио у року одбранивши рад под називом „Мрежна апликација за симулацију аукција у којима сви плаћају“

Основно и средњошколско образовање стекао је у Врњачкој Бањи. Као представник еколошких и друштвених организација учествовао на волонтерским пројектима у Србији, Норвешкој и Русији. Поред енглеског, течно говори и руски језик.

**Изјава о академској честитости**

*Попуњава кандидат:*

|  |  |
| --- | --- |
| Петровић Горан Јаков  Презиме, име једног родитеља, име | /3029  Број индекса |

**Студијски програм: Информациони системи и технологије**

**Модул:**Пословна интелигенција

**Аутор завршног рада под насловом:** **АПРОКСИМАТИВНИ АЛГОРИТАМ НАЈБЛИЖИХ СУСЕДА ЗА ВЕЛИКЕ ПОДАТКЕ**

Чија је израда одобрена на Седници Већа студијских програма мастер академских студија одржаној: 18. јуна 2018.

Потписивањем изјављујем:

* + Да је рад искључиво резултат мог сопственог истраживачког рада;
  + Да сам рад и мишљења других аутора које сам користио у овом раду назначио или цитирао у складу са Упутством;
  + Да су сви радови и мишљења других аутора наведени у списку литературе/референци који су саставни део овог рада и писани су у складу са Упутством;
  + Да сам довио све дозволе за коришћење ауторског дела који се у потпуносзи/целости уносе у предати рад и да сам то јасно навео;
  + Да сам свестан да је плагијат коришћење туђих радова у било ком облику (као цитата, парафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, веб сајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторским и сродним правима, Службени гласник Републике Србије, бр. 104/2009, 99/2011, 119/2012), као и других закона и одговарајућих аката Универзитета у Београду и Факултета организационих наука;
  + Да сам свестан да плагијат укључује и представљање, употребу и дустрибуирање рада предавача или других студената као сопствених;
  + Да сам свестан последица које код доказаног плагијата могу проузроковати на предати завшни мастер рад и мој статус;
  + Да је електронска верзија завршног рада идентична штампаном примерку и пристајем на његово објављивање под условима прописаним актима Универзитета и Факултета.

Београд, \_\_\_\_\_\_\_\_\_\_

Потпис студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_

АПСТРАКТ

Алгоритам најближих суседа припада наједноставнијим предиктивним алгоритмима машинског учења. Класификацију или регресију, алгоритам врши на основу скупа најближих познатих инстанци. Претрага најближих суседа (енг. *Nearest neighbour search*) је временски најкритичнија фаза алгоритма. Фаза претраге је и рачунски захтевна, па утиче на скалабилност алгоритма при раду са већим количинама података. На скалбилност не утиче само количина података, већ и број димензија које поседују. Како би се побољшале преформансе алгоритма, уводе се апроксимативне методе претраге суседа. Одабране апроксимативне технике биће представљене и анализиране у раду. Експерименталним путем тестираће се преформансе апроксимативних алгоритама над скуповима података под различитим параметрима и звести препоруке за њихово коришћење.

Кључне речи: апроксимативни к-НН, претрага најближих суседа, апроксимативна претрага најближих суседа, машинско учење

ABSTRACT

The algorithm of k-nearest neighbors is among the simplest predictive algorithms of machine learning. The algorithm performs classification and regression, based on a set of the nearest known instances. Nearest neighbor search is the most time-critical phase of the algorithm. The search phase is also very computationally demanding, so it affects the scalability of the algorithm when working with large amounts of data. Scalability is affected not only by the amount of data, but also by the number of dimensions they possess. Approximate neighbor search methods are introduced in order to improve the performance of the algorithm. Selected approximate techniques will be presented and analyzed in the paper. The performance of approximate algorithms over data sets under different parameters will be tested experimentally.

Key words: approximate kNN, nearest neighbour search, approximate nearest neighbour search, machine learning

АКРОНИМИ

|  |  |
| --- | --- |
| *2M-tree* | *2 means tree* |
| *HNSW* | *Hierarchial navigable small world graph* |
| *k-D tree* | *k-D* стабло (енг. *k-D tree*) |
| *k-NN* | к-најближих суседа (енг. *k-Nearest Neighbours*) |
| *mrpt* |  |
| *RP-tree* | *Random projection tree* |
| *SIFT* | *Scale Invariant Feature Transform* |
| *vp-Tree* | *Vintage point tree* |

ИЛУСТРАЦИЈЕ

[Илустрација 1: Апроксимативни алгоритам 10](#_Toc49559021)

[Илустрација 2: *k-D* подела простора 15](#_Toc49559022)

[Илустрација 3: *k-D* Бинарно стабло 15](#_Toc49559023)

[Илустрација 4: *HNSW* структура (Malkov, A, & Yashunin, 2018) 19](#_Toc49559024)

[Илустрација 5: Одзив и Време претраге SIFTSmall 27](#_Toc49559025)

[Илустрација 6: Одзив и конструкција индекса 28](#_Toc49559026)

[Илустрација 7: SIFT1M Почетни резултати 29](#_Toc49559027)

ТАБЕЛЕ

[Табела 1: Подаци 23](#_Toc49559096)

[Табела 2: Алгоритми 26](#_Toc49559097)

Садржај

[1. Увод 7](#_Toc49559187)

[2. Дефиниција проблема и преглед стања у области 9](#_Toc49559188)

[3. Апроксимативне технике претраге суседа 14](#_Toc49559189)

[3.1 Подела простора и стабла 14](#_Toc49559190)

[3.1.1 *k-D* стабла 15](#_Toc49559191)

[3.1.2 VP-tree 17](#_Toc49559192)

[3.1.3 Друге методе поделе 18](#_Toc49559193)

[3.2 Графовски приступ 18](#_Toc49559194)

[3.2.1 HNSW 18](#_Toc49559195)

[3.3 Приступи засновани на хеширању 21](#_Toc49559196)

[4. АНАЛИЗА МЕТОДА 22](#_Toc49559197)

[4.1 Скупови података, индикатори перформанси и библиотеке 23](#_Toc49559198)

[4.2 Експеримети над *SIFTsmall* и *SIFT1M* 27](#_Toc49559199)

[4.3 Експеримети над *MINST* 29](#_Toc49559200)

[4.4 Препоруке за одабир алгоритма 29](#_Toc49559201)

[5. Закључак 30](#_Toc49559202)

[6. Референце 31](#_Toc49559203)

[Прилози 33](#_Toc49559204)

# Увод

Рад се бави истраживањем апроксимативних техника за налажење најближих суседа над већим скуповима података. Претрага најближег суседа у метричком просотру, подразумева проналажење инстанце, којa има најмању удаљеност од инстанце у односу на коју се претрага врши.

Претрага најближих суседа је примењена у многим областима попут биомедицине, генетике, маркетинга, социјалних медија, машинског учења и оптимизације, користи се при обради слика и снимака, препознавању образаца и облика, те анализи и обради просторних и геолошких података. У већини случајева, записи из ових база се могу представити у векторском простору који има неколико десетина, па чак и хиљаду димензија. (Indyk & Motwani, 1998). У раду се са истим значењем користе појмови тачка у простору и вектор у простору.

Алгоритам к- најближох суседа (енг. *k-nearest neighbours*) је предиктивни алгоритам машинског учења, који непознату класу или вредност инстанце, одређује на основу особина из скупа најближих познатих инстанци и широко је коришћен метод класификације и регресије.

Предвиђање будућих догађаја је један од кључних изазова са којима се суочавају друштва, предузећа и појединци. У складу са предвиђањима, ентитети могу предузети акције у циљу повећања прихода или смањења штете.

Хоће ли ће нови комитент банке бити у могућности да отплати кредит? Хоће ли ће пацијент имати проблем при уградњи зубног импланта? Који жанр музике ће бити предложен потенцијалном купцу?

Добијање благовремених резултата применом алгоритама машинског учења, уз ефикасно коришћење ресурса, је императив. Једноставност је врлина КНН алгоритма, али је може бити узрок слабијих перформанси код података са шумовима, или кад је потребно обрадити велики број димензија и инстанци.

Претрага суседа је кључна и временски најкритичнија фаза алгоритма. Из претходно наведеног, намеће се закључак да су класификација и регресија завршне и једноставне фазе које следе захтеван и сложенен проблем претраге најближих суседа (енг. *Nearest neighbour search*). С тога проблем брзог налажења најближих суседа и структура које ту претрагу омогућавају представља окосницу овог рада. Укорењено је мишљење да ће линеарна, односно егзактна претрага суседа увек бити спорија од апроксимативне претраге. Апроксимативна претрага подразумева проналажење суседа који могу, али и не морају бити најближи суседи инстанце претраге. Најзначајнија особина апроксимативног алгоритма је да постигне готово исту прецизност као оригинални приступ са линеарном претрагом, али за значајно краћи временски интервал.

Управо је та чињеница мотивисала су аутора рада на изучавање апроксимативних метода претраге суседа. Резултат истраживања ће приказати рад алгоритма, преглед и анализу његових апроксимативних варијанти. Утврдиће се разлика у перформансама апроксимативних техника над скуповима података различитих величина и димензија, на корист будућим истраживачима.

У другом поглављу бавимо се формалним дефинисањем појмова претраге најближих суседа и апроксимативне претраге суседа.

