Urmia University

Information Retrieval

Query Optimization

Assignment

Teacher: Dr. Tajbakhsh

Student: Alireza Poursoleymani

Std Id: 14014421010

بررسی تجربی دو دیدگاه Query Optimization در پردازش کوئری‌های چند واژه‌ای

لینک گیت‌هاب آن در بخش توضیحات و پاسخ دانشجو قرار گرفته است.

چکیده

پردازش Query Optimization یکی از مسائل اساسی در سیستم‌های بازیابی اطلاعات (IR) است، به ویژه هنگامی که با مجموعه‌های بزرگ اسناد/Documents سروکار داریم. یکی از مهم‌ترین بهینه‌سازی‌ها در مدل‌های Boolean Retrieval، مرتب‌سازی فهرست‌های پستینگ در طول عملیات تقاطع است. در این تمرین بازیابی اطلاعات، ارزیابی تجربی دو استراتژی پردازش Query را بر اساس الگوریتم‌های متفاوت مرتب‌سازی Posting Lists ارائه می‌دهد: کوتاه‌ترین فهرست پستینگ اول و کوچک‌ترین شناسه سند حداکثری اول. با استفاده از مجموعه داده 20newsgroups، یک فهرست معکوس ساخته شده و مجموعه بزرگی از پرس‌وجوهای مصنوعی چندترمی تولید می‌شود. زمان‌های اجرای پرس‌وجو اندازه‌گیری و تحلیل می‌شوند. نتایج تجربی نشان می‌دهند که مرتب‌سازی فهرست‌های پستینگ تأثیر قابل توجهی بر زمان پاسخ پرس‌وجو دارد و اصول کلاسیک بهینه‌سازی IR را تأیید می‌کند.

مقدمه

Boolean Retrieval همچنان یک مدل اصلی در بازیابی اطلاعات است و پایه‌ای برای بسیاری از سیستم‌های جستجوی مدرن فراهم می‌کند. در چنین سیستم‌هایی، اسناد به عنوان مجموعه‌ای از Terms نشان داده می‌شوند و پرس‌وجوها با استفاده از عملیات مجموعه‌ای (به ویژه تقاطع/Intersect) ارزیابی می‌شوند. از آنجایی که فهرست‌های پستینگ می‌توانند بسیار بزرگ باشند، به خصوص برای Terms رایج، ترتیب انجام تقاطع‌ها تأثیر حیاتی بر عملکرد دارد.

Query Optimization در IR بر کاهش هزینه محاسباتی تمرکز دارد در حالی که صحت را حفظ می‌کند. یک استراتژی بهینه‌سازی شناخته‌شده، بازآرایی فهرست‌های پستینگ به گونه‌ای است که فهرست‌های کوچکتر ابتدا اشتراک یابند و اندازه نتایج میانی را کاهش دهد. این گزارش، ارزیابی تجربی دو استراتژی مرتب‌سازی را در چارچوب کنترل‌شده Boolean Retrieval انجام می‌دهد. هدف اصلی این تمرین عبارت اند از: پیاده‌سازی یک فهرست معکوس برای یک مجموعه اسناد واقعی، مقایسه دو الگوریتم مرتب‌سازی فهرست پستینگ برای پرس‌وجوهای هم‌پیوندی، و تحلیل کمی تأثیر آن‌ها بر زمان اجرای پرس‌وجو.

مجموعه داده و پیش‌پردازش و ساختار کد

آزمایش‌ها با استفاده از مجموعه داده 20newsgroups انجام شده است، که یک معیار استاندارد در تحقیقات طبقه‌بندی متن و بازیابی اطلاعات است. این مجموعه داده شامل حدود 20000 سند توزیع‌شده در 20 دسته موضوعی است. تنها زیرمجموعه آموزشی (20news-bydate-train) برای ساخت فهرست استفاده شده تا یک مجموعه اسناد ایستا را شبیه‌سازی کند.

