

March 2015

Evaluating Multiple Caching Strategies for Semantic Network Applications

John Davies French
Worcester Polytechnic Institute

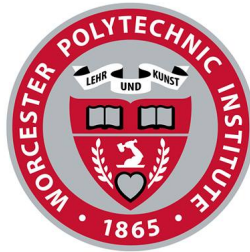
Steven Mark Malis
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

Repository Citation

French, J. D., & Malis, S. M. (2015). *Evaluating Multiple Caching Strategies for Semantic Network Applications*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/1754>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.



WPI

Evaluating Multiple Caching Strategies for Semantic Network Applications

A Major Qualifying Project Report submitted to the faculty of
Worcester Polytechnic Institute

In partial fulfillment of the requirements for the degree of Bachelor of Science
by:

John French (Computer Science),

Steven Malis (Computer Science)

March 27, 2015

Submitted to:
Professor Sonia Chernova, WPI Advisor

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Background | 2 |
| 2.1 | ConceptNet | 2 |
| 2.2 | Caching | 2 |
| 2.3 | Semantic Similarity Engine and Topological Topic Modeler | 3 |
| 3 | Methodology | 4 |
| 3.1 | Test Data Set | 4 |
| 3.2 | Performance Metrics | 5 |
| 3.3 | Caching Strategies | 5 |
| 3.4 | Testing Procedure | 7 |
| 3.5 | Cache Implementation Details | 7 |
| 4 | Results | 8 |
| 4.1 | Runtime | 8 |
| 4.2 | Memory | 9 |
| 4.3 | Number of Requests Made | 12 |
| 4.4 | Cache Hit Ratio | 13 |
| 4.5 | Request Times | 15 |
| 4.6 | Analysis | 16 |
| 5 | Conclusion | 17 |
| 6 | Bibliography | 18 |
| A | Word Frequency in Test Data Set | 20 |
| B | English Word Frequency | 23 |

List of Figures

| | | |
|---|--|----|
| 1 | A simple example of ConceptNet data | 3 |
| 2 | Average topic runtime for each caching strategy | 9 |
| 3 | Average analogy runtime for each caching strategy | 10 |
| 4 | Memory usage over time for each caching strategy | 10 |
| 5 | Maximum memory usage for each caching strategy | 11 |
| 6 | The average number of requests made while using each caching strategy | 12 |
| 7 | The average cache hit rate while using each caching strategy | 13 |
| 8 | The average cache hit rate adjusted for data size while using each caching strategy | 15 |
| 9 | The average time taken to complete each request to ConceptNet while using each caching strategy | 16 |

Abstract

Semantic networks are often used as a method of relating multiple pieces of data to each other. ConceptNet is a semantic network that contains information about words and how they relate to other words. ConceptNet and other semantic networks are often hosted remotely and accessed as a service, and data retrieval times can be large. This project examines multiple data caching strategies and their impact on the performance of two existing applications that make use of ConceptNet data. We found that the largest factor in whether or not caching improves the performance of semantic network applications is the access pattern of the particular application.

1 Introduction

Modern day semantic networks created as representations of knowledge date back to the 1960's [20] and are commonly used in artificial intelligence applications. Examples of widely used semantic networks include the Semantic Network Processing System 3 (SNePS3), a logic based knowledge representation and reasoning system developed and maintained by the State University of New York at Buffalo [18], BabelNet, a multilingual semantic network created by linked WordNet (an English lexical database) [14] to the encyclopedic knowledge base contained in Wikipedia [15], and ConceptNet [12], a component of the Open Mind Common Sense (OMCS) artificial intelligence project being developed by the Massachusetts Institute of Technology Media Lab [21].

SNePS has been used to create the "Cognitive Agent of the SNePS System - an Intelligent Entity" (CASSIE) projects, an attempt to create a natural language problem solving computational cognitive agent [19]. BabelNet itself can be viewed as a product of artificial intelligence, given that its method of creation was the automatic mapping of two databases, utilizing machine translation techniques [16]. BabelNet has been used to create a natural language semantic search engine that attempts to search for documents based on a user's intent, rather than the exact sequence of words entered [10].

ConceptNet is a semantic network database of information about the relationships between words in natural language. Two existing applications utilize ConceptNet's data: the Semantic Similarity Engine (SSE), and the Topological Topic Modeler (TTM). The large sizes of semantic network data sets, which are often hosted remotely and accessed as a service, can result in data retrieval delays. We therefore developed multiple caching and pre-fetching strategies to improve the performance of SSE and TTM when accessing ConceptNet data. We developed our caching strategies based on the layout of ConceptNet, existing research in caching, and observation of typical SSE and TTM data access patterns. We also created a test data set for SSE and TTM in order to evaluate our caching strategies. We utilized a number of metrics to evaluate each caching strategy in combination with our test data set in order to determine how best to improve the performance of these

programs, as well as others which use data from ConceptNet. Our results showed that caching can increase performance of semantic network applications, but the degree of the increase is highly dependent on the access patterns of the particular application.

2 Background

2.1 ConceptNet

ConceptNet is a semantic knowledge representation graph created with the goal of depicting the semantic relationships within languages in order to assist facilitation of semantic processing in software programs [12]. ConceptNet represents semantic knowledge by:

- identifying concepts - words and short phrases - each of which is represented by a “node” [12]; and
- identifying dozens of different types of relationships between concepts by depicting “relations” connecting two nodes [12, 8].

Relations create bidirectional links between nodes, representing the semantic relationships between concepts [12]. Each relation also contains information about the source of the data [7]. ConceptNet’s data is generated from multiple crowd-sourced resources (such as Wiktionary and Open Mind Common Sense), online games (such as Verbosity and Nadya), and expert-created resources (such as WordNet and JMDict) [12, 5].

A simple example of ConceptNet’s data is a “UsedFor” relationship between the “car” and “drive” concepts. These data points together represent the relationship “A car can be used for driving.” The data also include the source from which this relationship was generated [7].

There are multiple ways for developers to interact with ConceptNet’s data in their applications. One method is to make requests to a public web service maintained by the developers of ConceptNet [4]. If the latency of making requests to a web service is undesirable, or if the application is intended to be run without an internet connection, then the data can be downloaded ahead of time in multiple commonly used formats [6].

2.2 Caching

Previous research in the area of caching has been largely in the context of low-level memory architecture. Because this research often focuses specifically on these scenarios, much of it is not directly applicable to caching data from a web service like ConceptNet. For example, Afek, Brown, and Merritt [1] describes a “lazy caching”

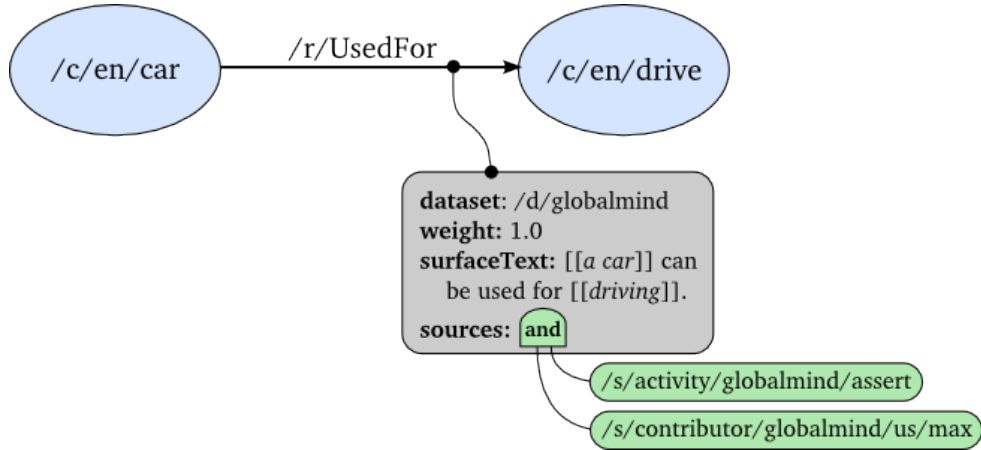


Figure 1: A simple example of ConceptNet data [7]

algorithm in the context of memory caches in multiprocessing systems, focusing largely on the cache coherency implications of the algorithm. Because ConceptNet’s data is read-only in our application, cache coherency is not a concern. Megiddo and Modha [13] describe a novel caching algorithm in more general terms, yet still make the assumption that the cache holds sequential “pages” of fixed size, which is true in memory caches but not necessarily in higher-level contexts such as our ConceptNet applications. We were unable to find previous research into evaluation of caching algorithms for application-level data retrieved over a network.

