An Anytime Algorithm for Interpreting Arguments

Sarah George, Ingrid Zukerman, and Michael Niemann

School of Computer Science and Software Engineering
Monash University, Clayton, VICTORIA 3800, AUSTRALIA
{sarahg,ingrid,niemann}@csse.monash.edu.au

Abstract. The problem of interpreting Natural Language (NL) discourse is generally of exponential complexity. However, since interactions with users must be conducted in real time, an exhaustive search is not a practical option. In this paper, we present an anytime algorithm that generates "good enough" interpretations of probabilistic NL arguments in the context of a Bayesian network (BN). These interpretations consist of: BN nodes that match the sentences in a given argument, assumptions that justify the beliefs in the argument, and a reasoning structure that adds detail to the argument. We evaluated our algorithm using automatically generated arguments and hand-generated arguments. In both cases, our algorithm generated good interpretations (and often the best interpretation) in real time.

1 Introduction

Discourse interpretation is a complex task that is essential for human-computer interaction. This complexity arises from the large number of choices to be made at different stages of the interpretation process, e.g., there are several possible referents for each noun, each sentence could have more than one meaning, and sentences may relate to each other in a variety of ways. As a result, the problem of finding an interpretation of Natural Language (NL) discourse is exponential. However, interactions with users must be conducted in real time. This precludes an exhaustive search for the best interpretation of a user's discourse, and leads us to the idea of a "good enough" interpretation.

Anytime algorithms were introduced by Dean [1] and Horvitz *et al.* [2] in the late 1980's to produce approximate solutions to complex problems in limited time. In this paper, we present an anytime algorithm for discourse interpretation. Our algorithm receives as input probabilistic arguments presented by users to an argumentation system called BIAS (*Bayesian Interactive Argumentation System*). These arguments are composed of NL sentences linked by means of argumentation connectives (a small sample argument is shown in Fig. 1).

Our system uses Bayesian networks (BNs) [3] as its knowledge representation and reasoning formalism. Our domain of implementation is a murder mystery, which is represented by a 32-node BN. An interpretation of an argument consists of nodes in the domain BN that match the sentences in the argument, assumptions (values for BN nodes) that account for the beliefs stated in the argument, and a reasoning structure (a subnet of the domain BN) that connects the identified nodes. For instance, the subnet in Fig. 1 illustrates an interpretation generated for the argument on the left-hand-side. The italicized nodes are those mentioned in the argument, and the grey, boxed node is an assumption (inferred by the system) that accounts for the beliefs in the argument.

In the next section, we discuss related research. We then define interpretations in the context of BNs, and describe our anytime algorithm for argument interpretation. In Section 5, we evaluate our algorithm's performance, followed by concluding remarks.

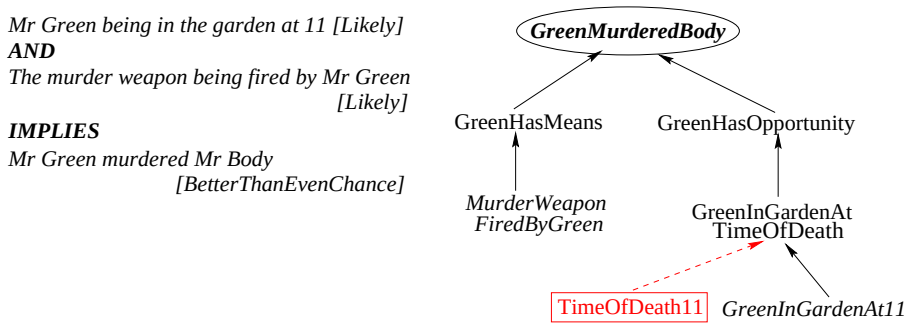


Fig. 1. Sample argument and interpretation

2 Related Research

Discourse interpretation systems typically employ different resources to fill in information omitted by a user, e.g., [4, 5]. However, most interpretation systems developed to date have focused on the procedures and knowledge sources required for generating interpretations, ignoring issues of efficiency and real-time performance.

These issues have been addressed by anytime algorithms in the context of planning, diagnosis and decision-making under uncertainty, e.g., [1, 6]. In areas more related to our work, anytime algorithms have been used in a system that generates proofs for hypotheses [7], and in spoken dialogue systems [8, 9].

