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.

L. Kanal et al. (eds.), *Search in Artificial Intelligence*

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.

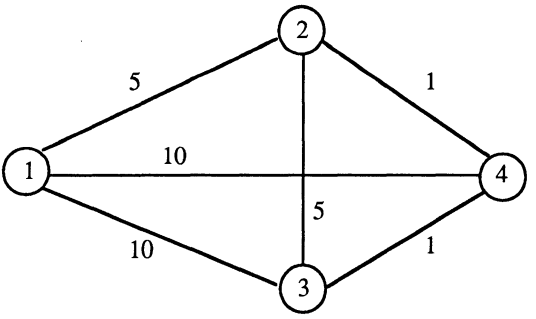
2. A MODEL FOR DISCRETE OPTIMIZATION

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. $V = \{a_i \mid 1 \le i \le n\}$, 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:

$$\left\{\begin{array}{l}(a_2, a_3, a_4, a_1)\\(a_2, a_4, a_3, a_1)\\(a_3, a_2, a_4, a_1)\\(a_3, a_4, a_2, a_1)\\(a_4, a_2, a_3, a_1)\\(a_4, a_3, a_2, a_1)\end{array}\right\}$$

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→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.

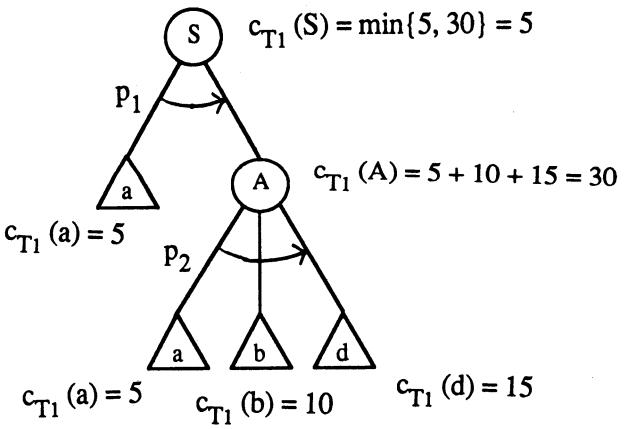
Context Free Grammar:

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

	Productions:	Cost Attributes:
P:	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: $s \rightarrow aS$,	$t_{p4}(r_1, r_2) = r_1 * r_2$,

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

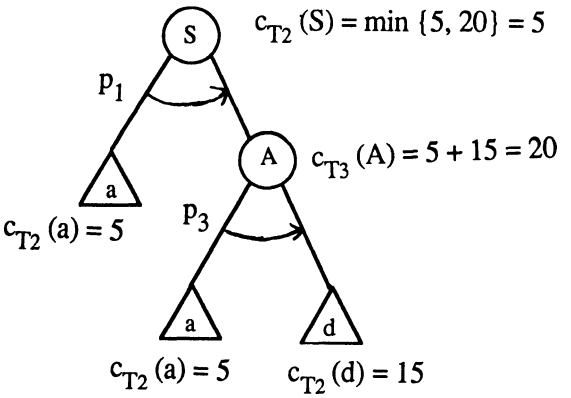
2. (a) A composite decision process $C = (G(V, N, S, P), t, p, c)$



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

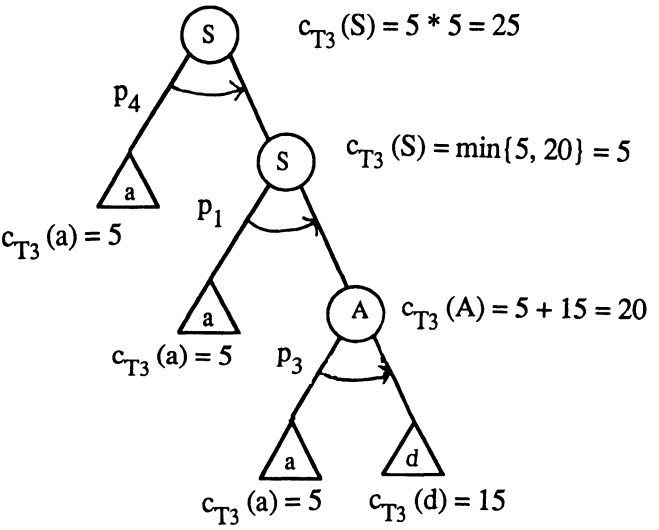
$$f(T_1) = c_{T1}(S) = 5$$

Figure 2.2



2. (c) The derivation tree T_2 depicting derivation of *aad* from *S*:

$f(T_2) = c_{T2}(S) = 5$



2. (d) The derivation tree T_3 depicting derivation of *aaad* from *S*:

$$f(T_3) = c_{T3}(S) = 25$$

Fig 2.2, continued

A composite decision process C = (G(V,N,S,P),t,c) is said to represent the discrete decision process D = (V,X,f) if $X \subseteq V^*$ is the set of strings generated from the grammar G, and if for all $x \in X$, f(x) = g(x). Clearly, any composite decision process represents a unique discrete decision process, and the minimization problems for a DDP and a CDP which represents it are equivalent.

We next introduce a type of composite decision process for which the minimization problem can be replaced by the problem of solving a system of recurrence equations.

2.3. Monotone Composite Decision Processes

A CDP C = (G(V,N,S,P),t,c) is called a **monotone composite decision process (MCDP)** if all of the k-ary cost attributes t_p associated with the productions p: $n \rightarrow n_1 \ldots n_k$ are monotonically nondecreasing in each variable, i.e., $x_1 \le y_1$ & ... & $x_k \le y_k \Rightarrow t_p(x_1,\ldots,x_k) \le t_p(y_1,\ldots,y_k)$.

