Efficient Computation of Spreading Activation Using Lazy Evaluation

Steven J. Jones (scijones@umich.edu)
Arthur R. Wandzel (awandzel@umich.edu)
John E. Laird (laird@umich.edu)
University of Michigan, 2260 Hayward Street
Ann Arbor, MI 48109-2121

Abstract

Spreading activation is an important component of many computational models of declarative long-term memory retrieval but it can be computationally expensive. The computational overhead has led to severe restrictions on its use, especially in real-time cognitive models. In this paper we describe a series of successively more efficient algorithms for spreading activation. The final model uses lazy evaluation to avoid much of the computation normally associated with spreading activation. We evaluate its efficiency on a commonly-used word-sense disambiguation task where it is significantly faster than a naive model, achieving an average time of 0.43ms per query for a spread to 300 nodes.

Keywords: cognitive architecture; context-sensitive retrieval; Soar; semantic memory; spreading activation.

Introduction

As cognitive modeling moves to more complex and real-world tasks, there is a challenge of maintaining efficient, scalable, context-sensitive access to long-term knowledge. In prior research, our group developed efficient and scalable algorithms for context-free cue-based retrievals (Derbinsky, Laird, & Smith, 2010). In this paper, we extend that work to context-sensitive retrievals by developing efficient and scalable algorithms for spreading activation (Anderson, 1983b).

In cognitive architectures, such as Soar (Laird, 2012) and ACT-R (Anderson, 1983a), working memory defines the *context*—the agent's task-relevant knowledge. Spreading activation supports context-sensitive retrieval by biasing the retrieval of elements from long-term declarative memory to those that have direct and possibly indirect long-term associations to structures in working memory. Previous work has focused on mitigating the cost of spreading activation through using high-performance computers, external data-base technology, and parallelism (Douglass & Myers, 2010; Chen, Petrovic, & Clark, 2014; Edmonds, Atahary, Taha, & Douglass, 2015). The expense of spreading activation has led cognitive modelers to severely limit the depth of spread or to avoid it completely. Efficient spreading could dramatically increase its use in cognitive models and decrease the time it takes to run simulations. It could also enable spreads to greater depth, so as to extract knowledge that is latent within the structure of an agent's long-term memory. Finally, efficient spreading would support its use in real-time cognitive models and AI agents.

As in our prior work, our investigations are within Soar. Soar provides an efficient platform, both in terms of overall performance, but more specifically in terms of an efficient implementation of long-term declarative memory. For cognitive modeling, Soar's decision cycle corresponds to the production firing cycle of ACT-R, which maps to approximately 50ms of human behavior. However, on a standard workstation, Soar's decision procedure runs at less than 0.3ms, even with large numbers of rules and declarative memory elements. The essence of this paper is adding spreading activation to Soar with a simple naive algorithm, then reconceptualizing that algorithm through a series of optimizations. Those optimizations take advantage of important regularities in the dynamics of Soar's long-term semantic memory. The ratio of changes to the context (working memory) to the total number of elements in the context is small. The ratio of long-term memory changes to the total number of elements in the long-term memory is even smaller. Many queries of long-term memory are unambiguous and even those with ambiguity are often constrained to only a few possibilities. We evaluate these optimizations on a word-sense disambiguation task that has proven usual for evaluating efficiency of long-term memory retrieval in the past (Derbinsky & Laird, 2011).

Background

In the Soar cognitive architecture, working memory maintains an agent's current knowledge of its task and environment, including active goals, results of perception, inferences, and retrievals from long-term memory. Behavior is conditional on the contents of working memory, so that for information to influence behavior, it must be in working memory.

Semantic memory contains the agent's long-term declarative knowledge, such as facts about the world, and corresponds to ACT-R's long-term declarative memory. Information can be retrieved from semantic memory into working memory via a query. A query is initiated using a *cue* that is composed of a single-level symbolic directed graph, anchored in a single node. Consider an example where there has previously been a retrieval for the word "activation" that returned a result with substructure `^meaning A1437`. The agent may then decide to retrieve a second word sense of "activation" using the following cue: (`<cue>^word-string activation ^meaning A1437 -`), where "-" is used to prohibit the retrieval of the previous word sense. All nodes in semantic memory that match the cue are found. If no node matches, then the query fails. If more than one node matches, a *bias term* is computed for every cue matching node and the node with the highest bias term is the result. One important component of the bias term is base-level activation (BLA). BLA combines information on the recency and frequency of

a node's previous accesses. Although BLA is useful, it does not support context-sensitive retrieval, where structures in the context (the contents of working memory) influence the bias term. One approach to incorporating context into the bias term is to use spreading activation (SA) as another component. Adding together the BLA component bla_m and the SA component sa_m for every cue matching node m gives us an overall bias term BT_m.