Haenni [7] applied an anytime algorithm based on the Dempster-Schaefer formalism for generating proofs for hypotheses. Like BIAS, his system aims for a concise proof. However, he used heuristics to select propositions to be included in the proof, while we use a complex function that combines different attributes of an interpretation. The anytime algorithm described in [8] improves the quality of the surface realization of spoken responses to users' queries, while the algorithm presented in [9] interprets users' spoken queries. Both systems are implemented in the travel timetable domain. Our work resembles most that of Fischer *et al.* [9] in the sense that their system also maps users' queries to points in a network (although they use a semantic network), and they also use a cost function to assess the quality of an interpretation. However, their domain of discourse is significantly more restricted than ours.

3 What is an interpretation?

An interpretation of an argument in the context of a BN is a tuple $\{NC,AC,IG\}$, where NC is a *node configuration*, AC is an *assumption configuration*, and IG is an *interpretation graph*.

A **Node Configuration** is a set of nodes in the domain BN that match the sentences in an argument. Each node in the BN is associated with one or more *canonical sentences* that express its content. For instance, the canonical sentence for GreenHasOpportunity in Fig. 1 is "Mr Green had the opportunity to murder Mr Body". BIAS uses a cosine similarity measure [10] complemented with an automatically-derived word-similarity score [11] to estimate the similarity between an input sentence and a canonical sentence associated with a node. For example, an input sentence such as "Mr Green had the chance to kill Mr Body" is considered similar to "Mr Green had the opportunity to murder Mr

Body" and (a little less similar) to "Mr Green murdered Mr Body". Hence, arguments that contain this sentence yield node configurations that include GreenHasOpportunity and GreenMurderedBody. The process for proposing candidate node configurations is described in Section 4.1.

An **Assumption Configuration** is a set of assumptions made by BIAS to account for the beliefs in an argument. For example, for the argument in Fig. 1 to make sense, the system has to assume that the time of death was 11. At present, our BN nodes are binary. Hence, the possible assumptions are: SET TRUE – assume that a node is True; SET FALSE – assume that a node is False; and UNSET – assume no direct evidence for this node.

An **Interpretation Graph** is a subnet of the domain BN which links the nodes that correspond to the antecedents in an argument (according to a particular node configuration) to the nodes that correspond to the consequents. Note that a good interpretation graph is not necessarily the minimum spanning tree that connects the nodes in question, as this spanning tree may have blocked paths (through which evidence cannot propagate in a BN [3]), which render an interpretation invalid.

4 Anytime Algorithm

Algorithm *GenerateInterpretations* (Fig. 2) receives as input an argument *Arg*, and returns the best N (=4) interpretations among those considered in the available time. To this effect, it matches the sentences in the argument to nodes in the domain BN (Step 1), makes assumptions that are warranted by the beliefs in the argument (Step 2), and proposes subnets of the BN that connect the nodes in the argument (Step 3). Each of these steps selects the "best" component of an interpretation based on local knowledge. However, this component is not necessarily the best overall when considered in conjunction with the other components. Fig. 3(a) depicts the search tree generated by our algorithm, where each level of the tree corresponds to a different component. Fig. 3(b) instantiates this search tree with respect to a small example.

4.1 Getting a Node Configuration

Algorithm *GetNodeConfig* (Fig. 2) receives as input the sentences in an argument, and returns a set of matching nodes – one node per sentence. The algorithm selects a node configuration from a list called *NodeConfigList*. This list comprises two main parts: {*PrevNodeConfigList*, *MakeNewConfig*}, where *PrevNodeConfigList* is a list of previously generated node configurations, and *MakeNewConfig* is a call to a function that returns the next best configuration of a given type, e.g., a node or assumption configuration, or an interpretation graph (Section 4.4).

For the example in Figure 3(b), *PrevNodeConfigList* is initially empty, hence *MakeNewConfig(Node)* must be called, returning node configuration NC_1. This configuration is added to *PrevNodeConfigList*, and returned by *GetNodeConfig*. Now, *NodeConfigList* contains two elements: {NC_1, *MakeNewConfig*}, where NC_1 is selected with probability 0.952, and *MakeNewConfig* with probability 0.048 (=5% of 0.952). If *MakeNewConfig* is selected next time *GetNodeConfig* is called, then *NodeConfigList* will contain three elements: {NC_1, NC_2,*MakeNewConfig*}.

