

search will be saved; the components are dynamically reordered (7) (see also Dependency-directed backtracking).

All of the above-mentioned methods cannot prevent that the solution of some problems still requires an extremely large computation time.

Application in Programming Languages

Floyd (8) was the first to propose a language construct facilitating the writing of backtrack programs and to point out the mechanical translation in constructs of conventional languages. A survey of useful primitive language constructs can be found in Ref. 2. Backtracking plays a fundamental role in a number of AI programming languages, most notably PLANNER (9) and PROLOG (10). In these languages the basic operation consists of nondeterministically applying an operator on a state to derive a new state. Backtracking is used to exhaustively explore all possibilities. Programming consists of formulating the problem in such a way that the backtracking search turns out to perform an efficient computation (see also Problem solving).

BIBLIOGRAPHY

1. R. L. Walker, "An Enumerative Technique for a Class of Combinatorial Problems," *Combinatorial Analysis (Proceedings of Symposium in Applied Mathematics, Vol. X)* Amer. Math. Soc., Providence, R.I., pp. 91-94, 1960.
2. J. Cohen, "Non-deterministic algorithms," *Comput. Surv.*, 11(2), 79-94 (1979).
3. J. R. Bitner and E. M. Reingold, "Backtrack program techniques," *CACM*, 18(11), 651-656 (1975).
4. P. W. Purdom, C. A. Brown, and E. L. Robertson, "Backtracking with multilevel dynamic search rearrangement," *Acta Inf.*, 15(2), 99-113 (1981).
5. E. C. Freuder, "A sufficient condition for backtrackfree search," *JACM*, 29(1), 24-32 (1982).
6. E. C. Freuder, "Synthesizing constraint expressions," *CACM*, 21(11), 958-966 (1978).
7. M. Bruynooghe, "Solving combinatorial problems by intelligent backtracking," *Inf. Proc. Let.*, 12(1), 36-39 (1981).
8. R. W. Floyd, "Nondeterministic algorithms," *JACM*, 14(4), 636-644 (1967).
9. D. G. Bobrow and B. Raphael, "New programming languages for artificial intelligence research," *Comput. Surv.*, 6(3), 155-174 (1974).
10. R. A. Kowalski, *Logic for Problem Solving*, North Holland, Rotterdam, 1979.

M. BRUYNOOGHE
Katholieke Universiteit Leuven, Belgium

R. VENKEN
BIM, Belgium

BACKTRACKING, DEPENDENCY DIRECTED

Dependency-directed backtracking is a problem-solving (qv) technique for efficiently evading contradictions. It is invoked whenever the problem solver discovers that its current state is

inconsistent. The goal is, in a single operation, to change the problem solver's current state to one that contains neither the contradiction just uncovered nor any contradiction encountered previously. This is achieved by consulting records of the inferences (see Inference) the problem solver has performed (called dependencies) and records of previous contradictions (called nogoods), which dependency-directed backtracking has constructed in response to previous contradictions.

Contrast to Chronological Backtracking

Dependency-directed backtracking was developed to avoid two deficiencies of chronological backtracking. Consider the application of chronological backtracking to the following task (see Fig. 1): First do one of **A** or **B**, then one of **C** or **D**, and then one of **E** or **F**. Assume that each step requires significant problem-solving effort and that **A** and **C** together or **B** and **E** together produce a contradiction that is only uncovered after significant effort. Figure 1 illustrates the sequence of problem-solving states that chronological backtracking goes through to find all solutions (6,7,11, and 14).

Backtracking to an Appropriate Choice. The first deficiency of chronological backtracking is illustrated by the unnecessary state 4. The contradiction discovered in state 3 depends on choices **A** and **C** and not **E**. Therefore, replacing the choice **E** with **F** and working on state 4 is futile, as this change does not remove the contradiction. Unlike chronological backtracking, which replaces the most recent choice, dependency-directed backtracking replaces a choice that caused the contradiction. The discovery that state 3 is inconsistent causes immediate backtracking to state 5. To be able to determine which choices underlie the contradiction requires that the problem solver store dependency records with every datum that it infers. When an inconsistency is encountered, these dependencies are consulted to determine which choices contribute to the inconsistency.

Avoiding Rediscovering Contradictions. The second deficiency of chronological backtracking is illustrated by the unnecessary state 13. The contradiction discovered in state 10 depends on **B** and **E**. As **E** is the most recent choice, chronological and dependency-directed backtracking are indistinguishable.

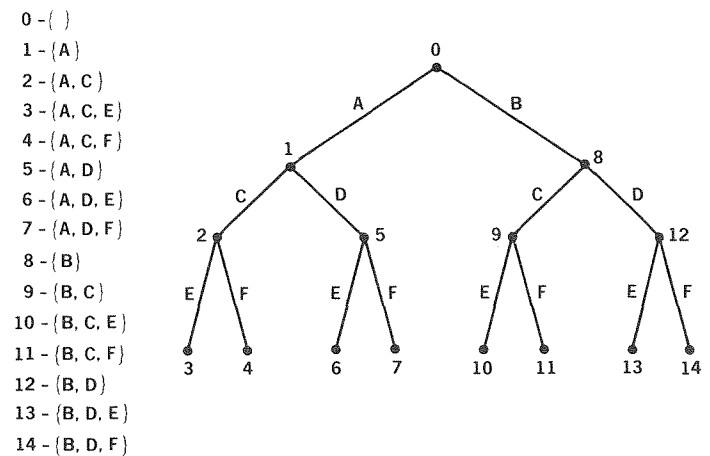


Figure 1. Chronological backtracking state sequence.