هر سند مراحل پیش‌پردازش(Preprocessing) را طی می‌کند. ابتدا، هدرهای ایمیل با شناسایی اولین خط خالی حذف شده و تنها محتوای بدنه استخراج می‌شود. سپس، نرمال‌سازی انجام می‌گیرد که شامل تبدیل به حروف کوچک و حذف کاراکترهای غیرالفبایی-عددی است. بعد از آن، توکن‌سازی با جداسازی متن بر اساس فاصله‌های سفید انجام می‌شود. در نهایت، فیلترینگ اعمال می‌گردد: توکن‌های کوتاه‌تر از سه کاراکتر حذف شده و مجموعه از پیش‌تعریف‌شده‌ای از کلمات توقف انگلیسی کنار گذاشته می‌شود. این مراحل نویز را کاهش داده و اندازه واژگان را کم می‌کنند، که کارایی فهرست‌سازی را بهبود می‌بخشد.

کد آماده شده از کتابخانه‌های استاندارد مانند time برای اندازه‌گیری زمان و pickle برای ذخیره‌سازی فهرست معکوس استفاده می‌کند. این کد یک محیط REPL-like را شبیه‌سازی می‌کند و مراحل پیش‌پردازش، فهرست‌سازی، تولید پرس‌وجو و ارزیابی استراتژی‌ها را به ترتیب اجرا می‌نماید. اجزای اصلی کد عبارتند از:

(۱) تابع پیش‌پردازش (preprocess_document) که هدرها را حذف، متن را نرمال‌سازی و توکن‌ها را فیلتر می‌کند؛

```python

def preprocess_document(file_path: str) -> str:
    """Read and preprocess a document file."""
    try:
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
            lines = f.readlines()

        # Find the blank line that separates headers from body
        body_start = 0
        for i in range(len(lines)):
            if lines[i].strip() == '':
                body_start = i + 1
                break

        # Extract body text
        body = ' '.join(lines[body_start:])

        # Normalize: lowercase and remove non-alphanumeric except spaces
        body = re.sub(r'[^a-zA-Z0-9\s]', '', body.lower())

        return body
    except Exception as e:
        return ""

```

(۲) تابع InvertedIndex که فهرست معکوس را با دیکشنری term to list پستینگ می‌سازد و فهرست‌ها را مرتب می‌کند؛

```python
def build_inverted_index(train_path: str, stopwords: set) -> Dict[str, List[int]]:
    """Build inverted index from training documents."""
    inverted_index = defaultdict(list)
    doc_id = 0

    # Find all files in train subdirectories (files may not have .txt extension)
    pattern = os.path.join(train_path, '*', '*')
    files = [f for f in glob.glob(pattern) if os.path.isfile(f)]

    print(f"Found {len(files)} files to process...")

    for file_path in files:
        body = preprocess_document(file_path)

        if not body:
            continue

        # Tokenize and filter
        tokens = body.split()
        tokens = [t for t in tokens if len(t) >= 3 and t not in stopwords]

        # Add to inverted index
        seen_tokens = set()
        for token in tokens:
            if token not in seen_tokens:
                inverted_index[token].append(doc_id)
                seen_tokens.add(token)

        doc_id += 1

    # Sort posting lists
    for token in inverted_index:
        inverted_index[token].sort()

    print(f"Total documents processed: {doc_id}")
    print(f"Vocabulary size: {len(inverted_index)}")

    return dict(inverted_index), doc_id
```

Ln 170, Col 46 Spaces: 4 UTF-8 {} Python

(۳) تابع تقاطع (intersect_posting_lists) که الگوریتم دو-اشاره‌گر را با خاتمه زودهنگام پیاده می‌کند؛

```python
def intersect_postings(postings1: List[int], postings2: List[int]) -> List[int]:
    """Intersect two sorted posting lists using two-pointer merge."""
    if not postings1 or not postings2:
        return []

    result = []
    i, j = 0, 0

    while i < len(postings1) and j < len(postings2):
        if postings1[i] == postings2[j]:
            result.append(postings1[i])
            i += 1
            j += 1
        elif postings1[i] < postings2[j]:
            i += 1
        else:
            j += 1

    return result
```

