Bi-Directional Inference

by

Stuart Shapiro, Joao Martins and Donald McKay*

Department of Computer Science
State University of New York at Buffalo
4226 Ridge Lea Road, Amherst, NY 14226

*(current address: Research & Development Activity, Special Systems Division, Federal and Special Systems Group, PO Box 517, Paoli, PA 19301)

This work was supported in part by the National Science Foundation under Grants MCS878-02274 and MCS80-06314 and by the Instituto Nacional de Investigacao Cientificia (Portugal) under Grant No. 20536.

Abstract

Inference can be viewed as a search through a space of inference rules. Backward and forward inference differ in the direction of the search: backward inference searches from goals to ground assertions; forward inference searches from ground assertions to goals. This paper describes an inference procedure, called bi-directional inference, which limits the number of inference rules searched. Bi-directional inference results from the interaction between forward and backward inference and loosely corresponds to bi-directional search. We show through an example that, when used throughout a session of related tasks, bi-directional inference sets up a conversational context and prunes the search through the space of inference rules by ignoring rules which are not relevant to that context.

1. Introduction

Bi-directional inference (BDI) combines forward inference (FI) and backward inference (BI) to limit the search through a space of inference rules by establishing a context on the basis of an ongoing session. We use the term "bi-directional inference" because the resulting search loosely corresponds to bi-directional search (Kowalski 72, Pohl, 71).

The benefits of BDI become clear during an extended session in which the user asks questions and adds assertions all of which are related. BDI sets up a conversational context and prunes the space of inference rules searched (either during BI or FI) by ignoring rules which are not relevant to the context.

In BDI there are two sets of inference frontiers, one growing from the assertions added in FI and the other growing during BI from the questions asked. Whenever two frontiers meet some answers are produced.

BDI has been implemented in SNIP, the SNePS Inference Package. We present examples of BDI and compare the results obtained using BDI with the results obtained using BI or FI only. Although SNIP has a much richer rule syntax than used in these examples (Shapiro, 79a, 79b) they suffice to illustrate BDI.

2. Basic notions of SNIP

SNIP relies on a declarative representation of inference rules (SNePS semantic network (Shapiro, 79a). Every rule may be used both in FI and BI. When a rule is used, it is activated, remaining that way until explicitly de-activated by the user. The activated rules are assembled into an active connection graph (acg) (McKay and Shapiro, 81), a collection of MULTI processes (McKay and Shapiro, 80) which carry out the inference. The acg also stores all the results generated by the activated rules. If during some deduction SNIP needs some of the rules activated during a previous deduction, it uses their results directly instead of rederiving them. The acg that is built for one query or assertion is not discarded after the query has been answered or the assertion "fully" understood by making all possible inferences from it. Rules of the network remain active, allowing a dynamic context to be constructed. The dynamic context is the collection of rules which have been activated. In addition, the active rules are more prominent: when searching for inference rules to be used, if any previously activated rules are appropriate then only those rules will be considered and no other rules will be activated. Hence rules apparently irrelevant to the current dynamic context are ignored.

3. Backward Inference

We present an example of BI, explaining very briefly how acg's work. A complete explanation can be found in (McKay and Shapiro, 81).

Suppose that SNIP is being used as a database retrieval system for some company interested in recruiting computer science (CS) majors. The recruiting policies of the company are stored as rules in the database (Lines 1-4, Fig. 1). The

```
V(x,y)[ Planning-to-visit(x) & CS-major-at(y,x) -> Good-prospect(y)]
V(x,y)[ Top-school(x) & CS-major-at(y,x) -> Good-prospect(y)]
V(x)[ Good-prospect(x) -> Send-literature-to(x)]
V(x)[ Good-prospect(x) & Graduating(x) -> Invite-for-interview(x)]
Top-school(MIT)
Top-school(CMU)
CS-major-at(Don,SUNY)
CS-major-at(Ted,CMU)
CS-major-at(Anna,MIT)
CS-major-at(John,UCLA)
```

Figure 1
Initial database

company's database also contains a list of top schools and a list of the CS majors at different schools (Lines 5-10, Fig. 1).