Algorithm *GenerateInterpretations(Arg)*
while {there is time}
{
1. // Get a node configuration in the domain BN that matches the argument
   $NC \leftarrow$ *GetNodeConfig(Arg)*
2. // Get an assumption configuration that accounts for the beliefs stated for the nodes in NC
   $AC \leftarrow$ *GetAssumptionConfig(Arg,NC)*
3. // Get an interpretation graph that connects the nodes in NC
   $IG \leftarrow$ *GetInterpretationGraph(NC,AC)*
4. Evaluate interpretation $\{NC, AC, IG\}$.
5. Retain top N (=4) interpretations.

}

Algorithm *GetNodeConfig(Arg)* (Section 4.1)

1. Select an element from *NodeConfigList* at random (all previously generated configurations are equiprobable, and the probability of generating a new configuration is Pr_{new}% (=5%) of the probability of any of the previous configurations).
2. If *MakeNewConfig(Node)* was called, (Section 4.4)
   Then insert the node configuration returned by this function in *PrevNodeConfigList*.
3. Return the chosen configuration.

Algorithm *GetAssumptionConfig(Arg,NC)* (Section 4.2)

1. If *AssumptionConfigList* is empty
   (a) Call *MakeNewConfig(Assumption)* K times (= 200), where each time *MakeNewConfig* returns the best assumption configuration. (Section 4.4)
   (b) Assign the top k (=3) assumption configurations to *AssumptionConfigList*.
2. Select an element from *AssumptionConfigList* at random.
3. Return the chosen configuration.

Algorithm *GetInterpretationGraph(NC,AC)* (Section 4.3)
Call *MakeNewConfig(InterpretationGraph)*, and return the configuration it produced.

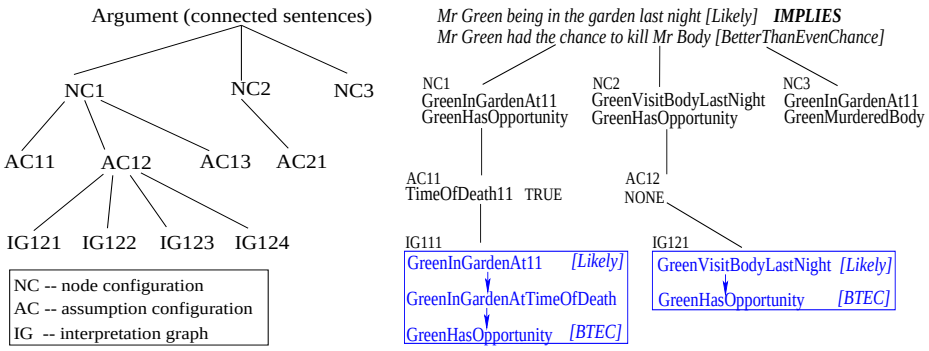
Algorithm *MakeNewConfig(ConfigType)* (Section 4.4)

1. If the priority queue is empty, propose an initial configuration, calculate its probability, and add the configuration and its probability to the priority queue.
2. Remove the first configuration from the queue.
3. Generate the children of this configuration, calculate their probability, and insert them in the queue so that the queue remains sorted in descending order of probabilities.
4. Return the chosen (removed) configuration.

Fig. 2. Anytime algorithm for argument interpretation

Algorithm *GetNodeConfig* reduces the fan-out factor of node configurations by having an initial low probability of generating a new node, and further reducing this probability as the list of configurations grows. This low fan-out is due to the fact that the intended node configuration is usually among the top three returned by our parser.

Note that when a node configuration is generated, a belief mentioned regarding a particular statement becomes associated with the node that matches that statement. For example, in Fig. 3(b), the belief of BetterThanEvenChance given for the sentence "Mr Green had the chance to kill Mr Body" is associated with node GreenHasOpportunity for node configuration NC_1, and with GreenMurderedBody for NC_3.



