General Branch and Bound, and Its Relation to A* and AO*

Dana S. Nau, Vipin Kumar and Laveen Kanal[†]
Laboratory for Pattern Analysis, Computer Science Department, University of Maryland College Park, MD 20742, U.S.A.

Recommended by Erik Sandewall

ABSTRACT
Branch and Bound (B&B) is a problem-solving technique which is widely used for various problems encountered in operations research and combinatorial mathematics. Various heuristic search procedures used in artificial intelligence (AI) are considered to be related to B&B procedures. However, in the absence of any generally accepted terminology for B&B procedures, there have been widely differing opinions regarding the relationships between these procedures and B&B. This paper presents a formulation of B&B general enough to include previous formulations as special cases, and shows how two well-known AI search procedures (A and AO*) are special cases of this general formulation.*

1. Introduction

A wide class of problems arising in operations research, decision making and artificial intelligence can be (abstractly) stated in the following form:

> Given a (possibly infinite) discrete set X and a real-valued *objective function* F whose domain is X, find an *optimal element* $x^* \in X$ such that $F(x^*) = \min\{F(x) \mid x \in X\}$.[1]

Unless there is enough problem-specific knowledge available to obtain the optimum element of the set in some straightforward manner, the only course available may be to enumerate some or all of the elements of X until an optimal element is found. However, the sets X and $\{F(x) \mid x \in X\}$ are usually

[†]This work was supported by NSF Grant ENG-7822159 to the Laboratory for Pattern Analysis at the University of Maryland.

[1]In some applications, the maximal element (i.e., x^* such that $F(x^*) = \max\{F(x) \mid x \in X\}$) is desired rather than the minimal element.

Artificial Intelligence **23** (1984) 29–58
0004-3702/84/$3.00

defined in such a way that their elements are not readily available, but instead require some computation to be generated. Thus, for problems of practical interest, exhaustive enumeration is often prohibitively time-consuming.

There are many examples of this kind of problem in AI. For example, the set X may be the set of solutions to an And/Or graph, game tree, or state-space search problem. While various comments have been made regarding the relationships of B&B procedures to heuristic search procedures, the comments have often been contradictory. Whereas Hall [2] and Ibaraki [4, 6] consider B&B and heuristic search procedures to be very similar, Pohl [22] does not agree. Similar differences in assessment may be found in Reingold et al. [23] and Knuth [7] concerning alpha-beta and B&B, in Kumar and Kanal [13] and Berliner [1] concerning B*, and in Ibaraki [6] and Martelli and Montanari [16] concerning AO*. A plausible explanation for some of the confusion concerning the relationships between B&B and AI search procedures is that B&B techniques have continued to evolve since the early 1960s, whereas the early survey by Lawler and Wood [14] is often the only reference used in the AI literature. Additional confusion results, however, from other factors as described below.

One reason for the present confusion about B&B is the differing conceptions of the basic entities operated on by B&B. For example, the characterization of B&B presented by Kohler and Steiglitz [8] is developed in the context of permutation problems and uses "partial combination vectors" as the basic entities. The one presented by Ibaraki [6] attempts to serve as a model for state-space search procedures, and formulates B&B as a procedure operating on strings over an alphabet. In Reingold, Nievergelt, and Deo [23], the basic entities are partial solution vectors.

In our opinion, much of the above confusion can be resolved by making a distinction between the entities operated on by B&B and the structures used to represent these entities. We argue that the fundamental entities operated upon by a B&B procedure are not the partial vectors, alphabetic strings, or other problem-specific entities. These are merely *representations* of subsets of the set of solutions X mentioned at the beginning of this section, and confusion results from the lack of distinction between the entity and its representation.

In this paper, we unify the various previous formulations of B&B by considering B&B as a procedure which operates on members of an arbitrary set of representations of subsets of X. These representations can be vectors, strings, paths in a graph, solution trees, or any other convenient structures.

Another source of confusion about B&B is that different writers define B&B to have different and non-equivalent pruning criteria. Until recently, the only test used for pruning in B&B was based on upper and lower bounds on subsets of X (e.g., Mitten [17]); hence the name Branch and Bound. The introduction of more sophisticated pruning techniques seems to have been initiated by Kohler and Steiglitz [9] with their concept of *dominance*, although similar ideas have been used heuristically in many other branch and bound algorithms

(Ibaraki [5] gives a long list). The formal introduction of dominance in B&B procedures was a major step towards formalizing the use of problem-specific knowledge in optimization procedures. Kohler and Steiglitz [9] and Ibaraki [6] have given a formal description of B&B with dominance. They have proved several results using this formulation.

To unify the varying conceptions of B&B pruning techniques, we allow any generated set to be pruned if it can be shown that at least one of the remaining sets contains an optimal element. A look at the references cited above will show that our approach to pruning is simpler than previous formulations while including them as special cases of our approach.

Using our formulation of B&B, we show that A* and AO* are special cases of B&B. A number of other search procedures (e.g., alpha-beta [20], SSS* [25], and B* [1]) can also be considered special cases of our general formulation; this is shown in another paper [12]. This general formulation can be used to shed light on the similarities and differences among these various search procedures. This topic is investigated further in [10].

Section 2 describes the general formulation, which we call General Branch and Bound (GBB). This section discusses the basic concepts of B&B, the necessity of distinguishing between a set and its representation, the use of auxiliary data in B&B, and a fundamental property of B&B. Section 3 discusses an important special case of GBB. This special case is used in subsequent sections to show that other procedures are special cases of GBB. Section 5 discusses how ordinary B&B (in which all pruning is done by means of upper and lower bounds) may be considered a special case of GBB, and Section 6 does the same thing for B&B with dominance. Sections 7 and 8 show that A* and AO*, respectively, are GBB procedures. Section 9 contains concluding remarks. Appendices A, B, C, and D contain proofs of results discussed in Sections 3, 4, 6, and 7, respectively.

2. A General Formulation of Branch and Bound

2.1. The basic concept of branch and bound

Consider the procedural scheme below (comments are indicated by double slashes ('//')):

```
procedure P0:
1.  ACT := {X}   //ACT is the current active set//
2.  loop   //the main loop//
3.     if ACT = {Z} for some Z and Z is a singleton {z} then
4.        return z
5.     endif
6.     SEL := select (ACT)
       //select some of the sets in ACT//
7.     SPL := split(SEL)   //split the sets in SPL//
```

```
8.    ACT := prune((ACT − SEL) ∪ SPL)
      //remove the selected sets from ACT, replace//
      //them by the newly generated sets, and then //
      //prune unneeded sets from ACT//
9.  repeat
end P0
```

– ACT, the *active set*, is a collection of subsets of X.
– select, the *selection function*, is any function which returns a collection SEL $\subseteq$ ACT. The domain of select is the set of all possible values which ACT might have at line 6 of P0.
– split, the *splitting function*, has as its domain the set of all possible values which the collection SEL might have at line 7 of P0. split(SEL) returns a collection SPL of subsets of X such that:

(1) every set in SPL is a subset of some set in SEL;

(2) $\bigcup\{Y' \mid Y' \in \text{SPL}\} = \bigcup\{Y \mid Y \in \text{SEL}\}$; i.e., the sets in SPL contain precisely those elements which are members of SEL.

– prune, the *pruning function*, has as its domain the set of all possible values which the collection of sets $R = (\text{ACT} - \text{SEL}) \cup \text{SPL}$ might have at line 8 of P0. prune returns a collection of sets $R' \subseteq R$ such that

$$\min\{F(y) \mid y \in Y \text{ for some } Y \in R'\} = \min\{F(y) \mid y \in Y \text{ for some } Y \in R\};$$

i.e., at least one of the minimum elements of R is also present in R'.

The procedure P0 describes our basic concept of B&B. We note that the selection and splitting functions of P0 are essentially those of Mitten [17]. Most of versions of B&B familiar to the reader (such as Ibaraki's formulation [6]) will select and split only one member of ACT at a time (see Sections 4 and 5). Our use of the capability of selecting and splitting several members of ACT at once turns out to be essential for explaining procedures which operate on And/Or graphs or game trees (which are special cases of And/Or graphs). Such procedures include AO*, which is discussed in Section 7, and SSS* and B*, which are discussed in [12].

The pruning function in P0 is more general than that of Mitten [17], which allows pruning only by bounding. Our pruning function is conceptually simpler than the formulation of pruning used by Ibaraki [6], and includes Ibaraki's formulation as a special case (see Section 5).