Caching data from a semantic network like ConceptNet is considerably different from memory page caching. Memory caching takes place on a much smaller timescale and at a much lower level. Memory caches are implemented in hardware and reduce access times on the order of 100 ns to approximately 1 ns [11]. Latency for a network request, as from ConceptNet, is much higher, on the order of 100 ms, and a cache for this sort of data is implemented at the application level. Though broad concepts such as cache replacement algorithms may apply to both varieties of caching, most research in the area of caching focuses very specifically on the unique challenges of memory caches.

2.3 Semantic Similarity Engine and Topological Topic Modeler

Our work focuses on optimizing ConceptNet access for a pair of existing programs which use ConceptNet.

The first of these programs is the Semantic Similarity Engine (SSE), described in Boteanu and Chernova [3]. The SSE is used to solve analogy problems similar to those found on some standardized tests, in the form “a is to b as c is to d”, where “a” and “b” are given and “c” and “d” must be selected from several possible pairs. The SSE solves these analogy problems by first extracting a path

connecting “a” to “b” in the ConceptNet network and then comparing this path to the paths for each of the possible pairs of “c” and “d”. It then selects the pair with the most similar path [3].

The other program is the Topological Topic Modeler (TTM), described in Boteanu and Chernova [2]. The TTM takes a list of words, referred to as a task, and separates them into groups of words related to common topics. It does this by first selecting a single word to use as a base for a new topic and then scores the similarity of each remaining word to the topic generated so far. This similarity score is generated by comparing the word to each word already in the topic by finding the distance between the two words in ConceptNet. The word is then sorted into the topic for which its score is the highest. If no score is above a threshold, the word becomes the first in a new topic [2].

In the current implementations of these programs, each time either program needs information about a node from ConceptNet it fetches the node, regardless of whether or not it has already fetched that same node. They do not perform any local caching of ConceptNet information. Since both programs frequently need to use information about words more than once, they must make many redundant requests to ConceptNet. Because nodes are retrieved by exploring the sub-graph around the nodes for the given words, it is likely that some common words will be accessed more frequently because they are more connected within ConceptNet (i.e. they have a higher connectedness).

3 Methodology

3.1 Test Data Set

In order to determine an optimal caching method, a two pronged approach was taken. Each of our caching strategies was evaluated by using both SSE and TTM in conjunction with each strategy and recording several performance metrics.

For TTM, a test set of 20 tasks was created. We created the test data set by exploiting the fact that a requirement of TTM is that the words given to it as a task be related to each other in ConceptNet. Accordingly, we created a script to generate tasks by first starting with a random word from an English dictionary, requesting the data on that word from ConceptNet, and using that data to pick between 4 and 7 related words. The script processed the same set of relations upon which TTM operates and ignores the same ones that TTM ignores. It also filtered the words retrieved to ensure that they would be acceptable for TTM operation. The script was run until 20 tasks were generated. As an example, one of the tasks generated was the words “custodian”, “greenskeeper”, “curator”, “keep”, “concierge”.

In addition, we included the existing analogy data set from Boteanu and Chernova [3] (which is based on the SAT data set created by Turney et al. [22]) and additional analogy problems from a public domain website. We took a subset of

both data sets to create our test data set. Specifically, we took 40 of the analogies sampled from those aimed at grades 1, 6, and 11, as well as 20 of the SAT analogies. As an example, one of the grade 1 analogies we used was “Good is to Bad as Happy is to _____”, with the options of “sad”, “great”, and “smile” to fill in the blank. SSE then processed all of the analogies we selected with each caching strategy.

3.2 Performance Metrics

We utilized several performance metrics to analyze the efficiency of each caching strategy:

- **Run time.** The main goal of this project is to shorten the amount of time in which the programs produce results. By measuring the time taken for the test set to complete, we can determine the gains in efficiency of each caching strategy.
- **Memory usage.** Caching ConceptNet data will require increased memory usage. Memory limitations will require measuring the memory requirements of each caching strategy in order to achieve a proper balance of increasing computation speed without requiring excessive memory usage.
- **Number of requests sent to ConceptNet.** Each request sent to ConceptNet requires the overhead of establishing an HTTP connection, constructing a request, waiting for a reply, and parsing the reply. Each step in this process utilizes computation time. By reducing the number of requests that need to be sent, the programs can produce results in a shorter period of time.
- **Cache hit/miss ratio.** When a word is requested and is not already in the cache, significant processing time is spent to fetch that word’s data. By measuring how often a word gets requested and is already in the cache, we can determine which caches are more efficient for the request patterns of our applications.
- **Time spent waiting for a response.** It is possible that ConceptNet can reply faster for some words and take more time to reply for others. By recording the time each request to ConceptNet takes, we can see if this is the case, and whether a particular caching strategy results in a more beneficial or harmful access pattern.

3.3 Caching Strategies

We implemented and evaluated several caching strategies:

- **No caching.** The test set was run and performance was analyzed with no caching to act as a baseline for each metric. The expectation is that this

strategy should use the least amount of memory, but take the longest to run. It will serve as one extreme when compared to other strategies. This cache will be referred to as the “Nothing Cache”.

- **Cache everything.** This strategy caches all words requested from ConceptNet and stores them in memory until the entire test data set has been processed. This strategy serves as the opposite extreme compared to the “Nothing Cache”; it should use the most memory but take the least amount of time to run. This cache will be referred to as the “Everything Cache”.
- **Cache everything per task.** This strategy caches every word requested from ConceptNet, but then empties the cache in between every task and analogy. This strategy could save significant memory without largely impacting computation time if tasks have very little overlap in the data they request. This cache will be referred to as the “Everything Per Task cache”.
- **Cache only a hard-coded list of highly connected words ahead of time.** It is possible that there exists a set of words in ConceptNet that are very highly connected and would thus be requested significantly more often than other words. By first running a calibration run of the test data set and logging the number of times each word is requested, we could identify if such words exist and what they are. This strategy would then request these words on startup and cache them in memory for the lifetime of the run, and never add anything else to the cache. This cache will be referred to as the “Frequency Cache”.
- **Cache a hard-coded list of frequent English words ahead of time.** This strategy is similar to the “Frequency Cache”, except that the list of words to pre-cache is based on word frequency in the English language rather than word frequency in our test data set. The cache would contain only this hard-coded list of words; they would be cached before computation begins and the cache would never change afterwards. This cache will be referred to as the “English Cache”.
- **Heuristic caching.** This strategy involves developing a heuristic based on the number of relations a word contains, in order to determine whether or not that word should be cached. Words with a large number of relations would stay in the cache, and words with very few relations will either stay in the cache for a very short time or not be cached at all. If the number of times a word’s data is accessed is related to the number of relations on that word, then this strategy should be very memory efficient. This cache will be referred to as the “Heuristic Cache”.
- **Fixed size caches.** This strategy is similar to the “Everything Cache”, but also involves setting a fixed size limit on how many words can be in the cache

at once. It also involves three different methods of determining which word to remove when the cache is full and a new word needs to be added. The first method is treating the cache as a FIFO (First In, First Out) queue; the oldest remaining word would be removed from the cache to make room for new words. The second is using an LRU (Least Recently Used) based strategy, so that the word that has been unread for the longest amount of time would be removed. The third and final strategy involves completely random removal of words in the cache to make room for new words. In order to determine the maximum size of these caches we observed the memory usage and number of cache entries while running other strategies and chose a size intended to balance cache usage patterns with memory limitations. These caches will be referred to as the “Fixed Size Caches”