(a) Search tree (b) Sample search tree

Fig. 3. Process for generating interpretations

4.2 Getting an Assumption Configuration

For each node configuration, the system determines whether it needs to make assumptions so that the beliefs in the BN resulting from Bayesian propagation match the beliefs stated in an argument. For instance, if "Mr Green being in the garden last night" is interpreted as GreenInGardenAt11 (Fig. 3(b)), then BIAS must assume that the time of death was 11 (TimeOfDeath11 TRUE) for the beliefs in the BN to match those in the argument (AC_{11}). In contrast, if this sentence is interpreted as GreenVisitBodyLastNight, no assumptions are required (AC_{12}).

Algorithm *GetAssumptionConfig* (Fig. 2) receives as input an argument *Arg* and a node configuration NC, and returns an assumption configuration, i.e., a set of nodes in the BN accompanied by their assumed beliefs. This algorithm randomly selects an assumption configuration from a list of configurations denoted *AssumptionConfigList*, which is composed of the top k (=3) assumption configurations among those returned by *MakeNewConfig* (Section 4.4).

For the example in Figure 3(b), assume that we are under node configuration NC_1, and that *AssumptionConfigList* is empty. Hence, *MakeNewConfig(Assumption)* is called K times, returning each time the best assumption configuration. After these K calls, the top k configurations are retained: $\{AC_{11}, AC_{12}, AC_{13}\}$. From now on, every time *GetAssumptionConfig* is called, it selects one of these configurations at random.

4.3 Getting an Interpretation Graph

There are many ways to connect the nodes in an argument, and each way yields a different interpretation graph. Interpretation graphs are generated after making assumptions (even though assumed nodes are not part of an interpretation graph), because the validity of an interpretation is influenced by the assumptions.

Algorithm *GetInterpretationGraph* (Fig. 2) receives as input an assumption configuration AC and a node configuration NC, and returns an interpretation graph – a Bayesian subnet that connects the nodes in NC. This algorithm simply calls *MakeNewConfig(InterpretationGraph)*, which returns the next best interpretation graph from a priority queue. The output of this algorithm is illustrated by the two interpretation graphs at the bottom of the search tree in Fig. 3(b).

Since this algorithm is activated for the last layer of the search tree, the interpretation graphs it returns do not need to be cached for further processing, as opposed to node configurations and assumption configurations.

4.4 Making a New Configuration

MakeNewConfig is at the core of our anytime algorithm. It returns a new configuration every time it is called (Fig. 2). This may be a node configuration, an assumption configuration or an interpretation graph. The algorithm maintains a priority queue of configurations and their probabilities. Each time it is called, it removes the configuration at the top of the queue, generates its "child configurations" (configurations derived from the selected one), inserts them in the queue, and returns the selected configuration.

Algorithm *MakeNewConfig* performs three activities that require calls to specialized functions: *propose an initial configuration*, *generate children of a configuration*, and *calculate the probability of a configuration*. A *static* approach is applied to perform the first two activities for node and assumption configurations, and a *dynamic* approach for interpretation graphs.

Static approach. In order to generate the next node or assumption configuration, the algorithm maintains a structure called *Score Table*, which maps an input element to a decision (e.g., a sentence to a matching node, or a node to an assumption about it). This structure is obtained from our sentence parser or our assumption generator.

The Sentence Score Table is a list where each element corresponds to a sentence in the argument. Each sentence in turn is associated with a list of <node: probability> pairs – one pair for each node in the domain BN – ordered in descending order of probability (Fig. 4(a)). Each pair represents the probability that this node is intended by the sentence in question, which is calculated based on the similarity between this sentence and the canonical sentences for the node [11].

Each element in the Assumption Score Table corresponds to a node in the BN. Each node is associated with a list of <assumption: probability> pairs – one pair for each type of assumption (Fig. 4(b)). Each pair represents the probability of making this assumption about the node in question, which is obtained using heuristics such as the following: not changing the belief in a node has the highest probability, and asserting the belief in a node has a higher probability than contradicting it (e.g., a belief of 0.8 has a higher probability of being assumed true than false).

The three activities mentioned above are performed as follows.

Propose an initial configuration – Select the first row from the Score Table.

Generate children of a configuration – The ith child is generated by moving down one place in list i in the Score Table, while staying in the same place in the other lists.