# Дефиниција проблема и преглед стања у области

У формалном облику, проблем налажења најближег суседа се састоји у следећем:

Нека је дат скуп тачака у простору . Потребно је претражити скуп , како би се нашла тачка која је најближа задатој тачки то јест:

где је функција раздаљине метричког простора.

За разлику од алгоритама који испрва изводе генерализацију података и генеришу класификациони модел, те на основу модела брзо процењују припадност непознате инстанце одређеној класи (енг. *eager learners*), овај метод припада групи лењих алгоритама (енг. *lazy learners*). Одлика лењих алгоритама је да се инстанце просто складиште у фази тренирања, или се извршава минимално претпроцесирање података. То значи да ће фаза тренирања алгоритма бити краћа, међутим предвиђање алгоритма ће трајати дуже. Како се фаза тренирања код ових алгоритама састоји у простом чувању инстанци, ова група алгоритама се назива и инстанцно-оријентисаном. Овај алгоритам je често веома рачунски интензиван и захтева ефикасне технике складиштења података, које треба да омогуће паралелно извршавање поступка проналажења суседа (Han, Pei, & Kamber, 2012).

Иако је проблем налажења суседа наизглед једноставан, постоје два изразита проблема у вези са скалабилношћу при раду са великом количином података:

* Време извршавања: Алгоритамска сложеност проналажења само једног најближег суседа је , где је број тачака у простору и број димензија. Алгоритам је рачунски захтевнији када је потребно пронаћи више суседа, што је најчешћи случај. Најзад, цео процес је потребно извести сваки пут када се појави инстанца коју треба класификовати.
* Меморија: За брзо израчунавање удаљености, неопходно је да је цео скуп постојећих тачака учитан у РАМ меморију, што може представљати изазов код великих база података. (Maillo, Ramirez, Triguero, & Herrera, 2017)

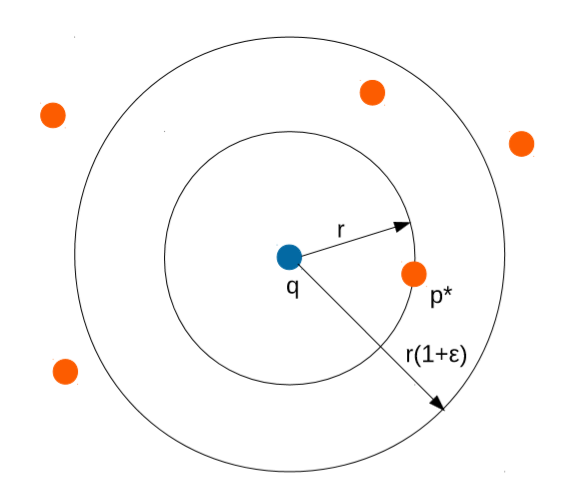
Ефикасно претраживање најближих суседа постаје учестали захтев према великом броју база података. Алгоритми који се директно ослањају на индексне структуре дају задовољавајуће резултате на малим или средњим количинама података, али не задовољавају потребе ефикасности када је реч o великим и високо димензионираним скуповима. (Seidl & Kriegel, 1998).

Очигледна тешкоћа да се осмисле алгоритми засновани на претрази суседа који ће бити ефикасни у случају када је број димензја већи (код неких аутора када је број димензија већи од 10), наводи да је потребно размотрити апроксимативне методе које ће проналазити приближно најближе суседе.

У том случају, можемо проширити претходно дату дефиницију:

Нека је дат скуп тачака у простору и нова тачка којој треба одредити суседа. За дато , тачка је приближан сусед ако је:

Где је најближи сусед задате тачке . У општем случају, када je , поступак се своди на проналажење приближних најближих суседа, где за сваког - тог важи да је апроксимација одговарајућег најближег суседа тачке (Arya, Mount, Netanyahu, Silverman, & Wu, 1998).



Илустрација 1: Апроксимативни алгоритам

Важно је напоменути да се међу имплементацијама апроксимативне претраге, ретко налазе оне, које дају гаранције да ће се пронађен сусед наћи у околини најближег суседа.

Проблем налажења суседа био је у пољу интересовања истраживача од педесетих година двадесетог века, међутим, област долази у фокус почетком деведесетих година када је објављено мноштво значајних радова из ове области. Према наводу (Kushilevitz, Ostrovsky, & Rabani, 2000), када је реч о веома димензионираним подацима, проблем је разматран од стране Добкина и Липтона (*Dobkin, D., & Lipton, R. J. (1976). Multidimensional searching problems*) који су генерисали алгоритам чија сложеност претраге експоненцијално зависи од броја димензија . Представљени метод је касније побољшан, али је време извршења још увек било у експоненцијалној зависности од димензије простора.

Препоруке за избор структуре погодне за претрагу у зависности од карактеристика података који се обрађују дали су (Muja & Lowe, 2009). Утврђено је да два алгоритма од којих се оба заснивају на стаблима (енг. *randomized kd-tree* и *hierarchical k-means tree*) пружају задовољавајуће резултате. Истраживања су показала да би број инстанци код оваквих структура, морао да буде значајно већи у односу на број димензија , јер у супротном не би било побољшања у времену у односу на линеарну претрагу свих инстанци (Arya, Mount, & Narayan, Accounting for boundary effects in nearest-neighbor searching, 1996)

Крајем века, (Kleinberg, 1997) је представио два алгоритма који дају значајна побољшања у односу на дотадашња достигнућа. Први алгоритам нуди сложеност претраге од и други, чија сложеност тежи линеарној (Kleinberg, 1997). Алгоритми подразумевају пројекцију вектора у простор са нижим бројем димензија и креирање графовске структуре.

Другачији приступ решавању проблема донели су (Indyk & Motwani, 1998) и објавили метод који има полиномну сложеност у зависности од и , дакле . Идеја је да се пронађе метод којим ће се међусобно слични објекти сврставати у исте категорије. Хеширање на основу положаја, или локацијски сензитивно хеширање (енг. *Locality-sensitive hashing, LSH*) је назив представљене технике. Дефиниција коју је дао (Samet, 2006) даје ближе објашњење:

Циљ *LSH* алгоритма је да се пронађе функција хеширања која ће сачувати информацију о удаљености инстанци, или приближне удаљености уз одређену толеранцију.

Општи кораци *LSH* алгоритма представљени су у раду под називом „Претраживање сличности при великом броју димензија путем хеширања“ (Gionis, Indyk, & Motwani, 1999).

Већина до данас представљених решења подразумева претпроцесирање постојећих инстанци, како би се над њима касније извршила претрага. Читав сет апроксимативних метода предвиђа преуређење скупа података тако што би се формирала структура стабла (*k-D tree*, *R-tree, VP-tree, SА-tree* и друга), чиме се обезбеђује убрзање алгоритма.

Друге методе предлажу хеширање инстанци како би се спровело груписање међусобно сличних, или пак организовање података у графовску структуру, где би блиске инстанце биле повезане гранама графа.

При одабиру апроксимативног алгоритма, пожељно је размотрити време, сложеност, као и меморијске захтеве обраде података. Велики број аутора даје одличан теоријски допринос, међутим, недостатак многих радова огледа се у изостанку свеобухватне експерименталне анализе. Често се недостаје прорамска имплементација алгоритма, или имплементација изводи на мањим и вештачки креираним скуповима података. Понекад изостаје упоредна анализа са изворним алгоритмом који се ослања на комплетну претрагу података.

Новија истраживања представљају решења проблема апроксимације ослањајући се на технологије за паралелизацију процеса и дистрибуиране рачунарске архитектуре. Неколико дистрибуираних алтернатива су предложене са циљем да омогуће апроксимативном алгоритму да обради велику количину података. Већина решења се базира на програмској парадигми *MapReduce* и његовој имплементацји отвореног кода – *Hadoop*. На тај начин се паралелизује извршавање алгоритма и ублажавају ефекти меморијских и рачунских ограничења. Недавно је представљен нови дизајн (Maillo, Ramirez, Triguero, & Herrera, 2017) који превазилази стандардне *Hadoop*-*MapReduce* приступе и обезбеђује флексибилну шему за класификацију великог броја непознатих инстанци над великим подацима, користећи *Apache Spark* архитектуру (Triguero, Maillo, Luengo, García, & Herrera, 2016).

Поменути радови објављени у прошлом веку инспирисали су истраживаче да осмисле нова, или дају значајна побољшања постојећих метода. Сва досадашња решења за претрагу суседа можемо оквирно сврстати у четири групе:

1. Приступи засновани на подели простора и стаблима
2. Графовски приступ
3. Приступ заснован на хеширању
4. Приступ заснован на векторској квантизацији

Подела је групо дефинисана и могуће је наћи примере који користе мешовите приступе попут (Douze, Sablayrolles, & Jégou, 2018) који комбинују графовске технике са приступима векторске квантизације. У наредним поглављима дати су описи неких представника ових група, који су имплементирани у програмским библиотекама отвореног кода.