For a symbol n of the grammar G, let $c^*(n)$ denote the minimum of the costs of the parse trees rooted at n. Note that if n is nonterminal, then $c^*(n)$ may be undefined, as there may be an infinite number of parse trees of decreasing costs rooted at n. The following theorem establishes that for a monotone CDP, $c^*(n)$ can be defined recursively in terms of $\{c^*(m) \mid$ m is a part of a string directly derivable from n$\}$.

Theorem 2.1: For a monotone composite decision process C = (G, t, c), if $c^*(n)$ is defined for all symbols n of G then the following recursive equations hold.

(2.3a) If n is a nonterminal symbol then

$$c^*(n) = \min\{t_p(c^*(n_1),\ldots,c^*(n_k)) \mid p: n \rightarrow n_1 \ldots n_k \text{ is a production in G}\}.$$

(2.3b) If n is a terminal symbol then

$$c^*(n) = c(n).$$

Proof: If p: $n \rightarrow n_1 \ldots n_k$ is a production in G, then $t_p(c^*(n_1),\ldots,c^*(n_k))$ is the cost of some derivation tree rooted at n. From the definition of c^*, $c^*(n)$ is the smallest of the costs of the derivation trees rooted at the symbol n. Hence,

(2.4) $c^*(n) \le \min\{t_p(c^*(n_1),\ldots,c^*(n_k)) \mid p: n \rightarrow n_1 \ldots n_k$ is a production in G$\}$.

Let T be a parse tree rooted at n for which $f(T) = c^*(n)$. Let p: $n \rightarrow n_1 \ldots n_k$ be the production used at the root of T, and let $T_1,\ldots,T_k$ be the subtrees of T rooted at $n_1,\ldots,n_k$ respectively. Then

(2.5) $c^*(n) = t_p(f(T_1),\ldots,f(T_k))$.

But from the definition of c^*, $c^*(n_i) \leq f(T_i)$ for any derivation tree T_i rooted at n_i, whence from the monotonicity of t_p,

(2.6) $t_p(f(T_1),\ldots,f(T_k)) \geq t_p(c^*(n_1),\ldots,c^*(n_k))$.

Hence,

(2.7) $c^*(n) = t_p(f(T_i),\ldots,f(T_k))$ (from (2.5))

$\geq t_p(c^*(n_1),\ldots,c^*(n_k))$ (from (2.6))

$\geq \min\{t_p(c^*(n_1),\ldots,c^*(n_k)) \mid p: n \rightarrow n_1 \ldots n_k \text{ is a production in } G\}$.

The theorem follows from (2.4) and (2.7).

[]

As discussed in Section 3, there does not exist a general algorithm for solving the system of recursive equations of an arbitrary monotone CDP. But for many interesting special cases there exist algorithms which solve these equations to compute $c^*(S)$, the smallest of the costs of the parse trees rooted at S. These algorithms can often be easily modified to build a least-cost parse tree rooted at S. In such a case, the minimization problem of a monotone CDP becomes equivalent to solving a system of recursive equations. The other alternative for solving the minimization problem of a CDP C = (G,t,c) - exhaustive generation and evaluation of all the derivation trees of G - turns out to be far more expensive than solving the recursive equations. Of course, the exhaustive generation is not even feasible if G generates an infinite number of strings.

2.4. Relationship with Dynamic Programming

Note that solving an optimization problem by Bellman's dynamic programming technique also involves converting the optimization problem into a problem of solving a set of recursive equations. Since most of the discrete optimization problems solvable by the conventional dynamic programming technique (and many more) can be stated in the monotone CDP format, we can consider the monotone composite decision process as a generalized

dynamic programming formulation, and the recursive equations of Theorem 2.1 as the generalized recursive equations of dynamic programming.

It is also possible to state a principle similar to Bellman's principle of optimality (all subpolicies of an optimum policy are also optimal). First, let us define the optimality criterion for a parse tree (the counterpart of Bellman's "policy" in our formulation). A parse tree rooted at symbol n of G is called an **optimum parse tree** if its cost (f-value) is the smallest of all the parse trees rooted at symbol n of G.

Lemma 2.1: For a monotone composite decision process C = (G,t,c), if $c^*(n)$ is defined for all symbols n of G, then for every symbol n of G, there exists an optimal parse tree rooted at n, all of whose subtrees (rooted at the immediate successors of n) are optimal.

Proof: See [18].

This statement is some what different (in fact "weaker") than Bellman's principle of optimality, because the monotonicity of t only guarantees that one can not build a **better** parse tree than T by replacing one of the subtrees of T by a worse subtree. But it is possible that the cost of a parse tree remains the same even though one of its subtrees have been replaced by an inferior counterpart. For example, in Fig. 2.2c, if the subtree of T_2 rooted at A (cost=20) is replaced by an inferior subtree (cost=30), we get the derivation tree T_1 (Fig. 2.2b), but $f(T_1) = f(T_2) = 5$. Hence, if there are more than one optimal parse trees rooted at a symbol n of G then subtrees of some of the optimal parse trees may not be optimal. Nevertheless, Lemma 2.1 guarantees that an optimal parse tree can always be built by optimally choosing from the aiternate compositions of only the optimal subtrees. This technique of first finding the optimal solution to small problems and then constructing optimal solutions to successively bigger problems is at the heart of all dynamic programming algorithms.

A stronger statement much similar to Bellman's principle of optimality can be made for a subclass of monotone CDP. A CDP C = (G,t,c) is called a **strictly-monotone CDP (SMCDP),** if all the k-ary functions t_p associated with productions p: $n \rightarrow n_1 \ldots n_k$ are strictly increasing in each variable, i.e., $\{x_i < y_i$ and $x_j \leq y_j$ for $j \neq i$ and $1 \leq j \leq k\} \Rightarrow \{t_p(x_1,\ldots,x_k) < t_p(y_1,\ldots,y_k)\}$.