Calculate the probability of a configuration – For node configurations, this probability is the product of the probabilities of the entries in a configuration. For assumption configurations, this probability is a function of the "cost" of making a set of assumptions (the higher the product of the probabilities of the entries in an assumption configuration, the lower the cost) and the "savings" due to a closer match between the beliefs stated in an argument and those in the BN as a result of making the assumptions (the calculation of this component is described in [12]). For instance, for the example in Fig. 1, the time of death assumption reduces the discrepancy between

sentence$_1$	*sentence*$_2$	...	*sentence*$_m$
n_1: 0.2	n_{15}: 0.4	...	n_3: 0.3
n_{24}: 0.15	n_{10}: 0.3	...	n_{20}: 0.1
...	...	...	...
n_{32}: 0.0	n_2: 0.01	...	n_{13}: 0.02

(a) Sentence Score Table

node$_1$		*node*$_2$		...	*node*$_{32}$	
UNSET:	0.8	SET TRUE:	0.95	... UNSET:		0.5
SET TRUE:	0.1	UNSET:	0.04	... SET TRUE:		0.3
SET FALSE:	0.1	SET FALSE:	0.01	... SET FALSE:		0.2

(b) Assumption Score Table

Fig. 4. Sample Score Tables for node configurations and assumption configurations

the user's stated belief in GreenMurderedBody and that in the BN in the absence of this assumption.

To illustrate the operation of this algorithm, consider the Sentence Score Table in Fig. 4(a). Initially, the first row is selected, yielding nodes $\{n_1, n_{15}, \ldots, n_3\}$ which have probability $0.2 \times 0.4 \times \ldots \times 0.3$. Prior to returning this configuration to *GetNodeConfig*, its children are generated: $\{\underline{n_{24}}, n_{15}, \ldots, n_3\}$, $\{n_1, \underline{n_{10}}, \ldots, n_3\}$, ..., their probabilities are calculated, and they are inserted in the queue in descending order of their probability. Next time *MakeNewConfig* is called for node configurations, the first configuration in the queue will be removed, its children will be generated, and so on.

Dynamic approach. This procedure is described in detail in [12]. Here we provide a brief outline.

Propose an initial configuration – Generate the minimum spanning tree that connects the nodes corresponding to the argument.

Generate children of a configuration – The children of an interpretation graph are generated by iteratively "growing" the graph, i.e., adding nodes and arcs.

Calculate the probability of a configuration – The probability of an interpretation graph is a function of its size (the larger the graph the lower its probability), its structural similarity with the argument, and the probability that its nodes were implied by the argument (nodes that were previously seen by the user are more likely than nodes with which the user is unfamiliar).

4.5 Algorithm Analysis

Our anytime algorithm may be classified as *interruptible* [6], as it can be interrupted at any time to produce results whose quality is described by its *Conditional Performance Profile (CPP)*. Notice, however, that our algorithm has a small fixed-time component due to the K calls to *MakeNewConfig* from *GetAssumptionConfig*.

The operation of *GetAssumptionConfig* differs from that of *GetNodeConfig* and *GetInterpretationGraph* as follows. *GetAssumptionConfig* makes a random selection from a static list of the k best assumption configurations (selected from K assumption configurations generated by *MakeNewConfig* for a particular node configuration), while the other two procedures iteratively call *MakeNewConfig* to obtain a new node configuration or interpretation graph. This difference is due to the fact that the process which generates the children of a node configuration or an interpretation graph reliably proposes items of decreasing goodness, while this is not necessarily the case for the process which generates the next assumption configuration. This is because the assumptions are generated from the top of the Assumption Score Table, but their goodness can be determined only after Bayesian propagation is performed.

5 Evaluation

Our evaluation focuses on the anytime algorithm, i.e., the time BIAS takes to produce a good interpretation, rather than on BIAS' ability to produce plausible interpretations (which was evaluated in [12]).

Our evaluation consists of two experiments: one where the system interprets automatically generated arguments, and one where it interprets hand-generated arguments. The arguments in both experiments were designed to test the effect of three factors on BIAS' performance: *argument size*, *interpretation complexity*, and *belief distortion*.

Argument size measures the number of nodes in an argument. We considered three argument sizes: Small (2-3 nodes), Medium (4-5 nodes) and Large (6-7 nodes).

