THE COMPOSITE DECISION PROCESS: A UNIFYING FORMULATION FOR HEURISTIC SEARCH, DYNAMIC PROGRAMMING AND BRANCH & BOUND PROCEDURES+

Vipin Kumar*
Laveen Kanal**

*Department of Computer Science, University of Texas at Austin, Austin, TX
**Department of Computer Science, University of Maryland, College Park, MD

ABSTRACT

In this short paper we present a brief exposition of a composite decision process - our unifying formulation of search procedures - which provides new insights concerning the relationships among heuristic search, dynamic programming and branch and bound procedures.

1. Introduction

Various heuristic procedures for searching And/Or graphs, game trees, and state space representations has appeared in the A.I. literature over the last few decades, and at least some of them have been thought to be related to dynamic programming (DP) and branch and bound (B&B) procedures of Operations research (O.R.). But the relationships between these classes of procedures have been rather contoversial.

For example, Pohl argues in [22] that heuristic search procedures are very different from B&B procedures, whereas Hall [5] and Ibaraki [8,10] claim that many heuristic procedures for searching state space representations are essentially B&B procedures. Knuth does not consider the alpha-beta game tree search algorithm to be a B&B procedure; he considers its less efficient version (called F1 in his classical treatment of alpha-beta [14] to be branch and bound. But, Reingold et al. [23] consider alpha-beta to be a type of B&B. While describing the algorithm HS (which is the same as the AO* And/Or graph search algorithm [21]) in [18], Martelli and Montanari state that their algorithm is different from B&B because "(B&B) technique does not recognize the existence of identical subproblems. But Ibaraki's B&B procedure [10] does recognize the existence of identical subproblems. While describing the heuristic search procedure A* for finding a shortest path in a state space, Nilsson [20] considers dynamic programming to be essentially a breadth-first search method. However, Dreyfus & Law [4] show that Dijkstra's algorithm for the shortest path [2], an algorithm very similar to A*, can be viewed as a dynamic programming algorithm. Morin & Marsten [19] permit DP computations to be augmented with bounds, which means that they do not consider it necessary that DP computations be breadth-first.

The relationship between B&B and dynamic programming techniques has also been rather controversial. Kohler [15] and Ibaraki [9] discuss how a number of dynamic programming procedures can be stated in the framework of B&B. Morin & Marsten [19] consider some classes of B&B procedures as dynamic programming procedures augmented with some bounding operations. Ibaraki's work [7,10] seems to imply that dynamic programming is a more general problem solving scheme than B&B for solving discrete optimization problems. Smith [24] presents a k-adic problem reduction system model as a model for optimization problems and considers, in that context, dynamic programming to be a bottom up approach, and B&B to be a top down procedure.

We have developed a methodology whereby most of these procedures can be viewed in a unified manner [17]. The scheme reveals the true nature of these procedures, and clarifies their relationships to each other. The methodology also aids in synthesizing (by way of analogy, suggestions, induction, etc.) new variations as well as generalizations of the existing search procedures.

In the rest of this short paper, we present a brief exposition of our unified approach to search procedures and discuss how it unveils the true nature and interrelationships of these procedures.

2. A Unified Approach

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: find a least cost (or largest merit) element of a given set X. In most of the problems of interest, X is 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 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

+Paper presented at the 1983 AAAI meeting; Proc. American Assoc. for A.I., Wash., DC, Aug. 83.

Research partially supported by NSF grants ECS-78-22159 and MCS-81-17391.

not suprising that the formal techniques developed look very different from each other, even though they are being used for similar purposes.

We have developed the concept of composite decision process (defined below) as a general model for formulating 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. We have also developed systematic procedures for exploiting such knowledge for the efficient discovery of an optimum element of X, and shown that many of the existing search procedures are special cases of our procedures.