2.2. Inadequacies of P0

Despite the generality of P0, it is not adequate to describe the behavior of all B&B procedures. (This is also true of all other B&B formulations which use abstract sets, e.g., [17, 24].) In practical implementations of B&B procedures, a subset Y of X is usually not given a direct or explicit representation (such as a

list of its elements), but instead is usually represented by a data structure from which the elements of Y can be obtained by computation. This representation often incorporates problem-specific knowledge which is used in selection, splitting, and pruning. Different choices of representation for a problem can lead to different B&B procedures.

Even within a single B&B procedure, there may be several different ways to represent the same set. Depending on which representations are used for some collection of sets $\{Y_1, Y_2, \ldots, Y_k\}$, the values returned by select($\{Y, Y_2, \ldots, Y_k\}$), split($\{Y_1, Y_2, \ldots, Y_k\}$), and prune($\{Y_1, Y_2, \ldots, Y_k\}$) may vary. This means that if select, split, and prune are considered as functions only of $Y_1, Y_2, \ldots, Y_k$, then they are not well-defined. Furthermore, it may not be readily apparent whether a data structure representing a subset of X represents a singleton set $Z = \{z\}$ or not, and thus the result of the termination test in line 3 of P0 may not be well-defined.

As an example, we consider the least-cost path problem. Let G be a directed graph, and (m, n) be an arc in G. Then m is called a *parent* of n, and n is called a *child* of m. If P is a path in G, then the last node in P is denoted by tip(P). Suppose that $P = (n_1, n_2, \ldots, n_j)$ and $Q = (n_j, n_{j+1}, \ldots, n_k)$ are paths in G, and (n_j, n) is an arc in G. Then Pn is the path $(n_1, n_2, \ldots, n_j, n)$, and PQ is the path $(n_1, n_2, \ldots, n_j, n_{j+1}, \ldots, n_k)$.

Suppose that each arc (m, n) in G has a cost $c(m, n) \geq 0$, and that for every path P in G, cost(P) is defined as the sum of the arc costs of P. Consider the problem of finding a path from a *source node* s in G to any member of a set T of *terminal nodes*. The set X of solutions to this problem is the set containing each path from s to any member of T. The *least-cost path problem* is the problem of finding a path P in X which minimizes the value of the objective function cost(P).

In B&B procedures to solve the least-cost path problem, a path P from s to some node n in G is typically used to represent the set of all *extensions* of P to members of T (i.e., the set of all paths PQ such that Q is a path from tip(P) to a member of T and the only member of T in Q is tip(Q)). The splitting function split is typically defined in the following way as a function of collections of representations of sets (rather than collections of sets):

$$\text{split}(\{P_1, P_2, \ldots, P_k\}) = \{P_i n \mid 1 \leq i \leq k \text{ and } n \text{ is a child of tip}(P_i)\}. \tag{2.1}$$

Let G be the directed graph having node set $\{a, b, c, d, e, f, g\}$ and arc set $\{(a, b), (b, c), (c, d), (d, e), (c, f), (f, g), (g, h)\}$, and let $s = a$ and $T = \{e, g\}$ (see Fig. 1). If split is defined as in (2.1), then

$$\text{split}(\{(a, b)\}) = \{(a, b, c)\} \tag{2.2}$$

and

$$\text{split}(\{(a, b, c)\}) = \{(a, b, c, d), (a, b, c, f)\}. \tag{2.3}$$

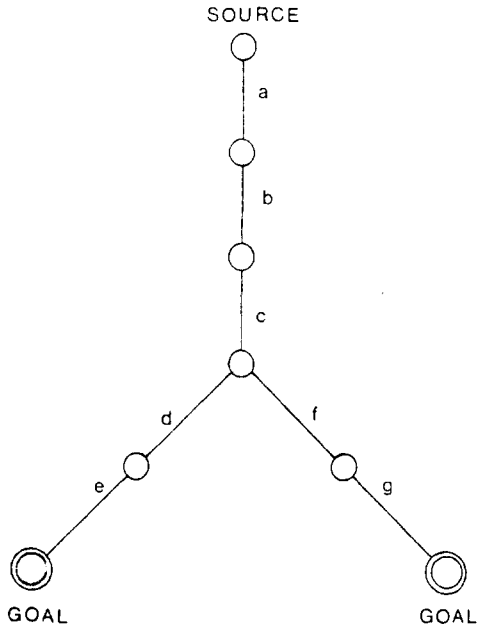

FIG. 1.

Suppose split were defined (as it is in P0) as a function of the sets represented rather than a function of the representations. The paths (a) and (a, b) both represent $\{(a, b, c, d, e), (a, b, c, f, g)\}$; (a, b, c, d) represents $\{(a, b, c, d, e)\}$; and (a, b, c, f) represents $\{(a, b, c, f, g)\}$. Thus (2.2) would be replaced by

$$\begin{aligned}&\text{split}(\{\{(a, b, c, d, e), (a, b, c, f, g, h)\}\}) = \\ &\quad = \{\{(a, b, c, d, e), (a, b, c, f, g, h)\}\},\end{aligned} \tag{2.2}'$$

and (2.3) would be replaced by

$$\begin{aligned}&\text{split}(\{\{(a, b, c, d, e), (a, b, c, f, g, h)\}\}) = \\ &\quad = \{\{(a, b, c, d, e)\}, \{(a, b, c, f, g, h)\}\}.\end{aligned} \tag{2.3}'$$

Since (2.2)′ and (2.3)′ assign two different values to split($\{\{(a, b, c, d, e), (a, b, c, f, g, h)\}\}$), split is ill-defined when considered as a function of the sets represented rather than their representations. This demonstrates that P0 is inadequate to describe the behavior of the splitting functions used in some B&B procedures. Similar examples can be found for the selection and pruning functions.

This example also illustrates that the termination test used in P0 does not adequately model the termination tests used in practical B&B procedures.

Suppose ACT were a collection containing the single path (a, b, c, d, f, g). This path represents the singleton set $\{(a, b, c, d, f, g, h)\}$. However, a B&B procedure for the least-cost path problem could not terminate before splitting $\{(a, b, c, d, f, g)\}$ to obtain $\{(a, b, c, d, f, g, h)\}$. This is because the tip node of the path (a, b, c, d, e, f, g) is not in T, and so it is not known whether (a, b, c, d, e, f, g) represents a single path (or more than one path, or any path at all) between s and some node in T. In order for the termination test in P0 to be well-defined, a goal function is needed to tell whether or not a representation is known to represent a singleton set $\{y\}$ such that $F(y)$ may be obtained directly.

2.3. An improved model of branch and bound

The previous section described the inadequacies of P0 as a general model of B&B. To eliminate these inadequacies, P0 is modified in this section to contain both a goal function and an explicit distinction between representations of sets and the sets represented.

A representation scheme is defined as a pair (S, rf), where S is a set of *representations* and $\text{rf} : S \to 2^X$ is a *representation function*. If $r \in S$ and $\text{rf}(r) = Y$, we say that r is an rf-representation of Y (or, if the identity of rf is obvious, that r is a representation of Y).

Throughout this paper, r denotes a representation and R denotes a collection of representations. For convenience, we define

$$F_{\min}(r) = \min\{F(x) \mid x \in \text{rf}(r)\}$$

and

$$F_{\min}(R) = \min\{F(x) \mid x \in \text{rf}(r) \text{ for some } r \in R\}.$$

A *goal function*, goal, is any predicate such that whenever goal(r) holds, rf(r) is a singleton set $\{x\}$ such that $F(x)$ can be computed directly from r.[2] Thus when goal(r) holds, $F_{\min}(r) = F(x)$. When goal(r) does not hold, it does not necessarily mean that rf(r) is not a singleton set.

When P0 is modified to include a goal function and to make explicit use of representations of sets, the result is the procedure P1 below.

procedure P1: //General Branch and Bound (GBB)//
1. ACT1 := {r0} //ACT1 is the current active set//
   //r0 is the initial representation of X; i.e., rf1(r0) = X//
2. loop //the main loop//

[2]It is sometimes useful to have goal(r) hold even if r does not represent a singleton. However, it is usually possible to define rf in such a way that all goals are singletons (as we have done for the search problems discussed in this paper). The modification of our model to include non-singleton goals is fairly straightforward but would require additional notation.

```
3.    if ACT1 = {r1} for some r1 and goal1(r1) holds then
4.      return r1
5.    endif
6.    SEL1 := select1(ACT1)
7.    SPL1 := split1(SEL1)
8.    ACT1 := prune1((ACT1 - SEL1), SPL1)
9.  repeat
end P1
```

In P1 above, rf1 is a representation function and ACT1 is a set of representations. The other functions, which are defined below, are analogous to the corresponding functions for P0 except that they refer to representations instead of sets.

– select1, the *selection function*, is any function which returns a subset of ACT1. The domain of select1 is the set of all possible values which ACT1 might have at line 6 of P1.