3.4 Testing Procedure

Each caching strategy was implemented and tested on the full test data set 5 times. In order to reduce the chance that a particular task ordering might benefit certain caching strategies, the order in which the tasks were run was randomized for each run. In order to reduce the number of variables during our testing all of the testing was run on the same computer, connected to WPI’s wired network. All requests were directed at another machine on WPI’s wired network, hosting a local cache of ConceptNet. The test computer on which we ran all of our testing was four-banger, one of WPI’s high performance computing servers. It contains 4 hex-core 2.6 GHz processors for a total of 24 processing cores, and 128 gigabytes of RAM. The abundance of computing resources available on fourbanger allowed us to run multiple strategies in parallel to save time, without having to worry about resource exhaustion. As each run was performed, all of our metrics were recorded. After all test runs were completed, the values of the metrics were compared to determine the optimal caching method.

3.5 Cache Implementation Details

Some of our caches required no special configuration, such as the “Nothing Cache” and the “Everything Cache”. However, others required special configurations before they could be used. The specifics of these configurations are detailed below:

- **Fixed Size Caches.** In order to determine the size for the fixed size caches, we performed a test run with the “Everything Cache” and observed its word request pattern. Each task used approximately 600 unique words on average, and in general each word was used for a short amount of time. We therefore determined that 300 words would strike a good balance between caching a sufficient number of words for good performance without requiring excess memory usage.

- **Frequency Cache.** Based upon the fixed size cache test runs, we applied the same 300 word size limit to the hard-coded highly connected words strategy. We used the same sample run data to determine the 300 most frequently requested words, which are listed in Appendix A. Implementation of this strategy consists of caching those 300 words before running any tasks and caching nothing else.
- **English Cache.** We again adopted the 300 word cache size limit for this cache. We obtained a list of the most frequently used English words [9] and utilized the top 300. As with the prior strategy, these 300 words are cached prior to running tasks and nothing else is cached. The words we used are listed in Appendix B.
- **Heuristic Cache.** The goal of the heuristic caching strategy is to cache words that seem likely to be used again in the future. Our hypothesis is that the connectedness of a word in ConceptNet will correlate with the number of times that word is requested by SSE and TTM. In order to accomplish its goal, the cache calculated a running mean and standard deviation of the number of relations connected to each word in the cache. Once 50 new words have been cached, the strategy removes words from the cache that have a connectedness that is less than one standard deviation below the mean, which is equivalent to being in the lowest 15.8% of connectedness. If the number of times a word is accessed correlates to its connectedness, and if the connectedness of words follows a normal distribution bell curve (which it should according to the Central Limit Theorem [17]), then this heuristic caching method should provide an optimal method of removing less commonly used words from the cache while preserving words more likely to be needed in the future.

4 Results

4.1 Runtime

The average task runtime for each caching strategy is shown in Figure 2 (for TTM tasks) and Figure 3 (for SSE tasks). Overall speedups were much better for the SSE than for the TTM.

For the TTM, the “English” and “Frequency” caches showed very little improvement over the baseline (the “Nothing Cache”). The “English” cache spent nearly as much time waiting for requests as the “Nothing” cache, indicating that a word being common in English does not necessarily mean it will be heavily requested by the TTM. The “Frequency” cache also showed relatively little improvement on request time, though it was somewhat better than “English”.

The “Everything” and “Heuristic” caches both resulted in a large speedup, which is as expected because they cache large amounts of data and never remove

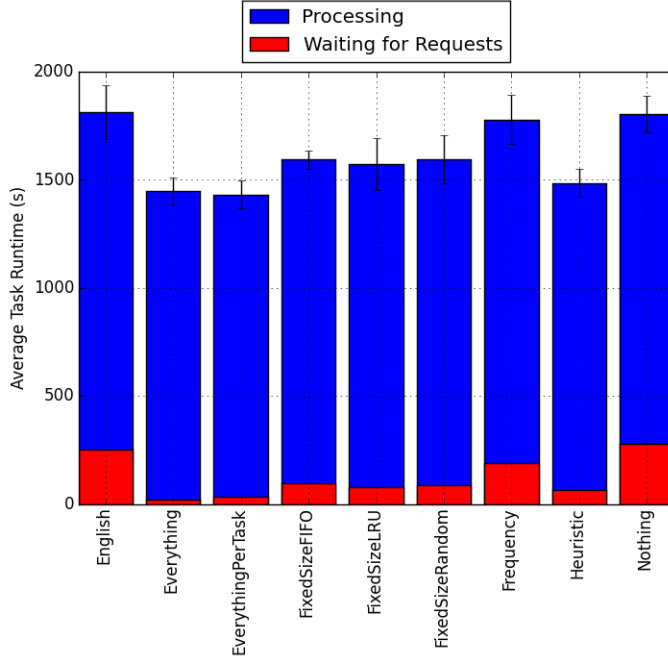


Figure 2: Average topic runtime for each caching strategy

items from the cache. The fixed-size caches each provided a speedup of 1.13-1.14x. The “Everything Per Task Cache”, surprisingly, showed a speedup of 1.26x, slightly better than the “Everything Cache”. This is almost certainly due to experimental error, but indicates that this cache performs approximately as well as the “Everything Cache”, despite using significantly less memory.

For the SSE, most caches showed negligible speedup at best, with the three fixed-size caches and the “Everything Per Task Cache” actually resulting in small slowdowns. The “Everything” cache produced a speedup of 1.04x. Though a relatively small speedup, this was the best of any cache for the SSE, as expected. The “English” and “Frequency” caches both produced a speedup of 1.01x. Although the overall times for the SSE were improved very little by caching, the amount of time spent waiting for requests decreased significantly. Adding caching increased the time spent processing data, due to cache lookups and replacement algorithms.

4.2 Memory

We logged the memory usage of each cache, sampling once per second, as seen in Figure 4. Unfortunately due to limitations in the way we measured memory usage, we were unable to separate the data for the SSE and TTM runs. The graph shows the data from a single run which includes both SSE and TTM tasks.

