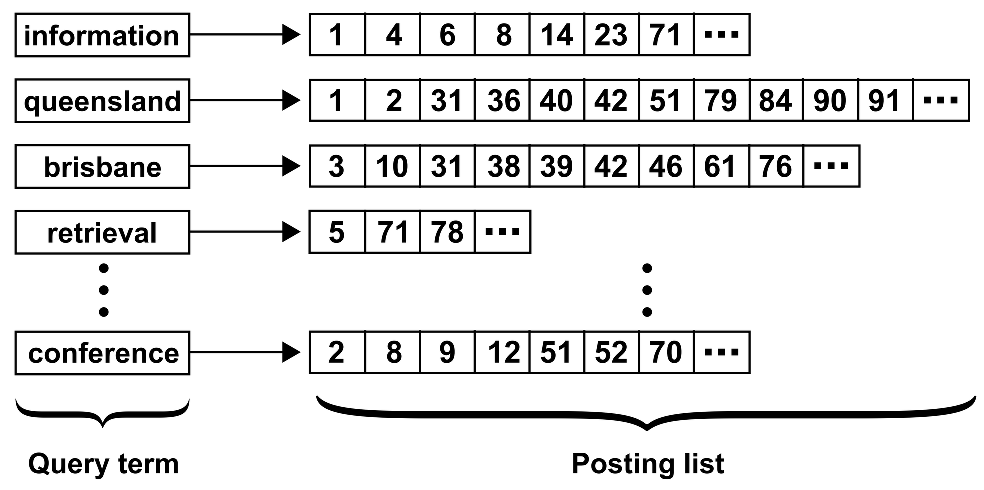
The Impact of using Combinatorial Optimisation for Static Caching of Posting Lists

Casper Petersen, Jakob Grue Simonsen, Christina Lioma
University of Copenhagen, Department of Computer Science
{cazz, simonsen, c.lioma}@di.ku.dk

UNIVERSITY OF COPENHAGEN

Introduction

- A *cache* facilitates query throughput and fast response times
- Caching *posting lists* (PL) can reduce the amount of disk I/O involved [6, 7]:



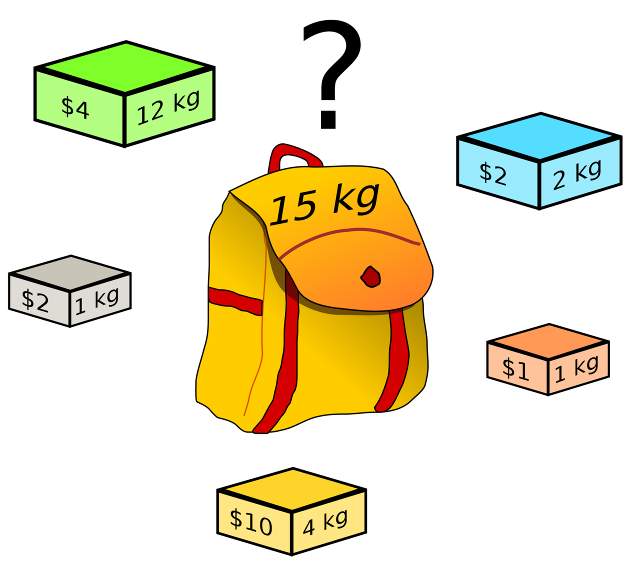
- PL caching offers higher cache utilisation and hit rates than result caching [5]
- Web search engines dedicate a large amount of main memory to PL caching [8]
- **How to select the terms/posting lists to be cached?**

Research Question

To what extent can constrained combinatorial optimisation (CCO) be used to optimally select posting lists for static caching, compared to strong baselines?