– split1, the *splitting function*, has as its domain the set of all possible values which SEL1 might have at line 7 of P1. split1(SEL1) returns a collection SPL1 of representations such that

$$\text{if } r' \in \text{SPL1, then there is an } r \in \text{SEL1}$$
$$\text{such that } \text{rl1}(r') \subseteq \text{rf1}(r) \tag{2.4}$$

and

$$\bigcup \{\text{rf1}(r) \mid r \in \text{SPL1}\} = \bigcup \{\text{rf1}(r) \mid r \in \text{SEL1}\}. \tag{2.5}$$

– prune1, the *pruning function*, has as its domain the set of all possible values which the pair of collections of representations (ACT1 – SEL1, SPL1) might have at line 8 of P1. prune1 returns a collection ACT1′ of representations such that

$$\{\text{rf1}(r') \mid r' \in \text{ACT1}'\} \subseteq \{\text{rf1}(r) \mid r \in (\text{ACT1} - \text{SEL1}) \cup \text{SPL1}\} \tag{2.6}$$

and

$$F_{\min}(\text{ACT1}') = F_{\min}((\text{ACT1} - \text{SEL1}) \cup \text{SPL}). \tag{2.7}$$

Most B&B procedures make use of auxiliary information not explicitly represented in P1. The auxiliary information, for example, might be information about various relationships among the members of ACT2 for use in pruning, or might take the form of maintaining ACT2 as an ordered list rather than an unordered set. If such information is made an explicit part of P1, the result is the procedure P2 given below.

```
procedure P2: //GBB with an auxiliary database (DB2)//
1. initialize DB2
   //DB2 consists of all auxiliary information used by P2//
2. ACT2 := {r0}
   //r0 is the initial representation of X//
```

```
 3.  loop  //the main loop//
 4.    if ACT2 = {r2} for some r2 and goal2(r2,DB2) holds then
 5.      return r2
 6.    endif
 7.    SEL2 := select2(ACT2,DB2)
 8.    (SPL2,DB2) := split2(SEL2,DB2)
 9.    (ACT2,DB2) := prune2(ACT2 - SEL2,SPL2,DB2)
10.  repeat
end P2
```

In P2, rf2 is a representation function. Except for the use of the auxiliary database DB2, the properties of select2, split2, and prune2 are identical to those of select1, split1, and split1.

In practice, nearly every B&B procedure makes use of auxiliary information, and hence has more in common with P2 than P1. However, it would be quite cumbersome to refer to this information explicitly every time such procedures are discussed. Thus we write:

(1) goal2($r2$) for goal2($r2$,DB2);
(2) select2(ACT2) for select2(ACT2,DB2);
(3) SPL2 = split2(SEL2) for (SPL2,DB2') = split2(SEL2,DB2);
(4) ACT2' = prune2(ACT2 − SEL2,SPL2) for (ACT2',DB2') = prune2(ACT2 − SEL2,SPL2,DB2).

From now on, when we speak of representation, goal, selection, splitting, and pruning functions, we mean functions having the properties of rf2, goal2, select2, and prune2, respectively.

2.4. A fundamental property

Each of the procedures discussed in this paper has a loop labeled 'the main loop' (e.g., lines 2–9 of P2). If V is a variable used by one of these procedures, then the value of V at the end of the ith iteration of the main loop is denoted by V^i. For example, let P2 be called with some initial representation $r0$ and instantiations of rf2, goal2, select2, split2, and prune2, and let t be the number of times the main loop is fully executed before a return occurs. (Thus, if the test at line 3 succeeds and P2 returns during the ith iteration of the loop, then $t = i - 1$; and if P2 never returns, then $t = \infty$.) We have $ACT2^0 = \{r0\}$, and for $0 \leq i < t$,

$$ACT2^{i+1} = \text{prune2}((ACT2^i - SEL2^{i+1}), SPL2^{i+1}). \tag{2.8}$$

We now present a correctness proof for P2. This theorem and its corollary are similar to results proved for previous formulations of B&B.

Theorem 2.1. *For every integer i such that $0 \leq i < t$, at least one of the optimal*

elements of X represented in $\mathrm{ACT2}^i$ *is also represented in* $\mathrm{ACT2}^{i+1}$; *i.e.*,

$$F_{\min}(\mathrm{ACT2}^{i+1}) = F_{\min}(\mathrm{ACT2}^i).$$

Proof.

$$\begin{aligned} F_{\min}(\mathrm{ACT2}^{i+1}) &= \min(\mathrm{prune2}((\mathrm{ACT2}^i \\ &\quad - \mathrm{SEL2}^{i+1}), \mathrm{SPL2}^{i+1})) && \text{(from (2.8))} \\ &= F_{\min}((\mathrm{ACT2}^i - \mathrm{SEL2}^{i+1}) \cup \mathrm{SPL2}^{i+1}) && \text{(by (2.7))} \\ &= F_{\min}((\mathrm{ACT2}^i - \mathrm{SEL2}^{i+1}) \cup \mathrm{SEL2}^{i+1}) && \text{(by (2.5))} \\ &= F_{\min}(\mathrm{ACT2}^i) \end{aligned}$$

since $\mathrm{SEL2}^{i+1} \subseteq \mathrm{ACT2}^i$.

Corollary 2.1.1. *For every integer i such that* $0 \leq i < t + 1$,

$$F_{\min}(\mathrm{ACT2}^i) = \min\{F(x) \mid x \in X\},$$

whence if P2 *terminates, it returns an optimal solution.*[3]

Proof. By induction on i using Theorem 2.1.

There are several different conditions under which P2 can be guaranteed to terminate. In the case of B&B procedures in which all pruning is done using bounding functions, some conditions under which termination can be guaranteed are discussed by Mitten [17].

Procedure P2 is our prototypical General Branch and Bound procedure. In this paper, for each procedure P claimed to be a special case of GBB, the claim is justified either by

(1) showing that P is an instantiation of P2 (or some other procedure known to be a special case of GBB); or

(2) exhibiting an instantiation of P2 (or some other procedure known to be a special case of GBB) such that on the ith iteration of the main loop of P, it computes representations of the same sets $\{\mathrm{rf2}(r) \mid r \in \mathrm{SEL2}^i\}$, $\{\mathrm{rf2}(r) \mid r \in \mathrm{SPL2}^i\}$, $\{\mathrm{rf2}(r) \mid r \in \mathrm{ACT2}^i\}$ computed on the ith iteration of the main loop of P2.

[3]Procedure P2 was formulated for the case in which a single optimal solution is desired. P2 may be generalized slightly by replacing lines 3 and 4 by

3. if goal(r) holds for every $r \in$ ACT2 and
   $F(\mathrm{rf2}(r)) = F(\mathrm{rf2}(r'))$ for every $r, r' \in$ ACT2 then
4. return ACT2

In this case, depending on the properties of prune2, P2 will return an optimal solution, some of the optimal solutions, or all of the optimal solutions.

3. A Special Case

For ease in pruning, many B&B procedures maintain a record of the 'best solution seen so far' separately from the active list. Such procedures usually are instances of the procedure P3 below. As discussed above, we assume the existence of an auxiliary database which is used implicitly by the functions in P3.

```
procedure P3:  //a special case of GBB//
1.   BEST3 := 'unknown'
     //we define rf3('unknown') to be ∅,//
     //whence F_min('unknown') = ∞//
2.   ACT3 := {r0>}
     //r0 is the initial representation of X//
3.   loop  //the main loop//
4.     if ACT3 = ∅ then return BEST3 endif
5.     SEL3 := select3(ACT3)
6.     if SEL3 is a singleton {r3} and goal3(r3) then
7.       if F_min(r3) < F_min(BEST3) then
8.         BEST3 := r3
9.       endif
10.    else
11.      SPL3 := split3(SEL3)
12.      ACT3 := prune3(ACT3 - SEL3,SPL3)
13.    endif
14.  repeat
end P3
```

In P3 above, the functions rf3, goal3, select3, and split3 are representation, goal, selection, and splitting functions, respectively. Often, instantiations of split3 are used which always select a single representation (i.e., they always return sets SEL3 containing exactly one element). prune3 is similar to a pruning function: prune3(ACT3 − SEL3,SPL3) returns a collection of representations ACT3′ such that

$$\{\text{rf3}(r') \mid r' \in \text{ACT3}'\} \subseteq \{\text{rf3}(r) \mid r \in (\text{ACT3} - \text{SEL3}) \cup \text{SPL3}\}$$

and

$$F_{\min}(\text{ACT3}' \cup \{\text{BEST3}\}) = F_{\min}((\text{ACT3} - \text{SEL3}) \cup \text{SPL3} \cup \{\text{BEST3}\}).$$

Any functions having the properties of rf3, goal3, select3, split3, and prune3 are called P3-representation, -goal, -selection, -splitting, and -pruning functions, respectively.

