A Parallel Algorithm for Text Inference

Sanda M. Harabagiu
University of Southern California
Dept. of Electrical Engineering-Systems
Los Angeles, CA 90089-2562
harabagi@usc.edu

Dan I. Moldovan
Southern Methodist University
Dept. of Computer Science & Engineering
Dallas, TX 75275
moldovan@seas.smu.edu

Abstract

In this paper we describe a highly parallel method for extracting inferences from text. The method is based on a marker-propagation algorithm that establishes semantic paths between knowledge base concepts. The paper presents the structure of the system, the marker-propagation algorithm, and results that show a large degree of parallelism.

1 Introduction to Text Inference

The problem of text inference

Text inference refers to the problem of extracting relevant, unstated information from a text. Humans have a great ability to perform correct inference from text or speech. This is perhaps because we have a great deal of world knowledge and can focus our thoughts and filter out irrelevant facts. The inferences derived from reading a text vary from person to person depending on background, context, mental state and other factors. However, basic inferences are in everyones ability. Consider the text:

S1: `John hit the ball with a bat.`
S2: `It landed far away.`

Verb `hit` has fourteen meanings, noun `ball` has nine meanings and noun `bat` has six meanings. A human is able not only to disambiguate words to construct a collective coherent meaning, but also to visualize how a ball hit by a bat is first touched by the bat, travels through the air and finally touches the ground far away from John. To automate the process of text inference by computers presents several difficulties unsolved yet. Since many of these technical challenges relate to the huge volume of processing required, it seems natural that parallel processing technology is likely to play an important role in implementing text inference applications.

Summary of our solution

Any reasoning system is concerned with three aspects: (1) *knowledge representation* and structure, (2) *rules of inference*, and (3) *control* of inference process. Our system represents linguistic knowledge as a large semantic network that can be regarded as nodes connected by relations. The inference rules are implemented as chains of selected relations that link semantically concepts in the knowledge base. We allow some processes, called markers, to travel along these inference patterns and check linguistic constraints along the way. The inference process is controlled by limiting marker propagations only along patterns that are specified by the user. Massive parallelism is achieved by allowing multiple markers to propagate throughout the large knowledge base spawning other markers along the way.

2 Structure of a Very Large Knowledge Base

WordNet and extensions

Common sense reasoning requires extensive knowledge. We decided to build our knowledge base on top of WordNet lexical database [4], [3] developed at Princeton. WordNet is a semantic dictionary because words are searched based on conceptual affinity with other words. It covers the vast majority of nouns, verbs, adjectives and adverbs from the English language. The words in WordNet are organized in synonym sets, called *synsets*. Each synset represents a concept. There is a rich set of relation links between words and other words, between words and synsets, and between synsets. A large linguistic knowledge base may be developed starting from WordNet.

We took a portion of WordNet and enhanced it in several ways. New case constraint relations were added that link verb concepts to their case roles such as agent, object, instrument, experiencer, and others. Another enhancement was to transform the gloss associated to each concept into defining feature relations that specify the properties of concepts. These improvements increased the connectivity between the knowledge base nodes.

Table 1 shows the number of words and concepts for each part of speech. In its current version WordNet 1.5 has

Part of speech	words	concepts
noun	107,484	60,560
verb	25,768	11,364
adjective	28,762	16,428
adverb	6,203	3,243
Total	168,217	91,595

Table 1. Knowledge base nodes

168,217 words organized in 91,595 synsets, thus a total of 259,812 nodes.

Semantic relations

The noun and verb concepts are structured into hierarchies by using ordering relations such as *isa*, *has_part*, and others. The adjectives and adverbs don't have hierarchies, instead are linked into clusters. The knowledge base relations, that link various nodes, are the building blocks to our solution for text inference. Altogether there are 345,264 links in the knowledge base. Note that, on average there are approximately 1.5 links per node.