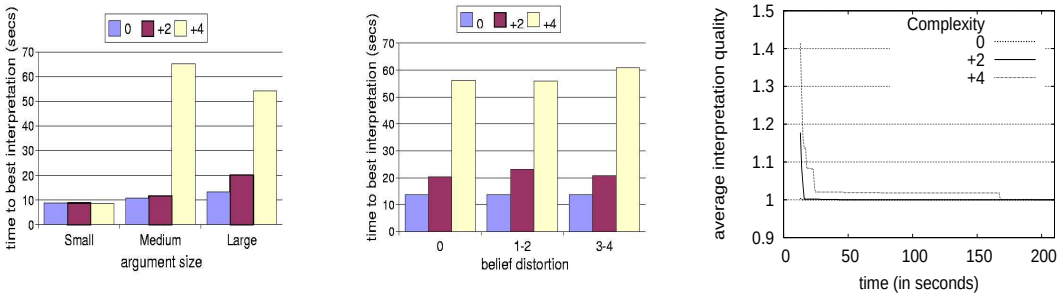
Interpretation complexity measures how much of an "inferential leap" is performed in order to connect between the nodes in an argument. The complexity of an interpretation is approximated by means of the number of nodes in the minimum spanning tree that connects the nodes in an argument. We considered three levels of complexity for our automatic evaluation: 0 (the minimum spanning tree includes only the nodes in the argument), +2 (the minimum spanning tree includes 2 additional nodes), and +4 (4 additional nodes). Our hand-generated evaluation had interpretations with a complexity of up to +7.

Belief distortion measures how far the beliefs in an argument are from those inferred by BIAS. For instance, if the argument states that a node is Likely while BIAS believes it is VeryLikely, there is a distortion of 1. Belief distortions are related to assumptions, since BIAS may be forced to make assumptions in order to reconcile the beliefs in an argument with BIAS' propagated beliefs. We considered 5 levels of *total distortion* between an argument and its interpretation (from 0 to 4) for both experiments. 0 distortion means that the beliefs in the argument match those in the interpretation, and a distortion of 4 means that the total discrepancy between the beliefs in the nodes in the argument and BIAS' beliefs is 4 – this may be due to a large discrepancy in one node, or small discrepancies in several nodes.

Our automatically generated arguments were produced by randomly selecting a certain number of nodes from our domain BN, and positioning them in the BN so that there are enough intervening nodes – this achieves a desired level of complexity. The propagated beliefs in the nodes in these arguments were then altered to achieve different levels of belief distortion. Our hand-generated arguments were based on arguments entered by people (obtained in previous system trials [12]). These arguments were manually modified in order to obtain enough sample arguments to test the above three factors.

Our experiments focused on two aspects of an interpretation: assumptions and interpretation graphs. Hence, our test arguments consist of implications that connect between nodes in the BN (rather than between NL sentences). Our interpretation algorithm was run for 5 minutes to obtain a performance profile over time. Its performance for our experiments is described below.

Automatically generated arguments. We ran our interpretation algorithm on a total of 228 automatically generated arguments, which varied in argument size, interpretation complexity and belief distortion. Fig. 5 depicts different aspects of the performance of our anytime algorithm for these experiments.



(a) Time-to-best versus arg. size (90 arguments)

(b) Time-to-best versus bel. distortion (138 Large arguments)

(c) Solution quality versus time (30 Large arguments)

Fig. 5. Performance of our anytime algorithm for automatically generated arguments

Fig. 5(a) shows the effect of argument size and interpretation complexity on the average time taken to find the best interpretation for 90 arguments, while keeping the belief distortion to 0. This chart indicates that argument size has a small influence on system performance (the best interpretation was found in less than 20 seconds), except when the complexity of the interpretation is +4 (4 nodes in addition to those in the argument) and the argument is Medium or Large. In these cases, the average time required to find the best interpretation exceeds 50 seconds. Although the performance for Medium arguments appears worse than that for Large arguments, this difference is not statistically significant, as the standard deviation is quite large.

Fig. 5(b) shows the effect of belief distortion and interpretation complexity on the average solution time for 138 Large arguments (the performance for Medium arguments is consistent with that for Large arguments). Belief distortion seems to have little effect on algorithm performance (with the best interpretation found in around 20 seconds), while once more, the main influencing factor is interpretation complexity.