Naive Spreading Activation

Activation spreads out from semantic memory nodes that are in working memory to adjoining nodes in semantic memory. One point of variability is whether activation spreads in the direction of edges (forward), opposite that direction (backward), or in both directions. Our algorithms are agnostic on spread direction and support all three directions. For simplicity and ease of analysis, our implementations track and record the SA- and BLA- component independently. Their only interaction is during the calculation of the overall bias term, simply related: $bla_m + sa_m = BT_m$

For the Naive Algorithm, spreading begins whenever a node n enters the context and becomes a *source* of spread. In Figure 1, node **A** is the source and spreads activation of .45 forward to the two nodes labeled **B** and **C**. Nodes that receive activation are spread *recipients*, and they can accumulate activation from many sources or even from the same source at varying depths, such as nodes **E** and **F**. The total accumulated activation for recipient r, is denoted as s_T.

The calculation of the activation of a recipient depends on three factors. The first is the initial activation of the source, which we set to 1. Second, as activation spreads deeper, there is a decay factor, $p < 1$. In this example, the decay factor is .9. Third, the activation of a parent node is divided equally among all of its children nodes, leading to the fan effect through the spread of activation (Anderson, 1983b). Thus, the calculation for the activation of a recipient node r, where there are a total of k children nodes from parent node s_n is $s_r = \frac{1}{k} \times p \times s_n$. For the children of **A** (nodes **B** and **C**), the calculation is $\frac{1}{2} \times .9 \times 1 = .45$. If a recipient receives activation from two distinct parents, the activations from both parents are summed together. Thus, for node **F**, **C** and **G** are both parents that respectively issue .45 and .135 to result in an activation value of .19575 for **F**.

We denote the activation that accumulates in a spread recipient, r, from a source, c, as $s_{r,n}$, so that the total activation for node $s_T = \sum_{c \in C} s_{r,c}$ where C is the set of all context nodes that are sources. If a node does not receive any activation, then $s_T = 0$.

The total spread from a source can be controlled in multiple ways. For example, a spread can be restricted to a fixed distance from the source node, called the *depth limit* or there can be a limit to the total number of nodes traversed, termed the *spreading size limit*. In our experiments, for simplicity we use a fixed spreading size limit, which is 300. The spreading size limit is applied within the context of a breadth-first

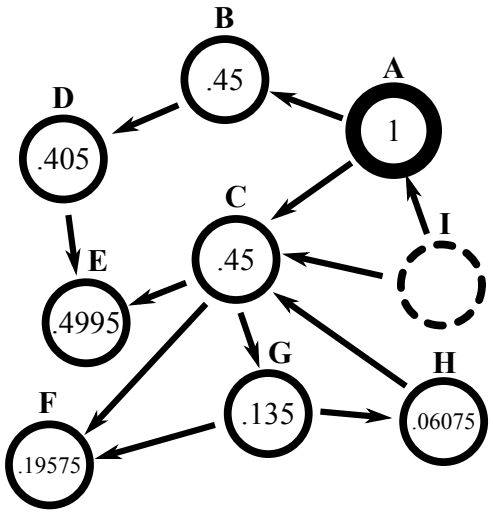

Figure 1: An example network, with forward spread from a single source **A**. Initial activation is 1, decay factor is .9, and depth limit is 3.

traversal of the graph, so that activation is spread to all nodes at the current depth before it is spread to nodes at the next depth level. With a size limit, it is possible to not spread to every node at the same depth. For example, if we use a size limit of 300 and we assume a regular graph where every node has exactly five children, after depth 1, 5 nodes will be visited; after depth 2, $5+25=30$ nodes will be visited; and after depth 3, $5+25+125=155$ nodes will be visited. An addition 145 nodes at depth 4 (out of 625) will be explored, but 480 nodes at depth 4 will not.