To justify the claim that P3 is a special case of GBB, functions rf2, goal2, select2, split2, and prune2 must be found such that P2 and P3 compute the same sets $\{\text{rf2}(r) \mid r \in \text{SEL2}^i\}$, $\{\text{rf2}(r) \mid r \in \text{SPL2}^i\}$, $\{\text{rf2}(r) \mid r \in \text{ACT2}^i\}$ on the ith iterations of their main loops. This is done in Appendix A. The representation, goal, selection, and splitting functions used in the appendix to instantiate P2

are almost the same as those for P3. The pruning function is somewhat more complicated: depending on whether a goal has been selected or not, it simulates either lines 7–9 or line 12 of P3.

In some versions of P3, the selection function always selects the 'best' member of ACT3 for splitting. Which member is best is always defined relative to a P3-*lower bound function*. This is any real-valued function $L3$ such that for any rf3-representation r,

(1) $L3(r) \leq F_{\min}(r)$;

(2) if goal3(r) holds (whence rf3(r) = $\{x\}$ for some $x \in X$), then $L3(r) = F(x)$.

Suppose there is always at least one $r^* \in$ SEL3 such that $L3(r^*) \leq L3(r)$ for every $r \in$ ACT3. Then select3 is called an $L3$-*best-first* selection function, and P3 is called an $L3$-best-first procedure.

If select3 is $L3$-best-first for some $L3$, then the first singleton set SEL3 = $\{r3\}$ selected at line 5 of P3 such that goal3($r3$) holds represents an optimal member of X. This is proven as Theorem A.2 in Appendix A. As a consequence of this result, if select3 is $L3$-best-first for some $L3$ then P3 can be rewritten as follows:

```
procedure P3B:  //best-first P3//
1.   ACT3 := {r0}
     //r0 is the initial representation of X//
2.   loop  //the main loop//
3.     if ACT3 = ∅ then return 'unknown' endif
4.     SEL3 := select3(ACT3)
5.     if SEL3 is a singleton {r3} and goal(r3) then
6.       return r3
7.     else
8.       SPL3 := split3(SEL3)
9.       ACT3 := prune3(ACT3 - SEL3,SPL3)
10.    endif
11.  repeat
end P3B
```

4. Ordinary Branch and Bound

Most conventional B&B procedures look similar to P3, except for the following properties.

(1) The selection function always selects a single representation.

(2) All pruning is done using a lower bound function similar to the one used for selection in P3B. Lower bounds are computed on all generated subsets of X, and a subset is pruned only if its lower bound is greater than the value of the best solution seen so far.

Every B&B procedure which operates in this manner we call an Ordinary Branch and Bound (OBB) procedure. As discussed later, there are many instantiations of GBB which are not OBB procedures, because their pruning functions are stronger than simple bounding functions. However, as shown in

Appendix B, every OBB procedure is an instantiation of P3 and hence is a special case of GBB.

We take an OBB procedure to be any procedure which can be rewritten as an instance of procedure P4 below.

```
procedure P4:  //OBB//
1.   BEST4 := 'unknown'
     //we define rf4('unknown') to be ∅,//
     //whence F_min('unknown') = ∞.//
2.   ACT4 := list containing r0
     //r0 is the initial representation of X//
3.   while ACT4 ≠ ∅ do  //the main loop//
       //select the first member of ACT4//
4.     {r4} := select4(ACT4)
5.     if goal4(r4) then
6.       if F_min(r4) < F_min(BEST4) then BEST4 := r4 endif
         //BEST4 is the best node seen so far//
7.     else
8.       SPL4 := split4({r4})
         //lines 9–14 compute ACT4'//
9.       ACT4 := ACT4 – {r4}
         //go through the members of SPL4 in order//
10.      for each r' ∈ SPL4 do
11.        if L4(r') < F_min(BEST4) then
             //insert r' into the list ACT4//
12.          ACT4 := insert4(r',ACT4)
13.        endif
14.      endfor
15.    endif
16.  endwhile
17.  return BEST4
end P4
```

Although ACT4 is an ordered list in procedure P4 above, it may be considered as a set, with the ordering information stored in the auxiliary database. rf4 is a P3-representation function, and goal4 is a P3-goal function. select4 removes the first element of a list, and hence is a P3-selection function. $L4$ is a P3-lower bound function. insert4 inserts a representation into a list of representations (thus, considering ACT4 as a set, insert4(r',ACT4) = ACT4 $\cup \{r'\}$).

The function split4 has the characteristics of a P3-splitting function, except that it returns a list of rf4-representations rather than an unordered set. By considering the ordering information to be auxiliary information, split4 can be considered a P3-splitting function. As shown in Appendix B, lines 9–14 of P4 constitute a P3-pruning function.

Theorem 4.1. P4 *is an instantiation of* P3.

Proof. The preceding discussion shows that the functions used in P4 have the properties of P3-goal, -selection, -splitting, and -pruning functions. It only remains to show that each of the functions is defined for all arguments which might be given to them during the operation of P4, and this is easily proved by induction.

From Theorem 4.1, it is clear that P4 is a special case of GBB.

Suppose that for every rf4-representation r', insert4(r', ACT4) inserts r' into ACT4 just after the last $r \in$ ACT4 for which $L4(r) < L4(r')$. Then the first member of ACT4 will always be the member with the lowest value of $L4$, whence select4 is an $L4$-best-first selection function. In this case (as with P3), the first $r4$ selected such that goal($r4$) holds will be an optimal solution, whence P4 may be modified analogously to the way P3 was modified to P3B.

5. Dominance Relations

Some researchers [6, 9] have augmented OBB to use dominance relations for pruning. The lower and upper bounding functions used for pruning in Horowitz and Sahni's description of the 0–1 Knapsack Problem [3] are a special case of dominance. As shown in Kohler and Steiglitz [9], the use of dominance relations allows pruning to be done which may not be possible in OBB procedures. Thus not every B&B procedure is an OBB procedure.

Several different ways [6, 9] have been proposed for the use of dominance in B&B procedures. However, all branch-and-bound procedures with dominance are similar to the procedure P5 given below.

```
procedure P5:  //OBB augmented by dominance//
1.   BEST5 := 'unknown'
     //we define rf5('unknown') to be ∅.//
     //whence F_min('unknown') = ∞.//
2.   ACT5 := list containing r0
     //r0 is the initial representation of X//
3.   GEN5 := ACT5 or ∅
     //GEN5 will contain some or all representations generated//
     //during the operation of P5, and will be updated by//
     //adding or deleting representations at various points.//
     //These points are not given explicitly below.//
4.   while ACT5 ≠ ∅ do  //the main loop//
       //select the first member of ACT5//
5.     {r5} := select5(ACT5)
6.     if goal5(r5) then
7.       if F_min(r5) < F_min(BEST5) then BEST5 := r5 endif
8.     else
9.      SPL5 := split({r5})
          //lines 10–16 compute ACT5//
10.       ACT5 := ACT5 – {r5}
          //go through the members of SPL5 in order //
11.       for each r' ∈ SPL5 do
```

```
12.          if L5(r') < F_min(BEST5) then
                //choose a dominated set R5, subject to //
                //restrictions (see the text)//
13.             R5 := choose(ACT5 ∪ {r'},GEN5)
14.             ACT5 := (ACT5 ∪ {r'}) - R5
15.          endif
16.       endfor
17.    endif
18. endwhile
19. return BEST5
end P5
```

The functions rf5, goal5, select5, split5, $L5$, and insert5 have the same properties as the corresponding functions in P4. Thus rf5, goal5, removetop5, split5, and $L5$ are P3-representation, -goal, -selection, -splitting, and -lower bound functions, respectively.

The function choose in line 13 makes use of a *dominance relation.* This is any relation D such that if q and r are rf5-representations and qDr, then $F_{\min}(q) \leqslant F_{\min}(r)$. choose(ACT5 $\cup \{r'\}$, GEN5) returns a *dominated* set; i.e., a set $R5 \subseteq \text{ACT5} \cup \{r'\}$ such that for every $q \in R5$ there is an $p \in \text{GEN5}$ such that pDq. There are additional restrictions on D and on the way $R5$ is selected. These restrictions, which vary depending on the particular version of P5 [9], are formulated to guarantee that

$$F_{\min}(\{\text{BEST5}\} \cup ((\text{ACT5} \cup \{r'\}) - R5)) = F_{\min}(\{\text{BEST5}) \cup \text{ACT5} \cup \{r'\}) .$$

Because of this, it can be shown that lines 10–16 of P5 have the properties of a P3-pruning function. The proof of this is similar to the corresponding proof for P4.