These results prompted us to examine the CPP (conditional performance profile) of our algorithm in relation to interpretation complexity. Fig. 5(c) plots average *interpretation quality* against time, where interpretation quality is defined below. This information was plotted for the 30 Large arguments from Fig. 5(a) for the three levels of interpretation complexity.

$$\textit{interpretation quality} = \frac{-\log_2 \Pr(\textit{best interpretation found so far})}{-\log_2 \Pr(\textit{best interpretation obtained in 5 minutes})}$$

As can be seen from this plot, the worst performance is obtained for interpretations of complexity +4, but even this performance improves significantly early in the process, with solutions reaching near-optimum obtained at around 25 seconds. The plot for complexity +2 reaches the best interpretation at about 16 seconds, and the plot for complexity 0 is nearly invisible, as it reaches the best interpretation quality immediately.

User-based arguments. Our results for user-based arguments are significantly better than those obtained for the automatically generated arguments. In fact, the best solution was obtained in under 3 seconds for all our hand-generated arguments, which had levels of complexity of up to +7. This indicates that the automatically generated arguments produce very harsh evaluation conditions, which constitute an upper bound for the performance expected under actual working conditions.

6 Conclusion

We have offered an anytime algorithm that generates "good enough" interpretations of probabilistic arguments in the context of a BN. These interpretations consist of BN nodes that match the sentences in an argument, assumptions that justify the beliefs in the argument, and a reasoning structure that adds detail to the argument. Our evaluation focused on interpreting arguments that required making additional assumptions and postulating interpretation graphs (rather than proposing different node configurations). Our anytime algorithm generated a good interpretation for automatically generated arguments in under 25 seconds, and the best interpretation for hand-generated arguments in under 3 seconds. In both cases creditable performance was achieved in real time. However, this discrepancy in run times suggests that our algorithm's performance for automatically generated arguments constitutes an upper bound for its performance under actual conditions, which is very encouraging.

7 Acknowledgments

This research is supported in part by the ARC Centre for Perceptive and Intelligent Machines in Complex Environments. The NL parser was implemented by Tony Ng.

References

1. Dean, T., Boddy, M.S.: An analysis of time-dependent planning. In: AAAI-88 – Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, Minnesota (1988) 49–54
2. Horvitz, E., Suermondt, H., Cooper, G.: Bounded conditioning: flexible inference for decision under scarce resources. In: UAI89 – Proceedings of the 1989 Workshop on Uncertainty in Artificial Intelligence, Windsor, Canada (1989) 182–193
3. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann Pub., San Mateo, California (1988)
4. Raskutti, B., Zukerman, I.: Generation and selection of likely interpretations during plan recognition. User Modeling and User Adapted Interaction **1** (1991) 323–353
5. Carberry, S., Lambert, L.: A process model for recognizing communicative acts and modeling negotiation subdialogues. Computational Linguistics **25** (1999) 1–53
6. Zilberstein, S., Russell, S.: Approximate reasoning using anytime algorithms. In Natarajan, S., ed.: Imprecise and Approximate Computation. Kluwer Academic Pub. (1995) 43–62
7. Haenni, R.: Anytime argumentative and abductive reasoning. Soft Computing Journal **8** (2003)
8. Jokinen, K., Wilcock, G.: Confidence-based adaptivity in response generation for a spoken dialogue system. In: Proceedings of the Second SIGdial Workshop on Discourse and Dialogue, Aalborg, Denmark (2001)
9. Fischer, J., Haas, J., Nöth, E., Niemann, H., Deinzer, F.: Empowering knowledge based speech understanding through statistics. In: ICSLP'98 – Proceedings of International Conference on Spoken Language Processing. Volume 5., Sydney, Australia (1998) 2231–2235
10. Salton, G., McGill, M.: An Introduction to Modern Information Retrieval. McGraw Hill (1983)
11. Zukerman, I., George, S., Wen, Y.: Lexical paraphrasing for document retrieval and node identification. In: IWP2003 – Proceedings of the Second International Workshop on Paraphrasing: Paraphrase Acquisition and Applications, Sapporo, Japan (2003) 94–101
12. Zukerman, I., George, S.: A probabilistic approach for argument interpretation. To appear in User Modeling and User-Adapted Interaction, Special Issue on Language-Based Interaction: User Modeling and Adaptation (2004)