As expected, the memory usage of the “Everything Cache” increased throughout the run. The rate of increase slowed as the run went on, as nodes were added

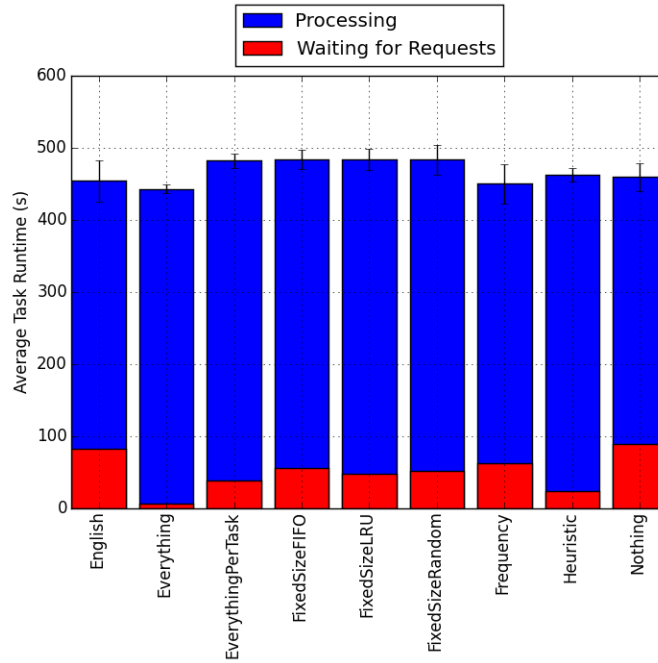


Figure 3: Average analogy runtime for each caching strategy

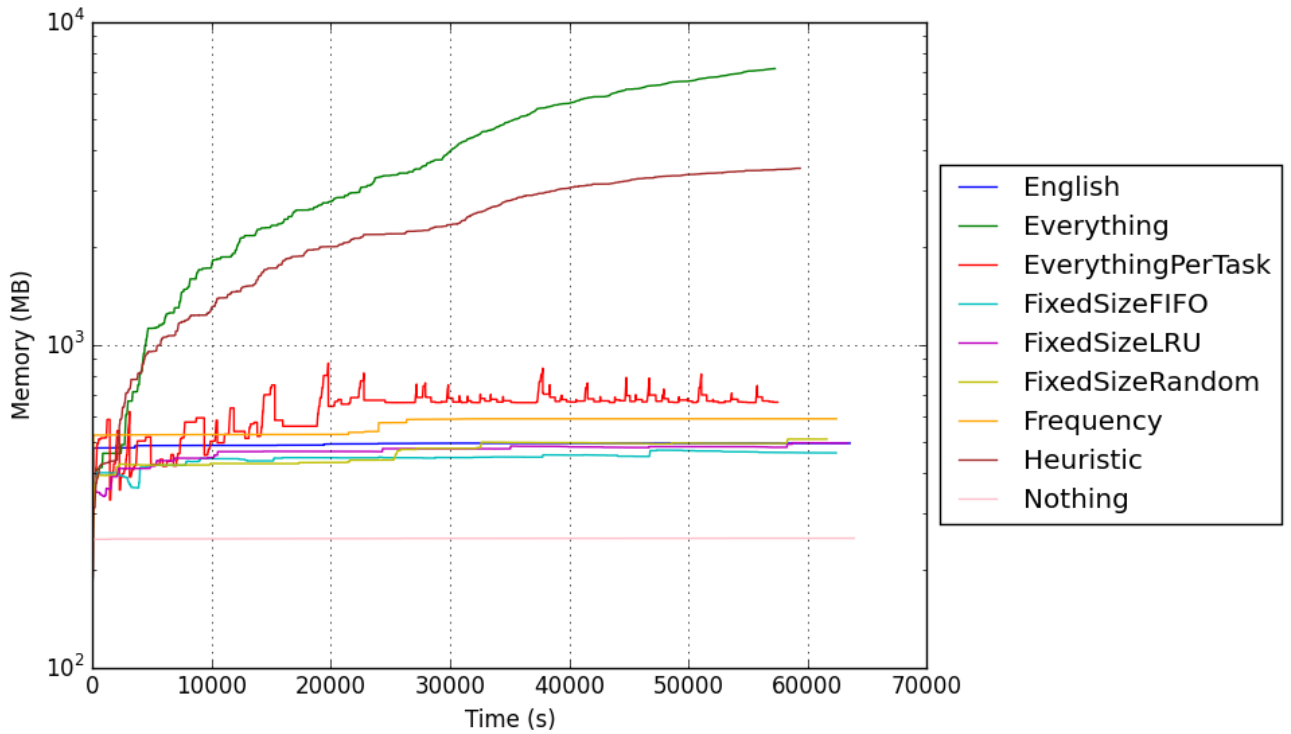


Figure 4: Memory usage over time for each caching strategy

to the cache. The “Heuristic Cache” also grew as the run went on, but used somewhat less memory than the “Everything Cache” overall. This makes sense, as these two caches differ from the others in that they have no size limit, never clear, and continue caching new data as it is retrieved.

The “Everything Per Task Cache” shows fluctuations in memory usage as each task runs and then completes, as expected. Its size grows gradually over the course of the first few tasks and then stabilizes, never dropping back down to the baseline memory overhead of the “Nothing Cache”. This is most likely due to imperfect garbage collection.

The three “Fixed Size Caches” all use a similar and relatively constant amount of memory, along with the “English Cache” and the “Frequency Cache”. The latter cache’s growth in memory usage between 2000 s and 3000 s is unexpected, but likely due to unexpected memory consumption fluctuations elsewhere in the program.

The “Nothing Cache”, as expected, uses the least memory of any of the caches, and has very consistent memory usage.

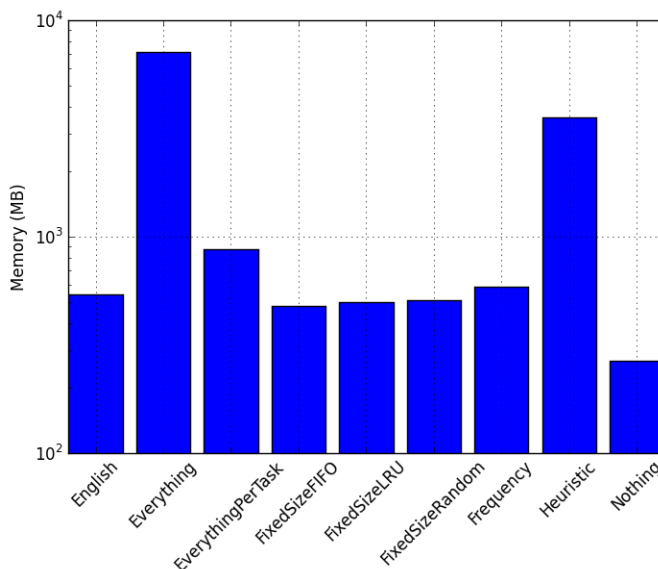


Figure 5: Maximum memory usage for each caching strategy

Figure 5 compares the maximum memory usage of each cache. The “Everything Cache” used the most memory, at around 7 GB. The “Heuristic Cache” used about half as much. The “English” and “Frequency” caches and the three “Fixed Size” caches used roughly the same amount of memory, around 500 MB. The “Nothing Cache” used around 250 MB, indicating that this is the base amount of memory used by the program without any caching.