# Апроксимативне технике претраге суседа

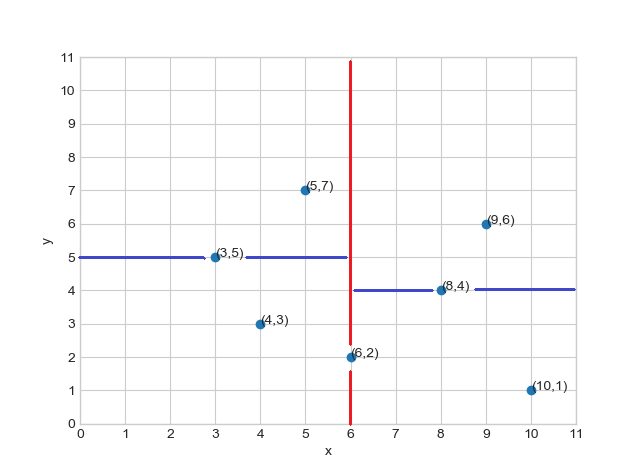
У претходном поглављу дат је осврт на правце истраживања у области претраге суседа. У поглављима која следе даје се детаљан опис најзначајнијих предтавника сваке класе. Објашњени су поступци претраге суседа и креирање структура података над којима се извршавају. Описано је неколико егзактних алгоритама који уз увођење додатних ограничења постају апроксимативни. Теоријски приказ сложености алгоритама омогућава тумачење резулата истраживања и олакшава одабир метода за тестирање.

## Подела простора и стабла

Како би убрзали поступак налажења суседа и смањили обим претраге, истраживачи су се окренули техникама за поделу простора. У највећем броју случајева, ради се о бинарној подели, која подразумева рекурзивно дељење на два дела, све док је испуњен критеријум поделе. Најмањи подпростор добијен овим путем се често назива ћелија. Бинарно стабло је структура података која је погодна за организацију елемената из тако подељеног простора. Тачке сваког дела, постају елементи левог и десног подстабла, респективно. Исправно је претпоставити да ће се најближи суседи наћи у ћелији којој припада тачка претраге, или пак у суседним ћелијама. Један од најбитинијих фактора по којима се разврставају алгоритми ове класе, је метод поделе простора. Код најпознатијег представника групе, *k-D* стабла, простор у сваком кораку дели хиперраван која је нормална на осу неке димензије к. Преставници ове групе су *RP* стабла, где је хиперраван насумично одређена , *2-means tree* где се подела спроводи на основу алгоритма калстеризације, *PCA-tree* које простор дели на основу анализе главних компоненти, *VP-tree* и друга.

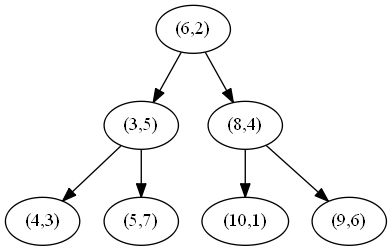
### *k-D* стабла

*k-D* стабло је један је од првих и најпознатијих предствника групе. Посматрајмо једноставан пример дводимензионалног простора (). У основном облику, алгоритам налази медијалну вредност међу кординатама одабране осе, у овом случају *x* осе. Медијална вредност ће дефинисати хиперраван поделе. У случају када имамо две димензије, хипераван ће заправо бити права, која дели простор на два дела.



Илустрација 2: *k-D* подела простора

Како је медијална вредност по *x* оси једнака 6, у првој итерацији, простор је подељен у односу на тачку са координатама (6,2) - која постаје корен стабла. Остале тачке из просотра постају чворови левог или десног подстабла, у завиности од положаја у односу на праву . У наредној итерацији простор се дели у односу на *y* осу, за сваки претходно добијен подпростор. Праве и деле леви и десни подпростор, па тачка (3,5) постаје корен левог подстабла, а тачка (8,4) десног. Бинарно стабло добијено на овај начин представљено је на Илустрацији 3: *k-D* Бинарно стабло.



Илустрација 3: *k-D* Бинарно стабло

Алгоритам је лако проширити на случај када је број димензија *k* већи () . У свакој итерацији простор се дели у односу на димензију *k*. Услов за заустављање поделе може бити дат корз ограничење броја нивоа које стабло може имати, дефинисањем минималног броја елемената који ће се наћи у листовима, или на неки други начин.

*k-D* је егзактан метод, што значи да ће резултат претраге бити оне инстанце које су прави суседи тачке претраге. Из тог разлога, алгоритам не проверава само саджај ћелије којој припада тачка претраге, већ и суседних ћелија уколико је потребно (Алгоритам 1.). Вратимо се на дводимензиони пример. Нека тачка претраге има координате (7,9). Претрага започиње од корена стабла који ћемо прогластити најближим. Како корени чвор није лист у стаблу, услов из 2. линије није задовољен. Алгоритам се наставља у десном подстаблу. Рекурзивним позивима долази се до чвора (9,6), који се налази у ћелији којој припада тачка претраге и који се проглашава најближим. Претрага се поново наставља у кореном чвору где се утврђује да је дистанца између тачке претраге и тачке (9,6), већа од удаљености од осе поделе (линија 7), па постоји могућност да се најближи сусед налази у левом подстаблу. На исти начин, алгоритам пролази кроз лево подстабло и налази правог најближег суседа (5,7).

|  |  |
| --- | --- |
| Алгоритам 1. Претрага у *k-D* стаблу | |
| 1: | pretraga(q, n, p, d) |
| 2: | if n.l = ∅ {\displaystyle \varnothing } ∅ ∧ n.r = ∅ |
| 3: | dc ≔ dist(q,n) |
| 4: | if dc < d then p ≔ n; d≔dc; return p,d; |
| 5: | else |
| 6: | if q.ax ≤ n.ax |
| 7: | If q.ax – d ≤ n.ax then p,d ≔ pretraga(q, n.l, p, d) |
| 8: | If q.ax + d > n.ax then p,d ≔ pretraga(q, n.r, p, d) |
| 9: | if q.ax > n.ax |
| 10: | If q.ax + d > n.ax then p,d ≔ pretraga(q, n.r, p, d) |
| 11: | If q.ax – d ≤ n.ax then p,d ≔ pretraga(q, n.l, p, d) |

Читалац ће приметити да се егзактан метод лако може претворити у апроксимативни, тако што ће се ограничити број чворова који се посећује, односно спречити претрага суседних ћелија. У повољном случају, сложеност претраге *k-D* стабла је , али се потврдило да сложеност може бити линеарна, што су потврдила и емпиријска истраживања (Дај референцу на истраживања) Метод је показао неефикасност када је број димензија већи од 12 (Marimont & Shapiro, 1979). Један од лорака који утиче на сложеност алгоритма је проналажење медијалне врености.

### VP-tree

Једна од најпознатијих подврста Лопта-стабала (енг. *Ball-tree*) je *VP-tree* (енг. *Vantage point tree*). Испрва је потребно из скупа тачака изабрати тачку која ће бити пивот - . Потом се израчунава медијална удаљеност између пивота и осталих тачака, па се скуп дели на два дела по принципу:

тако да су тачке из скупа унутар полупречника , док су остале тачке ван овог пречника. Рекурзивном применом овог правила гради се бинарно стабло где су унутрашњи чворови пивоти, чија су елементи левог и десног подстабла унутар и изван одговарајуће лопте (Samet, 2006). Пивот се може изабрати насумично, или на неки други начин. Један од предложених метода, од којег се очекује да резултира креирањем стабла које пружа добре преформансе, дефинисао је (Yianilos, 1993). Идеја је да се из насумично одабраног подскупа тачака пронађе она, која има најбољу расподелу удаљености у односу на друге тачке. Такав метод форсира избор оне тачке која је удаљена од средине, односно скрајнута на ивицама скупа. Поступак креирања стабла може се дефинисати једноставним алгоритмом:

|  |  |
| --- | --- |
| Алгоритам 2. Креирање *VP-tre* | |
| 1: | kreirajVP() |
| 2: | if = ∅ {\displaystyle \varnothing } ∅ then return ∅ |
| 3: | = ∅ {\displaystyle \varnothing } ∅ |
| 4: | = ∅ {\displaystyle \varnothing } ∅ |
| 5: | n ≔ kreirajCvor() |
| 6: | n.pt ≔ izaberiPivot() |
| 7: | n.med ≔ Medijana(dist(n.pt, )) |
| 8: | For each |
| 9: | If dist(, n.pt) < n.med then .add() |
| 10: | If dist(, n.pt) ≥ n.med then .add() |
| 11: | n.l = kreirajVP() |
| 12: | n.l = kreirajVP() |
| 11: | return n |

Претрага се врши на сличан начин као код *k-D* стабала, па и овде важи аналогија да се апроксимативна варијанта алгоритма може добити ограничавањем броја чворова који порцедура испитује.

### Друге методе поделе

Посматрајмо тачке скупа као векторе. За конструкцију *RP-стабла*, испрва се насумично бира вектор (често јединични вектор) , потом се врши пројекција свих вектора из на , да би се на крају одредила медијална пројекција и скуп поделио на два дела.

У дводимензионалном простору, геометријском смислу, простор би делила права која је ортогонална на насумично дефинисану праву. Ортогонална права дели простор на делове са једнаким бројем тачака.

У раду ће се тестирати и *k-means* стаблокоје се формира рекурзивним кластеровањем података и *randomised k-D tree* које се формира насумичним одабиром димензије по којој се врши подела код *k-D* стабала.

## Графовски приступ

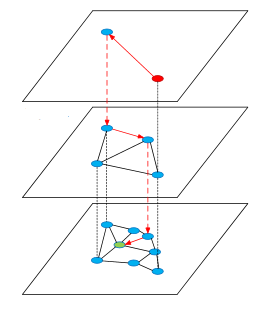
(Овде иде малени преглед о Делонеовом графу и SP јер су углавном инспирисани њима. Наведи проблем код ових алгоритама - када су подаци кластеризовани, може да се деси да граф није повезан и да се заглави у прерага у локалном минимуму.)