Lemma 2.2: For a strictly-monotone CDP, if $c^*(n)$ for some symbol n of G is defined then all the subtrees of an optimal parse tree rooted at symbol n are also optimal.
Proof: See [18].

2.5. Relationship with Sequential Decision Processes

A CDP C = (G(V,N,S,P),t,c) is called a **regular composite decision process** (Regular CDP) if the context-free grammar G is further restricted to be a regular grammar. Using the correspondence between regular grammars and finite state automata, the regular CDP formulation can be shown to be equivalent to the sequential decision process formulation which was introduced by Karp & Held [14] as a model for the problems solved by dynamic programming. In a sequential decision process, the discrete set X is represented (defined) by a finite state automaton, and a cost function is defined over the strings in V* with the help of a function associated with state transitions. A regular CDP C = (G(V,N,S,P),t,c) is called an **additive-regular CDP** if for all $p \in P$, $t_p(x_1,...,x_n) = x_1+...+x_n + k(p)$. (Here, k(p) is a constant specific to each production.) The class of additive-regular CDP is equivalent to additive processes (a subclass of SDP defined in [14]).

The concept of a sequential decision process is very important. It has been extensively studied in many areas in different guises. State space descriptions used in artificial intelligence (AI) to solve various problems are essentially sequential decision processes, as a finite-state-space graph is equivalent to a finite-state automaton. The cost functions used in AI problems are special cases of the general cost function of an SDP. The minimization problem of an SDP is essentially a generalized version of the well known shortest path problem studied extensively in the Operations Research literature [4]. Various Branch & Bound, dynamic programming and heuristic search procedures have been developed for problems which can be modeled by SDPs (e.g., [23], [11], [12], [13], [5], [27], [28]). Generalized versions of many of these procedures are also applicable to problems modeled by CDPs.

However, there are many discrete optimization problems (e.g., optimal binary search tree [16], finding the optimal sequence for matrix multiplication [7]) which can not be naturally formulated in terms of an SDP, but they

can be solved by a technique similar to conventional dynamic programming. As shown in the examples below, these problems can be formulated in a CDP.

Example 3.1: Finding the Optimal Sequence for the Multiplication of Matrices[4]

Problem Statement: Given a sequence of matrices $M_1,...,M_N$ ($r_0,...,r_N$ give the dimensions of the matrices; matrix M_i has the dimensions r_{i-1}, r_i), find an optimal way of multiplying them, i.e., find a way which requires the least number of multiplications.

Formulation as a Discrete Decision Process D = (V,X,f)

$V = \{a_{ijk} \mid 1 \le i \le j < k \le N\}$, where a_{ijk} means
"multiply the results of $M_i*...*M_j$ by the result of $M_{j+1}*...*M_k$".

X = {sequences of a_{ijk} corresponding to complete parenthesization of the N matrices}.

$f(e) = 0$.

$f(x\ a_{ijk}) = f(x) + r_{i-1}*r_j*r_k$.

Note that the definition of f is extended to all the sequences in V* and not just in S. This is done to simplify the definition of f. We need not be concerned about the f-values of the strings of V* not in S.

Formulation as a CDP C = (G(V,N,S,P),t,c)

V is the same as in DDP D.

$N = \{M_{i,j} \mid 1 \le i \le j \le N\}$

where $M_{i,j}$ represents the problem of multiplying matrices $M_i,...,M_j$.

$S = M_{1N}$

$P = \{ \{M_{i,j} \rightarrow M_{i,k}\ M_{k+1,j}\ a_{ikj} \mid 1 \le i \le k < j \le N\} \cup \{M_{i,i} \rightarrow \epsilon \mid 1 \le i \le N\} \}$.

---

[4]Examples 3.1 and 3.2 are taken from [3].

Cost Attributes t:

For a production $p : M_{i,j} \rightarrow M_{i,k}\, M_{k+1,j}\, a_{ikj}$,

$$t_p(x_1,x_2,x_3) = x_1 + x_2 + x_3.$$

For a production $p : M_{i,i} \rightarrow \in$,

$$t_p(x) = 0.$$

Terminal Cost Function c:

$c(a_{ijk}) = r_{i-1} * r_j * r_k$.

Example 3.2: Optimal Binary Search Trees

Problem Statement: We are given N names $A_1,...,A_N$ and 2N+1 frequencies $r_0,...,r_n$; $s_1,...,s_n$. Here s_i is the frequency of encountering name A_i, and r_i is the frequency of encountering a name which lies between A_i and A_{i+1}; r_0 and r_n have obvious interpretations.

Formulation as a Discrete Decision Process D = (V,X,f)

$V = \{a_{ijk} \mid 1 \le i < j \le k \le N\}$, where a_{ijk} means
join $r_i\, s_{i+1} \ldots r_{j-1}$ and $r_j\, s_{j+1} \ldots r_k$ with s_j.

X = {sequences of a_{ijk} corresponding to complete parenthesization of n+1 elements $(r_0,...,r_n)$}.

For $x \in X$, f(x) = Weight path length of the binary tree corresponding to the parenthesization x}.

Formulation as a CDP C = (G(V,N,S,P),t,c)

V is the same as in DDP D.

$N = \{M_{i,j} \mid 1 \le i \le j \le N\}$

$S = M_{1,N}$

$P = \{ \{M_{i,j} \rightarrow M_{i,k-1}\, M_{k,j}\, a_{ikj} \mid 1 \le i < k \le j \le N\} \cup \{M_{i,i} \rightarrow \in \mid 1 \le i \le N\}\}$.

