SYMBOLIC COMPUTATION

L. Kanal
V. Kumar
Editors

Search in Artificial Intelligence

Springer-Verlag

TP 18
K 16.2

8963522

Laveen Kanal Vipin Kumar
Editors

Search in Artificial Intelligence

With 67 Illustrations

Springer-Verlag
New York Berlin Heidelberg
London Paris Tokyo

Laveen Kanal
Department of Computer Science
University of Maryland
College Park, MD 20742
USA

Vipin Kumar
Computer Science Department
University of Texas at Austin
Austin, TX 78712-1188
USA

Library of Congress Cataloging-in-Publication Data
Search in artificial intelligence.
(Symbolic computation. Artificial intelligence)
Bibliography: p.
Includes index.
1. Artificial intelligence—Data processing.
2. Algorithms. 3. Information storage and retrieval
systems. I. Kanal, Laveen N. II. Kumar Vipin.
III. Series.
Q336.S4 1988 006.3 88-12247

Camera-ready text prepared by the editors.
Printed and bound by R.R. Donnelley & Sons, Harrisonburg, Virginia.
Printed in the United States of America

9 8 7 6 5 4 3 2 1

ISBN 0-387-96750-8 Springer-Verlag New York Berlin Heidelberg
ISBN 3-540-96750-8 Springer-Verlag Berlin Heidelberg New York

SYMBOLIC COMPUTATION

Artificial Intelligence

Preface

Search is an important component of problem solving in artificial intelligence (AI) and, more generally, in computer science, engineering and operations research. Combinatorial optimization, decision analysis, game playing, learning, planning, pattern recognition, robotics and theorem proving are some of the areas in which search algorithms play a key role.

Less than a decade ago the conventional wisdom in artificial intelligence was that the best search algorithms had already been invented and the likelihood of finding new results in this area was very small. Since then many new insights and results have been obtained. For example, new algorithms for state space, AND/OR graph, and game tree search were discovered. Articles on new theoretical developments and experimental results on backtracking, heuristic search and constraint propagation were published. The relationships among various search and combinatorial algorithms in AI, Operations Research, and other fields were clarified. This volume brings together some of this recent work in a manner designed to be accessible to students and professionals interested in these new insights and developments.

The first three papers are concerned with developing general formulations of search and investigating relationships among various search techniques of AI and Operations Research. The paper by Kumar and Kanal presents a model for discrete optimization problems and shows relationships among dynamic programming, branch-and-bound (B&B) and heuristic search. The paper by Helman presents a different model for discrete optimization and shows relationships between dynamic programming and B&B. This model applies to a smaller class of problems than the first paper, but allows an analysis of the complexity within a restricted class of computations. The third paper, by Kumar, Nau and Kanal, presents a general B&B procedure for AND/OR graph and game tree search, and shows that many important existing AI search procedures can be treated as special cases of this general B&B procedure.

The next three papers are concerned with the performance of the well known A* algorithm and its variations. Although the worst-case complexity of A* is probably exponential, the average case complexity depends upon the underlying nature of the search tree and the heuristic function. The paper by Bagchi and Sen shows that for certain characteristics of the search tree, the complexity of A* is exponential unless the heuristic function is very accurate. It is interesting to note a related result in Douglas Smith's 1979 dissertation. He showed that, for a different kind of search tree, the average search complexity for a best-first B&B procedure (which is essentially A*) can be polynomial (see JACM 1984).

Ever since its original development by Hart, Nilsson and Raphael, the optimality of A* over other heuristic search algorithms has been a controversial topic. The paper by Dechter and Pearl examines conditions under which A* is no better than other algorithms, and also under which A* is optimal. A recent paper by Mero presented a variation of A* in which the heuristic lower bounds are revised (to become more accurate) dynamically. Intuitively, it seems that the new "improved" bounds would only make the algorithm better, a claim also made in Mero's paper. The paper by Mahanti and Ray shows that this claim is incorrect; i.e., revising heuristic bounds can seriously hurt the performance in some cases. They present a new algorithm that revises heuristic bounds dynamically and does not have this drawback.