2.1 Composite Decision Processes

A *composite decision process* (CDP) C = (G,t,c) is a 3-tuple where G = (V,N,S,P) is a context-free grammar (V,N,P, and S are respectively the sets of terminal symbols, nonterminal symbols, productions and the start symbol), t denotes the set of cost attributes associated with productions of G (a real valued k-ary cost attribute tp(.,...,.) is associated with each production p = "w → w1 ... wk" of G), and c is a real valued cost function defined over the set of terminal symbols of G.

The set of parse trees rooted at the start symbol S of G represents the discrete set X of the optimization problem formulated by C. A cost f(T) is assigned to each parse tree T of G in terms of the set of cost attributes t and the function c. We first define a real valued function c_T over the nodes n of a parse tree T of G as follows:

(i) If n is a terminal symbol then

(1.a) $c_T(n) = c(n)$.

(ii) If n is a nonterminal symbol and n1,...,nk are descendents (from left to right) of n in T (implying that G has a production p = "n → n1 ... nk") then

(1.b) $c_T(n) = tp(c_T(n1),\ldots,c_T(nk))$.

If n is the root symbol of a parse tree T then we define

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

For a node of a parse tree T, $c_T(n)$ denotes the cost of the subtree of T rooted at n. For a production p: $n \to n_1 \ldots n_k$, $t_p(x_1,\ldots,x_k)$ denotes the cost of a derivation tree T rooted at n if the costs of the subtrees of T rooted at $n_1,\ldots,n_k$ are $x_1,\ldots,x_k$. Thus the cost of a parse tree is recursively defined in terms of the costs of its subtrees. See Fig. 1 for an illustration. 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 $f(T^*) = \min\{f(T) \mid T \text{ is a parse tree rooted as } S\}$.

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

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: $n \to n_1 \ldots n_k$ are monotonically nondecreasing in each variable, i.e.,

$$x_1 \leq y_1 \ \& \ \ldots \ \& \ x_k \leq y_k$$

$$\Longrightarrow$$

$$t_p(x_1,\ldots,x_k) \leq tp(y_1,\ldots,y_k).$$

For a symbol n of the grammar G, let c*(n) denote the minimum of the costs of the parse trees rooted at n. The following theorem proved in [17] establishes that for a monotone CDP, c*(n) can be defined recursively in terms of $\{c^*(m) \mid m \text{ is a part of a string directly derivable from } n\}$.

Theorem 1: For a monotone composite decision process C = (G,t,c), the following recursive equations hold.

(3a) If n is a nonterminal symbol then

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

(3b) If n is a terminal symbol then

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

For many interesting special cases there exist efficient algorithms which solve these equations to compute c*(S), the smallest of the costs of the parse trees rooted at S [17]. 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.

Relationships 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 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 1: For a monotone composite decision process C = (G,t,c), 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 [17].

This statement is somewhat different (in fact "weaker") than Bellman's principle of optimality. Nevertheless, Lemma 1 guarantees that an optimal parse tree can always be built by optimally choosing from the alternate 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 → $n_1 \dots n_k$ are strictly increasing in each variable, i.e., $x_i<y_i$ and $x_j<y_j$ for $j\neq i$ and $1\leq j\leq k$ ===> $t_p(x_1,\dots,x_k) < t_p(y_1,\dots,y_k)$.

Lemma 2: For a strictly monotone CDP C = (G,t,c), all the subtrees of an optimal parse tree rooted at a symbol n of G are also optimal.

Proof: See [17].

Relationships with Sequential Decision Processes

If the context-free grammar G of a CDP C = (G,t,c) is further restricted to be regular then we can use the direct correspondence between regular grammars and finite state automata to show that C is essentially a sequential decision process (SDP). The concept of SDP was introduced by Karp [13] as a model for the problems solved by dynamic programming.

The concept of a sequential decision process has been extensively studied in various areas in different guises. State space descriptions used in Artificial Intelligence to solve various problems are essentially sequential decision processes. The minimization problem of a SDP is essentially a generalized version of the well known shortest path problem studied extensively in the Operations Research literature [3]. Various Branch & Bound, dynamic programming and heuristic search procedures have been developed for problems which can be modeled by SDPs (e.g., [7], [10], [4], [20]). Generalized versions of many of these procedures are also applicable to problems modeled by CDPs (see [17]). In fact we came up with the concept of a CDP as a generalization of the concept of an SDP.