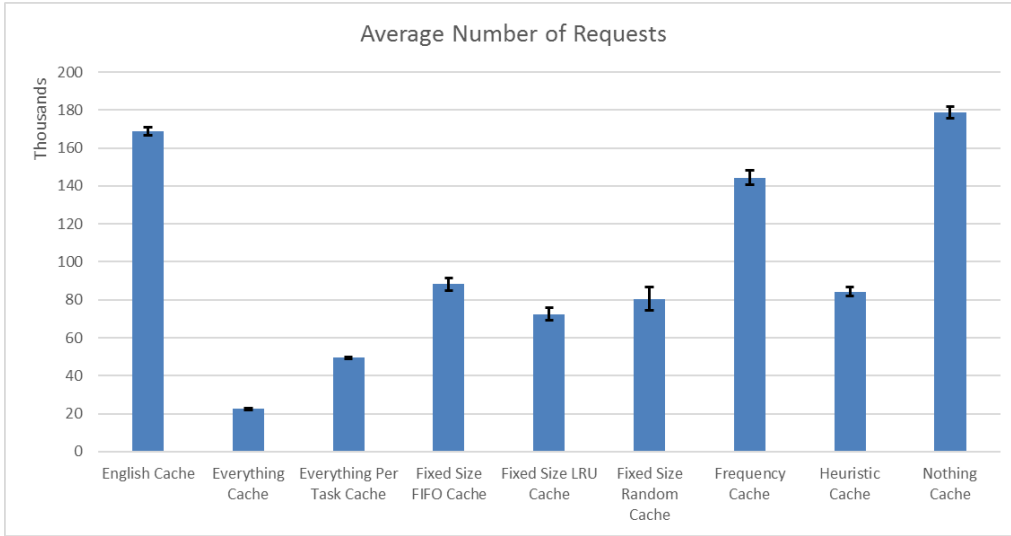


Figure 6: The average number of requests made while using each caching strategy

4.3 Number of Requests Made

We recorded every single request made to ConceptNet by SSE and TTM during their processing of our test data set, as seen in Figure 6. Unsurprisingly, the “Nothing Cache” made the most requests, coming in at just under 180 thousand requests on average, and the “Everything Cache” made the fewest, at just over 20 thousand on average. The “English Cache” didn’t fare much better than the “Nothing Cache”, which indicates that word frequency in the English language doesn’t correlate to the pattern of ConceptNet requests typically made by SSE and TTM.

The number of requests made while using the “Frequency Cache” is surprising; this strategy made almost a fifth fewer requests than the “Nothing Cache”, indicating that the 300 words it pre-caches were used relatively often. This suggests that it may be worthwhile to combine this strategy with another strategy designed to address the other four fifths of requests required by word absences in the “Frequency Cache”.

The three “Fixed Size Caches” all resulted in a very similar number of requests, with the LRU strategy performing the fewest and the FIFO strategy performing the most. Surprisingly, the random removal strategy outperformed the FIFO strategy, possibly indicating that a FIFO replacement strategy is not a good match for SSE and TTM’s typical request pattern.

The “Everything Per Task Cache” outperformed every cache except for the “Everything Cache”. Despite fully emptying the cache before starting each of the 80 test tasks, it made only roughly 2.2 times more requests than the “Everything Cache”. This indicates that while there is definitely some overlap between tasks, there isn’t a very large amount.

The “Heuristic Cache” performed about on par with the “Fixed Size Caches”,

suggesting that there were a significant amount of smaller sized words requested during task execution that were not stored in the cache.

4.4 Cache Hit Ratio

For each caching method we calculated a hit ratio as a raw percentage of the number of times a requested word was found in the cache divided by the total number of requests. This data is shown in Figure 7. We also adjusted the raw hit ratio for word connectedness in order to analyze the nature of the words stored by each caching method. To adjust for word connectedness we summed up the number of relations attached to each word and used the sums to calculate a similar hit-to-miss ratio. This data is shown in Figure 8.

4.4.1 Raw Cache Hit Rate

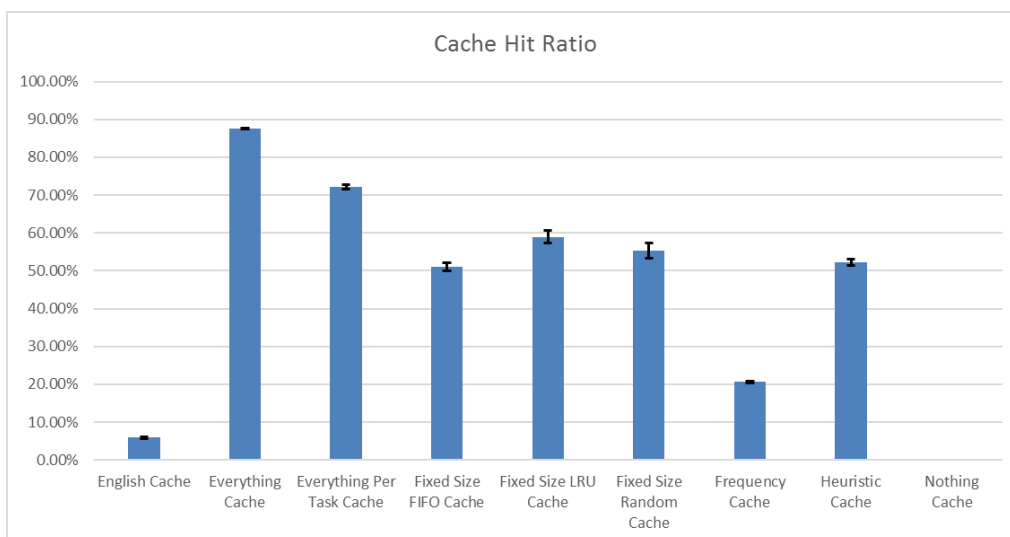


Figure 7: The average cache hit rate while using each caching strategy

Predictably, the “Everything Cache” produced the highest hit ratio at 87.54%. The second most efficient method in terms of raw hit ratio was the “Everything Per Task Cache”, which reduced the hit ratio to 72.14%. This 17.6% reduction in cached word availability appears to demonstrate that there was not an overly large amount of overlapping words between tasks.

Turning to the low end of this performance metric, the “English Cache” method produced an abysmally low hit ratio of 5.85%, demonstrating that word frequency in the English language does not correlate to the pattern of ConceptNet word requests typically made by SSE or TTM. The third lowest hit ratio was turned in by the “Frequency Cache”, suggesting that while some words may be requested

significantly more often, caching those words alone is not enough. The “Nothing Cache” of course produced a 0% hit ratio.

The remaining four caching methods all produced hit ratios between 50 and 60 percent. Given the closeness of these results, we compared the standard deviations of the hit ratio produced by each of the individual five runs for each caching method to ensure that our margin of error was small enough to effectively compare these methods:

| | Hit Ratios Per Run | | | | | Standard Deviation |
|-------------------------|--------------------|--------|--------|--------|--------|--------------------|
| Fixed Size FIFO Cache | 48.91% | 51.62% | 51.45% | 51.68% | 51.86% | 1.24% |
| Fixed Size LRU Cache | 58.85% | 56.33% | 58.43% | 60.47% | 60.75% | 1.78% |
| Fixed Size Random Cache | 58.78% | 56.72% | 54.59% | 53.66% | 53.55% | 2.25% |
| Heuristic Cache | 51.73% | 53.47% | 52.99% | 51.81% | 51.46% | 0.88% |

The caching method that produced the largest variability between runs was the “Fixed Size Random Cache”, likely due to the non-systematic replacement of cache data inherent in its randomness. However, even this cache did not have a large amount of variability between runs, suggesting that our margin of error is indeed low enough to compare these methods. The mechanistic FIFO replacement of cache data surprisingly produced the lowest hit ratio of the four, performing worse than removing words randomly. This indicates that a FIFO replacement strategy is not a good match for the pattern of requests typically made by SSE and TTM. The more attuned LRU method produced the highest hit ratio of the four.

The relatively low 52% hit ratio for the “Heuristic Cache” indicates that the words requested by these applications include a significant number of relatively lowly connected words, which the “Heuristic Cache” does not store for long. However, as seen in the following section, when adjusted for word connectedness the “Heuristic Cache” performance saw a significant increase.

4.4.2 Adjusted Cache Hit Rate

For most caches, adjusting the hit ratio calculations for word connectedness had a negligible effect, between a 2% to 5% increase. However, the “Frequency Cache” and “Heuristic Cache” saw much larger increases, 13% and 27% respectively. This was somewhat expected because these two caches are designed to cache highly connected words. If SSE and TTM spend more time processing more highly connected words, this increase could cause a significant decrease in processing time.