Observations on Naive Spreading Activation

In order to compute the overall bias term, a naive algorithm spreads from every context node when it is introduced into working memory and updates the SA component of every recipient per each spread. In such an implementation, the primary cost of spreading activation is the breadth-first traversal that computes the activation for nodes that receive spread from a given source. A secondary cost is updating the new value of the SA component and recalculating the overall bias term, which for the naive algorithm is performed for every spread recipient. In a naive algorithm, these costs scale with the number of source nodes multiplied by the spread size limit.

In the following sections, we identify properties of processing that suggest ways of reducing the costs of a naive algorithm. Many of these shortcuts are obvious, while others are more subtle, but in general the unnecessary computation can be avoided by one major approach: avoid calculations of spread until it is necessary, because it is possible they may never be necessary. This approach is called *lazy evaluation* and attempts to only compute activations that play a role in determining which candidate is retrieved from memory. In the naive algorithm, a significant proportion of nodes that re-

ceive spread do not influence which candidate is retrieved from memory. It is because of this that lazy evaluation can reduce computation while resulting in no change to which candidates are retrieved.

Our final algorithm incorporates optimizations made possible by the following observations.

There are Consistent Elements Shared Between Old and New Context

If there are no changes in the set of context elements, the results of spread will be exactly the same on subsequent cycles. Thus, spread only has to be computed when context elements change, which we call **Change-Only Processing**. This optimization is expected to have a large impact because working memory remains largely unchanged from one cycle to another. For an architecture where working memory has rapid changes, this may not be much of an improvement.

The Long-Term Memory Network Changes Slowly

The knowledge in semantic memory is not expected to change rapidly. Thus, the breadth-first traversal that computes the spread is relatively stable. We can take advantage of this stability by explicitly caching a trace of the breadth-first traversals and the activation values they produce. These can be used to directly access these values for updating the SA component without performing the breadth-first traversals. Whenever an edge is added to or removed from semantic memory, the traversals containing the parent of that edge are no longer valid. Under these circumstances, if these traversals are needed in the future, they will be recomputed. This improvement is simply referred to as **Caching**.

One implication is that we can compute traversals during task initialization. Thus, if the agent starts with a large knowledge base pre-loaded into its long-term semantic memory, it can pre-compute and cache the traversals for future use. Nevertheless, the agent must still recalculate traversals when the network changes according to the invalidation cases we outlined above. This is **Precalculation**.

Queries are Less Frequent than Context Changes

Even when there are context changes during a cycle, spread does not need to be computed unless there is a query. Even when the calculation of spread is tied to only changes in the context, this modification eliminates calculations for cases where context elements become active and then become inactive without any intervening queries. This improvement is called **Query-Deferred Calculation** and exists in ACT-R. This improvement also supports implementation of two further improvements.

A Query's Cue can be Unambiguous

If a cue is used that is constraining enough so that only one long-term memory node matches, there is no need to spread. We call this improvement **Ambiguity-Only Processing**.

The Number of Cue Matches is Small

A naive approach to spreading activation normally computes the activation of all recipients of a spread. However, it is rare that more than a small percentage of those nodes match the cue, and the SA component is needed only for those nodes that match. Thus, if a node does not match the cue, it is unnecessary to calculate the overall bias term for that node. Even when only a few constraints are included in a cue, they can eliminate a substantial proportion of the nodes from consideration.

In response to this observation, our approach flips the normal way of thinking about computing spread. Instead of updating the SA component of every recipient, our algorithm only computes the overall bias term for nodes that match the cue. This improvement is named **Candidate-Only Processing** and it is included in ACT-R. Note that if a cue has so little constraint such that every node is a candidate, this optimization will not help.

While this series of improvements generates our final algorithm, we restate the final algorithm explicitly below.

Algorithm Review

We reconceptualize spreading activation as the calculation required to provide the bias term necessary for context-sensitive retrieval. In algorithm 1, we follow the procedure PROCESSAGENTCYCLE() every cycle. The first step is to check whether or not a query is present. If not, processing stops. This corresponds to our **Query-Deferred Calculation** improvement. If a query is present, then there is a check as to whether there is only one node that matches the cue. If so, all spreading activation calculation is skipped and that node is returned. This is called **Ambiguity-Only Processing**. However, if the cue is ambiguous, then spreading activation is computed using DOTRAVERSALS() and DOAPPLICATIONS().