The Scope of the CDP Formulation

The concept of composite decision process is very important. In addition to the problems modeled by SDPs, it models a large number of problems in A.I. 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 A.I. applications to model problem reduction schema [6]. Due to this correspondence, the specification of a problem by an And/Or graph can often be viewed as its specification in terms of a 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, the problem of finding the minimax value of a game tree can also be represented in the CDP formulation. Furthermore, many other important optimization problems such as the problem of constructing an optimal decision tree, constructing an otpimal binary search tree, finding an optimal sequence for matrix multiplication can be naturally formulated in terms of CDPs (see [17]).

2.2 Solving the Minimization Problem

We have shown in [17] 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 as S can be provably identified by generating and evaluating only a finite number of parse trees of G, even though G may generate infinite number of 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 the simple enumeration.

The Generalized Dynamic Programming Technique

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 reverse 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. The term "bottom up" is quite suggestive of this theme and is often used for many search procedures which are special cases of the technique discussed above. 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 have named these bottom up procedures for minimization of CDPs as dynamic programming.

In [17] we have presented dynamic programming procedures to solve the minimization problem of the three classes of CDPs. Interestingly, 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 Montanati's bottom up search algorithm for constructing an optimal solution tree of an Acyclic And/Or graph, and many other optimization algorithms (usually termed as dynamic programming algorithms) can be considered as special cases of these procedures.

The Generalized Branch-and-Bound Technique

In this second technique, we start with some representation of the total (possible 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* [27], 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* [17].

3. Concluding Remarks

The generalized versions of DP and B&B (for solving the minimization problems of CDPs) provide a unifying framework for a number of heuristics search procedures. In particular, the B&B formulation for searching acyclic And/Or graphs has helped unveil the close relationship of alpha-beta with SSS*, showing that if a minor modification is made in the B&B formulation of SSS*, the resulting procedure is equivalent to alpha-beta (see [17,16]). This is most interesting, for alpha-beta as conventionally presented [14] appears very different from SSS* as described by Stockman [27]. Considering that alpha-beta has been known for over twenty years, it is noteworthy that SSS* was discovered only recently in the context not of game playing, but of a waveform parsing system [26]. Perhaps if an adequate B&B formulation for alpha-beta had been available earlier, SSS* would have been developed as a natural variation of alpha-beta. The B&B formulation also makes it easy to visualize many variations and parallel implementations of SSS* presented in [17,11,12]. In [17] we also proved the correctness of a general procedure for searching acyclic And/Or graphs. This greatly simplifies the correctness proofs of algorithms such as AO*, SSS*, and B*, since these procedures are special cases of the general procedure.

We have considered B&B and DP as two distinct ways (a top down search procedure and a bottom up search procedure) of solving the minimization problem of a CDP. However, it turns out that for an important subset of the problems formulated by the CDP, the class of DP algorithms becomes indistinguishable from the class of B&B algorithms (see [17]). This explains why DP and B&B algorithms for several optimization problems were thought to be related, and why B&B procedures of Ibaraki [10] for solving the minimization problem of a SDP could also be viewed as DP computations.

The general search procedures discussed in this paper make use of two types of information to efficiently find an element of a discrete set X - "syntactic" and "semantic". The syntactic information is present in the representation of X (e.g., by a context-free grammar, regular grammar, etc.), and is used in DP in the form of principle of optimality, and in B&B in the form of a dominance relation. But the only semantic information used in these procedures is in the form of heuristics or bounds associated with subproblems or subsets of X. It should be interesting to investigate what other types of problem specific knowledge can be integrated into these search procedures or their variations. We conjecture that such investigation will also be of help in improving problem solving search procedures which are not necessarily used for optimization problems.

