ELSEVIER

Decision Support Systems 21 (1997) 263–269

Decision Support Systems

Inference engine greediness: subsumption and suboptimality

Daniel E. O'Leary *

3660 Trousdale Parkway, University of Southern California, Los Angeles, CA 90089-1421, USA

Abstract

Greedy inference engines find solutions without a complete enumeration of all solutions. Instead, greedy algorithms search only a portion of the rule set in order to generate a solution. As a result, using greedy algorithms results in some unique system verification and quality concerns. This paper focuses on mitigating the impact of those concerns. In particular, rule orderings are established so that better solutions can be found first and those rules that would never be examined by greedy inference engines can be identified. The primary results are driven by rule specificity. A rule is more specific than some other rule when the conditions in one rule are a subset of the conditions in another rule. If the least specific rule is ordered first and it is true, then greedy algorithms will never get to the more specific rule, even if they are true. Since the more specific rules generally also have the greatest return this results in the 'wrong' ordering—the rule with the least return will be found. As a result, this paper focuses on generating orderings that will likely lead to higher returns. © 1997 Elsevier Science B.V.

Keywords: Subsumption; Inference engine greediness; Expert system optimality

1. Introduction

Greedy inference engines are frequently used in generating solutions using rule-based knowledge bases. Greedy inference engines do not generate all possible solutions, instead, they typically use only a subset of the rules and stop after a solution has been found. Greedy algorithms trade off speed of generating a solution with completeness of analysis. As a result, greedy algorithms are often used in real time systems or in systems that require substantial user interaction.

Although greedy algorithms rapidly generate solutions, there can be at least two related drawbacks. First, they can generate suboptimal solutions. In the case of medical systems, such suboptimality can result in life and death situations. While in the case of business systems, choice of the wrong solution might result in the death of the firm through litigation. Second, in some cases, some rules can never be investigated. Typically, this problem occurs when less specific rules are ordered before rules that are more specific (the conditions in one rule are a subset of the conditions in the other). This is referred to as the subsumption problem. Accordingly, this paper defines and classifies inference engine greediness, investigates how ordering of rules mitigates the impact of greediness on rule subsumption and studies ways that suboptimality can be mitigated through ordering rules.

* Corresponding author. Tel.: +1-213-740-4856; fax: +1-213-747-2815; e-mail: oleary@rcf.usc.edu

0167-9236/97/$17.00 © 1997 Elsevier Science B.V. All rights reserved.
PII S0167-9236(97)00042-0

1.1. Greediness inference engines

Suppose that the inference engine starts with the first rule, and sequentially goes through each condition to determine if the rule is true. If it is true, it stops, if it is not true, it goes to the next rule, where the process starts over again. Consider rule bases (or subsets of rule bases) that have two or more rules that each use a subset of the same conditions and each result in different conclusions. For example, in the 'bartender problem,' Winston [1] discussed a classic system with a goal of recommending drinks. That system included the following two rules:

(r-1) If guest is a health nut then drink = glop.

(r-2) If guest is a health nut and carrots are not served with the meal then drink = carrot juice.

Unfortunately, even if both are true, but r-1 is ordered first, r-2 will never be found true using a greedy algorithm. A greedy algorithm will only recommend 'Glop,' never 'Carrot Juice.' There are at least two problems with this occurrence. First, a developer or a user may examine the rule base, find that r-2 is in the knowledge base, and thus be satisfied that the knowledge is complete and correct. Unfortunately, they might not know that rule could never be found true. As a result, an apparently verified system will not function correctly or at least as anticipated.

Second, often the greater the specificity in the conditions, the greater the payoff associated with the conclusion. For example, with the above two rules if the system suggests carrot juice, when it is appropriate, as opposed to glop, the system will make more money for the bartender (or make the guest happier). As a result, given the above two rules, with a greedy inference engine, in a world where both are true, we generally prefer r-2 to succeed before r-1. However, with the current order only r-1 can be found true, thus, a greedy algorithm with the above ordering will guide the user to a 'less preferred result.'

This existence of a preference is critical since many inference engines (e.g., M.4, Cimflex Teknowledge, [2]) depend on the order of the rules.