Theorem 5.1. P5 *is an instantiation of* P3.

Proof. The proof is similar to the proof of Theorem 4.1.

Theorem 5.1 justifies calling P5 a special case of GBB.

Suppose that for every rf5-representation r', insert5(r',ACT5) inserts r' into ACT5 just after the last $r \in$ ACT5 such that $L5(r) < L5(r')$. Then select5 is an $L5$-best-first selection function. Thus, as with P3 and P4, the first $r5$ selected such that goal5($r5$) holds is the optimal solution, whence P5 may be modified accordingly.

6. A*

The least-cost path problem was described in Section 2.1. One procedure for solving this problem is the well-known A* procedure [20], which appears below as procedure P6.

In P6, $c(n, n')$ is the cost of the arc (n, n'); $g(m)$ is the cost of the least-cost path P seen so far from the source node s to m; and $P = \text{path}(m)$ is the path $(s, \ldots, \text{parent}(\text{parent}(n)), \text{parent}(n), n)$. $h(m) \geq 0$ is a lower bound on the cost of any path from m to a member of T, and $f(n) = g(n) + h(n)$ is a lower bound on the cost of extending P to a member of T.

```
procedure P6:  //A*//
2.   OPEN := list containing the source node s
3.   CLOSED := NIL
4.   while OPEN ≠ NIL do  //the main loop//
5.     n := removetop(OPEN)  //remove first element//
6.     insert n into CLOSED
7.     if n ∈ T then
8.       return path(n)
         //path(n) is the path//
         //(s, . . . , parent(parent(n)), parent(n), n)//
9.     else
10.      for every child n' of n in G do
           //compute g(n') and f(n')//
11.        gg := g(n) + c(n,n')
12.        ff := gg + h(n')
13.        for all nodes m in OPEN or CLOSED do
14.          if m = n' and f(m) ≤ ff then
15.            goto PRUNE  //prune n'//
16.          else if m = n' and ff < f(m) then
17.            call remove6(m)
18.          endif
19.        endfor
20.        parent(n') := n
21.        g(n') := gg
22.        f(n') := ff
23.        OPEN := insert6(n',OPEN)
           //insert n' into OPEN just after the last//
           //node n such that f(n) < f(n')//
24.      PRUNE: endfor
25.    endif
26.  endwhile
27.  return 'unknown'
end P6

procedure remove6(m)
1.  if m ∈ OPEN then remove m from OPEN endif
2.  if m ∈ CLOSED then remove m from CLOSED endif
3.  for every n such that parent(n) = m do
4.    call remove6(n)
5.  endfor
end remove6
```

In P6, each node n in OPEN or CLOSED represents the path

$$\text{path}(n) = (s, \ldots, \text{parent}(\text{parent}(n)), \text{parent}(n), n).$$

In order to make this representation explicit, P6 is rewritten as procedure P7 below. The active list for P7 is ACT7 = {path(n) | $n \in$ OPEN}, and the set GEN7 (similar to GEN5 in P5) is such that GEN7 = {path(n) | $n \in$ CLOSED}.

Let m be any node generated by P6, and let P = path(m). Note that tip(P) = m. To write P7, the following definitions are used.

(1) rf7(P) is the set of all extensions of P to members of the set of terminal nodes T (i.e., the set of all paths PQ such that Q is a path from tip(P) to a member of T and the only member of T in Q is tip(Q)). Thus rf7 is a representation function.

(2) goal7(P) holds if and only if tip(P) $\in T$.

(3) select7(L) returns a set whose only element is the first element of the list L. Since the list ACT7 is always kept ordered according to the lower bounds of its members, select7 corresponds to removetop in P6.

(4) split7(P) = {Pn | n is a child of tip(P)}. Expanding a node m in P6 corresponds to computing split({path(m)}) in P7.

(5) $L7(P) = \text{cost}(P) + h(\text{tip}(P))$. Thus from the definitions of f, g, and h, $L7(P) = f(\text{tip}(P))$, whence $L7(P)$ is a lower bound on $F_{\min}(P)$.

(6) insert7(P, L) inserts P into the list L just after the last path Q in L such that $L7(Q) < L7(P)$. Thus select7 is an $L7$-best-first selection function.

Using these definitions, it is clear that P6 can be rewritten as procedure P7 below.[4]

```
procedure P7:  //A*, rewritten//
2.   ACT7 := list containing the null path from s to s
3.   GEN7 := NIL
4.   while ACT7 ≠ NIL do  //the main loop//
5.     {P} := select7(ACT7)  //select first member P of ACT7//
6.     insert P into GEN7
7.     if goal7(P) then
8.       return P
9.     else
10.      SPL7 := split7({P})
11.      ACT7 := ACT7 - {P}
12.      for every path P' in SPL7 do
13.        for every Q in ACT7 or GEN7 do
14.          if tip(Q) = tip(P') and L7(Q) ≤ L7(P') then
15.            goto PRUNE  //prune P'//
16.          else if tip(Q) = tip(P') and L7(P') < L7(Q) then
17.            call remove7(Q)
18.          endif
19.        endfor
20.        predecessor(P') := P
```

[4] In some versions of A*, remove6(m) simply removes m from CLOSED. Thus any descendants of m which have already been generated suddenly represent a new path from s, but their f and g values represent the cost of the old path. Such versions of A* can also be described as GBB procedures simply by defining such nodes not to represent any sets of solutions. However, it is cleaner to use the version of A* presented above.

```
21.          ACT7 := insert7(P',ACT7)
             //insert P' into ACT7 after all paths Q//
             //such that f'(Q) < f'(P')//
22.        PRUNE: endfor
23.      endif
24.   endwhile
25.   return 'unknown'
end P7

procedure remove7(P)
1.  if P ∈ ACT7 then remove P from ACT7 endif
2.  if P ∈ GEN7 then remove P from GEN7 endif
3.  for every Q such that predecessor(Q) = P do
4.     call remove7(Q)
5.  endfor
end remove7
```

We show in Appendix C that P7 is an instantiation of P3B.[5]

7. AO*

AO* is an algorithm [21] for searching hypergraphs, which are conceptually the same as And/Or graphs. A *hypergraph* is a pair $G = (N, H)$, where N is a set of *nodes*, and $H \subseteq N \times 2^N$ is a set of *hyperarcs* or *connectors*. Members of N and H are called G-nodes and G-hyperarcs, respectively. Let m and n be G-nodes. Then m is a *G-parent* of n (or n is a *G-child* of m) if there is a G-hyperarc (m, K) such that $n \in K$. m is a *G-ancestor* of n (or n is a *G-descendant* of m) if

(1) m is a G-parent of n; or

(2) there is a G-descendant m' of m which is a G-parent of n.

m is a *G-leaf* node if m has no G-children. G is *acyclic* if no node in G is a G-ancestor of itself. If $G = (N, H)$ and $G' = (N', H')$ are hypergraphs, then the *union* of G and G' is the hypergraph

$$G \cup G' = (N \cup N', H \cup H'),$$

and the *intersection* of G and G' is the hypergraph

$$G \cap G' = (N \cap N', H \cap H').$$

When the identity of G is obvious, the prefix 'G-' will be dropped from 'G-node', 'G-hyperarc', 'G-parent', 'G-child', 'G-ancestor', 'G-descendant', and 'G-leaf'.

[5]Alternatively, it would be possible to show that P7 is an $L7$-best-first instantiation of P5 by noting that 'PDQ if $\text{tip}(P) = \text{tip}(Q)$ and $L7(P) \leq L7(Q)$' is a dominance relation.

Let $G = (N, H)$ be a hypergraph, and let $m \in N$ and $T \subseteq N$. Suppose that each hyperarc $(m, K) \in H$ has a cost $c(m, K) \geq 0$. A *hyperpath in G from m to T*, and the cost of that hyperpath, are defined recursively as follows.

(1) Suppose m is in T. Then the hyperpath is the hypergraph $(\{m\}, \emptyset)$, The cost of this hyperpath is

$$\text{cost}((\{m\}, \emptyset)) = 0 .$$

(2) Suppose m is not in T, and m is a leaf. Then there is no hyperpath from m to T.

(3) Suppose m is not in T, and m is not a leaf. Then there is at least one set $K = \{m_1, \ldots, m_k\}$ such that $(m, K) \in H$. Let G_0 be the hypergraph $(\{m\} \cup K, \{(m, K)\})$. Suppose that for every $m_i \in K$ there is a hyperpath G_i from m_i to T of cost C_i. Then the hypergraph

$$G' = G_0 \cup G_1 \cup \cdots \cup G_m$$

is a hyperpath from m to T of cost

$$\text{cost}(G') = c(m, K) + C_1 + C_2 + \cdots + C_k .$$

Note that there may be 0, 1, or many hypergraphs G' satisfying the above properties.