Context-Free Grammar

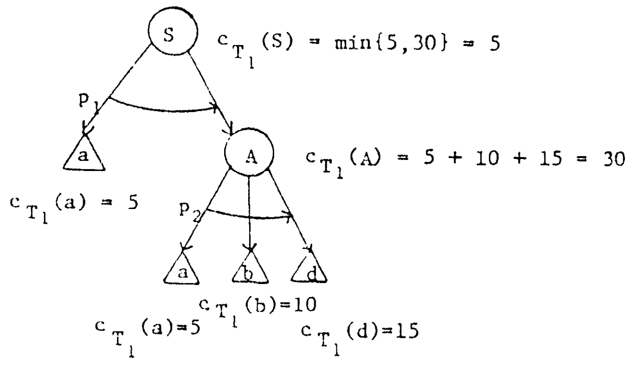
$$G = (\overset{U}{\{a,b,d\}},\overset{N}{\{S,A\}},S,P)$$

	Productions:	Cost Attributes:
	$p_1: S \to aA,$	$t_{p_1}(r_1,r_2) = \min r_1,r_2$,
P:	$p_2: A \to abd,$	$t_{p_2}(r_1,r_2,r_3) = r_1+r_2+r_3,$
	$p_3: A \to ad,$	$t_{p_3}(r_1,r_2) = r_1+r_2$
	$p_4: S \to aS,$	$t_{p_4}(r_1,r_2) = r_1+r_2$

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

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

Figure 1



1(b) The derivation tree T_1 depicting derivation of aabd from S:

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

Figure 1 con't.

REFERENCES

[1] Berliner, H., The B* Tree Search Algorithm: A Best-First Proof Procedure, Artificial Intelligence 12, pp. 23-40, 1979.

[2] Dijkstra, E.W., A Note on Two Problems in Connection with Graphs, Numer. Math. 1, pp. 269-271, 1959.

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

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

[5] Hall, P.A.V., Branch-and-Bound and Beyond, Proc. Second Int'l. Joint Conf. on Artificial Intelligence, pp. 641-658, 1971.

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

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

[8] Ibaraki, T., Theoretical Comparison of Search Strategies in Branch and Bound, Int'l. Journal of Computer and Information Science, 5, pp. 315, 344.

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

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

[11] Kanal, L. and Kumar, V., Parallel Implementations of a Structural Analysis Algorithm, Proc. IEEE Conf. Pattern Recognition and Image Processing, pp. 452-458, Dallas, August 1981.

[12] Kanal, L.N. and Kumar, V., A Branch and Bound Formulation for Sequential and Parallel Game Tree Search, Proc. 7th Int'l. Joint Conf. on A.I., pp. 569-571, Vancouver, August 1981.

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

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

[15] Kohler, W.H. and Steiglitz, K., Characterization and Theoretical Comparison of Branch and Bound Algorithms for Permutation Problems, J-ACM 21, pp. 140-156, 1974.

[16] Kumar, V. and Kanal, L., A General Branch and Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures, Artificial Intelligence 21, pp. 179-197, 1983.

[17] Kumar, V., A Unified Approach to Problem Solving Search Procedures, Ph.D. Dissertaion, Univ. of Maryland, College Park, 1982.

[18] Martelli, A. and Montanari, U., Optimizing Decision Trees Through Heuristically Guided Search, Comm. ACM 21, pp. 1025-1039, 1978

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

[20] Nilsson, N., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

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

[22] Pohl, I., Is Heuristic Search Really Branch and Bound?, Proc. 6th Annual Princeton Conf. Infor. Sci. and Systems, pp.370-373, 1972

[23] Reingold, E., Nievergelt, J. and Deo, N., Combinatorial Optimization, Prentice-Hall, 1977.

[24] Smith, D.R., Representation of Discrete Optimization Problems by Dynamic Programs, Tech. Rep. NPS 52-80-004, Naval P.

[25] Smith, D.R., Problem Reduction Systems, unpublished report, 1981.

[26] Stockman, G.C. and Kanal, L., Problem-Reduction Representation for the Linguistic Analysis of Waveforms, IEEE Trans. PAMI 5, 3, May 1983.

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