able, both backtracking to state 11. However, as **B** and **E** are known to be inconsistent with each other, there is no point in rediscovering this contradiction by working in state 13. Dependency-directed backtracking avoids this. Whenever a contradiction is discovered, the set of choices (called a nogood) contributing to the contradiction is added to a nogood database. Then, whenever backtracking introduces a new choice to the current state, this set is checked whether it contains any known nogood before problem solving is permitted to proceed. The two nogoods in this examples are {**A**, **C**} (discovered in state 3) and {**B**, **E**} (discovered in state 10).

Some Disadvantages

For some applications the disadvantages of dependency-directed backtracking may outweigh its advantages. Dependency-directed backtracking incurs a significant time and space overhead as it requires the maintenance of dependency records and an additional nogood database. Thus, the effort required to maintain the dependencies may be more than the problem-solving effort saved.

If the problem solver is logically complete (see Completeness) and finishes all work on a state before considering the next, the problem of backtracking to an inappropriate choice cannot occur (in this example, the contradiction found in state 3 would instead be found in state 2). In such cases much of the advantage of dependency-directed backtracking is irrelevant. However, most practical problem solvers are neither logically complete nor finish all possible work on a state before considering another.

Historical Perspective

The idea of dependency-directed backtracking grew out of applications of constraint propagation (qv) to electronic circuit analysis (1). However, dependency-directed backtracking, like chronological backtracking upon which it is an improvement, also suffers from avoidable deficiencies. For example, for both backtracking schemes inferences made in state 3 based on choices **C** and **E** are rederived in state 10. Fortunately, with more extensive use of dependency records, these and similar inefficiencies are avoided. The result is a truth maintenance system (2), which is a general problem-solving tool. Reference 2 contains the best description of dependency-directed backtracking, although it, as all the references, presents dependency-directed backtracking in the context of a larger truth maintenance system. Reference 3 describes the use of dependency-directed backtracking in the problem-solving system AMORD. Reference 4 describes the use of dependency-directed backtracking in another kind of truth maintenance system. Reference 5 includes extensive discussions and examples integrating dependency-directed backtracking into a constraint language. Reference 6 presents a truth maintenance system and an approach to problem solving that achieves most of the goals of dependency-directed backtracking but without backtracking (see also Belief revision).

BIBLIOGRAPHY

1. R. M. Stallman and G. J. Sussman, "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis," *Artif. Intell.* 9(2), 135-196 (1977).

2. J. Doyle, "A truth maintenance system," *Artif. Intell.* 12(3), 231-272 (1979).
3. J. de Kleer, J. Doyle, G. L. Steele, and G. J. Sussman, Explicit Control of Reasoning, in P. H. Winston and R. H. Brown (eds.), *Artificial Intelligence: An MIT Perspective*, Vol. 1, MIT Press, Cambridge, MA, pp. 94-116, 1979, also in R. J. Brachman and H. J. Levesque (eds.), *Readings in Knowledge Representation*, Morgan Kaufman, Los Altos, CA, pp. 345-355, 1986.
4. D. McAllester, *An Outlook on Truth Maintenance*, MIT Artificial Intelligence Laboratory, AIM-551, Cambridge, MA, 1980.
5. G. L. Steele, *The Definition and Implementation of a Computer Programming Language Based on Constraints*, MIT Artificial Intelligence Laboratory, TR-595, Cambridge, MA, 1979.
6. J. de Kleer, "An assumption-based TMS," *Artif. Intell.* 28(2), 127-196 (1986).

J. DE KLEER
Xerox PARC

BASEBALL

Two distinct AI systems have chosen the name of Baseball. One is a program that answers questions posed in English about baseball scores written in the early 1960s. It syntactically parses the sentences into templates (or specification lists) for the processor to look up in a data base (see B. F. Green, A. K. Wolf, C. Chomsky, and K. Laughery, "Baseball: An Automatic Question Answerer", in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, pp. 207-216, 1963).

The second system, written by Elliot Soloway, is a learning program that uses "snapshots" as its training instances. It uses domain knowledge (the rules of baseball), knowledge of physics, and the goals of the players to process these snapshots (see E. Soloway, "Learning = Interpretation + Generalization: A Case Study in Knowledge-Directed Learning", Report No. COINS-TR-78-12, Computer and Information Sciences Department, University of Massachusetts, Amherst, 1978; also see E. Soloway and E. M. Riseman, "Levels of Pattern Description in Learning", *Proceedings of the Fifth IJCAI*, Cambridge, MA, pp 801-811, 1977).

J. ROSENBERG
SUNY at Buffalo

BACKWARD CHAINING. See Processing, bottom-up and top-down.

BAYESIAN DECISION METHODS

Basic Formulation

Bayesian methods provide a formalism for reasoning about partial beliefs under conditions of uncertainty. In this formalism propositions are quantified with numerical parameters signifying the degree of belief accorded them by some body of knowledge, and these parameters are combined and manipulated according to the rules of probability theory. For example, if **A** stands for the statement "Ted Kennedy will seek nomina-