### HNSW

*HNSW* се сврстава у алгоритме који користе графовске структуре за проналажење најближих суседа у метричком простору. Тачке у простору представљају чворове графа, а неусмерене гране су везе међу суседима. Хеуристика који врши претрагу структуре користи се и при додавању нових чворова. Наиме, хеуристика претражује структуру у потрази за најближим суседима, када су услови претраге задовољени, елемент постаје чвор графа са гранама које га везују за суседе.

Аутори (Malkov, A, & Yashunin, 2018) представљају хијерархијски организовану структуру графова, за коју тврде да има логаритамску сложеност претраге. Главни чиниоци који доприносе логаритамској сложености су, експлицитна селекција улазног чвора у графовску струкуру и напредна хеуристика за брзо налажење суседа у оквиру графа. Структура се назива хијерархијском јер је организована у више међусобно повезаних нивоа. На свим нивоима хијерархије формира се граф, са тим што се број чворова који формирају граф смањује са порастом нивоа. На најнижем нивоу, све тачке векторског простора формирају граф. На сваком наредном нивоу, граф ће формирати подскуп тачака из претходног нивоа. Формалан опис претходно описане структуре може се дати на следећи начин:

Нека је дат скуп тачакa векторског простора и нека је дат скуп нивоа хијерархије . Све тачке скупа биће организоване у графовску структуру на најнижем нивоу хијерархије те можемо записати . За скп тачака које су саставни део графа на нивоу важи да је . У општем случају .



Илустрација 4: *HNSW* структура (Malkov, A, & Yashunin, 2018)

Из оваквог описа, јасно је да ће се поједине тачке векторског простора јављати на неколико нивоа хијерархије. Важно је напоменути да међу нивоима структуре постоје везе. Тачка која представља чвор графа на нивоу имаће референцу на чвор исте те тачке на нивоу .

Конструисање структуре

Графови се конструишу итеративним додавањем елемената из скупа . Елемент постаје чвор графа и повезује се са најближих суседа. Испрва се за сваки елемнт који се додаје у структуру одреди макслимални ниво на којем ће се појавити . Вероватноћа да се елемент нађе на вискоим нивоима је експоненцијално опадајућа, чиме се обезбеђује мали број чворова у вишим слојевима хијерархије. Процес додавања у структуру се угубо може поделити у две фазе:

* Проналажење улазног чвора, почевши од највишег нивоа па до максималног дефинисаног нивоа на ком се чвор додаје
* Проналажење најближих суседа на максималном нивоу за елемент и његовим поднивоима и повезивање са њима ()

Треба напоменути да се хеуристика која служи за претрагу нивоа користи у обе фазе. Разлика је што се у првој фази параметри постављају тако да се проналази само један најближи сусед.

Прва фаза додавања започиње у највишем нивоу где се проналазе најближи суседи за елемент који се додаје. Најближи сусед пронађен у највишем нивоу постаје улазни чвор за претрагу на нижем нивоу, где се пoступак претраге понавља. Суседи улазног чвора који су ближи елементу који се додаје у граф постају кандидати за најближег суседа. Како је у овој фази потребно наћи најближег суседа елемента који се додаје, хеуристика „прождрљиво“ бира само једног кандидата из листе суседа. У наредном кораку посматрају се суседи одабраног кандидата. На овај начин хеуристика у сваком чвору бира суседа који задовољава критеријум да је најближи елементу који се додаје. Претрага се завршава оном чвору, за којег важи да нема суседе који би били ближи елементу који се додаје. Тада се прелази на нижи хијерархијски ниво.

Друга фаза започиње када алгоритам спусти на ниво хиејарахије где ће елемент први пут посати чвор графа.

Ако би параметри алгортма били подешени тако да постоји само један ниво, алгоритам би се свео на нехијерархијску верзију дефинисану у (цитирај). Како би се избегло да се алгоритам претраге заустави у неком локалном минимуму, аутори су саветовали да се претрага започне у чворовима који имају висок степен повезаности (енг. 'hubs'). Како је број високо повезаних чворова растао са повећањем броја елемената, па је претрага тежила полилогаритамској сложености. Увођењем концепта хијерархије, очекивало се смањење сложености претраге и могућност заустављања у локалном минимуму, што је чест случај код кластеризованих података. Наведене претпоставке су се потврдиле у досадашњим експериментима. Аутори наводе да је фаза креирања графа погодна за паралелизацију.

## Приступи засновани на хеширању

Хеш функције пресликавају податке произвољне дужине у податке фиксне дужине – често је податак фиксне дужине у облику бинарног записа. Ситуација у којој два различита податка имају исту хеш вредност назива се колизија. Колизија је неминовна појава код хеширања и често је непожељна у области криптографије. Код проблема претраге сличности, колизија је веома пожељна! Тада два различита, али међусобно слична податка добијају исту хеш вредност. Циљ локацијски сензитивног хеширања (*LSH*), је да додели исту хеш вредност тачкама које су близу једна другој. Концепт су формулисали (Indyk & Motwani, 1998).

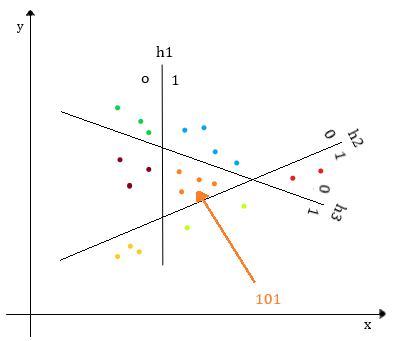
Нека је дат простор . Фамилија хеш финкција је сензитивна за функцију удаљености за свако када важи да:

ако онда

ако онда

Дакле, ако је удаљњност међу тачкама мања или једнака , вероватноћа њихове колизије треба да буде већа или једнака . Када је удаљеност већа од , вероватноћа колизије мора бити мања или једнака .

У духу претходно навединих техника поделе простора, можемо рећи да две тачке које су близу једна другој, треба да припадају истој партицији простора са вероватноћом не мањом од . У том случају хеш функције би дефинисале неку хиперраван за поделу.



Илустрација 5:Пример хеширања

Помоћу фамилије хеш функција, може се конструисати индексна структура, на следећи начин:

За цео број *М*, дефинишимо фамилију финкција такву да за важи да је , где и . Овде је заправо конкатенација *M* хеш функција.

За цео број *L*, насумични изаберимо функције из . Свака појединачна функција , користи се за конструисање појединачне хеш табеле. Дакле, креира се хеш табела.

У складу са претходно наведеним, потребно је

1. Извршити иницијализацију *L* хеш табела
2. За сваку табелу генерисати *М* хеш функција
3. Унети тачке рачунајући хеш вредност за сваку табелу .

Тачке којима је додељена иста хеш вредност у табели припадају истом бакету (енг. *bucket*). Претрага се врши тако што се се генерише хеш вредност тачке претраге за сваку табелу. Потом се саставља листа кандидата из бакета свих табела којима припада тачка претраге. Кандидати се напослетку сортирају по удаљености и извлачи се *k* најближих . Да би резултати претраге били смислени, потребан је велики број табела, који за последицу има велике меморијске захтеве. Овај проблем би се могао решити на уштрб времена претраге, тако што би се у свакој табели проверавало више бакета. Алгоритам *Multi-Probe LSH*, решава проблем тако што у свакој табели проналази „суседне“ бакете и додаје у листу кандидата.

# АНАЛИЗА МЕТОДА

Анализа перформанси апроксимативних техника за претрагу суседа представљена је у овом поглављу. У истражиавњу су коришћене библиотеке са постојећим имплементацијама претходно описаних алгоритама. Уобичајено је за овакве бибилиотеке да се изворни алгоритми делимично мењају, или комбинују ради побољшања претраге, или смањења рачунске сложености. На наредним страницама дат је опис кључних параметара за поређење техника и скупова података над којима се поређење врши.

Поглавље пред нама би могло да пружи одговоре на следећа питања:

* Које су најпопуларније апроксимационе технике?
* Колико је времена потребно за извршавање линеарне претраге и апроксимативне претраге?
* Колико се разликују резултати апроксимативне и комплетне претраге суседа?
* У којој мери је могуће паралелизовати извршење алгоритма?
* Колики је утицај структуре података на брзину извршавања апроксимативних варијанти алгоритма?
* Које ће перформансе и резултате апроксимативни алгоритми постићи при раду над скуповима различитих величина и броја атрибута?
* Да ли је подешавање параметара једноставно и у којој мери утиче на перформансе?
* Да ли је могуће извести препоруке за коришћење одговарајућег апроксимативног алгоритма у зависности од података и корисничких захтева?

## Скупови података, индикатори перформанси и библиотеке

Сетови података за тестирање најчешће садрже три скупа података: тренинг, тест и контролни скуп података. Скуп података за тренирање је онај у коме се врши претрага најближих суседа. Тест скуп садржи тачке претраге, односно оне инстанце за које је потребно одредити суседе. Контролни скуп (енг. *ground truth*), садржи правилна решења претраге, то јест садржи листу правих суседа из тренинг скупа.