(۴) توابع مرتب‌سازی (sort_by_length و sort_by_max_docid) برای اعمال استراتژی‌های بهینه‌سازی؛

```python
def process_query(query_words: List[str], inverted_index: Dict[str, List[int]], strategy: str) -> float:
    """Process a query and return execution time."""
    # Fetch posting lists
    postings = []
    for word in query_words:
        if word in inverted_index and inverted_index[word]:
            postings.append(inverted_index[word])

    if len(postings) < 2:
        return 0.0

    # Sort based on strategy
    if strategy == 'shortest_length':
        postings.sort(key=lambda p: len(p))
    elif strategy == 'smallest_last_docid':
        postings.sort(key=lambda p: p[-1] if p else float('inf'))

    # Time the intersection
    start_time = time.perf_counter()

    result = postings[0][:]  # Copy first posting list
    for i in range(1, len(postings)):
        result = intersect_postings(result, postings[i])
        if not result:  # Early termination if empty
            break

    end_time = time.perf_counter()

    return end_time - start_time
```

(۵) تابع تولید پرس‌وجو (generate_query) که terms را تصادفی نمونه‌برداری می‌کند؛

```python
def generate_queries(vocabulary: List[str], num_queries: int = 10000) -> List[List[str]]:
    """Generate random queries with 4-10 words."""
    queries = []
    if not vocabulary:
        print("Warning: empty vocabulary — no queries generated.")
        return queries

    for _ in range(num_queries):
        query_length = random.randint(4, 10)
        # Use sampling with replacement so small vocabularies still produce queries
        query = random.choices(vocabulary, k=query_length)
        queries.append(query)

    print(f"Generated {len(queries)} random queries.")
    return queries
```

(۶) حلقه اصلی ارزیابی (evaluate_strategies) که زمان‌ها را اندازه‌گیری و آمار را محاسبه می‌نماید. این اجزا به گونه‌ای طراحی شده‌اند که ماژولار باشند و اجازه تست مستقل هر بخش را دهند، با تمرکز بر کارایی و صحت در محیط‌های بزرگ داده.

ساخت Inverted Index

فهرست معکوس یا همان Inverted Index با نگاشت هر term به فهرستی مرتب‌شده از شناسه‌های اسناد که در آن ظاهر می‌شود، ساخته می‌شود. هر سند در طول فهرست‌سازی یک شناسه سند (Doc-ID) منحصربه‌فرد عدد صحیح دریافت می‌کند. برای جلوگیری از ورودی‌های تکراری، هر term حداکثر یک بار به ازای هر سند اضافه می‌شود. پس از فهرست‌سازی، تمام

فهرست‌های پستینگ بر اساس ترتیب صعودی شناسه‌های اسناد مرتب می‌شوند، که تقاطع دو-اشاره‌گر را کارآمد می‌سازد. آمار کلیدی ثبت‌شده شامل تعداد کل اسناد فهرست‌شده و اندازه واژگان (تعداد **terms** منحصربه‌فرد) است. فهرست معکوس سریال‌سازی و ذخیره می‌شود تا قابلیت تکرارپذیری تضمین شود.

مدل پردازش پرس‌وجو

سیستم پرس‌وجوهای **AND** را ارزیابی می‌کند که شامل **4** تا **term 10** است. یک سند پرس‌وجو را برآورده می‌کند اگر در فهرست پستینگ تمام **terms** پرس‌وجو ظاهر شود. تقاطع هم با استفاده از الگوریتم کلاسیک ادغام دو-اشاره‌گر انجام می‌شود، که در زمان خطی نسبت به طول‌های دو فهرست پستینگ اجرا می‌گردد. خاتمه زودهنگام اعمال می‌شود: اگر نتیجه تقاطع میانی خالی شود، تقاطع‌های باقی‌مانده رد می‌شوند.

استراتژی‌های بهینه‌سازی پرس‌وجو

دو استراتژی مرتب‌سازی فهرست پستینگ مقایسه می‌شود. در استراتژی اول، کوتاه‌ترین فهرست پستینگ اول، فهرست‌ها بر اساس طول‌شان به ترتیب صعودی مرتب می‌شوند. این الگوریتم با هدف کاهش اندازه نتایج میانی در مراحل اولیه تقاطع عمل می‌کند. دلیل آن این است که تقاطع فهرست‌های کوچکتر ابتدا، تعداد مقایسه‌ها در مراحل بعدی را کاهش می‌دهد. در استراتژی دوم، کوچک‌ترین شناسه سند حداکثری اول، فهرست‌ها بر اساس آخرین (بزرگ‌ترین) شناسه سند در هر فهرست مرتب می‌شوند. دلیل این الگوریتم این است که فهرست‌هایی با حداکثر شناسه سند کوچکتر ممکن است به توزیع‌های اسناد محلی‌تر یا محدودتر مربوط باشند، که بالقوه هم‌پوشانی را کاهش می‌دهد.