Cost Attributes t:

For a production $p : M_{i,j} \rightarrow M_{i,k-1}\ M_{k,j}\ a_{ikj}$,

$t_p(x_1,x_2,x_3) = x_1 + x_2 + x_3$.

For a production $p : M_{i,i} \rightarrow \in$,

$t_p(x) = 0$.

Terminal cost Function c:

$c(a_{ijk}) = r_i+s_{i+1} + ... + s_j+r_j$.

2.6. The Scope of the CDP Formulation

The composite decision process is more general than the sequential decision process. It is able to model a large number of problems in AI and other areas of computer science which can not be naturally formulated in terms of SDPs. The wide applicability of CDPs becomes obvious when we notice that there is a direct natural correspondence between context-free grammars and AND/OR graphs used in AI applications to model problem reduction schema [8]. (See Fig. 2.3 for an AND/OR graph equivalent to the grammar G of a CDP C = (G,t,c) which model the problem of multiplying three matrices in Example 3.1.) Due to this correspondence, the specification of a problem by an AND/OR graph can often be viewed as its specification in terms of an CDP, and the problem of finding a least cost solution tree of an AND/OR graph becomes equivalent to the minimization problem of its corresponding CDP. Due to the correspondence between AND/OR trees and game trees [22], the problem of finding the minimax value of a game tree can also be represented in the CDP formulation. Figure 2.4 depicts the scope of CDP.

2.7. Relationship with Other Work

Since Karp & Held's SDP is not able to model all the problems solved by dynamic programming, many researchers have tried to develop generalized models. Martelli and Montanari take a generalized view of dynamic programming [24], [25] as they allow recursive equations of dynamic programming to be polyadic. Note that the recursive equations obtained from an SDP are monadic. Using an algebraic framework, Martelli and Montanari prove that solving a system of monotone polyadic functional

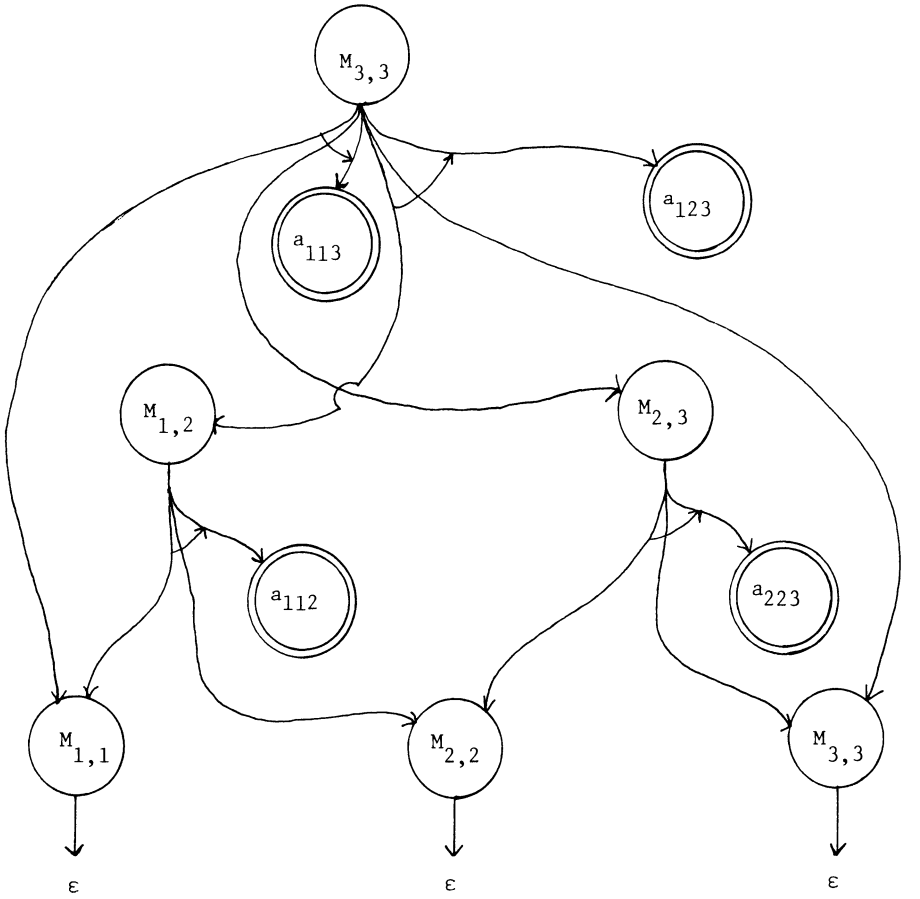


Fig. 2.3. The problem reduction graph for optimizing the multiplication of a sequence of matrices M_1, M_2, M_3.