Скупови података који се најчешће користе у референтној литератури, над којима се врши анализа и у овом раду, приказани су у табели:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Назив | Број димензија | Тренинг инстанце | Тест инстанце | Величина | Број суседа |
| *SIFTsmall* | 128 | 10000 | 100 | 4.92 *MB* | 100 |
| *SIFT1M* | 128 | 1000000 | 10000 | 492 *MB* | 100 |
| *FashionMINST* | 784 | 60000 | 10000 |  | 100 |
|  |  |  |  |  |  |

Табела 1: Подаци

Колоне имају следеће значење :

* Број димензија – величина векторскоског простора; Свака инстанца из скупа података је вектор за назначеним бројем димензија
* Тренинг инстанце – број инстанци, односно вектора у тренинг скупу података
* Тест инстанце – број вектора у тест скупу за које је потребно наћи најближе суседе из скупу података за тренинг
* Величина – меморијски простор који скуп заузима на диску рачунара
* Број суседа – показује колико је суседа потребно пронаћи за сваку инстанцу из тест скупа

Број суседа који се тражи једнак је за сваки скуп пдатака и сваки алгоритам и изности једну стотину.

*SIFT1M* и *SIFTsmall* садрже векторе, чије димензије представљају дескрипторе фотографија који су погодни за претрагу. Подаци су преузети са . Датотеке су у бинарном формату, а свака димензија је представљена тридесетдвобитним децималним бројем.

Дефинисано је пет кључних идентификатора перформанси. У квантитативне индикаторе убрајају се:

* Грешка удаљености () – разлика у вредности функције удаљености од непознате инстанце између апроксимативних и правих суседа
* Одзив () – број идентификованих правих суседа у односу на број тражених
* Време претпроцесирања () – процесорско време потребно за конструисање индексне структуре у секундама
* Време претаге () – процесорско време потребно за проналажење суседа у секундама

Процесорско време указује на укупно време које је процесор утрошио на конструисање индекса и претрагу. У њега не спада време које је утрошено на друге процесе система. Оно међутим укључује утрошено време на свим процесорима код вишепроцесорских система, па се може догодити да буде веће од реално протеклог времена. У хипотетичком случају, за алгоритам који користи две нити, реално време извршавања може бити 5 секунди, али је у том случају процесорско време 10 секунди.

Грешка удаљености се формално дефинише као количник просечнне удаљености суседа и тачака претраге у скупу апроксимативних суседа и правих суседа :

Одзив се показује удео броја пронађених парвих суседа:

Једноставност подешавања параметара намеће се као квалитативни индикатор перформанси. Јадноставно подешавање параматеара подразумева да је за генерисање добрих квантитативних идикатора потребно подесити један параметар, или да је подешавање аутоматско. Средње тешко подешавање подразумева подешавање 2 параметара, док комплексно захтева сложене комбинације више параметара за добијање добрих резултата.

Већина алгоритама је имплементирана у библиотекама отвореног кода у језицима *C* и *C++*. Постоје имплементације у другим програмским језицима као што је *Python*, али се оне ретко користе због слабијих перформанси тог програмског језука у односу на горе наведене. Коришћене бибилиотеке имају омотаче (енг. *wrappers, Python bindings*) у *Python* језику, што олакшава њихово коришћење, али отежава корекције самог алгоритма.

У разматрање се узимају следеће бибилиотеке са припадајућим алгоритмима:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назив алгоритма | Библиотека | Структура | Језик | Врста |
| *HNSW* | *Nmslib* | Граф | *C++* | Апр. |
| *vp-tree* | *Nmslib* | *vp-tree* | *C++* | Апр./Екз. |
| *Annoy* | *Annoy* | *2-means tree forest* | *C++* | Апр. |
| *mrpt* | *Mrpt* | *RP-tree forest* | *C++* | Апр. |
| *kdTree* | *flann* | *randomized k-D* | *C++* | Апр. |
| *kmeans* | *flann* | *k-means tree* | *C++* | Апр. |
| *linear* | *flann* | Линеарна претрага |  | Егз. |
| *k-D* | *Scikit-learn* | *k-D* | *Python* и *Cyton* | Егз. |
| *ball-tree* | *Scikit-learn* | *Ball-tree* | *Python* и *Cyton* | Егз. |
| *LSH* | *FALCONN* | *LSH*- табеле | C++ | Апр. |

Табела 2: Алгоритми

Анализа започиње над мањим скуповима података и врши се на персоналном рачунару са процесором од два језгра јачине 2×2.4 *Ghz* и 4*Gb* меморије, под Windows оперативним системом. Сложеније анализе се спроводе на удаљеном рачунару на *Databricks* платформи, са огрниченим ресурсима у погледу меморије, процесора и времена коришћења. Доступно је 15 *Gb* меморије и процесор са два језгра и максималним периодом мировања рачунара од 2 сата.

ОВДЕ О БИБЛИОТЕКАМА

Једина бибилиотека која првобитно намењена за *Linux* оперативни систем је *falconn*, па су њене могућности тестиране само на удаљеном рачунару.

Без подешавања параметара, *vp-tree* имплементира линеарну претрагу. Најлакши начин коришћења овог алгоитма на апроксимативан начин, подразумева ограничавање броја листова који се могу посетити. То се постиже подешавањем параметра *maxLeavesToVisit*.

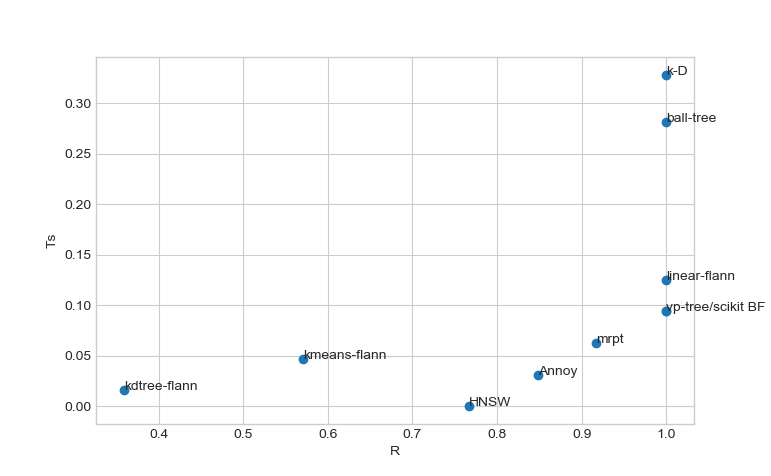
*Annoy* алгоритам се ослања на *2-means-tree* структуру и векторске пројекције. У свакој се итерацији при креирању стабла, хеуристиком бирају два центроида који дефинишу хиперраван поделе. Креирањем више таквих стабала, то јест подешавањем параметра *n\_trees*, повећава се прецизност алгоритма, али и величина индекса и време конструисања. Као и *vp-tree,* постоји параметар за ограничавање броја чворова који се обилазе *search\_k.*

*Mrpt* se zaniva na *RP-tree.* На случајан начин бира се хиперраван за поделу, вектори се пројектији на раван, па се елементи смештају у лево и десно подстабло у односу на медијалну вредност пројекције. Могуће је креирање више оваквих стабала. Испрва се резулатаи генеришу за свако стабло, а потом се из тако добијене листе кандидата селектују најближи. Ова библиотека нуди самоподешавајућу методу за коју је потребно дефинисати тражени одзив. Могуће је ограничавање броја стабала. Параметри су *target\_recall* и *trees\_max*.

Aлгоритами у *flann* библиотеци названи су по алгоритмима којима су били инспирисани. Алгоритам *kdTree* је заправо *randomized k-D tree,* док је *k-means* заправо хијерархијско *k-means* стабло.

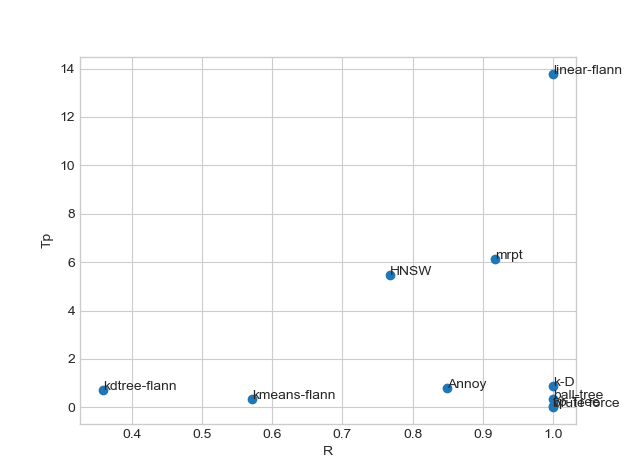
## Селекција алгоритама

Како је за анализу већег броја података потребно више времена, пожељно је извршити тестирање и селекцију алгпритама на мањем узорку, са минималним подешавањем параметара. Први круг евалуације је започет са малом скупу података како би се на брз начин проверили ефекти параметара. Значајно ће бити поређење перформанси на мањем и сто пута већем скупу *SIFT1M.* Као мера удаљености на оба скупа корити се еуклидско растојање. Поједине бибилиотеке нуде аутоматско прилагођавање параметра скупу података (*flann*, *mrpt*), док се друге ослањају на предефинисане вредности (*scikit-learn*) и ручно унете параметре (*Annoy*, *nmslib*).