3 Marker-Propagation Programming Environment

Marker-Propagation Networks

Marker-propagation computational paradigm is especially suitable for applications where control flow is incompletely specified. Reasoning on knowledge bases falls into this broad category. An SIMD implementation of a much simpler marker propagation model is the SNAP experimental computer [6]. In this paper, we use a more powerful MIMD model that runs only on a simulator.

The marker propagation model can be briefly described as a network consisting of *nodes* and *links*, and a set of *markers* moving through the network according to some propagation rules.

The nodes can independently execute a set of functions, store data, and communicate with other nodes. Nodes may have several identities or labels simultaneously. The links are directional, connecting source nodes to destination nodes. A link has a name, indicating the link type, and it may have some associated functions that are executed when some markers are propagated along it. This is the static part of the model.

The markers are process threads that carry a variable number of fields. They reside inside the nodes and propagate to other nodes through links using user-defined propagation rules. Some of the required marker fields are: the marker type, the propagation function, and the node where the marker originated. The arguments of a propagation rule may be node labels, regular expressions of link labels, and/or functions which govern the interactions between a marker and other markers or nodes. In addition to the required fields, user may define any other fields. A set of functions are normally included to specify the behavior of markers inside nodes.

An application program is loaded into the nodes such that each node has the same set of programs. However, the application functions are triggered only by the arrival of markers to a node. Thus, different nodes execute different functions. This is why we call this a *process flow* model of computation. The processing operands may be any combination of one or more marker data and node data. Processing changes the states of nodes and possibly marker data.

Marker processing

The basic mechanism for handling markers is summarized below. An MPN is mapped into a multiprocessor by allocating several nodes to a processor.

---

1. Fetch marker from the processor input queue in round-robin;
2. Find node to which marker was sent;
3. Execute node function $F_N(node, marker)$;
  3.1 Execute node function prior to any marker synchronization;
  3.2 If marker synchronization
    then search for corresponding marker in waiting list;
  3.3 If the synchronization marker is not found
    marker is placed on waiting list;
  3.4 Execute node function after marker synchronization;
  3.5 Goto 3.2 if there is another synchronization;
4. Evaluate marker propagation rule;
  4.1 Execute propagation rule prior to any marker synchronization
  4.2 Repeat procedures from steps 3.2 and 33
  4.3 Execute propagation rule code after marker synchronization
  4.4 Goto 4.1 if more synchronization is necessary
5. Update marker propagation rule
6. If necessary spawn marker
7. Execute link function $F_L(link, marker)$
8. Place markers on output queues

---

4 Path-Finding Algorithm

Below we describe a marker propagation algorithm for finding relational paths between words in a text. Markers are placed on the nodes corresponding to words in the input text, and then let free to propagate through the knowledge base to establish semantic connections between sentences.

The *inputs* to the algorithm are: (1) a semantic knowledge base as described above, and (2) semantically tagged input text. The input text is considered to be disambiguated, in other words, the correct senses of each word have already been identified. This is necessary in order to start from correct concepts. The *outputs* of the algorithm consist of semantic paths that link a pair of input concepts. To each path it corresponds some inferred sentences that explain closely how the two concepts are logically related. Although

this is an asynchronous MIMD algorithm, the system host has the role of initiating and collecting results.

Step 0. Create and load the knowledge base. The knowledge base is in the form of a semantic network consisting of nodes and relations. It is partitioned arbitrarily into as many parts as processors. Each part is then loaded into that processor memory.

Step 1. Place markers on words in knowledge base. This step consists of creating markers and placing them on the knowledge base nodes corresponding to the input words. The marker creation is done by selecting and filling the marker fields with the values provided by the input.

Step2. Propagate markers. After markers are placed on nodes, they start to propagate according to their propagation rule. New markers are spawned every time a node contains more than one outgoing link that is part of its propagation rule. Whenever a marker reaches a node that does not have an outgoing relation as part of its propagation rule, marker stops. The propagation trail of a marker becomes part of that marker history, thus markers become fatter as they propagate. For reasons that will become evident in the next step, nodes keep a copy of each passing marker.