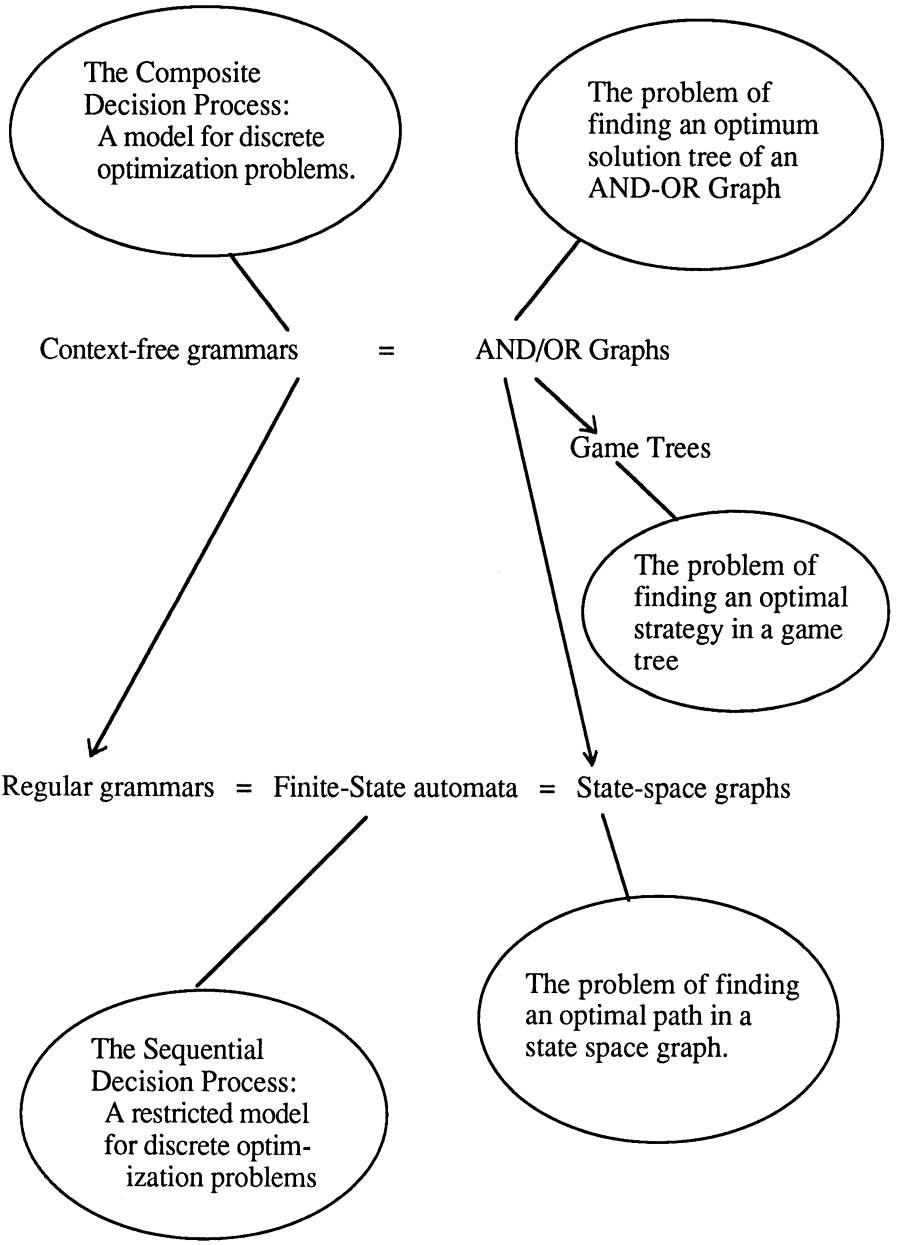


Figure 2.4: The scope of the CDP formulation.

equations is equivalent to searching an AND/OR graph with monotone cost functions [6]. But they consider a procedure for solving the functional equations to be a dynamic programming procedure only if the corresponding AND/OR graph is acyclic and it is searched in a bottom-up manner ([6], p748). This means that they do not consider many of the procedures developed by Ibaraki [11] for various classes of SDPs to be dynamic programming procedures.

Helman has developed a different model for DP and B&B [9]. In this model, a search procedure is viewed as a sequence of lateral and vertical combinations. The model distinguishes between B&B and DP primarily on the basis of the dominance relation used in lateral combinations - B&B uses a bounding relation (i.e., lower bounds), and dynamic programming uses a comparability relation. This view leaves out many algorithms that are considered as DP by other researchers (e.g., Dijkstra's algorithm for finding a shortest path in a graph is considered a DP algorithm by Dreyfus and Law [5]), and is not able to model many game tree search and AND/OR graph search procedures. But it helps in determining an optimal search strategy within a class of search strategies using a certain kind of dominance relation in lateral combinations.

By using a context-free grammar instead of a finite state automaton to describe the discrete set X, we came up with the concept of a CDP as a generalization of the concept of an SDP. Interestingly, most of the procedures developed by Ibaraki to solve the minimization problem of an SDP, can be easily generalized to the case when the set X of strings is described by a context-free grammar instead of a finite state automaton. Moreover, the natural correspondence between AND/OR graphs and context-free grammars [8] immediately extends the application domain of the composite decision process to a very important class of problems. The utility of the concept of CDP is further enhanced because we have identified two general classes of algorithms to solve its minimization problem, and most of the existing algorithms to solve the problems modeled by CDPs fall naturally

into one of the two categories.

3. SOLVING THE MINIMIZATION PROBLEM

We have shown in [18] that, in its full generality, the minimization problem of a monotone CDP (hence of a CDP) is unsolvable. However we identified three interesting special cases (acyclic-monotone CDP, positive-monotone CDP, strictly-monotone CDP) of monotone CDPs for which the minimization problem is solvable. In all the three cases a least cost parse tree of G rooted at S can be provably identified by generating and evaluating only a finite number of parse trees of G, even though G may generate infinitely many parse trees. This is a sufficient proof for the solvability of their minimization problems. But even these finite parse trees can be too many. In the following we briefly discuss two general techniques to solve the minimization problems of these CDPs in a manner which can be much more efficient than simple enumeration.

3.1. The Bottom-up Search Method

One way of solving the minimization problem of a given monotone CDP C = (G(V,N,S,P),t,c) is to successively find (or revise the estimates of) $c^*(n)$ for the symbols n of the grammar G until $c^*(S)$ is found. Viewed in terms of AND/OR graphs, bigger problems are successively solved starting from the smaller problems. This bottom-up search method is used in many search procedures. Historically, many of these procedures have also been called Dynamic Programming computations. Furthermore, the basic ideas of Dynamic Programming procedures - Bellman's principle of optimality and the recursive equations - are associated with monotone composite decision processes in their generalized forms. Hence we name all these bottom-up procedures for minimization of CDPs as dynamic programming.