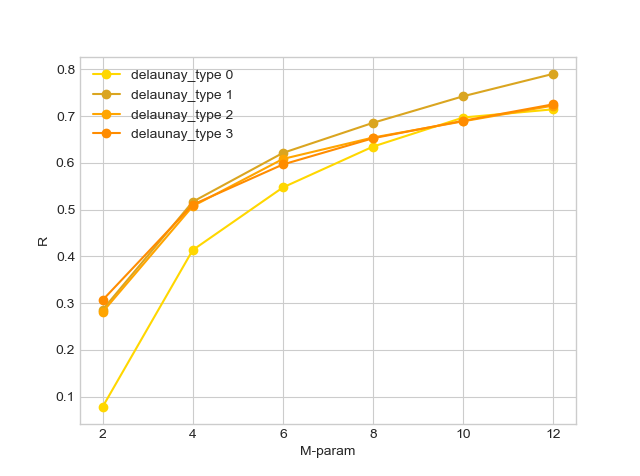
Илустрација 6: Одзив и Време претраге SIFTSmall

На малом узорку са већим бројем димензија, *k-D* показује слабије перформансе, што потврђује теоријске претпоставке. Предефинисани максимални број листова у стаблу је 40. Егазктне методе које се ослањају на структуру бинарних стабала биће искључена из даље анализе (*k-D* и *ball-tree* ).За *Annoy* алгоритам је дефинисано 15 стабала. Код конструкције *mrpt* алгоритма, унет је параметар за аутоматско подешавање *target\_recall* = 0.91, који преставља жељени одзив. У сладу са тим параметром, алгоритам је самосатлно дефинисао максималну дубину сатабала на 6 нивоа и број стабала на 99. Најбржи алгоритам је *HNSW* , али по цени нижег одзива (Илустрација 6). Параметар *М* регулише максималан број грана по чвору, на свим нивоима графа осим на најнижем. Његова вредност је подешен на 20. Исти параметар утиче на број нивоа хијерархије, меморијске захтеве и време креирања структуре. *HNSW* има високо процесорско време, али за конструкцију индекса користи две процесорске нити



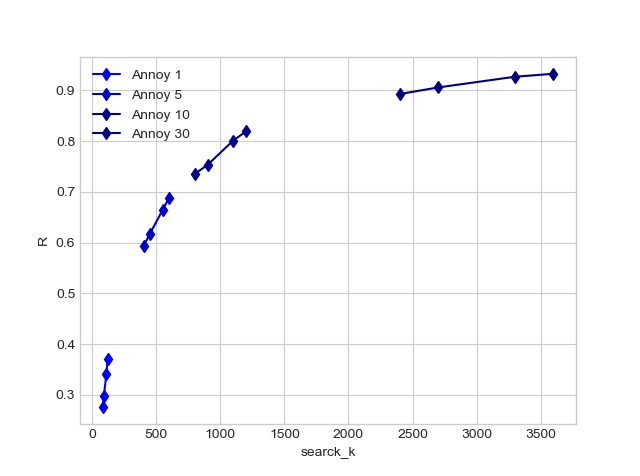
Илустрација 7: Одзив и конструкција индекса

Аутоматско подешавање је укључено у време конструисања *mrpt*, па је за индексирање било потребно више од 6 секунди прoцесорског времена. Алфоритам *linear-flann* има активирано аутоматско подешавање које утиче на *Tp.* Пошто су *Annoy* и *HNSW* апркосимативне методе које имају задовољаваћући одзив, над њима се врши анализа утицаја параметара.



Илустрација 7:*HNSW* -утицај параметара

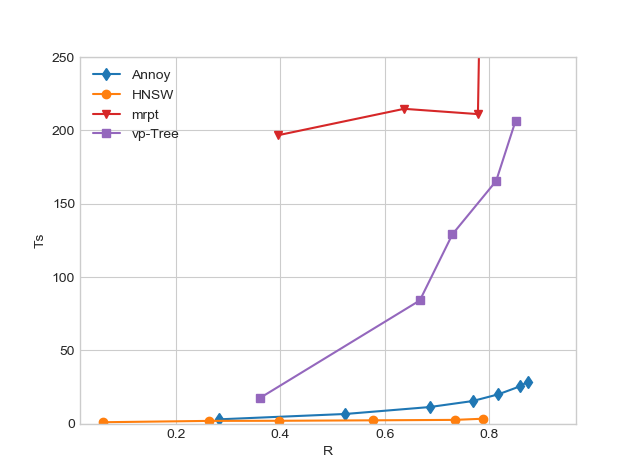
У теоријском опису *HNSW* је поменуто да постоји више хеуристика које врше претрагу, а користе се и за креирање графовске структуре. На тесту се показало да хеуристика 1 има бољи одзив када вредност параметра *M* расте, то јест када је број грана међу чворовима већи. Разлика између нулте хеуристике и осталих је већа, када је број грана у графу нижи. Када је број грана расте, не постоји значајна разлика мађу хеуристикама 0, 2 и 3. Граф је боље повезан у томслучају, па је претпоставка да је за добар одзив хеуристике 0 заслужна управо повезаност графа. Параметар *M* утиче на време креирања индекса и величину меморије коју они заузимају, те ако постоје меморијска ограничења на кластеру, за бољи одзив је пожењно користити хеуристике 1, 2 и 3. Треба имати у виду да оне утичу на време претраге, па је потребно извршити анализу, како би се постигао одговарајући одзв за жељено време.



Илустрација 8:*Annoy -* утицај параметара

Код *Annoy* алгоритма је тестиран утицај параметара за претрагу *search\_k*, при различитом броју конструисаних стабала. Његова предефинисана вредност једнака броју стабала помноженом са бројем суседа који се траже (Илустрација 8:*Annoy -* утицај параметара). Овај параметар директно утиче на време претраге. Примећуе се да при већем броју стабала, параметар у мањој мери утиче на одзив алгоритма. Број стабала директно утиче на време конструисања индекса и меморију коју заузима, док параметар *search\_k* утиче на одзив и време претраге. Број стабала такође утиче и на време претраге и одзив, јер са повећањем броја стабала расте време претраге, али се и побољшавају резултати претраге. Овде је потребно наћи прави однос параметара, како би се постигао одређени одзив у жељеом времену, што је случај и са осталим алгоритмима. Ако меморија представља ограничавајући фактор, повећање параметра *search\_k* по цени споријег одзива је смислено. Ако је време критичан фактор, и потребан је виши одзив, може се креирати више стабла, али смањити вредност параметра *search\_k.*

Почетни резултати на већем скупу података *SIFTM*, на персоналном рачунару, указују да алгоритми *Annoy* и *HNSW* дају резултате за значајно краћи временски период од других алгоритама (Илустрација 9: SIFT1M Почетни резултати). Време претраге *HNSW* мери се секундама, док се код осталих ради о десетинама и стотинама секунди. И поред подешавања параметра *М*, алгоритам није забележио задовољавајући одзив. Најбољи одзив постигнут је са параметром *М*=40, а препоручене вредности параметра су између 2 и 48. Веома добре резултате постиже *Annoy,* где је најбољи одзив од 0.85 постигнут са 80 стабала. Одзив од 0.81 постигнут је са 60 стабала. Иако је спор, изненађујуће резултате постиже *vp-Tree* коме је максималан број елемената у листу подешен на десет хиљада, што је око 1% тест скупа података. Најбољи одзив од 0.85 постиже се за параметар *maxLeavesToVisit* на 25 – што не може бити више од 25% скупа података. Алгоритам *mrpt* подешен је преко методе за аутоматско формирање параметара и одзив од 0.77 постиже са 262 стабла. Максималан број стабала који је прослеђен као параметар претраге је 300. Са 458 стабала, постиже се одзив од 0.92 али за изузетно дуг временски период, па је овај алгоритам изузет из даље анализе.



Илустрација 8: SIFT1M Почетни резултати

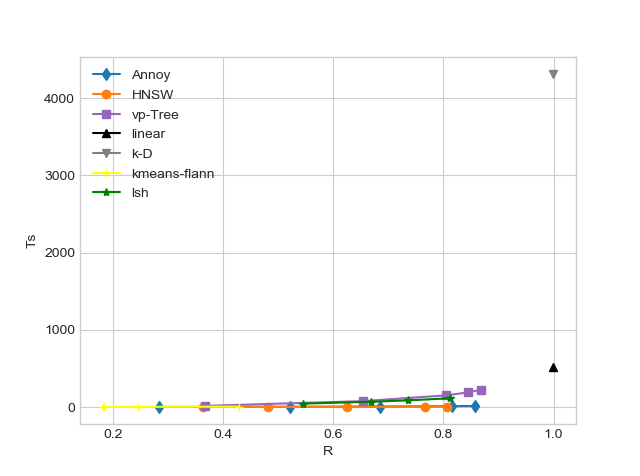
Даље истраживање се спроводи на удаљеном рачунару, под *Linux* оперативним системом, па је могуће тестирати и *LSH* алгоритам из *falconn* библиотеке. Ради поређења резултата, извршена је и линеарна претрага и *k-D* алгоритам. Избор параметара код *flann* библиотеке, препуштен је функцији за аутоматско тражење параметара, али резултати алгоритма нису задовољавајући. Наведена функција бира параметре претраге и алгоритам претраге на основу узорка из скупа података и на основу коефицијената који регулишу време креирања структуре и меморијске захтеве (*build\_weight ,memory\_weight*).