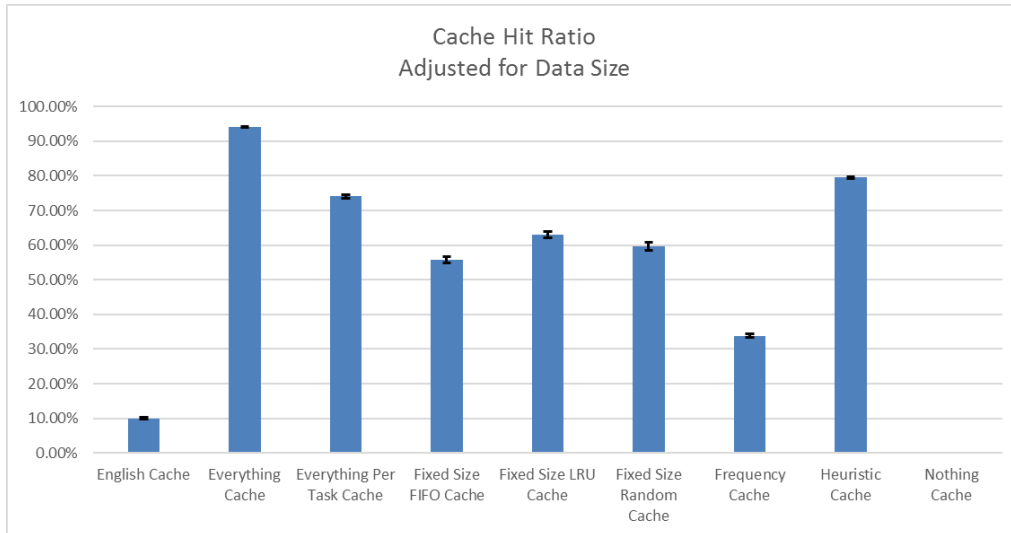


Figure 8: The average cache hit rate adjusted for data size while using each caching strategy

4.5 Request Times

For each caching strategy we calculated the average time required to obtain each un-cached word from ConceptNet, as shown in Figure 9. Six of the caching methods produced average access times between .06 and .063 seconds. The “Frequency Cache” produced a similar average access time of .054 seconds. The two outliers are the “Everything Cache” and the “Heuristic Cache”, with access times of .035 and .032 seconds respectively. This is an almost 50% decrease compared to the other types of caches. These results can be explained if ConceptNet takes less time to retrieve word data for words with a lower connectedness. This seems logical, as words with more data attached should take longer to load from storage and to send over the network. The results for the “Everything Cache” would then be caused by SSE and TTM requesting a larger amount of lower connectedness words, which would lower the average word retrieval time. The same logic can be applied to the “Heuristic Cache”, as it would only request highly connected words once, but may request words with a lower connectedness multiple times, thus lowering the average request time. This would also serve to explain the smaller decrease in average request time with the “Frequency Cache”, as it would never need to request the 300 words it already contains, and these words are all sufficiently highly connected and used frequently enough to affect the average. However it seems the size limit of 300 prevented the “Frequency Cache” from exhibiting a larger decrease in average request time.

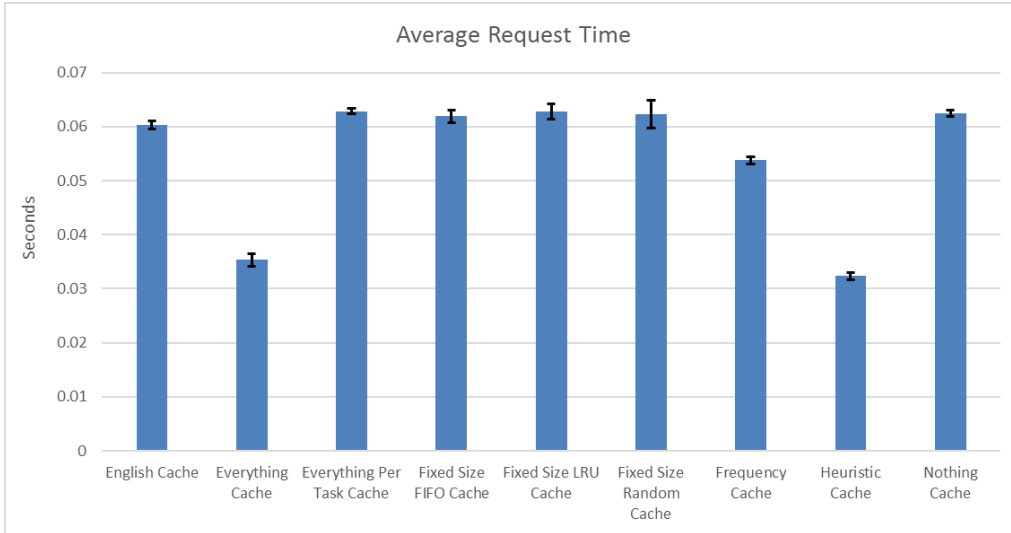


Figure 9: The average time taken to complete each request to ConceptNet while using each caching strategy

4.6 Analysis

The caching strategies which never remove items from the cache (“Everything” and “Heuristic” caches) provided good speedups, but use a large amount of memory (and the memory footprint will continue to grow as long as the application runs). The caching strategies with limited cache size did effectively limit memory consumption, but reduced time spent waiting for requests only slightly, and provided negligible speedups due to additional overhead from deciding what to cache. The “Everything Per Task Cache” provided a good balance, significantly reducing request time with little additional overhead while using only marginally more memory than the fixed size caches and keeping relatively constant memory usage throughout a run.

The degree to which caching affects performance varied significantly between the TTM and the SSE applications. This is because the TTM’s access patterns for conceptnet data involve expanding the same node many times in a given task, because it pulls the subgraphs of a number of words, some of which will have significant overlap, to group the words into topics. The SSE, on the other hand, has only two sets of two nodes as input for each problem, so the algorithm it uses will not request the same node from ConceptNet multiple times nearly as frequently as the TTM.

As a result, the data access pattern of the target application is likely the most important factor in selecting an appropriate caching mechanism. Based on our data, for semantic network applications which use algorithms which are likely to request the same word multiple times, we recommend using a cache which caches all semantic network requests indiscriminately and keeps that data cached within the scope of the task which requested it, similar to our “Everything Per Task Cache”. For

applications which do not employ algorithms which do not make redundant requests by their nature, there seems to be little value in caching nodes based on external metrics such as word frequency in prose or connectedness heuristics. Though they may reduce the number of requests made, our data shows that the overhead of the cache itself may negate any speedup.

5 Conclusion

In this project we investigated the use of caching algorithms to speed access to remotely-stored semantic network data. We compared the performance characteristics of a number of different caching strategies and found some to be much more effective and efficient than others. We found that algorithms which never remove data from the cache result in high speedups but use significant amounts of memory, while algorithms which impose a size limit on the cache incur overhead which limits the overall speedup provided. A cache strategy which periodically clears the cache offers a trade off, providing reasonably high speedups with less memory usage than fixed-size caches. We found that the performance improvement provided by caching varies depending on the application and its access patterns. One of the two applications we used showed significant speedups with caching, while the other did not. Future research might attempt to quantify the relationship between semantic network access patterns and cache speedup.

