Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Concurrent Reasoning with Inference Graphs

Daniel R. Schlegel and **Stuart C. Shapiro**
Department of Computer Science and Engineering
University at Buffalo, Buffalo NY, 14260
<drschleg,shapiro>@buffalo.edu

1 Introduction

Since at least the early 1980s there has been an effort to utilize multiple processors for logical reasoning. Prior to the rise of the multi-core desktop computer, this often meant parallel algorithms on specialized hardware. Parallel logic programming systems designed during that same period were less attached to a particular parallel architecture, but parallelizing Prolog is a very complex problem (Shapiro 1989). Parallelizing Datalog has been more successful, but it is a less expressive subset of Prolog. Recent work in statistical inference has returned to large scale parallelism using GPUs, but while GPUs are good at statistical calculations, they do not do logical inference well (Yan, Xu, and Qi 2009).

We present *inference graphs*, a graph-based natural deduction inference system which lives within a KR system and is capable of taking advantage of multiple cores and/or processors using concurrrent processing techniques rather than parallelism. Inference graphs extend propositional graphs so that the representation of knowledge is also the inference system, something we believe is unique among logical inference systems. We chose to use natural deduction inference, despite the existence of very well performing refutation based theorem provers, because our system is designed to be able to perform forward, bi-directional (Shapiro, Martins, and McKay 1982), and focused reasoning in addition to the backwards inference used in resolution. Natural deduction also allows formulas generated during inference to be retained in the KB for later re-use, whereas refutation techniques always reason about the negation of the formula to be derived, making intermediate derivations invalid. In addition, our system is designed to allow formulas to be disbelieved, and to propogate that disbelief to dependent formulas. We believe inference graphs are the only concurrent inference system with all these capabilities.

2 Propositional Graphs

Propositional graphs in the tradition of the SNePS family (Shapiro and Rapaport 1992) are graphs in which every well-formed expression in the knowledge base, including individual constants, functional terms, atomic formulas, or non-atomic formulas (which we will refer to as "rules"), is represented by a node in the graph. A rule is represented in the graph as a node for the rule itself (henceforth, a *rule node*), nodes for the argument formulas, and arcs emanating from the rule node, terminating at the argument nodes. Arcs are labeled with an indication of the role the argument plays in the rule, itself. Every node is labeled with an identifier. Nodes representing individual constants, proposition symbols, function symbols, or relation symbols are labeled with the symbol itself. Nodes representing functional terms or non-atomic formulas are labeled `wft`i, for some integer, i. No two nodes represent syntactically identical expressions; rather, a single node is used if there are multiple occurrences of one subexpression.

3 Inference Graphs

An inference graph is a propositional graph in which certain arcs and certain reverse arcs are augmented with *channels* through which information can flow – meaning the inference graph is both a representation of knowledge and the method for performing inference upon it (see Fig. 1). Channels come in two forms. The first type, *i-channels*, are added to the reverse antecedent arcs – named as such since they carry messages reporting that "I am true" or "I am negated" from the antecedent node to the rule node. Channels are also added to the consequent arcs, called *y-channels*, since they carry messages to the consequents which report that "you are true" or "you are negated." Rules are connected by shared subexpressions (such as `d` in Fig. 1).

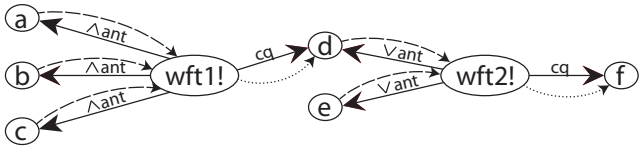

Figure 1: The propositional graph for the propositions that if {`a`,`b`,`c`} are all true, then `d` is true, and if one of {`d`,`e`} is true, then `f` is true. Inference graph channels are also shown: dashed lines are i-channels (drawn from antecedents to rule nodes), and dotted lines are y-channels (drawn from rule nodes to consequents).

Each channel contains a valve. Valves enable or prevent the flow of messages forward through the graph's channels.

When a valve is closed, any new messages which arrive at it are added to a waiting set. When a valve opens, messages waiting behind it are sent through.

Messages of several types are transmitted through the inference graph's channels, serving two purposes: relaying newly derived information, and controlling the inference process. A message can be sent to relay the information that its origin has been asserted or negated (an I-INFER message), that its destination should now be asserted or negated (Y-INFER), or that its origin has either just become unasserted or is no longer sufficiently supported (UNASSERT). These messages flow forward through the graph. Other messages flow backward, controlling inference by affecting the channels: BACKWARD-INFER messages open them, and CANCEL-INFER messages close them. The use of messages to control valves allows inference graphs to perform forward, backward, bi-directional, and focused inference, and the messages are prioritized so these operations are all performed efficiently (discussed further in Sec. 4).