The paper by Korf deals with the problem of finding a path between two different states in the presence of different sources of knowledge. He shows that if knowledge about the domain is available only in terms of heuristic lower bounds, then an algorithm that performs cost bounded depth-first search repeatedly over a search space is indeed asymptotically optimal over all other admissible search algorithms. If more knowledge is available (in the form of subgoals, macro-operators or abstractions), then more powerful search methods can be used.

Means-end analysis is an important problem solving technique, which is embodied in the General Problem Solver (GPS) of Newell, Shaw and Simon. Banerji and Ernst examine a number of developments related to GPS. In particular they set forth the strengths and weaknesses of work by Goldstein and by Korf. The paper concludes with a negative view of heuristic search techniques based on numerical methods, and states the authors' belief that symbol manipulation or problem descriptions represents a more promising direction. While this belief is reasonable and is not uncommon in AI, it remains to be backed up by success on more than toy problems. On the other hand, heuristic search techniques have been used successfully in many of the areas listed at the beginning of the preface.

The next four papers deal with algorithms for solving the constraint satisfaction problem (CSP). Nadel presents a unified view of a large number of constraint satisfaction algorithms that were developed by many researchers at different times. He shows that all these algorithms can be viewed as having different degrees of tree search and consistency checking components. Freuder examines the conditions under which a CSP can be solved by doing a bounded amount of search. Dechter and Pearl show that the idea of solving a relaxed problem to create heuristic guidance can be effectively used in solving constraint satisfaction problems. Traditional formulations of the CSP deal only with binary constraints. Montanari and Rossi present an abstract formulation of the CSP which allows the constraints to be n-ary. They show that many properties known to be true of binary constraint networks are also true of the n-ary constraint networks.

The final paper, by Chi and Nau, presents an investigation of competing back-up rules for game tree search. Most game playing programs generate the game tree only up to a limited depth, and use the minimax back-up rule to propagate the heuristic values of the leaf nodes. Since these heuristic values can be in error, the selected move is not guaranteed to be the best. But one would still expect the overall performance (in terms of selecting the best move) to improve as the search depth is increased. Surprisingly, investigations by Nau and by others have shown that the discriminating power of the minimax back-up rule can decrease, rather than increase, with increasing depth of the search tree. This has led to investigations of alternatives to the minimax rule, such as the product rule. Chi and Nau show that there are real games for which the product rule does better than the minimax back-up rule, and investigate a possible way to predict the relative performance of the product rule against minimax.

The articles in this book should provide the reader a broad view of recent developments on search in AI and some of the relationships among different search algorithms. Even as this preface is being written, highly interesting work on parallel implementations of search algorithms and on their experimental performance on parallel computer architectures is appearing. The potential of parallel processing for AI search algorithms is so great that from now on it is likely to be the focus of work on search in AI. The new insights and developments provided by this volume should also be helpful to researchers interested in parallel search algorithms.

The editors thank all those who submitted papers for possible inclusion in this book. Even though some of the selected papers have been published elsewhere, we hope that having the papers revised for this book has resulted in greater readability and cohesiveness. We are grateful to the authors for their cooperation in revising their papers. We also thank the staff of Springer-Verlag, for their cooperation and help in producing the book.

Laveen Kanal and Vipin Kumar

Contents

THE CDP: A UNIFYING FORMULATION FOR HEURISTIC SEARCH, DYNAMIC PROGRAMMING, AND BRANCH-AND-BOUND

Vipin Kumar
Artificial Intelligence Laboratory
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Laveen N. Kanal
Machine Intelligence and Pattern Analysis Laboratory
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

This paper presents the composite decision process (CDP), a general model for discrete optimization problems. Using certain relationships between formal grammars, AND/OR graphs, and game trees, it is shown that a large number of search problems in artificial intelligence can be formulated in terms of this model. Two general classes of algorithms are discussed, and it is shown that most of the existing search algorithms to solve problems represented by this model fall into one of two categories. This approach to formulating and solving discrete optimization problems makes it possible to view several search procedures in a unified manner and clarifies the relationships among them. The approach also aids in synthesizing new variations and generalizations of existing search procedures.