6 Bibliography

- [1] Yehuda Afek, Geoffrey Brown, and Michael Merritt. “Lazy Caching”. In: *ACM Trans. Program. Lang. Syst.* 15.1 (Jan. 1993), pp. 182–205. ISSN: 0164-0925. DOI: 10.1145/151646.151651. URL: <http://doi.acm.org/10.1145/151646.151651>.
- [2] Adrian Boteanu and Sonia Chernova. “Modeling Discussion Topics in Interactions with a Tablet Reading Primer”. In: *Proceedings of the 2013 International Conference on Intelligent User Interfaces*. IUI '13. Santa Monica, California, USA: ACM, 2013, pp. 75–84. ISBN: 978-1-4503-1965-2. DOI: 10.1145/2449396.2449409. URL: <http://doi.acm.org/10.1145/2449396.2449409>.
- [3] Adrian Boteanu and Sonia Chernova. “Solving and Explaining Analogy Questions Using Semantic Networks”. In: (2014).
- [4] ConceptNet. *API*. Nov. 6, 2014. URL: <https://github.com/commonsense/conceptnet5/wiki/API>.
- [5] ConceptNet. *ConceptNet 5*. Nov. 6, 2014. URL: <https://github.com/commonsense/conceptnet5/wiki/>.
- [6] ConceptNet. *Downloading*. Oct. 10, 2014. URL: <https://github.com/commonsense/conceptnet5/wiki/Downloading>.
- [7] ConceptNet. *Graph structure*. Nov. 6, 2014. URL: <https://github.com/commonsense/conceptnet5/wiki/Graph-structure>.
- [8] ConceptNet. *Relations*. Mar. 19, 2014. URL: <https://github.com/commonsense/conceptnet5/wiki/Relations>.
- [9] Mark Davies. *Word frequency data*. June 2014. URL: <http://www.wordfrequency.info/free.asp?s=y>.
- [10] Khadija Elbedweihy et al. “Using BabelNet in Bridging the Gap Between Natural Language Queries and Linked Data Concepts.” In: *NLP-DBPEDIA@ISWC*. 2013.
- [11] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. English. Upper Saddle River, New Jersey: Prentice Hall, 2014. ISBN: 9780133390094, 0133390098.
- [12] Hugo Liu and Push Singh. “ConceptNet-a practical commonsense reasoning tool-kit”. In: *BT technology journal* 22.4 (2004), pp. 211–226.
- [13] Nimrod Megiddo and D.S. Modha. “Outperforming LRU with an adaptive replacement cache algorithm”. In: *Computer* 37.4 (Apr. 2004), pp. 58–65. ISSN: 0018-9162. DOI: 10.1109/MC.2004.1297303.
- [14] George A Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

- [15] Roberto Navigli and Simone Paolo Ponzetto. “BabelNet: Building a very large multilingual semantic network”. In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics. 2010, pp. 216–225.
- [16] Roberto Navigli and Simone Paolo Ponzetto. “BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network”. In: *Artificial Intelligence* 193 (2012), pp. 217–250.
- [17] John Rice. *Mathematical Statistics and Data Analysis*. Second. Duxbury Press, June 1994.
- [18] Stuart C Shapiro. *An introduction to SNePS 3*. Springer, 2000.
- [19] Stuart C Shapiro. “The CASSIE projects: An approach to natural language competence”. In: *EPIA 89*. Springer, 1989, pp. 362–380.
- [20] Stuart C Shapiro. *The SNePS semantic network processing system*. State University of New York at Buffalo, Department of Computer Science, 1978.
- [21] Push Singh et al. “Open Mind Common Sense: Knowledge acquisition from the general public”. In: *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*. Springer, 2002, pp. 1223–1237.
- [22] Peter Turney et al. “Combining independent modules to solve multiple-choice synonym and analogy problems”. In: (2003).

A Word Frequency in Test Data Set

| Count | Word | Count | Word | Count | Word |
|-------|--------------|-------|---------------|-------|---------------|
| 7190 | album | 913 | die | 635 | paper |
| 5554 | person | 909 | idea | 633 | understand |
| 3627 | book | 907 | dog | 633 | actor |
| 2896 | film | 899 | burn | 630 | snore |
| 2512 | band | 890 | school | 620 | fly |
| 2196 | single | 885 | man | 614 | bedroom |
| 2138 | someone | 867 | sing | 612 | text |
| 2128 | something | 865 | music | 606 | sound |
| 2065 | think | 846 | agree | 604 | town |
| 1935 | movie | 842 | store | 603 | bed |
| 1823 | human | 836 | fish | 597 | home |
| 1767 | organisation | 822 | cook | 596 | fire |
| 1688 | sleep | 796 | newspaper | 594 | money |
| 1661 | read | 793 | child | 591 | bad |
| 1587 | learn | 789 | room | 587 | teach |
| 1514 | play | 786 | food | 584 | earth |
| 1403 | animal | 785 | live | 579 | disease |
| 1395 | water | 778 | city | 578 | cover |
| 1373 | build | 766 | cogitate | 577 | talk |
| 1372 | fun | 756 | contemplate | 567 | theater |
| 1305 | rest | 750 | student | 564 | drink |
| 1273 | study | 744 | often | 562 | table |
| 1184 | good | 743 | software | 562 | drive |
| 1158 | house | 742 | relaxation | 559 | activity |
| 1117 | change | 739 | run | 557 | see |
| 1088 | eat | 732 | plant | 552 | lie |
| 1080 | place | 726 | love | 542 | life |
| 1076 | cat | 714 | cabinet | 542 | entertainment |
| 1075 | light | 710 | computer | 541 | street |
| 1057 | kill | 708 | knowledge | 541 | sentence |
| 1031 | magazine | 704 | colloquialism | 532 | supermarket |
| 1011 | information | 692 | word | 530 | clothe |
| 999 | work | 686 | brain | 527 | disagree |
| 990 | write | 684 | kitchen | 525 | stupid |
| 976 | relax | 672 | smoke | 525 | closet |
| 960 | agreement | 665 | game | 523 | car |
| 951 | library | 657 | university | 522 | consider |
| 945 | dream | 651 | woman | 521 | dance |
| 928 | death | 638 | communicate | 519 | listen |

| Count | Word | Count | Word | Count | Word |
|-------|-----------|-------|---------------|-------|---------------|
| 519 | letter | 436 | cogitation | 389 | pay |
| 519 | head | 430 | floor | 385 | chicken |
| 518 | heat | 429 | wood | 384 | field |
| 516 | heavy | 428 | thick | 384 | bone |
| 512 | page | 427 | hospital | 383 | owner |
| 510 | make | 425 | farm | 383 | moon |
| 506 | pain | 424 | usually | 381 | faith |
| 503 | class | 424 | object | 377 | god |
| 496 | ocean | 422 | testify | 376 | illness |
| 496 | hot | 422 | mean | 376 | clean |
| 495 | sport | 422 | deliberation | 375 | duck |
| 495 | company | 422 | bore | 374 | hero |
| 494 | compete | 421 | tell | 374 | compact |
| 493 | forest | 421 | box | 374 | belief |
| 491 | country | 419 | energy | 374 | action |
| 491 | act | 419 | consideration | 373 | mountain |
| 490 | breathe | 417 | opera | 373 | loose |
| 488 | entertain | 416 | shelf | 371 | door |
| 484 | show | 414 | janitor | 370 | plural |
| 482 | story | 413 | grind | 369 | mouse |
| 469 | doctor | 412 | art | 369 | boat |
| 469 | desk | 411 | record | 368 | education |
| 469 | business | 411 | plan | 366 | perform |
| 467 | speak | 411 | paint | 366 | lizard |
| 465 | move | 411 | hotel | 366 | assent |
| 465 | classroom | 410 | organization | 365 | countryside |
| 462 | big | 408 | dense | 363 | ship |
| 455 | sex | 408 | artist | 361 | map |
| 454 | travel | 406 | eukaryote | 360 | hard |
| 453 | law | 405 | concentrate | 359 | remember |
| 453 | apartment | 404 | language | 359 | murder |
| 452 | thickness | 404 | give | 358 | village |
| 451 | pretend | 401 | park | 358 | sky |
| 448 | bird | 399 | gift | 357 | restaurant |
| 447 | horse | 396 | baby | 357 | name |
| 445 | song | 395 | know | 357 | contemplation |
| 443 | stone | 394 | bar | 354 | state |
| 442 | tree | 394 | apple | 353 | reason |
| 440 | sun | 393 | settlement | 353 | product |
| 440 | shop | 390 | end | 353 | picture |
| 436 | specie | 389 | time | 352 | deal |