Let $G = (N, H)$ be a hypergraph, $s \in N$ be a *source node*, and $T \subseteq N$ be a set of *terminal nodes*. The *least-cost hyperpath problem* is the problem of finding a hyperpath (N', H') from s to T which minimizes the value of the objective function $\text{cost}((N', H'))$.

One procedure for solving the least-cost hyperpath problem is the procedure AO* discussed by Nilsson [19]. Other similar procedures are discussed by Nilsson [21] and Martelli and Montanari [15]. AO* is given below as procedure P8.

AO* makes use of a lower bound $h(m)$ on the least cost of any hyperpath from m to T. As discussed by Nilsson [21], h must be such that:

(1) $h(m) \geq 0$ for every G-node m (whence $h(m) = 0$ if $m \in T$);

(2) for every G-hyperarc (m, K),

$$h(m) \leq c(m, K) + \sum_{n \in K} h(n) .$$

procedure P8: //AO*//
1. $(N,H) := (\{s\}, \emptyset)$ //the portion of G searched so far//
2. $q(s) := h(s)$
   //$q(n)$ is the best known lower bound for each node n//
3. if $s \in T$ then solved(s) := 1

```
4.    else solved(s) := 0 endif
5.    loop  //the main loop//
6.       let BEST be the hyperpath in (N,H) formed by tracing
            the 'best' pointers from s through (N,H) to the
            leaves of (N,H)
            //these pointers are set in line 20//
7.       if solved(s) = 1 then return BEST endif
8.       let m be a BEST-leaf not in T
            //Nilsson discusses several possible ways//
            //of choosing m, but the procedure will work//
            //regardless of how m is chosen.//
         //expand m//
9.       for every K such that (m,K) is a G-hyperarc do
10.         for every n ∈ K − N do
11.            if n ∈ T then solved(n) := 1 else solved(n) := 0 endif
12.            q(n) := h(n)
13.         endfor
14.         (N,H) := (N ∪ K, H ∪ {(m,K)})
15.      endfor
         //update the 'best' pointers and the q-values by//
         //searching bottom-up//
16.      V := {m}
17.      while V ≠ ∅ do
18.         remove from V a node u such that no (N,H)-descendants
               of u are in V
19.         q(u) := min{c(u,K) + Σ_{v∈K} q(v) | (u,K) ∈ H}
20.         K' := any K such that (u,K) ∈ H and (u,K) produces the
               minimum value q(u) found in line 19
            //(u,K') is currently the best hyperarc from u//
21.         best(u) := K'
22.         if solved(v) = 1 for every v ∈ K' then solved(u) := 1 endif
23.         if solved(u) = 1 or q(u) was changed in line 19 then
24.            V := V ∪ {v ∈ N | u ∈ best(v)}
25.         endif
26.      endwhile
27.   repeat
end P8
```

We now discuss how to rewrite P8 as an instantiation P9 of P3B.

Let P be a hyperpath in G from some node m of G. An *extension* of P in G is any hyperpath P' from m containing P. We define the representation function rf9 by

$$\text{rf9}(P) = \{P' \mid P' \text{ is an extension of } P \text{ to } T\}.$$

Note that if P' is an extension of P then $\text{rf9}(P') \subseteq \text{rf9}(P)$, and that

$$F_{\min}(P) = \min\{\text{cost}(P^*) \mid P^* \text{ is an extension of } P \text{ to } T\}.$$

We define

$$L9(P) = \text{cost}(P) + \sum \{h(n) \mid n \text{ is a } P\text{-leaf}\}.$$

To see that $L9$ is a P3-lower bound function, we note that

(1) if P' is any extension of P in G to T, then

$$\begin{aligned}\text{cost}(P') &= \text{cost}(P) + \text{cost}(P_1) + \cdots + \text{cost}(P_k) \\ &\geqslant \text{cost}(P) + h(n_1) + \cdots + h(n_k) \\ &= L9(P),\end{aligned}$$

whence $L9(P) \leqslant F_{\min}(P)$;

(2) if P is a hyperpath to T, then $L9(P) = \text{cost}(P) = F_{\min}(P)$.

Let goal9(P) hold if and only if P is a hyperpath from s to T. Clearly, goal9 is a goal function.

Let ACT9 be the set of all hyperpaths in (N,H) from s to the leaves of (N,H). Let SEL9 = select9(ACT9) be {BEST} if goal9(BEST) holds, and otherwise let it be the set of all hyperpaths in ACT9 containing the node m chosen in line 8 of P8. Clearly, select9 is a selection function.

It may be proved by induction that for every $u \in N$, lines 16–26 of P8 maintain $q(u)$ such that

$$q(u) = \begin{cases} h(u), & \text{if } u \text{ is an } (N,H)\text{-leaf} \\ \min\{c(u,K) + \sum_{v \in K} q(v) \mid (u,K) \in H\} & \text{otherwise}. \end{cases}$$

Thus, by induction on the distance from u to the leaves of (N,H), it may be proved that

$$q(u) = \min\{L9(P) \mid P \text{ is a hyperpath in } (N,H) \text{ from } u \text{ to the leaves of } (N,H)\}.$$

The hyperpath found by tracing the 'best' pointers from u to the leaves of (N,H) is the one which achieves this minimum. In particular, $q(s) = L9(\text{BEST})$. Since it is always the case that BEST $\in$ SEL9, select9 is $L9$-best-first.

Let SPL9 = split9(SEL9) be the set of all hyperpaths from s to the leaves of (N',H') which contain m, where (N',H') is the expanded version of (N,H) computed in lines 9–15 of P8. Let prune9(ACT9 − SEL9, SPL9) be the set of all hyperpaths in (N',H') from s to the leaves of (N',H'). It is proved in Appendix D that split9 and prune9 have the properties of splitting and pruning functions, respectively.

Using the above definitions, P8 may be rewritten as procedure P9 below.

```
procedure P9:  //AO*, rewritten//
1.   ACT9 := {({s},∅)}
2.   loop
       //the test below will never succeed, and is included//
       //merely to illustrate that P9 is an instantiation//
       //of P3B//
3.     if ACT9 = ∅ then return 'unknown' endif
4.     SEL9 := select9(ACT9)
5.     if SEL9 is a singleton {r9} and goal(r9) then
6.       return r9
7.     else
8.       SPL9 := split9(SEL9)
9.       ACT9 := prune9(ACT9 - SEL9,SPL9)
10.    endif
11.  repeat
end P9
```

Theorem 7.1. P9 *is an instantiation of* P3B.

Proof. From the preceding discussion, we see that the functions used in P9 have the properties of P3-goal, -selection, -splitting, and -pruning functions, respectively, and that select9 is $L9$-best-first. It only remains to show that each of the functions is defined for all arguments which might be given to them during the operation of P9, and this is easily proved by induction.

8. Summary and Conclusions

This paper contains a general formulation of B&B called General Branch and Bound (GBB). The main features of GBB include

(1) a formal treatment of the way subsets of the domain of solutions are represented;

(2) the formulation of a procedural scheme for GBB using abstract goal, selection, splitting, and pruning functions;

(3) the discussion of how these functions may be generalized to make use of problem-specific auxiliary data;

(4) a discussion of the conditions under which an arbitrary optimization procedure may be considered a special case of GBB.

GBB is powerful enough to include as special cases the formulations of B&B used by Mitten [17], Kohler and Steiglitz [9], and Ibaraki [6]. In addition, the AI search procedures A* and AO* have been proven to be instances of GBB.

It can be shown that a number of other AI search procedures are also special cases of GBB [13]. It is possible to visualize many variations of existing search procedures being generated from this general branch-and-bound paradigm, which provides a theoretical basis for a better understanding of the per-

formance of such algorithms and the relationships among them (for example, see [11, 13, 18]). In particular, we conjecture that all procedures for top-down search of problem-reduction representations can be examined and understood as instantiations of this general branch-and-bound procedure.