1.2. The remainder of this paper

The remainder of this paper proceeds in the following manner. Section 2 investigates greedy inference engines. Section 3 analyzes the verification required for greedy algorithms. Section 4 investigates the use of greedy algorithms on rules with common conditions. Section 5 analyzes the use of greedy algorithms on rules with unrelated sets of conditions. Section 6 briefly reviews some extensions.

2. Greedy inference engines

Greedy inference engines are those inference engines that do not generate all possible solutions (to the rule-base system). Instead, they stop short of complete enumeration, stopping when, e.g., they find a feasible solution to the goal. For example, both M.1 (Teknowledge, [3], p. 4–10) and M.4 (Cimflex Teknowledge, [2], p. 4–8) start with the first rule and stop when they find the first rule that is true, that also solves the goal. Other inference engines, such as Prolog knowledge-based systems and Kappa PC, also employ greedy algorithms. Further, virtually all expert system 'shells' provide greedy capabilities that allow the user the ability to stop when the first solution is found.

2.1. Why are greedy inference engines used?

There are a number of reasons why greedy algorithms are developed and used. First, in the case where solutions are required in 'real time,' systems and users are sometimes limited in the amount of time that can be spent waiting for the system to inference through the rule set. As a result, greedy algorithms speed the determination of a solution.

Second, generally as inference engines process rules, they require that users provide input, regarding the conditions. Thus, complete enumeration requires not only computation time, but more importantly, user time. Greedy algorithms have the important advantage of limiting the time with which the user of the system must spend providing the system information, since greedy algorithms find a solution without complete enumeration.

Greedy algorithms can be 'forced' into complete enumeration. However, for the above reasons, there is often a need for a greedy solution.

2.2. Types of greediness

There can be different types of greediness. To facilitate the discussion, it is assumed that rules i are of the type 'If $X_{1,i}$, $X_{2,i}, \ldots, X_{n,i}$ then Y_i', where Y_i is a conclusion and also a goal for which the search is being done. $X_{j,i}$ is the value in the ith rule of the jth condition. $(X_{1,i}, X_{2,i}, \ldots, X_{n,i})$ is called the condition set c-i. Since Y_i is a goal, if the rule is true, then the goal is achieved. The results of the paper can be extended to other forms of rules, but this form of rules facilitates discussion. In addition, oftentimes at least subsets of knowledge bases take this form.

There are at least two ways that a greedy algorithm can stop early: stopping when the 'first rule satisfied' is found; and stopping when all the 'conditions to find first rule satisfied.' In the last case, the algorithm stops after information gathered from the user has been tested in each of the remaining rules in the knowledge base ('with completion'). In addition, greedy algorithms can be based on either 'breadth-first' (one rule at a time) or 'depth-first' (one condition at a time) searches.

'First rule satisfied' (FRSG) 'breadth-first' greediness is used by inference engines that stop after they find the first rule that is satisfied. M.1 (Teknowledge, [3]) and M.4 (Cimflex Teknowledge, [2]) are examples of expert system shells that employ 'first rule satisfied' inference engines. Generally, the inference engine starts with the first rule, determines if it is true or not and if the goal is satisfied. If it is true and the goal is satisfied, it stops, however, if some condition is false, it proceeds to the next rule. Conditions that are true are held in memory and applied sequentially to each new rule that is processed, before eliciting additional information from the user. As a result, solutions generated by FRSG are heavily dependent on the order in which the rules are processed.

FRSG 'depth-first' greediness elicits the conditions one at a time. Then rules are examined one at a time to determine if a rule is true or if more information is necessary, based on the conditions elicited so far. If a rule succeeds then the inferencing stops, otherwise it goes to the next condition in the first rule. This approach starts with r-1, and continues with r-2,...,r-m, that had not yet be found to not succeed. Rules are held in memory until they are determined to succeed or fail. Partially processed rules are eliminated from memory if they are found to have conditions that failed. Rules are added to memory if some condition from the rule was found to succeed.

'Conditions to find first rule satisfied' greediness (CFRSG) are used by inference engines that stop after they find the first rule satisfied. They complete the search through the remaining rules, with the gathered information. All other rules are investigated that employ conditions that were used in the process of finding the first satisfied rule. This approach requires no additional information gathering from users. It operates similar to FRSG, but after a rule is found true, then information gathering stops and determination is made as to the truth of any other rules are true, given the available information.