Problem Description

Selecting posting lists is a **0-1** KNAPSACK PROBLEM [2]:



Given n items $x_i : 1 \leq i \leq n$ with values v_i and weights w_i, and a knapsack with capacity c, select the subset $\hat{x} \subseteq x$ items that maximise the total value without exceeding the knapsack's capacity

Methods For Selecting Posting List

Greedy (Baseline):

- Approach by Baeza-Yates et al. [2]

  Step 1. $f_q(t)$: Frequency of query term t (*query term popularity*)

  Step 2. $f_d(t)$: Number of documents where t occurs (*posting list length*)

  Step 3. Sort in decreasing order of $\frac{f_q(t)}{f_d(t)}$ and load into cache

- A variation of the *profit-to-weight ratio* approach by Dantzig [4]

The 0-1 knapsack problem is NP-hard and cannot, in general, be solved optimally using a greedy approach [3, Chap. 16]

Constrained Combinatorial Optimisation (CCO):

Posting list selection as an *integer linear program*:

$$\begin{array}{rll} \max & \sum_{i=1}^{n} v_i x_i & \textit{(Objective function)} \\ \text{subject to} & \sum_{i=1}^{n} w_i x_i \leq c & \textit{(Total weight constraint)} \\ & x_i \in \{0,1\}, 1 \leq i \leq n & \textit{(Binary decision variables)} \end{array}$$

CCO theoretically guarantees an approximately optimal solution to a 0-1 KNAPSACK PROBLEM

Simulating Queries

Construct *simulated query logs* [1]

- Let t_i be a term and d_k a document in some collection
- $p\left(t_i|\Theta_m^{d_k}\right) = (1-\lambda) \cdot p\left(t_i|d_k\right) + \lambda \cdot p(t_i)$
- $p\left(t_i|d_k\right)$ is one of two language models (LM):
  - **Popular LM**...........: $p(t_i|d_k) = \text{tf}(t_i, d_k) / \sum_{t_j \in d_k} \text{tf}(t_j, d_k)$
    where $\text{tf}(t_j, d_k)$: term frequency of t_j in d_k
  - **Discriminative LM**: $p(t_i|d_k) = b(t_j, d_k)/p(t_i) \cdot \sum_{t_j \in d_k} b(t_j, d_k)/p(t_j)$
    where $b(t_j, d_k) = 1$ if $t_j \in d_k$
- $p\left(t_i\right)$ is the maximum likelihood probability of t_i in the collection

Experimental Setup

- Cache sizes of 200, 600 and 1000 MB
- ClueWeb09 cat. B. as collection (no stemming, stop word removed)
- 1M, 5M and 10M query logs, with queries of length $l = 1, 2, 3$
- COIN-OR SYMPHONY mixed integer linear programming solver
- Estimate $v_i = f_q(t)$ from each simulated query log and $w_i = f_d(t)$ from the collection
- Cache hit for a query iff *at least one* query term is found in the cache

Results

- **OC** is the cache term *overlap coefficient* $= \frac{|X \cap Y|}{\min(|X|,|Y|)}$
- **CT** is the number of *cache terms* loaded into each cache
- **CH** is the number of *cache hits* in the caches
- **Diff** is the difference in CH. **A positive Diff favours CCO**