In [18] we have presented dynamic programming procedures to solve the minimization problem of the three solvable classes of CDPs. Ibaraki's procedures for solving the minimization problems of SDPs, Dijkstra's algorithm for shortest path, Knuth's generalization of Dijkstra's algorithm, Martelli and Montanari's bottom-up search algorithm for constructing an optimal solution tree of an Acyclic AND/OR graph, and many other

optimization algorithms (usually called dynamic programming algorithms) can be considered as special cases of these procedures.

3.2. The Top-Down Search Method

In this technique, we start with some representation of the total (possibly infinite) set of parse trees out of which a least cost parse tree needs to be found. We repeatedly partition this set (a partitioning scheme is usually suggested by the problem domain). Each time the set is partitioned, we delete all members of the partition for which it can be shown that even after eliminating the set, there is a least cost parse tree in one of the remaining sets. This cycle of partitioning and pruning can be continued until only one (i.e., a least cost) parse tree is left.

This "top-down" process of partitioning and pruning for determining an optimal element of a set has been used for solving many optimization problems in Operations Research, where it is known as branch and bound (B&B). It is easy to see that the central idea of B&B - the technique of branching and pruning to discover an optimum element of a set- is at the heart of many heuristic procedures for searching state space, AND/OR graphs, and game trees. But none of the B&B formulations presented in the O.R. literature adequately model AND/OR graph and game tree search procedures such as Alpha-Beta, SSS* [26], AO* and B* [1]. This has caused some of the confusion regarding the relationship between heuristic search procedures and B&B.

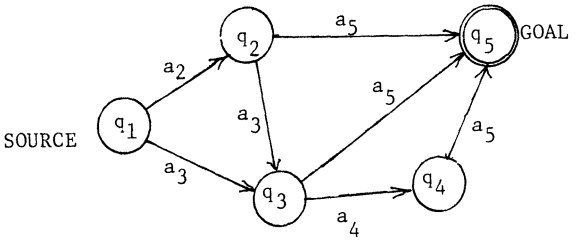
To remedy this situation, we have developed a formulation of B&B which is more general and also much simpler than existing formulations. We have further developed a B&B procedure to search for an optimum solution tree of an acyclic AND/OR graph (i.e., to solve the minimization problem of an acyclic monotone CDP) which generalizes and unifies a number of AND/OR graph and Game tree search procedures such as AO*, SSS*, Alpha-Beta, and B* [18], [22].

3.3. The Equivalence of Top-Down and Bottom-Up Computation for a Subclass of CDP

Due to an interesting property of regular grammars, for a subclass of regular CDPs, top-down search methods are indistinguishable from bottom-up search methods. For every regular grammar, there exists another equivalent "dual" regular grammar such that the AND/OR graph equivalent to one regular grammar is a "reversed" version of the AND/OR graph equivalent to its dual grammar. Figures 3.1b and 3.1c show two regular grammars that are dual of each other; both grammars are equivalent to the finite-state automaton Π of Fig. 3.1a. The AND/OR graphs of Fig. 3.1b and Fig. 3.1c represent the problem of going from the source state to the goal state in Π. In the AND/OR graph of Fig. 3.1b, q_i denotes the problem of going from q_1 to q_i in Π. In the AND/OR graph of Fig. 3.1c, q_i denotes the problem of going from q_i to q_5 in Π.

Let us call a regular CDP $C = (G,t,c)$ **invertible** if it is possible to construct its **inverted** regular CDP $C^I = (G^d,t^d,c^d)$ such that G^d is the dual of G, and C and C^I represent the same discrete decision process. It is easy to see that the bottom-up search of the AND/OR graph equivalent to G can also be viewed as a top-down search of the AND/OR graph equivalent to G^d, and vice versa. Also, the minimization problems of C and C^I are equivalent.

Even though we have not been able to precisely characterize the subclass of regular-CDPs which can be inverted, it is easy to see that additive-regular CDPs are invertible. Additive-regular CDP models a number of important problems (e.g., the traveling salesman problem, the shortest path problem). This explains why many algorithms (e.g., Dijkstra's algorithm for finding a shortest path in a graph) were considered dynamic programming by some researchers and B&B by others. This also shows that B&B procedures of Ibaraki [13] for solving the minimization problem of an SDP could also be viewed as DP computations. Traditionally, the A^* algorithm [27] is considered a top-down search procedure. But it can be viewed as a bottom-up search procedure as well, since it solves the minimization problem of an additive-regular CDP. This insight was used to develop a heuristic bottom-up search procedure for CDP as a generalization of A^*.



(a)

A left linear grammar

$q_5 \rightarrow q_2\ a_5$

$q_5 \rightarrow q_3\ a_5$

$q_5 \rightarrow q_4\ a_5$

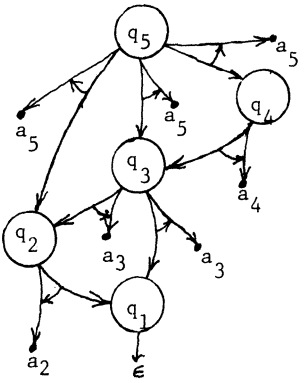
$q_4 \rightarrow q_3\ a_4$

$q_3 \rightarrow q_2\ a_3$

$q_3 \rightarrow q_1\ a_3$

$q_2 \rightarrow q_1\ a_2$

$q_1 \rightarrow \epsilon$



(b)

A right linear grammar

$q_1 \rightarrow a_2\ q_2$

$q_1 \rightarrow a_3\ q_3$

$q_2 \rightarrow a_3\ q_3$

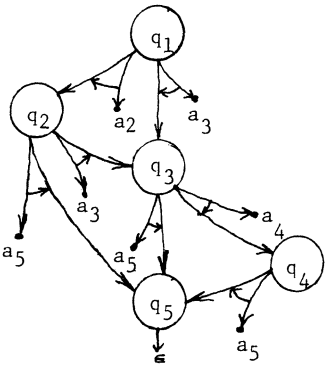
$q_2 \rightarrow a_5\ q_5$

$q_3 \rightarrow a_4\ q_4$

$q_3 \rightarrow a_5\ q_5$

$q_4 \rightarrow a_5\ q_5$

$q_5 \rightarrow \epsilon$



(c)

Fig 3.1 (a) A finite state automaton Π. (b) A left linear grammar and an AND/OR graph equivalent to Π. (c) A right linear grammar and an AND/OR graph equivalent to Π.