There is no greediness in complete enumeration. Instead, complete enumeration requires that a complete analysis of all conditions from all rules be investigated to determine the entire set of rules that are satisfied. Such an approach can be quite time consuming and require substantial user participation.

2.3. Limitations of greedy inference engines

However, there are at least two limitations with greedy inference engines. First, the solutions that they find are not necessarily optimal. If the user or the developer does not know that the inference engine is greedy or is unaware of the impact of greedy inference engines on solutions generated, then the user will not be aware that solutions are not optimal. If the user is aware that greedy algorithms result in suboptimality, then it might be assumed that they can trade-it off a lack of optimality against time spent supplying a system with information or the costs of waiting for a solution, in the case of real time expert systems.

Second, using greedy inference engines, some rules would never be executed. Users could see the knowledge base, see rules in there, think that the knowledge base was representative of the underlying knowledge to solve the problem, but not know that some rules could never be executed. For example, as noted in the example in the introduction, r-2 would never be found true if r-1 was ordered first and the inference engine was greedy. Considering only knowledge bases (an approach consistent with 'classic' verification processes), is deceptive since problem solution is a function of knowledge base and inference engine. As a result, as noted in O'Leary [4] verification must includes the knowledge base, inference engine and the interaction between the two.

3. Verification in greedy algorithm knowledge bases

Greedy algorithms may use only a portion of the knowledge base, while a complete enumeration algorithm may use every rule. As a result, an important question is how does verification differ when there is a greedy algorithm, as opposed to a complete enumeration algorithm? In most 'classic' verification tests (e.g., Nguyen et al. [5]), if the test holds for the entire knowledge base, then it also holds for any subset of that knowledge base. For example: (a) If there are no cycles in the entire knowledge base, then there are no cycles in any subset of the knowledge base. (b) If there are no redundant rules in the entire knowledge base, then there are no redundant rules in any subset, etc.

As a result, in most cases, verification of a knowledge base for complete enumeration will provide the necessary verification for greedy use of the knowledge base. However, as noted in the introduction there are problems in knowledge bases subject to greedy inference, when compared to analysis of the same knowledge using complete enumeration.

Accordingly, the focus of this paper is on those concerns that occur by reducing consideration from the entire knowledge base to some subset through the use of greedy techniques. One area where greedy and complete enumeration verification issues differ is that of 'subsumption.'

The primary concern of previous verification research (e.g., Nguyen et al. [5]) of knowledge-based systems has been on subsumed rules where 'One rule is subsumed by another if the two rules have the same conclusions, but one contains additional constraints on the situations in which it will succeed.' In such situations, if both rules are correct, one of the two rules will be true before the other, resulting in the same consequence. As a result, it is critical to find such rules and include whichever one is correct, while removing the other. Subsumption of this type is important whether or not the inference engine is greedy. However, the current research considers subsumption where the situations in which the rules will succeed are subsets of each other (one rule subsumes another) yet the conclusions differ. The results in a greedy world are similar: only one rule will succeed.

Ordering rules depends on the extent to which the conditions on the rules are related. As a result, the Section 4 examines rules where the conditions on one rule are subsets of conditions on another rule. Section 5 then examines rules that have no overlap in condition sets.

4. Ordering related sets of rules

An important type of knowledge representation occurs with a set of rules with differing specificity across the same conditions, i.e., one rule's conditions are a subset of another rule's conditions.

4.1. Impact on subsumption and suboptimality: informal analysis

If there is complete enumeration and both rules are satisfied, then the conclusions from both rules r-1 and r-2 would be listed as solutions. However, if the inference engine employs a greedy approach, then that will not be the case.

If there is a breadth-first FRSG or CFRSG approach being used and r-1 is ordered before r-2, then only r-1 will be found. For FRSG and CFRSG, r-2 would never be completely elicited. This is the case even if r-2 was also true. The order of the rules drives the process.

In a depth first FRSG or CFRSG, where the condition 'guest is a health nut' is ordered first, and r-1 is ordered before r-2, then only r-1 will be found. This is the case even if r-2 was also true. Even if r-2 is ordered first, r-2 will never be found true. In this situation, the order of the rules and conditions in the rules drive the results.