Параметри осталих метода дати су у табели:

|  |  |
| --- | --- |
| Алгоритам | Параметри |
| *Annoy* | Број стабала *num\_trees* = 5, 15, 20, 60, 80 |
| *HNSW* | Број грана *M =* 5, 8, 15, 30, 38 |
| *vp-Tree* | Величина листа *bucketSize* = 10000; *maxLeavesToVisit* = 2, 10, 20, 25, 30 |
| *k-D* | Величина листа *leafSize* = 10000 |
| *flann* | *target\_precision* = 0.3, 0.6, 0.7, 0.85 *build\_weight* = 0; *memory\_weight* = 0,*sample\_fraction* = 0.05 |
| *LSH* | Број табела *t*= 15; Број хеш функција *k* = 4;  *number\_ probes* = 15, 30. 45, 75 |

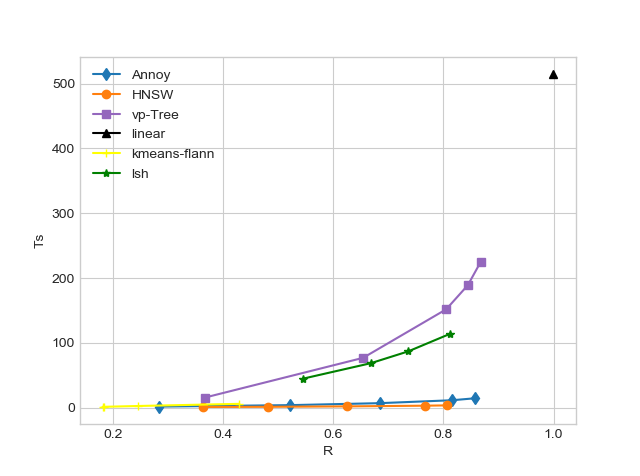
Табела 3: Параметри на *SIFT1M* *DataBricks*

Све апроксимативне методе, осим оних из *flann* библиотеке, постигле су одзив већи од 0.8. Најгоре време претраге имао је *k-D* и чије време извршавања износи више од 4300 секунди. То је скоро 9 пута спорије од линеарне претраге за коју је задужена *scikit-learn* библиотека.



Илустрација 9: Одзив и Ts на *SIFT1M DataBricks*

Табеларни приказ добијених резултата се налази у Прилогу 1. Због јаснијег прегледа је искључен *k-D* приступ на илустрацији 10.

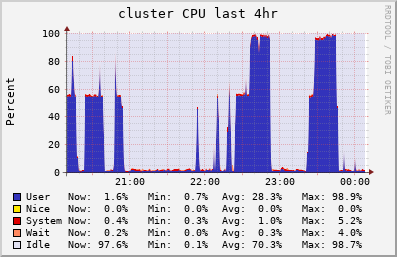
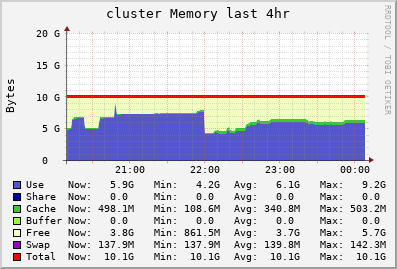


Илустрација 10: Време претраге и одзив на *SIFT1M*

Линеарна претрага над овим скупом података је релативно брза и завршена је за 515 секунди. Апроксимативни алгоритми су остварили одзив већи од 0.8 за време које је и до 100 пута мање у односу на линеарну претрагу (*HNSW*). Просечна удаљеност тачака претраге од идентификованих суседа је незнатно већа од стварне, која је одређена потпуном претрагом скупа података (*linear*).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритам |  |  |  | Просечна удаљеност |  |
| Annoy-trees-80 | 244.18 | 14.5923 | 0.8574 | 236.89 | 100.53% |
| HNSW-M-38 | 3269.57 | 3.627 | 0.8064 | 237.35 | 100.73% |
| vp-Tree-10k-mL30 | 17.5 | 225.437 | 0.8681 | 237.83 | 100,93% |
| linear | 0.09 | 515.1839 | 1.0 | 235.63 | / |
| lsh-L15k4t75 | 132.35 | 114.1396 | 0.8125 | 0.0 | / |

Приметна је велика разлика у времену потребном за конструисање индекса. *HNSW* је утрошио највише времена и ресурса за изградњу структуре и томе је нејвише допринео сложен начин формирања индекса и већи меморијски захтеви због већег броја грана у графу (*M*=48). У појединим интервалима, процес креирања је у потпуности заузео ресурсе процесора (Илустрација 11), што није био случај код других алгоритама. Резултати показују да *HNSW*, *Annoy* и *LSH* имају боље перформансе у односу на остале алгоритме. Они су уједно и представници група које су описане у другом поглављу.

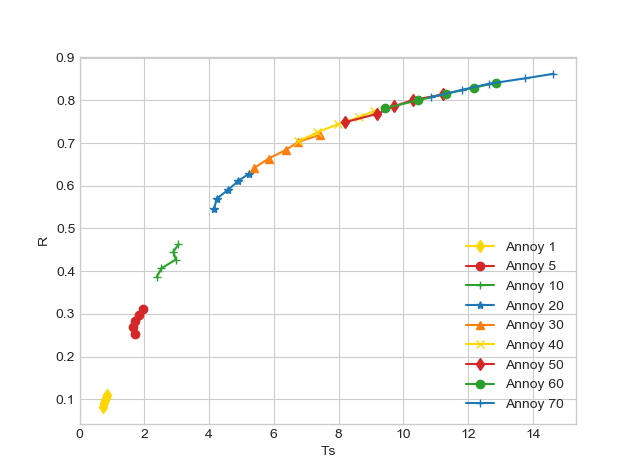


Илустрација :Меморијски (лево) и рачунски (десно) захтеви

Сва три алгоритма захтевају већи меморијски простор и пружају могућност складиштења индекса на диску рачунара. После једог од експеримената над *SIFT1M* подацима на персоналном рачунару, сачувани индекси су заузели више меморије од самог скупа података! *Annoy* од 80 стабала заузео је 1.4 *Gb*, а *HNSW* 0.9 *Gb* меморије. Параметри ова три алгоритма биће даље анализирани.

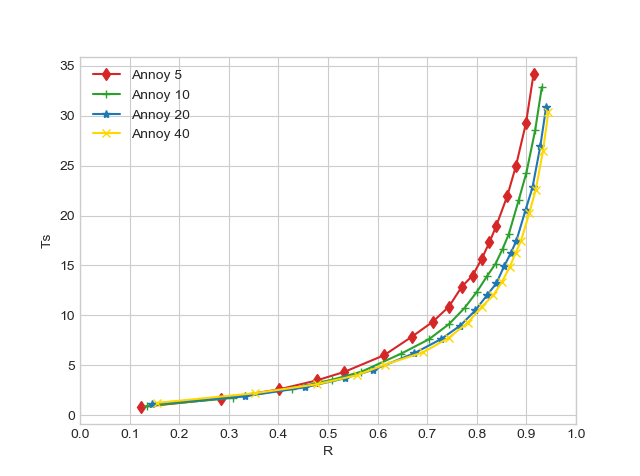
## Анализа параметара

На илустрацији је приказан однос времена претраге и одзива, у зависности од броја стабала и параметра *search\_k*. Утицај броја стабала на однос индикатора перформанси је занемарив у појединим случајевима. Разлика у одзиву између алгоритма са 60 и 70 стабала у дванаестој секунди је минорна, па се намеће питање колико је стабала потребно да би се постигао жељени однос одзива и времена претраге. Већи број стабала заузима више меморије , па је оптимизација оправдана када су мемориjски капацитети ограничени.



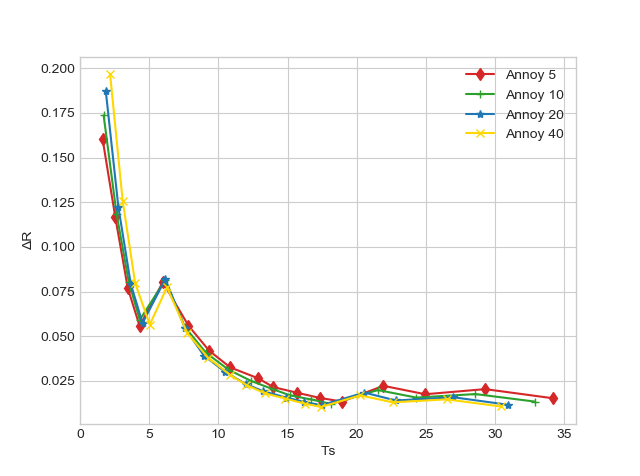
Илустрација : *Annoy -* Број стабала и перформансе

Може се претпоставити да је у наведеном случају коришћено много више стабала него што је засита потребно, па сличан бој стабала има готово истоветне резултате. Наведену претпоставку проверавамо искуственим путем на мањем броју стабала. При нижем одзиву, све варијанте алгоритма са различитим бројм стабала, постижу сличне перформансе, а исти ефекат јавља се и у претходно описаном случају. Са порастом одзива долази до раслојавања и алгоритми са мањим бројем стабла имају дуже време претраге – одзив од 0,9 је постигнут за 20 секунди код експеримента са 40 стабала, што је скоро 10 секунди брже експеримента са 5 стабала.