4. CONCLUDING REMARKS

The composite decision process has greatly enhanced our understanding of the nature and interrelationships of dynamic programming, branch-and-bound(B&B) and heuristic search procedures, and has provided us with a general paradigm for formulating and analyzing a large number of search procedures. In the light of our model these procedures can be viewed as if they are searching for an optimal element of a discrete set X in either top-down or bottom-up fashion by making use of the problem-specific structure present in X and f. (See Figure 4.1.) The structure in X is present in its description as a set of parse trees of a context-free grammar G(V,N,S,P). The structure in f is present in its definition in terms of the attributes associated with the productions of G. Additional problem-specific knowledge is used in the form of heuristic bounds on $c^*(n)$ for symbols $n \in N$.

According to our model, B&B is a top-down search method, whereas dynamic programming is a bottom-up search procedure. B&B uses the structure in X and f via the dominance relation, whereas dynamic programming uses it via the principle of optimality. Lower bounds used in B&B are essentially heuristic bounds on $c^*(n)$. Traditional bottom-up search procedures do not use these bounds, which led some researchers to believe that dynamic programming is essentially breadth-first (see [27], p76). But bottom-up search methods (hence, dynamic programming) can certainly make use of heuristic information to speed up search [19], [26]. Most heuristic search methods of AI use both kinds of information. Some of them can be classified as B&B and others as dynamic programming. As discussed in Section 3.3, for an important subset of the problems formulated by the CDP, top-down algorithms are indistinguishable from the bottom-up algorithms. This explains why many algorithms were considered as B&B by some researchers and dynamic programming by others.

The model has also helped us devise a general top-down search procedure which subsumes a large number of well known search procedures [22]. For example, a top-down (B&B) formulation for searching AND/OR graphs subsumes a number of heuristic procedures such as AO*, SSS*, Alpha-Beta, and B*, and uncovers hitherto unknown interrelationships of these procedures. By viewing A^* as a bottom-up search procedure, we were able to develop a general heuristic bottom-up procedure for searching

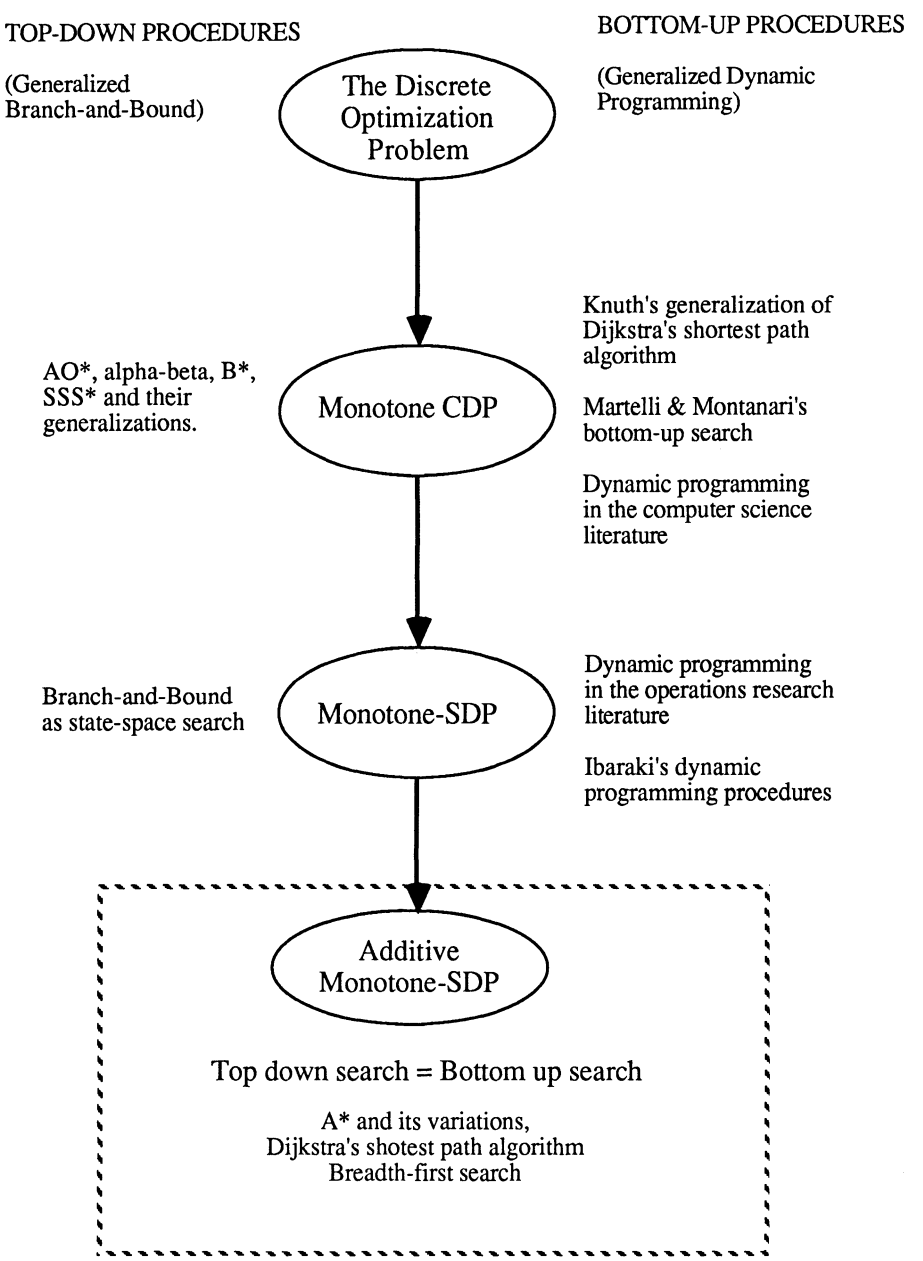


Figure 4.1: Classification of Search Procedures.