Second, since there are differential returns to rule conclusions and since in general, only one rule will succeed, it is assumed that rule bases should be structured so that the most desirable conclusion is found, using a greedy approach. If there is a breadth-first FRSG approach being used, then by ordering r-2 before r-1, if one of r-1 or r-2 is found true, it will be r-2. Since only one conclusion could be found, and since the conclusion to r-2 is preferred then this is the generally preferred solution. Similarly, if there is a CFRSG approach being used and r-2 is ordered before r-1, then both conclusions will be found.

In a depth-first FRSG approach, by ordering r-2 first and the condition set 'guest is a health nut' second, r-2 could be found to be satisfied before r-1. Since the conclusion to r-2 is preferred, that order of rules and conditions is generally preferred.

4.2. Formal analysis: breadth-first search

In this section, I first provide a more formal analysis of finding an ordering that provides better solutions and second, provide an ordering that mitigates the subsumption problem.

Assume that a breadth-first greedy search is executed. Suppose that there is a set of rules (r-1,...,r-*m*), where each set of conditions for rule r-*i* (c-*i*) is a subset of the conditions c-($i-1$). Assume that the conclusion for rule r-*i* ($k-i$) has a greater return than all $k-j$, for $j > i$. (The approach used to prove these theorems is analogous to the approach used in the scheduling literature to show that one schedule is as good or better than another, e.g., Conway et al. [6])

Theorem 1: Under those conditions, if the inference engine is a FRSG then the ordering of rules (r-1,...,r-*m*) will provide a solution at least as good as any other ordering.

Proof

Proof by contradiction. Assume that instead of r-1, we put r-*i* first. There are two different orderings to compare and there are three cases.

1. If r-*i* is false, then r-1,...,r-($i-1$) would be false so there would be no difference in solution, with either ordering of the rules.
2. If r-*i* is true and r-1,...r-($i-1$) are false, then either ordering would give the same solution.
3. If r-*i* is true and some r-1,...,r-($i-1$) is true, then ordering r-*i* first can yield a solution that is not as good, and the ordering with r-*i* as the *i*-th rule can yield a better solution.

Theorem 2: Suppose that c-1 is a subset of all c-*j*, for $j = 2,\dots,m$. If rule r-1 is ordered first, then rules r-$j = 2,\dots,m$ will never be found true, under FRSG.

Proof

If c-1 is true, then the greedy algorithm will stop with r-1. If it is not true, then none of c-*j*, $j = 2,\dots,m$, will be true either.

As an extension, under CFRSG, the additional rules (r-1,...,r-($m-1$)) also will not be executed unless, the conditions are in some previously examined rule. As a result, if there exists a set of rules like (r-1,...,r-*m*) in the context of a larger knowledge base, and the developer is using a heuristic approach like CFRSG, then for any rules ordered before (r-1,...,r-*m*), the conditions in those rules should be ordered first in the other rules. Otherwise the information will not be available for analysis.

Another view of theorem 2 is provided in theorem 3. In particular, if there is a set of rules where one rule contains a 'super set of the conditions', then ordering that rule first will ensure that it is examined before other rules in the set with common conditions, otherwise it will not be examined since one of the other rules will be found true first.

Theorem 3: Suppose that c-2,...,c-*m* are a subset of c-1. If rule r-1 is true, then under FRSG it must be ordered first or else it will never be found true.

Proof: Similar to Theorem 1

4.3. Formal analysis—depth-first search

Similar results can be developed for depth-first greedy algorithms. The primary difference in the depth first approach is that not only do the rules need to be 'properly' ordered, but so do the conditions. Accordingly, we make the same set of assumptions as in theorem 1, and make an assumption on the order of the conditions.

Theorem 4: Assume the conditions of theorem 1. If the conditions are ordered so that those conditions used only in r-1 are ordered first in rule 1, those used only in r-1 and r-2 are ordered first in rule 2, etc., then that ordering of rules and conditions will provide at least as good a solution as any other ordering.

Proof

Assume an alternative ordering of the conditions, e.g., where the conditions that are common to rules *i* are ordered first in rule r-1.

1. If those conditions are true, then rule *i* (or some rule *j*, for $j = i, \ldots, m$) will be executed before any other rule. In this case, if any of rules r-1 through r-*j* were true then they still would not be found true and the resulting solution would be suboptimal.
2. If the conditions are false, then all rules r-1, . . . ,r-*j*, are false so either the solution with the initial ordering is at least as good.