Every year the company updates its database with the names of all students graduating in CS and all the schools that the company will visit during that year (Fig. 2). The company then uses SNIP to find out

```
Planning-to-visit(SUNY)
Planning-to-visit(CMU)
Graduating(Don)
Graduating(Ted)
Graduating(John)
```

Figure 2
Information updating the database

which CS majors should be invited for interviews, which ones should be sent the company's literature, etc.

We now consider the acg describing the reasoning of SNIP when it is asked who should be invited for an interview.

An acg is represented as rectangles and circles. Each rectangle represents a rule instance (a deduction rule together with a substitution for the variables in the rule); the antecedents appear to the left of the double line and the consequents to the right. Circles (called goal nodes) represent goals to be proved. Rule instances and goal nodes are connected by directed edges. Substitutions flow through the edges. Rule instances and goal nodes can be viewed as producers of formulas sent out on the edges leaving them and as consumers of formulas coming in on the edges pointing to them. Some edges have switches (represented by square brackets) which have the effect of renaming the variables in the substitutions flowing through them. For ease of reference, rule instances have labels of the form An (where n is an integer). Those labels are used for notational convenience only and have no relation with the way acg's work.

Initially, a request is created which contains the atomic formula being sought. The rule instance labeled A1 in Figure 3 represents the request to

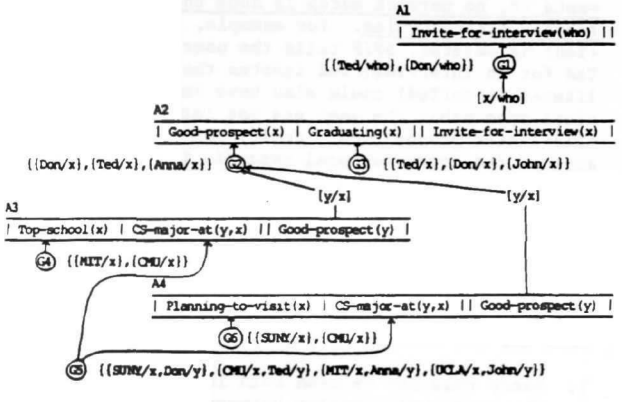

Figure 3
acg for backward inference

deduce all instances of the atomic formula Invite-for-interview(who). The next step is to create a goal node for the atomic formula. A goal node (G1) is added below the instance being sought. One of the jobs of the goal node is to match its atomic formula against the network to find all formulas which unify with it. If there are ground instances the goal node produces them immediately. For every matching formula in consequent position of some rule, a new rule instance is added to the acg. The other job of the goal node is to remember all substitutions it receives (these substitutions are represented enclosed in curly brackets next to the goal node). When a goal node receives a new substitution, it sends it to all rule instances to which it points. In this case, G1 can't find any ground instances of Invite-for-interview(who) but the rule $\forall(x)$[Good-prospect(x) & Graduating(x) → Invite-for-interview(x)] may be used to derive such instances. Rule instance A2 is thus created by G1 (Fig. 3). Notice that the variable 'x' in A2 should be bound to 'who' in A1 when an answer is produced by A2. For this reason a switch ([x/who]) is inserted in the link between A2 and G1 and has the effect of translating between variable contexts. Switches are computed by the network matching function (Shapiro, 77) which was used by G1. For details of how this is done see (McKay and Shapiro, 81).

Goal nodes G2 and G3 are created for the antecedents of A2. G2 finds two rules which can produce instances of Good-prospect(x) and creates the corresponding rule instances (A3 and A4, Fig. 3). G3 finds three ground instances of Graduating(x), namely Graduating(Ted), Graduating(Don) and Graduating(John). The substitutions {Ted/x}, {Don/x} and {John/x} are stored by G3 and sent to its consumer (A2).

Goal nodes are created for the antecedents of A3 and A4. G4 finds two top schools (MIT and CMU), and sends the substitutions to A3. G5 finds the CS majors at different schools, informing both A3 and A4. A3 deduces that both Ted and Anna are good prospects. A4 deduces that both Don and Ted are good prospects after receiving from G6 the information that SUNY and CMU will be visited.

The information about good prospects flows through the acg reaching A2 which deduces that both Ted and Don (good prospects who are graduating) should be invited for interviews and the answer is finally produced by A1.

Notice that BI tries to get each answer in all possible ways, and so the same answer can be produced several times. In this particular case the answer Good-prospect(Ted) was produced twice, by rule instances A3 and A4.