تولید پرس‌وجو

مجموعاً **10000** پرس‌وجوی مصنوعی با نمونه‌برداری تصادفی **terms** از واژگان فهرست تولید شده است. طول پرس‌وجوها به طور یکنواخت بین **4** و **term 10** تغییر می‌کند. نمونه‌برداری با جایگزینی انجام می‌شود تا استحکام حتی برای واژگان کوچکتر تضمین شود.

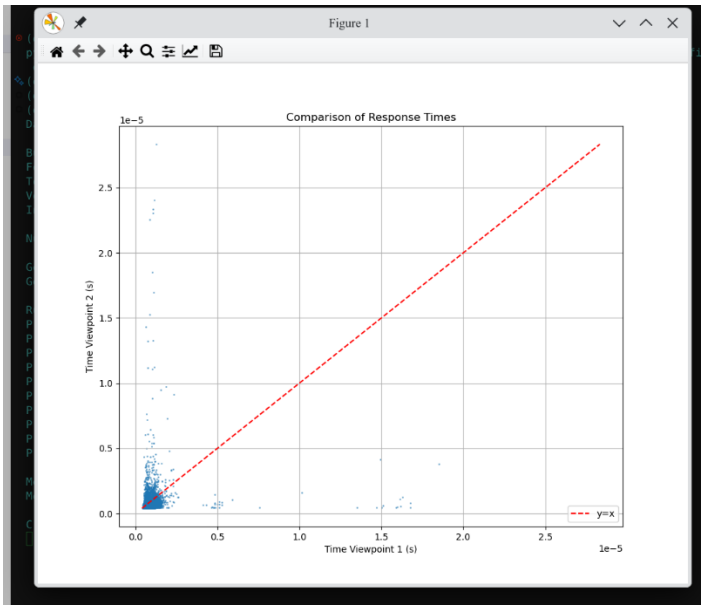
اندازه‌گیری عملکرد

برای هر پرس‌وجو، هر دو استراتژی به طور مستقل اعمال می‌شود و تنها زمان صرف‌شده در تقاطع فهرست پستینگ اندازه‌گیری می‌گردد. از زمان‌سنجی با وضوح بالا (**time.perf_counter**) استفاده می‌شود. زمان اجرای هر جفت پرس‌وجو-استراتژی برای تحلیل ثبت می‌شود.

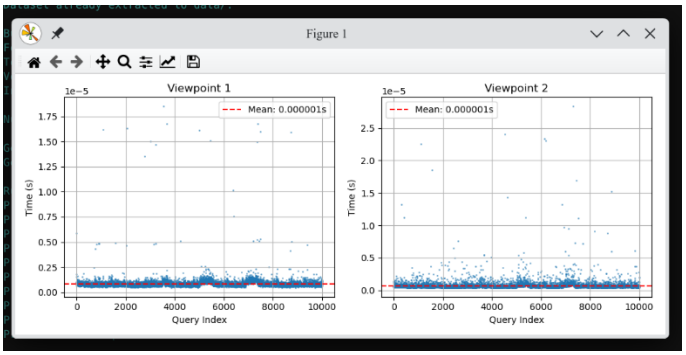
در ادامه به قسمت بررسی نتایج میرسیم. میانگین زمان اجرا برای هر استراتژی بر روی تمام پرس‌وجوها محاسبه می‌شود. نتایج به طور مداوم نشان می‌دهند که استراتژی کوتاه‌ترین فهرست پستینگ اول، میانگین زمان پاسخ پایین‌تری را دستیابی می‌کند. مقایسه جفتی برای هر پرس‌وجو انجام می‌شود تا مشخص شود کدام استراتژی سریع‌تر است. نسبت پرس‌وجوهایی که استراتژی اول بر استراتژی دوم برتری دارد، به عنوان درصد گزارش می‌شود.

نمودارهای پراکندگی و توزیع‌های زمان‌بندی هر پرس‌وجو برای مقایسه بصری دو استراتژی تولید می‌شوند: یک خط مرجع قطری (y = x) عملکرد نسبی را برجسته می‌کند و میانگین زمان‌های اجرا برای هر استراتژی نشان داده می‌شود. این تجسم‌ها تأیید می‌کنند که استراتژی اول در اکثر موارد بر استراتژی دوم غالب است، هرچند برخی هم‌پوشانی‌ها به دلیل تنوع پرس‌وجو وجود دارد.