4.4. Implications

These theorems can provide a basis for ordering knowledge bases for using greedy algorithms. Although the conditions in those theorems are unlikely to apply to entire knowledge bases, they can be used for ordering subsets of rules. In addition, if the greedy algorithms are done 'with completion' (CFRSG), then rules with subsets of conditions will be enumerated by the system, making those solutions available to the user.

What these theorems do not address is how to order those rules where the conditions are not related. That issue is discussed in Section 5.

5. Ordering non-related rules

The above results provide partial orders of rules meeting a 'relatedness' of the condition sets. However, it says nothing about rules whose conditions are not related. One approach is to order the consequence with the largest payoff first. This leads us to the following theorem.

Theorem 5: Assume that the conditions in the *i*-th rule and the *j*-th rule are not related. Assume that the consequence of the *i*-th rule is worth more than that of the *j*-th rule. Assume breadth-first greediness. The ordering of the *i*-th rule before the *j*-th rule will result in a solution that is at least as good as any other ordering.

Proof

Assume instead that r-*j* is ordered before r-*i*. There are two cases.

1. If r-*j* is false, then ordering r-*j* first makes no difference in the solution found.
2. Assume r-*i* and r-*j* are both true. If r-*j* is ordered before r-*i*, then r-*j* will be executed and r-*i* will never be found true or executed. Accordingly, the solution under this ordering is not as good as the one with r-*i* ordered first.

Accordingly, we generate better solutions with r-*i* ordered first.

The results of this section and Section 4 provide a basis on which to order the rules in a knowledge base. Together they provide the ability to generate partial orderings of rules in preparation of those rules being subjected to solution analysis by greedy algorithms.

6. Extensions

The research discussed in this paper has a number of extensions. For example, theorems 2 and 3 could be extended to depth first reasoning. In addition, the paper could be extended to include alternative forms of greediness, the impact of greediness on quality of solution, and the extension of the results to other types of rules.

6.1. Alternative types of greediness

There are a number of different types of greediness that can be generated for inference engines. For example, one extension to FRSG is to allow the system to find, say *k* satisfied rules, generating '*k*-optimal' solutions, before the inference engine stops. Another approach is to continue to gather information from the user to try to satisfy all rules that had satisfied conditions and unelicited conditions at the time the first rule was satisfied.

6.2. Greediness and quality of solution

Another set of relatively unexplored issues is the quality and cost of using different greedy algorithms. An empirical analysis of different greedy algorithms could be done, possibly examining the quality of solution and the amount of effort that the user had to provide for different levels of greediness in some sample rule-based systems. Unfortunately, few knowledge-based systems account for value (e.g., dollar value of goals) in the knowledge.

6.3. Probabilities and weights on rules and subsumption and other rule types

General subsumption is not an issue when there are probabilities or weights on rules. However, under some forms of greedy inference engines, it can still be a critical issue. For example, if the inference engine stops when it finds a solution of a certain quality, such as a probability greater than 0.8, then the same issues would need to be addressed.

This paper has focused on rules of the type r-1 and r-2. However, the results are not limited to these kinds of rules. Other rule structures could also be explored.

Acknowledgements

An earlier version of this paper was presented at the Workshop on Verification and Validation of Knowledge-based Systems, at IJCAI, Montreal, August 1995. The author acknowledges the comments of the workshop referees and the referees on the revised version.

References

[1] P. Winston, Artificial Intelligence, Addison-Wesley, Reading, MA, 1984.
[2] Cimflex Teknowledge, M.4 Users Guide, Cimflex Teknowledge, Palo Alto, 1992.
[3] Teknowledge, M.1 Reference Manual, Teknowledge, Palo Alto, December 1986.
[4] D. O'Leary, Methods of validating expert systems, Interfaces 18 (6) (1988) 72–79.
[5] T. Nguyen, W. Perkins, T. Laffey, D. Pecora, Knowledge base verification, AI Mag. 8 (2) (1987) 69–75.
[6] R. Conway, W. Maxwell, L. Miller, Theory of Scheduling, Addison-Wesley, Reading, Ma, 1967.

Professor Daniel E. O'Leary has been on the faculty at the University of Southern California since getting his Ph.D. from Case Western Reserve in 1985. Dan is the editor of IEEE Expert and on the editorial board of a number of other journals. His research has been published in a wide range of journals including Management Science, Communications of the ACM and Computer.