An example of a propagation rule for markers placed on verbs is any combination of relations `ISA-V`, `Caused_by`, `Caused_to`, `Entail` and `reverse_Entail`. Markers continue to propagate as long as the propagation rule allows. The algorithm detects and avoids cycles by simply checking whether or not a marker has visited already a node. .

Step 3. Detect collisions. A path between two concepts is established when the marker originated from one concept collides with the marker originated from the other concept. Figure 1 shows a path found between verb `hit` from sentence S1 and verb `land` from sentence S2.

Step 4. Extract inferences. The nodes along a path between two concepts provide a rich source of information that explains the semantic connections and coherence relations between these two concepts. Our markers carry case relations which allow us to express inferences as English sentences. The reader may trace the highlighted path from Figure 1 and correlate the nodes with the inference sentences from Table 2.

5 Simulation Results

This section describes some of the results obtained with the algorithm above when running on a WordNet-like knowledge base of 392 nodes, and 605 links. This knowledge base contains all the semantic senses of the verbs and nouns in the examples below. Thus, the knowledge base is a "window" of the knowledge base described in Section 2.

Inference sequence
John hit the ball with a bat
John propelled the ball with a bat
John moved forward the ball with a bat
Ball moved ⇐ COLLISION
It moved
It flew
It landed far away

Table 2. Inferences resulted from coherence path

Experiment 1 Find the coherence paths between S1 and S2-1.
S1: `John hit the ball with a bat.`
S2-1: `It landed far away.`
For these sentences, the algorithm found 30 paths; one path is illustrated in Figure 1. Out of these 30 paths, only 20 were semantically correct. The algorithm started with seven markers and the total number of markers in the system was 54, thus 47 new markers were created. Out of the 392 nodes, 181 nodes were visited by markers. In the next section, we introduce some metrics for algorithm performance and summarize the results from several experiments.

Experiment 2 Find the coherence paths between S1 and S2-2.
S1: `John hit the ball with a bat.`
S2-2: `He felt relieved.`
The first sentence is the same, but the second one is slightly changed from the previous example. The algorithm found 11 paths of which 6 were semantically correct. One is:

`hit#1v` $-isa\rightarrow$ `propel#1v` $-isa\rightarrow$ `move#2v` $-cause\_to\rightarrow$ `move#3v` $-df\_isa\rightarrow$ `change#1v`$**$

`relieved#1adj` $-pertain\rightarrow$ `relieve#1v` $-isa\rightarrow$ `ameliorate#1v` $-isa\rightarrow$ `change#1v`$**$

The path starts from `hit#1`, which is move forward, which is displace, which causes to change position, which is a synonym of change. Here collision took place with the path starting from `relieve#1` which is synonym with alleviate, which is to ameliorate, which is to change.

Experiment 3 Find the coherence paths between S1 and S2-3.
S1: `John hit the ball with a bat.`
S2-3: `It broke.`
For this text, the algorithm found 16 paths of which 11 were semantically correct. One correct path is:

`ball#1n` $-df\_isa\rightarrow$ `object#1n` $-dfc\_r\_obj\rightarrow$ `hit#1v` $-isa\rightarrow$ `propel#1v` $-isa\rightarrow$ `move#2v` $-cause\_to\rightarrow$ `move#3v` $-df\_isa\rightarrow$