| | | | | | |
|-------|---------------|-------|--------------|-------|-------------|
| Count | Word | Count | Word | Count | Word |
| 352 | backpack | 335 | jump | 325 | rich |
| 351 | air | 334 | emotion | 324 | writer |
| 348 | undergraduate | 333 | ring | 324 | nightmare |
| 347 | mind | 333 | refrigerator | 324 | imagine |
| 344 | metal | 332 | swallow | 323 | meat |
| 343 | insect | 332 | grow | 323 | bargain |
| 343 | expensive | 332 | fiddle | 322 | carry |
| 341 | memorize | 332 | control | 321 | keep |
| 341 | fruit | 331 | sea | 321 | blowfish |
| 340 | important | 331 | religion | 320 | firefighter |
| 340 | deep | 331 | believe | 320 | break |
| 340 | beach | 331 | beautiful | 319 | zoo |
| 339 | inform | 330 | egg | 319 | loosen |
| 338 | steal | 330 | debate | 319 | dirty |
| 337 | memory | 328 | gold | 319 | cultivation |
| 337 | dirt | 327 | war | 318 | mammal |
| 337 | awake | 327 | thinker | 318 | chair |
| 336 | rock | 326 | sell | 317 | property |
| 336 | feel | 326 | garage | 317 | airport |
| 335 | sweep | 326 | accept | 316 | orgasm |

B English Word Frequency

| | | | | | |
|-----------|-------|-----------|--------|-----------|---------|
| Frequency | Word | Frequency | Word | Frequency | Word |
| 1 | the | 40 | if | 79 | our |
| 2 | be | 41 | would | 80 | two |
| 3 | and | 42 | her | 81 | more |
| 4 | of | 43 | all | 82 | these |
| 5 | a | 44 | my | 83 | want |
| 6 | in | 45 | make | 84 | way |
| 7 | to | 46 | about | 85 | look |
| 8 | have | 47 | know | 86 | first |
| 9 | to | 48 | will | 87 | also |
| 10 | it | 49 | as | 88 | new |
| 11 | I | 50 | up | 89 | because |
| 12 | that | 51 | one | 90 | day |
| 13 | for | 52 | time | 91 | more |
| 14 | you | 53 | there | 92 | use |
| 15 | he | 54 | year | 93 | no |
| 16 | with | 55 | so | 94 | man |
| 17 | on | 56 | think | 95 | find |
| 18 | do | 57 | when | 96 | here |
| 19 | say | 58 | which | 97 | thing |
| 20 | this | 59 | them | 98 | give |
| 21 | they | 60 | some | 99 | many |
| 22 | at | 61 | me | 100 | well |
| 23 | but | 62 | people | 101 | only |
| 24 | we | 63 | take | 102 | those |
| 25 | his | 64 | out | 103 | tell |
| 26 | from | 65 | into | 104 | one |
| 27 | that | 66 | just | 105 | very |
| 28 | not | 67 | see | 106 | her |
| 29 | n't | 68 | him | 107 | even |
| 30 | by | 69 | your | 108 | back |
| 31 | she | 70 | come | 109 | any |
| 32 | or | 71 | could | 110 | good |
| 33 | as | 72 | now | 111 | woman |
| 34 | what | 73 | than | 112 | through |
| 35 | go | 74 | like | 113 | us |
| 36 | their | 75 | other | 114 | life |
| 37 | can | 76 | how | 115 | child |
| 38 | who | 77 | then | 116 | there |
| 39 | get | 78 | its | 117 | work |

| | | | | | |
|-----|-----------|-----|----------|-----|------------|
| 118 | down | 159 | let | 200 | play |
| 119 | may | 160 | great | 201 | government |
| 120 | after | 161 | same | 202 | run |
| 121 | should | 162 | big | 203 | small |
| 122 | call | 163 | group | 204 | number |
| 123 | world | 164 | begin | 205 | off |
| 124 | over | 165 | seem | 206 | always |
| 125 | school | 166 | country | 207 | move |
| 126 | still | 167 | help | 208 | like |
| 127 | try | 168 | talk | 209 | night |
| 128 | in | 169 | where | 210 | live |
| 129 | as | 170 | turn | 211 | Mr |
| 130 | last | 171 | problem | 212 | point |
| 131 | ask | 172 | every | 213 | believe |
| 132 | need | 173 | start | 214 | hold |
| 133 | too | 174 | hand | 215 | today |
| 134 | feel | 175 | might | 216 | bring |
| 135 | three | 176 | American | 217 | happen |
| 136 | when | 177 | show | 218 | next |
| 137 | state | 178 | part | 219 | without |
| 138 | never | 179 | about | 220 | before |
| 139 | become | 180 | against | 221 | large |
| 140 | between | 181 | place | 222 | all |
| 141 | high | 182 | over | 223 | million |
| 142 | really | 183 | such | 224 | must |
| 143 | something | 184 | again | 225 | home |
| 144 | most | 185 | few | 226 | under |
| 145 | another | 186 | case | 227 | water |
| 146 | much | 187 | most | 228 | room |
| 147 | family | 188 | week | 229 | write |
| 148 | own | 189 | company | 230 | mother |
| 149 | out | 190 | where | 231 | area |
| 150 | leave | 191 | system | 232 | national |
| 151 | put | 192 | each | 233 | money |
| 152 | old | 193 | right | 234 | story |
| 153 | while | 194 | program | 235 | young |
| 154 | mean | 195 | hear | 236 | fact |
| 155 | on | 196 | so | 237 | month |
| 156 | keep | 197 | question | 238 | different |
| 157 | student | 198 | during | 239 | lot |
| 158 | why | 199 | work | 240 | right |

| Frequency | Word | Frequency | Word | Frequency | Word |
|-----------|----------|-----------|-----------|-----------|-----------|
| 241 | study | 261 | since | 281 | ever |
| 242 | book | 262 | long | 282 | stand |
| 243 | eye | 263 | provide | 283 | bad |
| 244 | job | 264 | service | 284 | lose |
| 245 | word | 265 | around | 285 | however |
| 246 | though | 266 | friend | 286 | member |
| 247 | business | 267 | important | 287 | pay |
| 248 | issue | 268 | father | 288 | law |
| 249 | side | 269 | sit | 289 | meet |
| 250 | kind | 270 | away | 290 | car |
| 251 | four | 271 | until | 291 | city |
| 252 | head | 272 | power | 292 | almost |
| 253 | far | 273 | hour | 293 | include |
| 254 | black | 274 | game | 294 | continue |
| 255 | long | 275 | often | 295 | set |
| 256 | both | 276 | yet | 296 | later |
| 257 | little | 277 | line | 297 | community |
| 258 | house | 278 | political | 298 | much |
| 259 | yes | 279 | end | 299 | name |
| 260 | after | 280 | among | 300 | five |