		Simulated 5M					
		Discriminative			**Popular**		
		qlen=1	qlen=2	qlen=3	qlen=1	qlen=2	qlen=3
OC	200M	0.851	0.884	0.923	0.990	0.973	**0.977**
	600M	0.962	**0.934**	0.899	0.997	**0.987**	**0.999**
	1000M	0.952	**0.952**	**0.942**	**0.995**	**0.994**	**0.998**
CT	200M	24,938 / 24,951	24,922 / 24,920	24,973 / 24,974	9,311 / 9,310	13,799 / 13,782	**16,841 / 16,845**
	600M	74,648 / 74,483	**74,725 / 74,740**	74,780 / 74,752	13,804 / 13,803	**21,483 / 21,473**	**27,568 / 27,557**
	1000M	114,269 / 115,850	**122,655 / 123,358**	**124,525 / 124,546**	**16,309 / 16,307**	**25,704 / 25,696**	**33,194 / 33,209**
CH	200M	217,626 / 217,626	228,183 / 228,266	228,692 / 228,877	19,185 / 19,185	28,655 / 28,671	**35,242 / 35,238**
	600M	546,566 / 546,566	**596,812 / 596,790**	600,376 / 600,733	28,537 / 28,537	**44,579 / 44,573**	**57,352 / 57,351**
	1000M	765,793 / 765,793	**859,142 / 857,718**	**880,740 / 880,132**	**33,768 / 33,767**	**53,531 / 53,529**	**69,370 / 69,366**
DIFF	200M	0	-83	-185	0	-16	**4**
	600M	0	**22**	-357	0	**6**	**1**
	1000M	0	**1,424**	**608**	**1**	**2**	**4**
		Simulated 10M					
		Discriminative			**Popular**		
		qlen=1	qlen=2	qlen=3	qlen=1	qlen=2	qlen=3
OC	200M	0.949	0.957	0.865	0.961	**0.929**	0.977
	600M	0.890	**0.909**	**0.885**	0.989	0.982	**0.985**
	1000M	**0.910**	**0.891**	0.957	**0.999**	**0.989**	**0.999**
CT	200M	24,988 / 24,958	24,955 / 24,926	24,954 / 24,935	13,892 / 13,886	**19,404 / 19,335**	22,775 / 22,799
	600M	74,589 / 74,537	**74,468 / 74,591**	**74,642 / 74,562**	21,623 / 21,614	32,962 / 32,919	**41,095 / 41,170**
	1000M	**124,251 / 124,127**	**123,925 / 124,091**	124,632 / 124,350	**25,872 / 25,876**	**40,122 / 40,096**	**51,247 / 51,247**
CH	200M	240,056 / 240,056	246,439 / 246,440	246,117 / 246,293	28,634 / 28,634	**40,152 / 40,145**	48,251 / 48,258
	600M	629,405 / 629,405	**662,230 / 662,148**	**664,890 / 664,515**	44,710 / 44,710	68,199 / 68,203	**86,191 / 86,154**
	1000M	**957,964 / 957,963**	**1,033,190 / 1,033,109**	1,044,362 / 1,044,852	**53,631 / 53,629**	**83,379 / 83,374**	**107,333 / 107,332**
DIFF	200M	0	-1	-176	0	**7**	-7
	600M	0	**82**	**375**	0	-4	**37**
	1000M	**1**	**81**	-490	**2**	**5**	**1**

Table 1: Results for the Discriminative and Popular 5M and 10M query logs. *x/y* means *CCO / baseline*.

Conclusions

- Both methods perform similarly. Modest gains for the CCO method for queries of length $l = 2, 3$
- Overlap coefficients suggest cache hit differences be attributed to a small set of infrequent terms
- Quality of solution depends on the problem, the solver and the settings of the solver's parameters

References

[1] Leif Azzopardi, Maarten de Rijke, and Krisztian Balog. Building simulated queries for known-item topics: an analysis using six european languages. In *SIGIR*, pages 455–462, 2007.

[2] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. The impact of caching on search engines. In *SIGIR*, pages 183–190. ACM, 2007.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press Cambridge, 2001.

[4] George B. Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[5] Myron Papadakis and Yannis Tzitzikas. Answering keyword queries through cached subqueries in best match retrieval models. *JIIS*, pages 1–40, 2014.

[6] Paricia C. Saraiva, Edleno Silva de Moura, Novio Ziviani, Wagner Meira, Rodrigo Fonseca, and Berthier Riberio-Neto. Rank-preserving two-level caching for scalable search engines. In *SIGIR*, pages 51–58. ACM, 2001.

[7] Gabriel Tolosa, Luca Becchetti, Esteban Feuerstein, and Alberto Marchetti-Spaccamela. Performance improvements for search systems using an integrated cache of lists+intersections. In *SPIRE*, pages 227–235. 2014.

[8] Jiangong Zhang, Xiaohui Long, and Torsten Suel. Performance of compressed inverted list caching in search engines. In *WWW*, pages 387–396. ACM, 2008.