In DOTRAVERSALS(), if there are changes to the context elements then the following processing occurs. If a source's traversal has never been calculated or there has been a change to the network that invalidates the traversal, then it is necessary to recalculate the traversal via breadth-first search (TRAVERSE()). A map from node to traversal, *cachedSpread*, maintains a history of currently usable traversals. If these conditions do not hold, then a cached traversal is retrieved to bypass additional calculation.

After all source node traversals are calculated and retrieved, DOAPPLICATIONS() updates the SA component of all the recipients in these traversals that also match the cue as looked up against a recorded history of currently usable traversals. Updating the SA component of only the cue-matched nodes corresponds to **Candidate-Only Processing**.

Finally, the cue-matched node with the highest overall bias term is returned as the result to the query.

The worst case for this algorithm is when there are frequent changes to declarative memory that invalidate the cached traversals, and when there are frequent changes to context elements that require continual recalculation of the traversals. In

Algorithm 1 : Lazy algorithm for spreading activation

```
cachedSpread                                   ▷ global variable

1: function DOTRAVERSALS(contextChanges)
2:     for source ∈ contextChanges do
3:         if source ∉ cachedSpread OR
           ISINVALID(cachedSpread[source]) then
4:             spread ← TRAVERSE(source)
5:             cachedSpread[source] ← spread

1: function DOAPPLIES(cueMatches, contextChanges)
2:     for match ∈ cueMatches do
3:         if match ∈ cachedSpread[contextChanges] then
4:             UPDATEBIASTERMOF(match)

1: procedure PROCESSAGENTCYCLE()
2:     if agent issues a query command then
3:         cueMatches ← DOQUERY(cue)
4:         if SIZEOF(cueMatches) == 1 then
5:             ADDTOWMEM(match)
6:         else
7:             DOTRAVERSALS(contextChanges)
8:             DOAPPLIES(cueMatches, contextChanges)
9:             contextChanges ← ∅
10:        ADDTOWMEM(MAX(cueMatches))
```

this worst case, our algorithm essentially performs the naive algorithm.

Evaluation

Our hypothesis is that our proposed algorithm can result in a significant reduction in the time spent computing spreading activation and that each component of the algorithm provides some benefit. To test these claims, we evaluate the time efficiency as we incrementally incorporate each component of the final algorithm.

The task we use is the word sense disambiguation (WSD) task that we previously used for evaluating implementations of base-level activation (Derbinsky & Laird, 2011). In this task, the agent must disambiguate the word senses used in a sentence. Each input word is annotated with its name and part-of-speech (e.g. noun) but not its sense. When an agent encounters and issues a query for the word, such as the word "English" with part-of speech "noun", it must choose one of the following possible senses: 1) the West Germanic language; 2) the humanities discipline; 3) the people of England; 4) the spin given to a ball in pool or billiards. The agent keeps retrieving senses until the retrieved sense matches the correct sense.

The test sentences and the ground truth are provided by SemCor, a popularly used sense-tagged corpus. SemCor consists of 352 texts from the Brown corpus (Kucera & Francis, 1967), with every word linked to its correct sense in the English lexical database WordNet, version 3.0 (Miller, 1995). Our construction of WordNet 3.0 includes all synset and lemma links for every part-of-speech, and our construction of SemCor includes all available sense-tagged words, numbering 217,918, of which approximately 75% are multi-sense[1].

In our experiment, there are seven different agents, corresponding to different spreading activation algorithms. These are listed in Table 1. All agents are preloaded with our construction of WordNet 3.0 in their semantic memory and they all use Soar's existing base-level activation mechanism in addition to spreading activation. We compare as well to one agent that does not use spreading activation, instead using only base-level activation.

All agents iterate through all SemCor sentences, maintaining in working memory the retrieved correct word sense for all words previously encountered within a paragraph as context. Table 1 displays the time spent on spreading activation during the task. The execution of the task is deterministic with negligible variance in execution times. All spreading activation agents have a spread size limit of 300, and they all compute exactly the same spread values (and bias terms) for all the candidate retrievals, and they retrieve the same node from semantic memory. Thus, the seven algorithms differ only in the efficiency of computing the retrieved nodes.

All agents ran for a total of 1,644,058 decision cycles while issuing a total of 565,223 queries. The naive algorithm, omitting all improvements, took over 100,000 seconds. Every change to the algorithm decreased execution time. The amount of time our final algorithm took on spreading activation alone was 245 seconds. On average, the amount of time spent on the rest of the agent's processing was 290 seconds (not shown in Table 1). When examined at the individual query level, the final algorithm spent an average of 0.43ms per query on spreading activation compared compared to 5.87ms for the naive algorithm with change-only processing.