[1] A preliminary version of this paper appeared in the proceedings of the 1983 National Conference on Artificial Intelligence.

This work was supported in part by Army Research Office grant #DAAG29-84-K-0060 to the Artificial Intelligence Laboratory at the University of Texas at Austin, and in part by NSF grants to the Machine Intelligence and Pattern Analysis Laboratory at the University of Maryland.

1. INTRODUCTION

Because of the somewhat differing requirements of the various areas in which search algorithms are used, different search procedures have evolved. Examples are, Alpha-Beta [17] and SSS* [30] for minimax search; A^* and other heuristic search procedures for state space search [1], [29]; AO* [28] for problem reduction search; and various branch-and-bound (B&B) [23], [20], [13] and dynamic programming (DP) procedures [2], [31], [5] for combinatorial optimization. Although several of these procedures are thought to be interrelated, their true relationships have not been properly understood. Furthermore, similar procedures (with minor modifications) have been invented and reinvented independently in different disciplines. The following statement by Stuart E. Dreyfus [4] regarding the search algorithms for discrete shortest path problems accurately reflects the state of affairs in the wider area of search algorithms.

> In the never-ending search for good algorithms for various discrete shortest-path problems, some authors have apparently overlooked or failed to appreciate previous results. Consequently, certain recently reported procedures are inferior to older ones. Also, occasionally, inefficient algorithms are adopted to new, generalized problems where more appropriate modifiable methods already exist.

A better understanding of these procedures would also have helped in seeing how improvements in one type of search procedure could be incorporated in other search procedures.

A large number of problems solved by dynamic programming, heuristic search, and B&B can be considered as discrete optimization problems, i.e., the problems can be stated as follows: find a least cost[2] element of a discrete set X. In many problems of interest, X is usually too large to make the exhaustive enumeration for finding an optimal element practical. But the set X is usually not unstructured. Often it is possible to view X as a set of policies in a multistage decision process, or as a set of paths between two states

[2] In some problems the function f represents the "merit" of the elements in the set X, and a largest merit element of X is desired. Unless otherwise stated, we shall deal with the minimization problem. With obvious modifications, the treatment for the maximization problem runs parallel to the treatment for the minimization problem.

in a state space, or as a set of solution trees of an AND/OR graph. These and other ways of representing X immediately suggest various tricks, heuristics, and short cuts to finding an optimal element of X. These short cuts and tricks were developed by researchers in different areas, with different perspectives and for different problems; hence, it is not surprising that the formal techniques developed look very different from each other, even though they are being used for similar purposes.

We have developed a general model for discrete optimization problems. This model provides a good framework for representing problem specific knowledge such that it can be usefully exploited for finding an optimum element of X. Using certain relationships between formal grammars, AND/OR graphs, and game trees, it is shown that a large number of search problems in Artificial Intelligence can be formulated in terms of this model. Two general classes of algorithms are discussed, and it is shown that most of the existing search algorithms to solve problems represented by this model fall into one of two categories. This approach to formulating and solving discrete optimization problems makes it possible to view most of the procedures mentioned before in a unified manner. It reveals the true nature of these procedures, and clarifies their relationships to each other. The approach also aids in synthesizing new variations as well as generalizations of existing search procedures.