Appendix A. Theorems about P3

To justify the claim that P3 is a special case of GBB, functions rf2, goal2, select2, split2, and prune2 must be found such that P2 and P3 compute representations of the same sets $\{\text{rf2}(r) \mid r \in \text{SEL2}^i\}$, $\{\text{rf2}(r) \mid r \in \text{SPL2}^i\}$, $\{\text{rf2}(r) \mid r \in \text{ACT2}^i\}$ on the ith iterations of their main loops. P3 is a version of P2 in which the active list ACT2 is separated into two parts: ACT3 and {BEST3}. Furthermore, the pruning operation which would be done by prune2 in P2 is split into the two different operations performed in lines 7–9 and 12 of P3. To reproduce this behavior in P2, we define prune2 to flag every representation in ACT2 to indicate whether it is in ACT3 or (BEST3}. Let

$$\text{rf2}(r) = \text{rf3}(r), \qquad \text{goal2}(r) = \text{goal3}(r),$$

and

$$\text{select2}(\text{ACT2}) = \{\text{select3}(\text{ACT2} - \{\text{BEST3}\})\}.$$

Let split2 and prune2 be the procedures given below.

```
procedure split2(SEL2):  //simulate lines 6 and 10–11 of P3//
1.  if SEL2 is a singleton {r} and goal2(r) then
2.     goalfound := 1
3.     return SEL2
4.  else
5.     goalfound := 0
6.     return split3(SEL2)
7.  endif
end split2

procedure prune2(ACT2,SPL2):  //simulate lines 7–9, 12 of P3//
     //for every representation r, flag(r) is taken to be 0//
     //unless it is set to 1 in line 5 below//
1.  if goalfound = 1 then
        //select2 has selected a goal r2 and SPL2 = {r2}//
        //simulate lines 7–9 of P3//
2.      if flag(r*) = 1 for some r* ∈ ACT2
        //at most one such r* exists; this r* is BEST3//
        and F_min(r*) ≤ F_min(r²) then
3.          return ACT2
4.      else
            //make r2 the new BEST3//
5.          flag(r2) := 1
6.          return {r ∈ ACT2 | flag(r) = 0} ∪ {r2}
```

```
7.  else
8.      return {r ∈ ACT2 | flag(r) = 1} ∪ prune3({r ∈ ACT2 | flag(r) = 0})
9.  endif
end prune2
```

Theorem A.1. *Suppose* P2 *is instantiated using the definitions of* rf2, goal2, select2, split2, *and* prune2 *given above. Then* rf2 *and* goal2 *are representation and goal functions, and for every initial representation* $r0$, *the computations of* P2 *and* P3 *are such that for every* i,

(1) $\{r \in \text{ACT2}^i \mid \text{flag}(r) = 0\} = \text{ACT3}^i$;

(2) $\{r \in \text{ACT2}^i \mid \text{flag}(r) = 1\} = \{\text{BEST3}^i\}$;

(3) $\text{SEL2}^i = \text{SEL3}^i$;

(4) $$\text{SPL2}^i = \begin{cases} \text{SEL3}^i, & \textit{if } \text{SEL3}^i \textit{ is a singleton } \{r\} \textit{ and } \text{goal}(r) \textit{ holds,} \\ \text{SPL3}^i, & \textit{otherwise;} \end{cases}$$

(5) select2(ACT2^i), split2(SEL2^i), *and* prune2($\text{ACT2}^{i-1} - \text{SEL2}^i, \text{SPL2}^i$) *are defined for the arguments given to them, and hence are selection, splitting, and pruning functions, respectively.*

Proof. By induction on i.

Corollary A.1.1. *Let* t *be the number of times the main loop of* P3 *is fully executed for some initial representation* $r0$ *and instantiations of* rf3, goal3, select3, split3, *and* prune3. *For every integer* i *such that* $0 \leq i < t + 1$,

$$F_{\min}(\text{ACT3}^i \cup \{\text{BEST3}^i\}) = \min\{F(x) \mid x \in X\} .$$

Proof. Immediate from Theorem A.1 and Corollary 2.1.1.

Theorem A.1 justifies the claim that P3 is a special case of GBB. Theorem A.2 below justifies rewriting P3 as P3B when the selection function is best-first.

Theorem A.2. *If* select3 *is* L3*-best-first for some* L3, *then the first singleton set* $\text{SEL3} = \{r\}$ *selected at line* 5 *of* P3 *such that* goal3($r3$) *holds represents an optimal member of* X.

Proof. Consider the first singleton set $\text{SEL3} = \{r3\}$ selected at line 5 of P3 such that goal3($r3$) holds, and suppose $r3$ is selected during the ith iteration of the main loop of P3. Then for every $r \in \text{ACT3}^{i-1}$,

$$F_{\min}(r3) = L3(r3) \leq L3(r)$$
$$\leq F_{\min}(r) = \min\{F(x) \mid x \in \mathrm{rf3}(r)\}.$$

Thus since

$$F_{\min}(\mathrm{BEST3}^{i-1}) = F_{\min}(\text{'unknown'}) = \infty,$$

it follows from Corollary A.1.1 that

$$F_{\min}(r3) = F_{\min}(\mathrm{ACT3}^{i-1}) = \min\{F(x) \mid x \in X\}.$$

Appendix B. Properties of P4

Theorem B.1. *Lines* 9–14 *of* P4 *constitute a* P3-pruning function.

Proof. When lines 9–14 of P4 are executed during the ith iteration of the main loop, they compute

$$\mathrm{ACT4}^i = (\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \{r \in \mathrm{SPL4}^i \mid L4(r) < L4(\mathrm{BEST4}^i)\}$$
$$\subseteq (\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \mathrm{SPL4}^i.$$

Thus

$$\{\mathrm{rf4}(r') \mid r' \in \mathrm{ACT4}^i\} \subseteq \{\mathrm{rf4}(r) \mid r \in (\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \mathrm{SPL4}^i\}.$$

The above is the first property of a P3-pruning function.

To prove that lines 9–14 of P4 have the second property of a P3-pruning function, there are two cases to be considered.

Case 1.

$$F_{\min}(r4^i) \geq F_{\min}((\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \{\mathrm{BEST4}^i\})$$

In this case,

$$F_{\min}((\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \{\mathrm{BEST4}^i\}) =$$
$$= F_{\min}((\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \{r4^i\} \cup \{\mathrm{BEST4}^i\})$$
$$= F_{\min}((\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \mathrm{SPL4}^i \cup \{\mathrm{BEST4}^i\}).$$

Thus since

$$(\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \{\mathrm{BEST4}^i\} \subseteq \mathrm{ACT4}^i \cup \{\mathrm{BEST4}^i\}$$
$$\subseteq (\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \mathrm{SPL4}^i \cup \{\mathrm{BEST4}^i\},$$
$$F_{\min}(\mathrm{ACT4}^i \cup \{\mathrm{BEST4}^i\}) = F_{\min}((\mathrm{ACT4}^{i-1} - \{r4^i\}) \cup \mathrm{SPL4}^i \cup \{\mathrm{BEST4}^i\}).$$

Case 2.

$$F_{\min}(r4^i) < F_{\min}((\text{ACT4}^{i-1} - \{r4^i\}) \cup \{\text{BEST4}^i\}).$$

According to the definition of split4, there is an $r \in \text{SPL4}^i$ such that $F_{\min}(r) = F_{\min}(r4^i)$. This means that

$$\begin{aligned} L4(r) &\leq F_{\min}(r) = F_{\min}(r4^i) \\ &< F_{\min}(\text{BEST4}^i) = F_{\min}(\text{BEST4}^i), \end{aligned}$$

whence $r \in \text{ACT4}^i$. Thus

$$\begin{aligned} F_{\min}(\text{ACT4}^i) &= F_{\min}(r) \\ &= F_{\min}((\text{ACT4}^{i-1} - \{r4^i\}) \cup \text{SPL4}^i \cup \{\text{BEST4}^i\}). \end{aligned}$$

In both cases, the second property of a P3-pruning function is satisfied.

Appendix C. Properties of A*

In this appendix, it is shown that P7 (and hence A*) is a special case of GBB.

Let G be a graph for a shortest path problem with start node s and goal set T, and let P and Q be paths in G from s. Then P *covers* Q if $F_{\min}(P) \leq F_{\min}(Q)$ and there are paths P' and P'' such that $P = P'P''$ and $\text{tip}(P') = \text{tip}(Q)$. If P covers Q, this means that for every extension Q^* of Q to a member of T, there is an extension P^* of P to a member of T such that

$$F_{\min}(P^*) \leq F_{\min}(Q^*),$$

whence Q can be pruned if $Q \neq P$ and $P \in \text{ACT7}$. Note that the covering relation is transitive: if P covers Q and Q covers R, then P covers R.

Lemma C.1. *If P prunes Q in lines* 15 *or* 17 *of* P7, *then P covers Q.*

Proof. If P prunes Q, then $\text{tip}(P) = \text{tip}(Q)$ and $L7(P) \leq L7(Q)$. But

$$L7(P) = \text{cost}(P) + h(P)$$

and

$$L7(Q) = \text{cost}(Q) + h(Q) = \text{cost}(Q) + h(P),$$

so

$$\text{cost}(P) \leq \text{cost}(Q).$$

Since $\text{tip}(P) = \text{tip}(Q)$, this means that $F_{\min}(P) \leq F_{\min}(Q)$, whence P covers Q.