4. Forward Inference

In this section we discuss the results obtained if the company chooses to use FI. We will assume that the information represented in Figure 1 is stored in the database and that FI is done with the information represented in Figure 2.

Doing FI with Planning-to-visit(SUNY) generates the acg of Fig. 4: rule instance A1 is created along with goal nodes for its antecedents (G1 and G2). G1 is immediately satisfied, and G2 finds CS-major-at(Don, SUNY), sending to A1 the substitution {Don/y}. Notice that G2 is performing some amount of BI, reflecting a characteristic of SNIP in which BI and FI are closely interconnected. A1 deduces Good-prospect(Don), creating rule instances A2 and A3 to do further FI. A2 deduces its consequent but A3 doesn't since Graduating(Don) is not in the database yet.

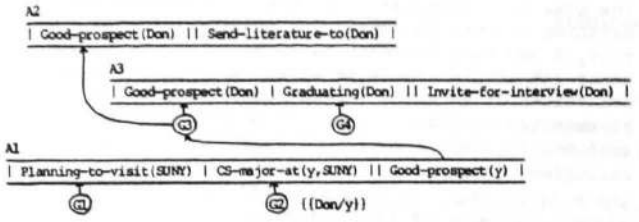

Figure 4
acg for forward inference

After entering all the information of Figure 2, SNIP has deduced (acg not shown) that Don, Ted and Anna should be sent literature and that Don and Ted should be invited for interviews. In other words, all possible inferences were made, even if the user was only interested in some of them. FI does not take the user's interests into account filling the database with assertions which may never be used.

5. Bi-directional Inference

In this section, we introduce BDI and show that it establishes conversational contexts, focusing SNIP's inferences within those contexts and thereby limiting the space of rules searched. BDI results from the interaction of FI and BI and can be obtained either by doing BI following FI or by doing FI following BI. We consider each of these cases in turn.

5.1. Backward Inference Following Forward Inference

Suppose that the user says "I am planning to visit SUNY, who shall I invite for an interview?". In this context, by asking 'Invite-for-interview (who)?' the user wants to consider only the CS majors from SUNY. We show how FI can be used to set up the 'SUNY context' which is then used to answer the user's query. In a pure BI system, finding the CS majors from SUNY who should be invited for an interview requires finding the intersection between all CS majors from SUNY and all persons who should be invited for an interview (or, in some systems, generating all of one and testing each to see if it satisfies the other).

The user begins by doing a small amount of FI with Planning-to-visit(SUNY). The amount of inference can be defined by the number of network pattern matches performed. Let us assume, for the sake of argument, that by "small amount of FI" we mean that FI is only allowed two network matches. The first match finds the rule $\forall(x,y)$[Planning-to-visit(x) & CS-major-at(y,x) → Good-prospect(x)], setting up the rule instance A1 (Fig. 5) and the second match is used by G2 to look for instances of CS-major-at(y,SUNY), finding CS-major-at(Don,SUNY). This is enough to deduce Good-prospect(Don) but nothing can be done with this because finding unactivated rules requires a match. Therefore, the inference stops, leaving behind the active rule instance A1 (Fig. 5).

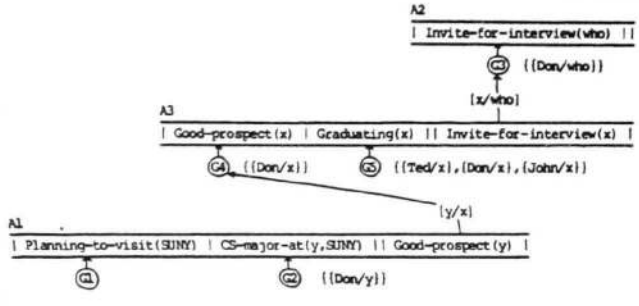

Figure 5
acg for bi-directional inference

If the user now asks the question 'Invite-for-interview(who)?' rule instances (A2 and A3) and goal nodes (G3, G4 and G5) are created (Fig. 5) as discussed in section 3. Here, however, goal node G4 finds that there is an active rule that can produce instances of Good-prospect(x), namely A1. Instead of doing a network pattern match to find additional rules, it uses rule instance A1 immediately. The substitution {Don/y} flows through the acg producing the answer Invite-for-interview(Don). In this case the CS majors from other schools were not even considered since SNIP had set up the "SUNY context".