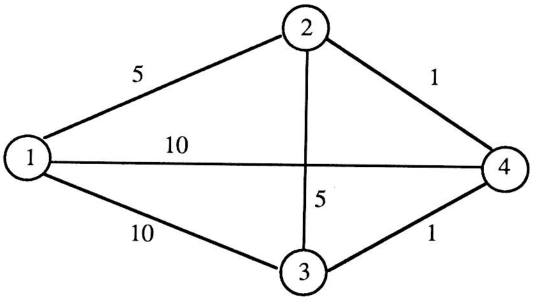
2. A MODEL FOR DISCRETE OPTIMIZATION

2.1. Discrete Decision Processes

A large subset of discrete optimization problems can be modeled by a Discrete Decision Process. A **discrete decision process (DDP)** is a 3-tuple $D = (V,X,f)$, where V is a finite alphabet, X is a subset of the set of all strings formed from the finite alphabet V (i.e., $X \subseteq V^*$), and f is a real valued cost function defined over the elements of X. The **minimization problem** for D is to find the least cost element of the set X.

For example, the problem of finding a shortest tour of the classical n-city traveling salesman problem can be stated in DDP format as follows. $V = \{a_i \mid 1 \leq i \leq n\}$, where a_i means "go to city i from the present city". X

Problem: Find the shortest tour which starts from the city 1, visits each city exactly once, and returns back to the city 1.



The alphabet A: $\{a_1, a_2, a_3, a_4\}$

The set of strings:
$$\left\{\begin{matrix} (a_2, a_3, a_4, a_1) \\ (a_2, a_4, a_3, a_1) \\ (a_3, a_2, a_4, a_1) \\ (a_3, a_4, a_2, a_1) \\ (a_4, a_2, a_3, a_1) \\ (a_4, a_3, a_2, a_1) \end{matrix}\right\}$$

The cost function f:

for a tour x, $f(x) =$ the sum of the intercity distances in the tour x.

Figure 2.1: Traveling Salesman Problem as a DDP.

consists of precisely those strings in V* which are permutations of $(a_1,a_2,...,a_n)$ ending in a_1 (e.g., $(a_2,a_3,...,a_n,a_1)$) (see Fig. 2.1). It is assumed that the tour starts at city 1. A string x in the set X is a traveling tour, and f(x) is the sum of the inter-city distances encountered in the tour x.

The DDP was originally introduced by Karp & Held [14] as a standard format for stating various optimization problems. A natural question that can be asked is how to specify the discrete set X and the cost function f. This is a rather important question, as the efficiency of the search process for finding x* in X is significantly influenced by the way in which X and f are specified. This will become clearer as we proceed with our discussion.

2.2. Composite Decision Processes

In many problems, the discrete set X can be naturally specified in a structured form, and this structure can be exploited to perform efficient search of an optimum element x* of X. Of particular interest is the case in which the set X can be specified as generated by a context-free grammar and f(x), for strings x in X, is defined in terms of composition of the cost of smaller strings in a recursive manner.

A context-free grammar is a 4-tuple G = (V,N,S,P). Here, V, N, P, and S are respectively the sets of terminal symbols, nonterminal symbols, productions, and the start symbol for the grammar. Each production in P is of the form: $w_1 \rightarrow w_2$, where w_1 is a nonterminal symbol in N and w_2 is a string in $(N \cup V)^*$. S is a symbol in N.

A string w in (N U V)* is said to be derived from a nonterminal symbol B in N if w can be obtained from B by sequential application of some number of productions in P. A grammar G generates precisely those strings in the set V* which can be derived from the start symbol S. Derivation of a string x in V* from a nonterminal symbol B of N can be described pictorially by a parse tree (also called derivation tree). If B is the root of a parse tree T, and x is the sequence of terminal symbols (or nodes) of T in left-to-right order, then the tree T denotes a derivation or parse of the string x from B. A detailed treatment of context-free grammars and parse trees can be found in [10].

Let us associate a k-ary cost attribute t_p with each production p: $n \rightarrow n_1...n_k$ in a way similar to the synthesized attributes defined by Knuth

[15]. Let c: $V \rightarrow R$ be a real valued function defined over the set of terminal symbols. The function c_T can be recursively defined on nodes n of a parse tree T as follows:

(i) If n is a terminal symbol of G then
(2.1a) $c_T(n) = c(n)$.