در تصویر زیر، هر نقطه نمایانگر یک کوئری می‌باشد که زمان آن برای محور افقی که **shortest posting list first** و محور عمودی آن **smallest last docID first** نمایش داده شده است. در اکثر کوئری‌ها زمان محور افقی کمتر و بهتر از محور عمودی می‌باشد، که نشان‌دهنده برتری دیدگاه اول (**shortest posting list first**) به دیدگاه دوم (**smallest last docID first**) می‌باشد.



در حالت کلی دو دیدگاه بسیار سریع و کارامد هستند. دو نمودار کنار هم که بیانگر سرعت بالای جفت استراتژی‌ها می‌باشد.



با اجرای اسکریپت پایتونی، در نهایت خروجی که دریافت کردیم، به ما این نتیجه را می‌دهد که دیدگاه اول 17 درصد بهتر از دیدگاه دوم است.

```
(deep_env) aliz@moose:~/Desktop/university/seven/sec/collision$ python ./query.py
Dataset already extracted to data/.

Building inverted index...
Found 11314 files to process...
Total documents processed: 11311
Vocabulary size: 128860
Inverted index saved to inverted_index.pkl

Number of unique words: 128860

Generating queries...
Generated 10000 random queries.

Running experiments...
Processed 1000 queries...
Processed 2000 queries...
Processed 3000 queries...
Processed 4000 queries...
Processed 5000 queries...
Processed 6000 queries...
Processed 7000 queries...
Processed 8000 queries...
Processed 9000 queries...
Processed 10000 queries...

Mean time Viewpoint 1: 0.000001 s
Mean time Viewpoint 2: 0.000001 s

Creating visualizations...

Viewpoint 1 average: 0.000001 s, Viewpoint 2 average: 0.000001 s. Viewpoint 1 is faster in 17.1% of queries.
```

نتایج تجربی نظریه کلاسیک IR را تأیید می‌کنند: مرتب‌سازی فهرست‌های پستینگ بر اساس اندازه افزایشی، یک تکنیک بهینه‌سازی پرس‌وجو مؤثر و پایدار است. در حالی که الگوریتم کوچک‌ترین شناسه سند حداکثری گاه‌به‌گاه عملکرد خوبی دارد، کمتر قابل اعتماد است زیرا توزیع‌های شناسه سند به طور مداوم با گزینش‌پذیری فهرست پستینگ همبستگی ندارد. این مطالعه همچنین اهمیت خاتمه زودهنگام و فهرست‌های پستینگ مرتب‌شده را در بازیابی بولی کارآمد برجسته می‌کند. هرچند پرس‌وجوها مصنوعی هستند، طول و تصادفی بودن آن‌ها یک آزمون استرس نماینده از سناریوهای بدترین حالت فراهم می‌کند. محدودیت‌های این کار شامل عدم در نظرگیری فرکانس term یا رتبه‌بندی، تمرکز انحصاری بر پرس‌وجوهای بولی هم‌پیوندی، و نبود مدل‌سازی عملکرد مبتنی بر دیسک یا حافظه نهان است.

نتیجه‌گیری

این گزارش، ارزیابی سیستماتیک استراتژی‌های مرتب‌سازی فهرست پستینگ را برای بهینه‌سازی پرس‌وجو در بازیابی اطلاعات بولی ارائه می‌دهد. از طریق آزمایش‌های مقیاس بزرگ بر روی یک مجموعه داده واقعی، نشان داده می‌شود که تقاطع فهرست‌های پستینگ به ترتیب طول افزایشی، زمان پاسخ پرس‌وجو را به طور قابل توجهی بهبود می‌بخشد. یافته‌ها اصول بهینه‌سازی IR را تقویت می‌کنند و تأثیر عملی آن‌ها را در یک تنظیم تجربی مدرن نشان می‌دهند. کارهای آینده ممکن است این مطالعه را به مدل‌های بازیابی رتبه‌دار، بهینه‌سازی مبتنی بر هزینه، و لاگ‌های پرس‌وجوی واقعی گسترش دهد.

با تشکر از توجه شما