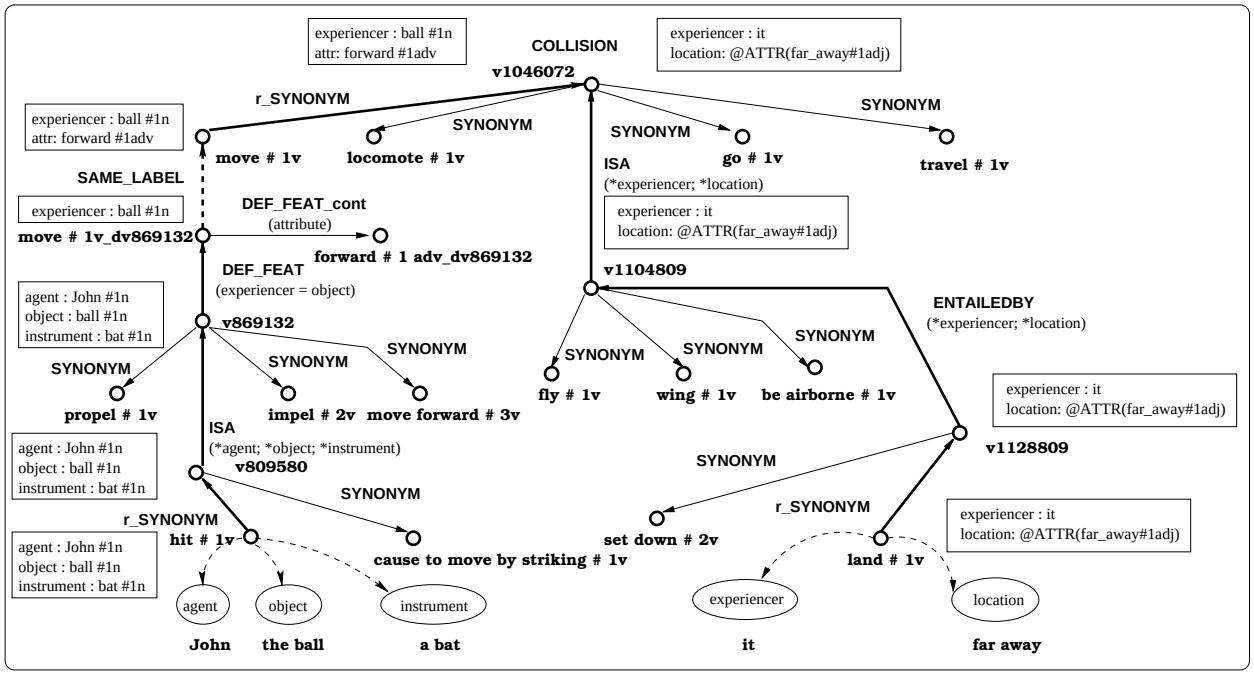

Figure 1. Coherence path between S1 and S2

change#1v**

---

break#4v –*isa*→ change#1v**

This path shows a connection from ball to brake. Noun ball#1 has the meaning of a sport ball which has defining feature that is hit. From here, the path takes us to propel, which is a displacement, which causes to change position , which is a change. Here is the collision with the path from break which causes a change of state, which is a change.

6 Algorithm Performance and Parallelism Analysis

Algorithm performance metrics

Little is known about inference algorithms, mainly because there are not many such algorithms. Algorithm performance should reflect both the quality of inferences, as well as the computational efficiency measured in terms of processing time and memory space. While we realize intuitively that some inferences are more important than others, it is not known yet how to rank inferences. Unfortunately, we can not yet link algorithm performance to the "goodness of inferences".

In this paper, we propose two metrics that indicate how many useful inferences were found by the algorithm and how many were missed. Since inferences are closely related to the semantic paths in the knowledge base, we will refer to these paths instead. The algorithm may find paths that are semantically incorrect and lead to erroneous inferences, or algorithm may miss some correct paths that exist in the knowledge base.

Let us denote with n the total number of paths found by the algorithm, n_c the number of correct paths found by the algorithm, and N_c the number of correct paths that exist in the knowledge base. A correct path is one that is semantically correct, meaning that contains concepts leading to inferences acceptable to an average human.

Definition 1. *Recall* is the ratio between the number of correct paths found by the system over the number of correct paths that exist in the knowledge base.

$$Recall = \frac{n_c}{N_c}$$

Definition 2. *Precision* is the ratio between the number of correct paths found by the system over the total number of paths found.

$$Precision = \frac{n_c}{n}$$

A human evaluator needs to determine n_c out of all the paths produced by the algorithm. Also, the evaluator needs to inspect the knowledge base and determine all the correct

paths. While this is in general difficult, it may be possible for some examples, for the purpose of tuning the algorithm marker propagation rules.