We confirm that precalculation (and thus the corresponding reduction in the breadth-first traversals during the task) speeds up the agent. The memory cost to precalculation is storage of the traversal to 300 nodes for each node. While query-deferred processing has little direct impact, it supports candidate-only processing and ambiguity-only processing. While the effect of ambiguity-only processing is modest, candidate-only processing shows a significant improvement associated with selectively updating only spread recipients that are potential query results. The naive algorithm, which omits all improvements, is the slowest.

The amount of time to calculate spread from a single node is expected to scale linearly with spreading size limit. As a check, we used a test agent that first randomly selects a word and then adds new word information to the network. The randomly-selected word serves as a context element. The agent then initiates an artificially constrained query, such that

[1]The SemCor and WordNet 3.0 data sets are available to download at `http://web.eecs.umich.edu/~mihalcea/downloads.html#semcor` and `http://wordnet.princeton.edu`, respectively.

Spreading Activation Mechanism	Spread Time (s)	Spread Time Per Query (ms)
Naive Algorithm	$> 100{,}000$	
+ Change-Only Processing	3,316	5.87
+ Caching	1,200	2.12
+ Precalculation	810	1.43
+ Query-Deferred Processing	803	1.42
+ Ambiguity-Only Processing	778	1.38
+ Candidate-Only Processing	245	.43

Table 1: Timed performance on the WSD task across seven spreading activation variants. Rows with prefaced with "+" denote incremental cumulative improvements to our algorithm.

the breadth-first traversal must be calculated for the new context element and that query has a sufficiently general cue such that the candidate set includes all spread recipients. The test agent thus induces the maximum cost of a single network change. As expected, the maximum spreading times shown in this figure are significantly greater than the average times achieved in the WSD task.

The timing results for this test are found in Figure 2. Graph points that fall below the linear trend reflect spread traversals that exhaust the network before reaching the spread size limit. We note that given our random selection of words, there is some noise and furthermore that a traversal of a given size can have variable cost depending on whether repetition in the traversal reduces the number of elements requiring application further below the spread size limit. However, it is overwhelmingly the spread size limit that determines the total cost and we observe the expected linear scaling.

While the termination criterion of spreading size limit allows for direct control over computational cost and is convenient for the above analysis, we add an additional termination criterion. A spreading size limit of 300 does not provide a meaningful bound in terms of the influence spreading has on retrieval. In the presence of noise or uncertainty, small values of spreading activation may be irrelevant. We thus introduce a threshold termination criterion such that spreading traversals terminate if spreading activation values generated in the traversal are below the threshold. In other words, we assume a minimum acceptable spreading activation value as an adjustable parameter. We also change the traversal so that instead of breadth-first traversal, the traversal is biased to where there is still the most spread to distribute.

The intuition is to pick a value such that spreading activation is not applied if it would be "lost in the noise." Note that in ACT-R, such a noise term is added to activation. Consider an ACT-R noise set to .1. A threshold of .0025 in Figure 3 represents a 95% chance that such a noise magnitude is larger than the terminated spread. Figure 3 shows that such a termination criterion would result in spread sizes of approximately 65 nodes. Per query, a spread size of 65 nodes is expected to take an average of .094ms. The threshold has the potential to

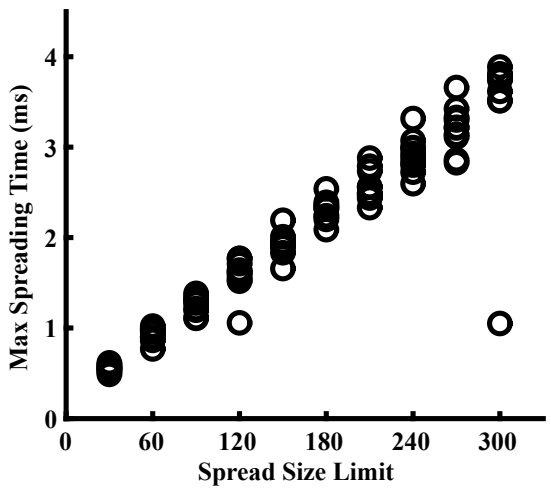

Figure 2: The maximum time spent on spreading activation from a single context node, with varying spread size limit, for randomly selected words.