Suppose that CS-major-at(Don,SUNY) were not in the network and thus rule instance A1 could not produce any answer even though instances of Invite-for-interview(who) could have been derived for CS majors of other schools. Following the query 'Invite-for-interview(who)?', SNIP would return an "I don't know" answer. This, at first glance, seems to be wrong. However, taking into account that the user only wants to consider the CS majors from SUNY this makes perfect sense, showing a feature of BDI in which derivable instances which are irrelevant to the context are effectively ignored by SNIP.

5.2. Forward Inference Following Backward Inference

Suppose that the database contained the information of Figure 1 and the user asked who should be invited for an interview. SNIP builds an acg as shown in Figure 3, except that goal nodes G3 and G6 have no stored data. The acg produces no answers since the information in the database is insufficient. If the user now does FI with any of the propositions of Figure 2, the waiting goal nodes are found. Whenever a new assertion is produced for FI, and a goal node already exists that wants it, no network match is done to find additional relevant rules. For example, if Graduating (Ted) is entered, SNIP tells the user to invite Ted for an interview, and ignores that Send-literature-to(Ted) could also have been derived, since presumably the user was not interested in this latter proposition. Again, BDI takes into account the conversational context, ignoring the rules irrelevant to the active context.

6. Conclusions

We presented an overview of BDI, pointing out the two characteristics required by a system to make the BDI behavior possible:

1. Every rule may be used both in FI and BI.
2. There is a distinction between rules which have been activated and rules which haven't.

Relying on these two characteristics, when SNIP (a system which uses BDI) searches for rules to be used, it looks for activated rules first and just in case of failing to find any activated rule, non-activated rules are considered. In addition, as a matter of efficiency, activated rules remember all the results produced, not solving the same problem twice. The resulting inference loosely

corresponds to a bi-directional search. We say 'loosely corresponds' because not only may there be several bi-directional searches going on in parallel (one for each question asked) which can intersect each other, but also there are two levels of search, the first through the activated rules, and the second, which is tried only after failure of the first, through the non-activated rules.

The example presented, although very small and simplistic, shows that BDI effectively prunes the search through the space of inference rules by focusing the system's attention towards the interests of the user.

In BDI, some of the disadvantages of pure FI and pure BI do not exist. One of the disadvantages of pure FI is that it may fill the database with derived propositions which may never be used. We showed that BDI ignores some derivations which do not interest the user. One of the disadvantages of BI is that all apparently relevant rules are tried, regardless of the actual data. We showed that BDI ignores inactive rules in favor of rules activated by previous (forward or backward) deduction.

7. Acknowledgements

Many thanks to Terry Nutter, Ernesto Morgado, Jeannette Neal and the other members of SNeRG (the SNePS Research Group) for their comments on earlier versions of this paper and for their general discussions while the research was in progress.

8. References

1. Kowalski, R., And-or Graphs, Theorem-proving Graphs and Bi-directional Search, in Machine Intelligence 7, Meltzer and Michie (eds.), Halsted Press, 1972.
2. McKay D. and Shapiro S., "MULTI - a LISP Based Multiprocessing System", Proc. 1980 LISP Conference, pp. 29-37.
3. McKay D. and Shapiro S., "Using Active Connection Graphs for Reasoning with Recursive Rules", Proc. IJCAI-81, pp. 368-374.
4. Pohl I., Bi-directional Search, in Machine Intelligence 6, Meltzer and Michie (eds.), American Elsevier, 1971, pp. 127-140.
5. Shapiro S., "Representing and Locating Deduction Rules in a Semantic Network", in Proc. Workshop on Pattern-Directed Inference System, SIGART Newsletter 63, 1977, pp. 14-18.
6. Shapiro S., "The SNePS Semantic Network Processing System", in Associative Networks, N.V. Findler (ed.), Academic Press, 1979a, pp. 179-203.
7. Shapiro S., "Numerical Quantifiers and their use in Reasoning with Negative Information", Proc. IJCAI-79, 1979b, pp. 791-796.
8. Shapiro S. and McKay D., "Inference with Recursive Rules", Proc. First AAAI Conference, 1980, pp. 151-153.