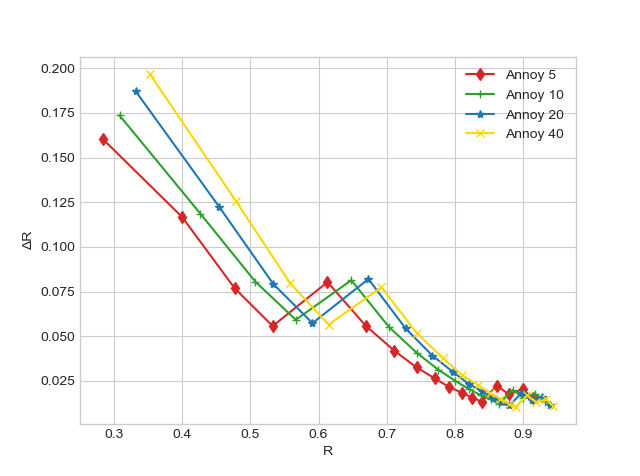
Илустрација 13: *Annoy*- раслојавање

Маргинални одзив се смањује са дужином претраге. Дијаграм показује да је маргинални одзив незнатно нижи код већег броја стабала при дужем времену претраге.



Илустрација 14: *Annoy-* маргинални одзив и време претраге

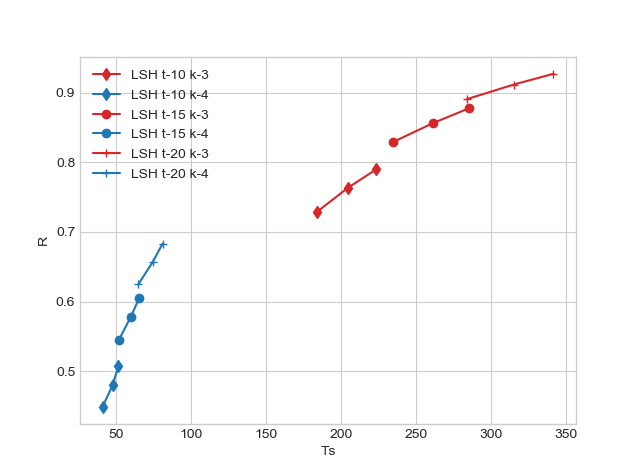
Када се маргинални одзив упореди са постигнутим одзивом (Илустрација 15), јасно је да су алгоритми са већим бројем стабала у предности.



Илустрација 15:*Annoy* - Маргинални одзив и одзив

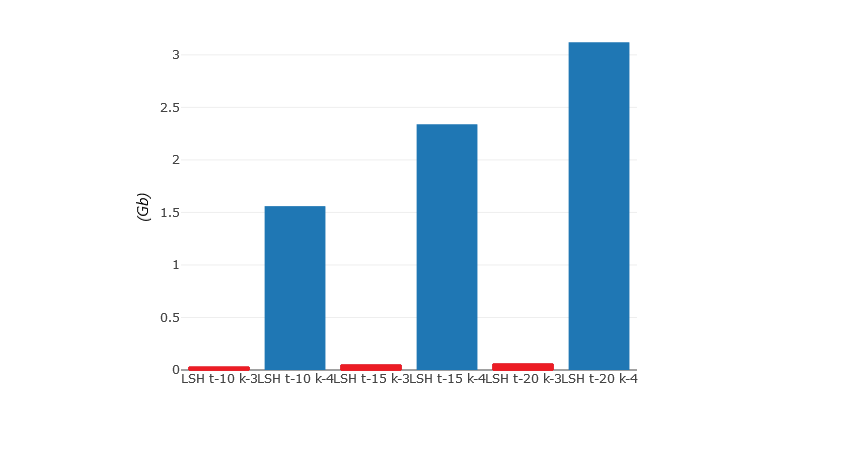
Скокови на дијаграму су последица методолошког поступка. Параметар *search\_k је* у првих пет итерација увећаван за 500, а у следећим итерацијама се увећавао за 1000 због бржег добијања резултата.

Утицај броја табела *t* и хеш функција *k* код *LSH* приказан је на Илустрацији 16. Резултати се групишу у односу на величину параметра *k.* Плавом бојом означене су варијанте у којима је *k = 4*, а црвеном када је *k =3.* Из приказаног се закључује да параметар има висок утицај на одзив, јер нижи број хеш функција резултира мањем броју бакета који постају гломазни. Величина бакета утиче на раст времена претраге, али је чини прецизнијом јер се испитује велики број кандидата. Ипак, време претраге од 200 секунди је неприхватљиво када имамо у виду резултате других алгоритама, па би даљу требало наставити са параметром *k = 4*. Виши одзив се може постићи повећањем броја „суседних“ бакета који бити претражени.



Илустрација : *LSH -* утицај параметара на перформансе

Параметар *k* не утиче само на одзив и време претраге, већ и на меморијске захтеве *LSH*.



Илустрација : *LSH*- меморија

## Експеримети над *FASHION* *MINST*

## Препоруке за одабир алгоритма

# Закључак

Мало апстракт мало сумирање резултата

# Референце

Abbasifard, M. R., Ghahremani, B., & Naderi, H. (2014). A survey on nearest neighbor search methods. *International Journal of Computer Applications, 95*, 39-52.

Arya, S., Mount, D. M., & Narayan, O. (1996). Accounting for boundary effects in nearest-neighbor searching. *Discrete & Computational Geometry, 16*, стр. 155-176.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Јournal of the ACM (JACM), 45*, 891-923.

Douze, M., Sablayrolles, A., & Jégou, H. (2018). Link and code: Fast indexing with graphs and compact regression codes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (стр. 3646-3654).

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. *Vldb*, *99*, стр. 518-529. Edinburgh.

Han, J., Pei, J., & Kamber, M. (2012). *Data mining: concepts and techniques* (3rd изд.). Waltham, Massachusetts, United States of America: Elsevier.

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. ACM.

Kleinberg, J. M. (1997). Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (стр. 599--608). El Paso,Texas, USA: АCM.

Kushilevitz, E., Ostrovsky, R., & Rabani, Y. (2000). Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing, 30*, 457-474.

Maillo, J., Ramirez, S., Triguero, I., & Herrera, F. (2017). kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems, 117*, 3-15.

Marimont, R., & Shapiro, M. (1979). Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics*, стр. 59-70.

Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 331-340.

Samet, H. (2006). *Foundations of multidimensional and metric data structures.* (J. Gray, Ур.) San Francisco, California, United States of America: Morgan Kaufmann.

Seidl, T., & Kriegel, H.-P. (1998). Optimal Multi-Step k-Nearest Neighbor Search. *27.* Seattle: ACM SIGMOD Record.

Triguero, I., Maillo, J., Luengo, J., García, S., & Herrera, F. (2016). From big data to smart data with the k-nearest neighbours algorithm.

*UCI Machine Learning Repository*. (n.d.). Преузето са University of California, Irvine: https://archive.ics.uci.edu/ml/datasets.html

Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. *e 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, (стр. 311-321). Austin, USA.

# Прилози

Прилог 1:Почени резултати на *SIFT1M* подацима

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритам | Tp | Ts | R | Просечна удаљеност |
| Annoy-trees-5 | 35.93 | 1.7853 | 0.2835 | 254.7 |
| Annoy-trees-15 | 62.39 | 4.0106 | 0.5221 | 243.33 |
| Annoy-trees-30 | 102.71 | 7.0456 | 0.6841 | 239.47 |
| Annoy-trees-60 | 185.83 | 11.5536 | 0.8158 | 237.4 |
| Annoy-trees-80 | 244.18 | 14.5923 | 0.8574 | 236.89 |
| HNSW-M-5 | 633.89 | 1.1619 | 0.3629 | 251.36 |
| HNSW-M-8 | 856.31 | 1.4383 | 0.4822 | 244.77 |
| HNSW-M-15 | 1347.66 | 2.0843 | 0.6244 | 240.4 |
| HNSW-M-30 | 2519.09 | 3.1523 | 0.7668 | 237.86 |
| HNSW-M-38 | 3269.57 | 3.627 | 0.8064 | 237.35 |
| vp-Tree-10k-mL2 | 17.5 | 15.7579 | 0.3673 | 252.0 |
| vp-Tree-10k-mL10 | 17.5 | 76.9463 | 0.6541 | 241.96 |
| vp-Tree-10k-mL20 | 17.5 | 151.8395 | 0.8051 | 238.83 |
| vp-Tree-10k-mL25 | 17.5 | 189.0328 | 0.8444 | 238.17 |
| vp-Tree-10k-mL30 | 17.5 | 225.437 | 0.8681 | 237.83 |
| linear | 0.09 | 515.1839 | 0.9999 | 235.63 |
| k-D | 30.08 | 4317.1864 | 0.9999 | 235.63 |
| autotuned-flann-build005 | 463.1 | 1.4064 | 0.1831 | 277.67 |
| autotuned-flann-build005 | 445.64 | 1.3903 | 0.1811 | 278.17 |
| autotuned-flann-build005 | 454.99 | 2.6843 | 0.246 | 260.7 |
| autotuned-flann-build005 | 753.91 | 6.0138 | 0.4294 | 246.94 |
| lsh-l15k4t15 | 132.35 | 44.8974 | 0.5454 | 0.0 |
| lsh-l15k4t30 | 132.35 | 68.6406 | 0.6679 | 0.0 |
| lsh-l15k4t45 | 132.35 | 86.9347 | 0.7362 | 0.0 |
| lsh-l15k4t75 | 132.35 | 114.1396 | 0.8125 | 0.0 |

Прилог 2: *Annoy* - меморијски захтеви