Inference operations take place in the rule nodes. When a message arrives at a rule node the message is translated into *Rule Use Information*, or RUI (Choi and Shapiro 1992). RUIs contain information about how many (and specifically which) antecedents of a rule are known to be true or false, along with a set of support. All RUIs created at a node are cached. When a new one is made, it is combined with any already existing ones. The output of the combination process is a set of new RUIs created since the message arrived at the node. By looking at the number of known true or false antecedents, this set is used to determine if the rule node's inference rules can be applied. RUIs prevent re-derivations and cut cycles by ignoring arriving RUIs already in the cache. The disadvantage of our approach is that some rules are difficult to implement such as negation introduction and proof by cases. For us, the advantages in capability outweigh the difficulties of implementation.

4 Concurrent Reasoning

The inference graph's structure lends itself naturally to concurrent inference. For example, each antecedent of `wft1` in Fig. 1 could be derived concurrently. The RUIs generated from the messages would then need to be combined, before a Y-INFER message could be sent to `d`. Since there is shared state (the RUI cache) we perform the combination of RUIs synchronously, guaranteeing we don't "lose" results.

In order to perform inference concurrently, the inference graph is divided into *inference segments* (henceforth, *segments*). A segment represents the inference operation – from receipt of a message to sending new ones – which occurs in a node. Valves delimit segments. When a message passes through an open valve a new *task* is created – the application of the segment's inference function to the message. When tasks are created they enter a global prioritized queue, where the priority of the task is the priority of the message. When the task is executed, inference is performed as described above, and any newly generated messages are sent toward its outgoing valves for the process to repeat.

If we arrange an inference graph so that a user's request (in backward inference) is on the right, and channels flow (where possible, cycles can exist) from left to right, we can see the goal of inference as trying to get messages from the left side of the graph to the right side of the graph. Every inference operation begins processing messages some number of levels to the left of the query node. Since there are a limited number of tasks which can be running at once due to hardware limitations, we must prioritize their execution, and remove tasks which we know are no longer necessary. Therefore,

1. tasks for relaying newly derived information using segments to the right execute before those to the left, and
2. once a node is known to be true or false, all tasks attempting to derive it (left of it in the graph) are canceled, as long as their results are not needed elsewhere.

Together, these two heuristics ensure that messages reach the query as quickly as possible, and time is not wasted deriving unnecessary formulas. The priorities of the messages (and hence, tasks) allow us to reach these goals. All UNASSERT messages have the highest priority, followed by all CANCEL-INFER messages. Then come I-INFER and Y-INFER messages. BACKWARD-INFER messages have the lowest priority. The priority of I-INFER and Y-INFER messages increase as they flow to the right, but they remain lower than that of CANCEL-INFER messages.

5 Evaluation Summary

To evaluate the speedup in inference as the number of processors increases, we automatically generated graphs of chaining entailments with multiple antecedents and a single consequent. The system then backchained on and derived a consequent when the entailments required either all the antecedents to be true, or only one. We found, when ignoring the intrinsically sequential portions of inference, nearly linear speedup with the number of processors. In addition, the heuristics presented resulted in a nearly 10x speedup over using LIFO queues, and 20-40x over FIFO queues.

References

Choi, J., and Shapiro, S. C. 1992. Efficient implementation of non-standard connectives and quantifiers in deductive reasoning systems. In *Proc. HICSS-25*. Los Alamitos, CA: IEEE Computer Society Press. 381–390.

Shapiro, S. C., and Rapaport, W. J. 1992. The SNePS family. *Comput Math Appl* 23(2–5):243–275. Reprinted in Lehmann, F. 1992. Semantic Networks in Artificial Intelligence. Pergamon Press.

Shapiro, S. C.; Martins, J. P.; and McKay, D. P. 1982. Bi-directional inference. In *Proc. CogSci-4*.

Shapiro, E. 1989. The family of concurrent logic programming languages. *ACM Comput. Surv.* 21(3):413–510.

Yan, F.; Xu, N.; and Qi, Y. 2009. Parallel inference for latent dirichlet allocation on graphics processing units. In *Proc. NIPS*, 2134–2142.

This work has been supported by a Multidisciplinary University Research Initiative (MURI) grant (Number W911NF-09-1-0392) for Unified Research on Network-based Hard/Soft Information Fusion, issued by the US Army Research Office (ARO) under the program management of Dr. John Lavery.