AND/OR graphs as a generalization of A* [19]. The methodology has also helped us formulate parallel search procedures for AND/OR graphs and game trees [21].

REFERENCES

[1] A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence, Vol I,* William Kaufman, Inc., Los Altos, CA, 1981.

[2] R. E. Bellman, *Dynamic Programming,* Princeton University Press, Princeton, NJ, 1957.

[3] K. G. Brown, Dynamic Programming in Computer Science, Tech. Report CMU-CS-79-106, Computer Science Dept., Carnegie-Mellon Univ., 1979.

[4] S. E. Dreyfus, An Appraisal of Some Shortest Path Algorithms, *Operations Research 17,* pp. 395-412, 1969.

[5] S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming,* Academic Press, New York, 1977.

[6] S. Gnesi, A. Martelli, and U. Montanari, Dynamic Programming as Graph Searching, *JACM 28,* pp. 737-751, 1982.

[7] S. S. Godbole, On Efficient Computations of Matrix Chain Products, *IEEE Transactions on Computers ∈22: 9,* pp. 864-866, 1973.

[8] P. A. V. Hall, Equivalence between AND/OR Graphs and Context-Free Grammars, *Comm. ACM 16,* pp. 444-445, 1973.

[9] P. Helman, An Algebra for Search Problems and Their Solutions, in *Search in Artificial Intelligence,* ed. Kanal and Kumar, Springer-Verlag, 1988.

[10] J. Hopcroft and J. Ullman, *Introduction to the Theory of Languages,*

Addison Wesley, 1979.

[11] T. Ibaraki, Solvable Classes of Discrete Dynamic Programming, *J. Math. Analysis and Applications 43,* pp. 642-693, 1973.

[12] T. Ibaraki, The Power of Dominance Relations in Branch and Bound Algorithms, *J. ACM 24,* pp. 264-279, 1977.

[13] T. Ibaraki, Branch-and-Bound Procedure and State-Space Representation of Combinatorial Optimization Problems, *Inform and Control 36,* pp. 1-27, 1978.

[14] R. M. Karp and M. H. Held, Finite-State Processes and Dynamic Programming, *SIAM J. Appl. Math 15,* pp. 693-718, 1967.

[15] D. E. Knuth, *The Art of Computer Programming-Volume 1,* Addison-Weseley Publ. Co., 1968.

[16] D. E. Knuth, Optimal Binary Search Trees, *Acta Informatica 1,* pp. 14-25, 1971.

[17] D. E. Knuth and R. W. Moore, An Analysis of Alpha-Beta Pruning, *Artificial Intelligence 6,* pp. 293-326, 1975.

[18] V. Kumar, A Unified Approach to Problem Solving Search Procedures, Ph.D. thesis, Dept. of Computer Science, University of Maryland, College Park, December, 1982.

[19] V. Kumar, A General Bottom-up Procedure for Searching And/Or Graphs., *Proc. National Conference on Artificial Intelligence (AAAI-84),* Austin, Texas, pp. 182-187, 1984.

[20] V. Kumar, Branch-and-Bound Search, pp. 1000-1004 in *Encyclopedia of Artificial Intelligence,* ed. S.C. Shapiro, Wiley-Interscience, 1987.

[21] V. Kumar and L. N. Kanal, Parallel Branch and Bound Formulations

for And/Or Tree Search, *IEEE Trans. on Pattern Analysis and Machine Intelligence,* November 1984.

[22] V. Kumar, D. S. Nau, and L. N. Kanal, A General Branch-and-Bound Formulation for AND/OR Graph and Game Tree Search., in *Search in Artificial Intelligence,* ed. Kanal and Kumar, Springer-Verlag, 1988 .

[23] E. L. Lawler and D. E. Wood, Branch-and-Bound Methods: A Survey, *Operations Research 14,* pp. 699-719, 1966.

[24] A. Martelli and U. Montanari, Additive AND/OR Graphs, *Proc. Third Internat. Joint Conf. on Artif. Intell.,* pp. 1-11, 1973.

[25] A. Martelli and U. Montanari, Programmazione dinamica e punto fisso., *Proc. Convegno di Informatica Teorica,* Mantova, Italy, pp. 1-19, 1974.

[26] T. L. Morin and R. E. Marsten, Branch and Bound Strategies for Dynamic Programming, *Operations Research 24,* pp. 611-627, 1976.

[27] N. Nilsson, *Problem-Solving Methods in Artificial Intelligence,* McGraw-Hill Book Company, New York, 1971.

[28] N. Nilsson, *Principles of Artificial Intelligence,* Tioga Publ. Co., Palo Alto. CA, 1980.

[29] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving,* Addison-Wesley, 1984.

[30] G. C. Stockman, A Minimax Algorithm Better than Alpha-Beta?, *Artificial Intelligence 12,* pp. 179-196, 1979.

[31] D. J. White, *Dynamic Programming,* Holden-Dey, San Francisco, CA, 1969.