(ii) If n is a nonterminal symbol and $n_1,...,n_k$ are descendants (from left to right) of n in T, i.e., p: $n \rightarrow n_1,...,n_k$ is a production in G, then
(2.1b) $c_T(n) = t_p(c_T(n_1),...,c_T(n_k))$.

If n is the root symbol of a parse tree T then a cost function f can be defined on solution trees of G as:

(2.2) $f(T) = c_T(n)$.

For a node n of a parse tree T, $c_T(n)$ denotes the cost of the subtree of T rooted at n. For a production p: $n \rightarrow n_1\ n_k$, $t_p(x_1,...,x_k)$ denotes the cost of a derivation tree T rooted at n if the costs of the subtrees of T rooted at $n_1,...,n_k$ are $x_1,...,x_k$. Thus the cost of a parse tree is recursively defined in terms of the costs of its subtrees. See Fig. 2.2 for an illustration.

A **composite decision process**[3] **(CDP)** is a 3-tuple C = (G(V,N,S,P),t,c), where G=(V,N,S,P) is a context-free grammar, and the functions t and c are as defined above. A cost f(T) is associated with each parse tree T of G from Equation 2.2. The **minimization problem** for the composite decision process C can be stated as follows: find a parse tree T* rooted at the start symbol S such that f(T*) = min{ f(T) | T is a parse tree rooted at S}.

A cost function g can be defined on strings x generated by the grammar G as follows: g(x) = min{ $c_T(S)$ | T is some derivation tree of x rooted at S}. Note that the grammar G can be ambiguous, and thus there can be more than one derivation tree of x rooted at S.

---

[3]As developed in [18] the CDP concept requires a 4-tuple formulation. (the fourth parameter specifies the problem instance.) For the sake of notational simplicity, only a 3-tuple formulation is presented here. This simple formulation is adequate for the discussion in this paper.

Context Free Grammar:

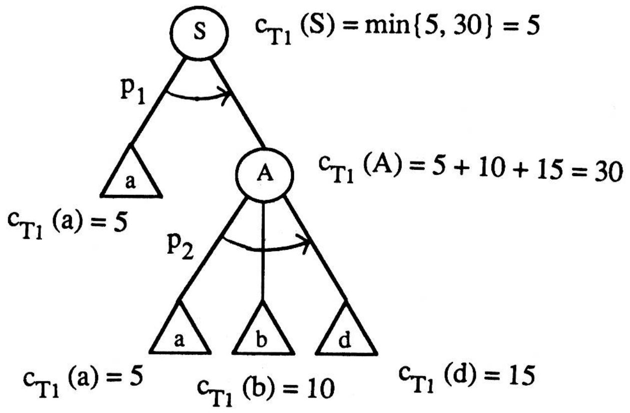
$$G = (\overset{V}{\{a, b, d\}}, \overset{N}{\{S, A\}}, S, P)$$

$$P: \begin{cases} p_1: S \rightarrow aA, & t_{p1}(r_1, r_2) = \min\{r_1, r_2\}, \\ p_2: A \rightarrow abd, & t_{p2}(r_1, r_2, r_3) = r_1 + r_2 + r_3, \\ p_3: A \rightarrow ad, & t_{p3}(r_1, r_2) = r_1 + r_2, \\ p_4: s \rightarrow aS, & t_{p4}(r_1, r_2) = r_1 * r_2, \end{cases}$$

(Headings: **Productions:** / **Cost Attributes:**)

terminal costs: $c(a) = 5$, $c(b) = 10$, $c(d) = 15$.

2. (a) A composite decision process $C = (G(V, N, S, P), t, p, c)$



2. (b) The derivation tree T_1 depicting derivation of *aabd* from *S*:

$$f(T_1) = c_{T1}(S) = 5$$

Figure 2.2