For example, in Experiment 1 described above $n = 30, n_c = 20$, and $N_c = 25$. The numbers n and n_c result by inspecting the path, and N_c was found by inspecting the knowledge base. This results in a 66% recall and 80% precision.

Parallelism performance

Several sources contribute to the large degree of parallelism available. First, it is the fact that markers may originate in parallel from many words in the sentence pair. Another source of parallelism results from the branching in the knowledge base. When a marker is processed in a node, it may spawn other markers whenever there are several outgoing relations on which its propagation rule allows markers to move. These two sources of parallelism are reflected in the number of markers produced by the system.

The algorithm speed up may be computed as the ratio between the total number of marker propagations (sequential time), over the number of propagations along the longest path (parallel time). Here we assume that a marker propagation includes communication as well as processing time. Let P_{total} be all marker propagations, and P_{path} the propagations along the longest path.

$$Speedup = \frac{P_{total}}{P_{path}}$$

For example, in Experiment 1 there were 54 markers in the system, and the total number of nodes marked was 181. The total number of marker propagations was 338, and the longest path had 12 propagations, which results in a speed up of 28.1

Summary of results

Table 3 summarizes our measurements for the three experiments discussed above. The number of nodes and relations gives information about the knowledge base. The next entry indicates the number of nodes that were marked, in each case. Then, the number of markers indicates the *degree of parallelism*. Recall, precision and speed up are computed for each case.

Parameter	Exp. 1	Exp. 2	Exp. 3
Nodes	392	346	278
Relations	605	532	418
Nodes marked	181	132	116
Markers total	54	44	40
n	30	11	16
n_c	20	6	11
N_c	25	8	13
$Recall$	66%	55%	69%
$Precision$	80%	75%	85%
P_{total}	338	272	258
P_{path}	12	10	11
$Speedup$	28.1	27.2	23.4

Table 3. Measurements summary

7 Conclusions

We have identified a parallel method for extracting plausible inferences from text. A practical way of constructing large, scalable, general-purpose knowledge bases is to expand WordNet.

The markers we used are considerably more complex then spreading activation markers proposed by Quillian [7], and used later by [2], [1], [5], and others. Our markers have fields for case constraints, synchronization with other markers, and propagation rules. We think that by allowing the user to program the marker fields is a clear advantage. Markers may be regarded as lightweight processes that execute asynchronously. Some problems with marker propagation, however, are the detection of the end of propagation, and using a large amount of memory to keep copies of markers.

Acknowledgment

We are grateful to Takashi Yukawa, visiting scientist from NTT, Japan, who has contributed to the construction of the larger knowledge base and conducted several experiments. This work was partially supported by NSF grant CCR-9406998.

References

[1] E. Charniak. A neat theory of marker passing. In *Proceedings of the Fifth National Conference on Artificial Intelligence AAAI-86*, pages 584–588, 1986.

[2] S. Fahlman. *NETL: A System for Representing and Using Real-World Knowledge*. The MIT Press, Cambridge, Massachusetts, 1979.

[3] C. Fellbaum. English verbs as a semantic net. Technical Report CSL 90-43, Princeton University, 1990.

[4] C. Fellbaum, D. Gross, and G. Miller. Wordnet : A lexical database organized on psycholinguistic principles. In U. Zernik, editor, *Lexical Aquisition: Exploiting On-Line Resources to Build a Lexicon*, pages 141–170. Lawrence Erlbaum Associates Publishers, Hillsdale, N.J., 1991.

[5] J. Hendler. *Integrating Marker-Passing and Problem-Solving*. Lawrence Erlbaum Associates Publishers, 1988.

[6] D. Moldovan, W. Lee, and C. Lin. Snap: A marker-propagation architecture for knowledge processing. *IEEE Transactions on Parallel and Distributed Systems*, 3:1–13, 1992.

[7] M. Quillian. *Semantic Memory*. PhD thesis, Carnegie Institute of Technology (Carnegie Mellon University), 1966.