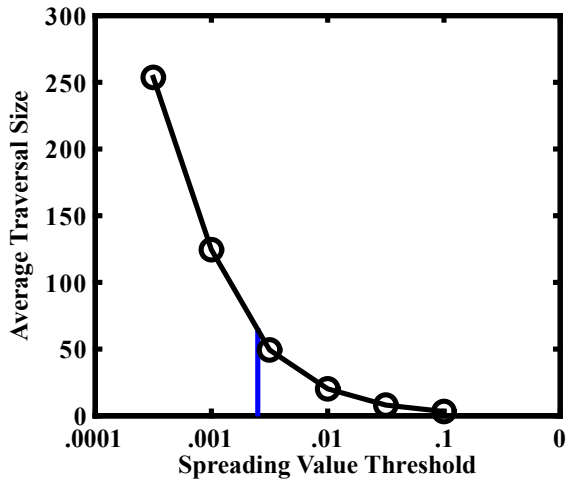

Figure 3: The average sizes of spreading traversals are plotted with varying thresholds for termination. The vertical line denotes a threshold value of .0025.

change which candidates are retrieved.

Conclusion and Future Work

A central motivation of implementing spreading activation is to support context-sensitive retrieval for cognitive agents. To satisfy the constraints of a cognitive architecture while meeting the demands of complex, dynamic, or real-world environments, spreading activation must be efficient and reactive. We have developed an optimized algorithm for spreading activation that has an average time of .43ms for a spread to 300 nodes. Although optimized, there is no compromise in correctness – results of the spread are *exactly the same* as the results of a straightforward (naive) algorithm. Adding a threshold-based termination criterion for spread based on noise or confidence further reduces the cost to .094ms, albeit potentially changing query results. We expect that such efficient spreading activation will change how spreading is used in cognitive architectures. It will be possible to explore deeper spreads where there are more indirect associations between concepts, and it will be possible to use it for real-world applications.

In the future, we plan to further evaluate this algorithm on much larger networks and networks with more varied structure to get a better profile of its performance characteristics for different network organization and dynamics.

We also plan to extend our algorithm so that it includes a temporal decay for spreading activation. Our plan is to initialize the magnitude of the spread from a source node with that source node's base-level activation. Additionally, we plan to extend the representation of semantic memory so that it includes association strengths between nodes. These two changes should have only minimal impact on the spreading algorithm and its efficiency while allowing us to study algorithms that dynamically modify those association strengths based on the co-occurrence of nodes in working memory. This suite of changes has the potential to allow spreading activation to adapt to an agent's experience, which is lacking in our current implementation.

Acknowledgments

The work described here was supported by the Office of Naval Research under grant number N00014-08-1-0099. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the ONR or the U.S. Government.

References

Anderson, J. R. (1983a). The architecture of cognition. *Cambridge, Mass.: Harvard University Pres*.

Anderson, J. R. (1983b). A spreading activation theory of memory. *Journal of Verbal Learning & Verbal Behavior*.

Chen, Y., Petrovic, M., & Clark, M. (2014). Semmemdb: In-database knowledge activation. In *Flairs conference*.

Derbinsky, N., & Laird, J. E. (2011). A functional analysis of historical memory retrieval bias in the word sense disambiguation task. *Ann Arbor*, *1001*, 48109–2121.

Derbinsky, N., Laird, J. E., & Smith, B. (2010). Towards efficiently supporting large symbolic declarative memories. In *Proceedings of the 10th international conference on cognitive modeling* (pp. 49–54).

Douglass, S. A., & Myers, C. W. (2010). Concurrent knowledge activation calculation in large declarative memories. In *Proceedings of the 10th international conference on cognitive modeling* (pp. 55–60).

Edmonds, M., Atahary, T., Taha, T., & Douglass, S. A. (2015). High performance declarative memory systems through mapreduce. In *Software engineering, artificial intelligence, networking and parallel/distributed computing (snpd), 2015 16th ieee/acis international conference on* (pp. 1–8).

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, *19*(1), 17–38.

Kucera, H., & Francis, W. N. (1967). *Computational analysis of present-day american english*. Providence, RI: Brown University Press.

Laird, J. E. (2012). *The Soar cognitive architecture*. MIT Press.

Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, *28*, 29–41.