Let t be the number of times the main loop of P7 is fully executed for some input graph G (thus t may be infinite). Let $k(i)$ be the number of times lines 12–22 of P7 are executed during the ith iteration of the main loop. Note that

$$k(i) = |\text{split7}(P^i)| .$$

We define (i, j) to be a *loop index* of P7 if it corresponds to the jth iteration of the inner loop of P7 during the ith iteration of the main loop; i.e., if $0 \leq i < t+1$ and $0 \leq j \leq k(i)$. We say that (i', j') is *older* than (i, j) if (i', j') corresponds to an iteration of the inner and main loops of P7 previous to the iteration corresponding to (i, j); i.e., if $i' < i$ or if $i' = i$ and $j' < j$.

Let $\text{ACT7}^{i,j}$ and $\text{GEN7}^{i,j}$, respectively, be the values of ACT7 and GEN7 computed on the jth iteration of lines 12–22 of P7 during the ith iteration of the main loop. Let $P_{i,1}, P_{i,2}, \ldots, P_{i,k(i)}$ be the members of SPL7^i computed at line 10 of P7, and let

$$\text{frontier}(i, j) = \text{ACT7}^{i,j} \cup \{P_{i,j+1}, \ldots, P_{i,k(i)}\} .$$

Note that for each i,

(1) $\text{ACT7}^{i,0} = \text{ACT7}^{i-1} - \{P^i\}$;

(2) $\text{GEN7}^{i,0} = \text{GEN7}^{i-1}$;

(3) $\text{frontier}(i, 0) = (\text{ACT7}^{i-1} - \{P^i\}) \cup \text{SPL7}^i$;

(4) $\text{GEN7}^{i,k(i)} = \text{GEN7}^i$,

(5) $\text{frontier}(i, k(i)) = \text{ACT7}^{i,k(i)} = \text{ACT7}^i$.

Theorem C.2. *For every loop index* (i, j), *every loop index* (i', j') *older than* (i, j), *and every path* $V \in \text{frontier}(i', j')$, *there is a path* $W \in \text{frontier}(i, j)$ *such that* W *covers* V.

Proof (by induction on (i, j)). There is no (i', j') older than $(1, 0)$, so the theorem holds vacuously for $(i, j) = (1, 0)$. For the induction hypothesis, let $0 \leq i < t+1$ and $0 \leq i \leq k(i)$, and suppose that the theorem holds for every (i'', j'') older than (i, j). To prove that the theorem holds for (i, j), there are two possible cases to consider: $j = 0$ and $j > 0$.

Case 1. $j = 0$. Let (i', j') be older than $(i, 0)$. If $i' < i - 1$ or if $i' = i - 1$ and $j' < k(i-1)$, then by the induction hypothesis every $V \in \text{frontier}(i', j')$ is covered by a $W \in \text{frontier}(i-1, k(i-1))$. Thus since covering is transitive, it suffices to show that every $V \in \text{frontier}(i-1, k(i-1))$ is covered by a $W \in$

frontier$(i, 0)$. If $V \in$ frontier$(i-1, k(i-1))$, then either $V \in \text{ACT7}^{i,0}$ or $V = P^i$. In the first case, V covers itself. In the second case, it follows from the definition of split7 that

$$F_{\min}(\{P_{i,1}, \ldots, P_{i,k(i)}\}) = F_{\min}(P^i),$$

whence one of $P_{i,1}, \ldots, P_{i,k(i)}$ covers P^i. In either case, the theorem holds for (i, j).

Case 2. $j > 0$. Let (i', j') be older than (i, j). If $i' < i$ or if $i' = i$ and $j' < j-1$, then by the induction hypothesis every $V \in$ frontier(i', j') is covered by a $W \in$ frontier$(i, j-1)$. Thus since covering is transitive, it suffices to show that every $V \in$ frontier$(i, j-1)$ is covered by a $W \in$ frontier(i, j). There are three possible cases to consider.

Case 2(a). $V \in \{P_{i,j+1}, \ldots, P_{i,k(i)}\}$. Then $V \in$ frontier(i, j) and V covers V.

Case 2(b). $V = P_{i,j}$. If $V \in \text{ACT7}^{i,j}$, then $V \in$ frontier(i, j) and V covers V. Otherwise, V was pruned at line 15 of P7 by some a $Q \in \{\text{ACT7}^{i,j-1} \cup \text{GEN7}^{i,j-1}$. By Lemma C.1, Q covers V, and by the induction hypothesis, there is a $W \in$ frontier$(i, j-1)$ covering Q, whence W covers V. Since V could not have pruned W,

$$W \in \text{ACT}^{i,j} \subseteq \text{frontier}(i, j).$$

Case 2(c). $V \in \text{ACT7}^{i,j-1}$. If $V \in \text{ACT7}^{i,j}$, then V covers V. Otherwise, V was pruned at line 17 of P7 by $P_{i,j}$, whence from Lemma C.1, $P_{i,j}$ covers V. Thus, since covering is transitive, this case reduces to Case 2(b).

Corollary C.2.1. P7 *is an instance of* P3B.

Proof. It is clear that goal7 is a goal function, and that select7 and split7 have the properties of P3-selection and -splitting functions, and that select7 is $L7$-best-first. If we define prune7 to be lines 11–22 of P7, it follows from Theorem C.2 it follows that prune7 has the properties of a P3-pruning function. It only remains to show that select7 split7, and prune7 are defined for all arguments which might be given to them during the operation of P7, and this is easily proved by induction.

From Corollary C.2.1, it is clear that P7 (and hence A*) is a special case of GBB.

Appendix D. Properties of AO*

Theorem D.1. split9 *satisfies properties* (2.4) *and* (2.5).

Proof. Let (N, H) and (N', H') be as in Section 8. We first show that if $P' \in$ SPL9, then there is a $P \in$ SEL9 such that rf9$(P') \subseteq$ rf9(P). Let $P' \in$ SPL9, and let $Q = P' \cap (N, H)$. Clearly, Q contains at least one hyperpath P from s

to the leaves of (N, H) containing m. Since P is a subgraph of P', P' is an extention of P, whence $\mathrm{rf}(P') \subseteq \mathrm{rf}(P)$.

It follows directly from the above that $\bigcup\{\mathrm{rf9}(P) \mid P \in \mathrm{SPL9}\} \subseteq \bigcup\{\mathrm{rf9}(P) \mid P \in \mathrm{SEL9}\}$. To prove that $\bigcup\{\mathrm{rf9}(P) \mid P \in \mathrm{SEL9}\} \subseteq \bigcup\{\mathrm{rf9}(P) \mid P \in \mathrm{SPL9}\}$, let $P \in \mathrm{SEL9}$ and let P^* be an extension of P in G to T. Let $Q = (N', H') \cap P^*$. Clearly, Q contains at least one hyperpath P' from s to the leaves of (N', H'), and by definition of SPL9, $P' \in \mathrm{SPL9}$. Since P' is a subgraph of P^*, P^* is an extension of P' to T, whence $P^* \in \mathrm{rf}(P')$.

Theorem D.2. prune9 *satisfies properties* (2.6) *and* (2.7).

Proof. Let (N, H), m, and (N', H') be as in Section 8. Since prune9((ACT9 − SEL9) ∪ SPL9) is the set ACT9′ of all hyperpaths in (N', H') from s to the leaves of (N', H'), the theorem will follow trivially if we show that

$$\mathrm{ACT9}' = (\mathrm{ACT9} - \mathrm{SEL9}) \cup \mathrm{SPL9}\,.$$

Let P be a path in SPL9. Then P is a hyperpath in (N', H') from s to the leaves of (N', H') containing m, so P is clearly in ACT9′. Let P be a path in ACT9 − SEL9. Then P is a hyperpath in (N, H) from s to the leaves of (N, H) which does not contain m. Since the only difference between (N, H) and (N', H') is the addition of some hyperarcs at m, P is also a hyperpath in (N', H') from s to the leaves of (N', H'). Thus $P \in \mathrm{ACT9}'$. Finally, let P be a hyperpath in ACT9′. If P does not contain m, then $P \cap (N, H) = P$, whence $P \in \mathrm{ACT9} - \mathrm{SEL9}$. If P does contain m, then P is an extension of some $P' \in \mathrm{SEL9}$, whence $P \in \mathrm{SPL9}$.

REFERENCES

1. Berliner, H., The B* tree search algorithm: A best-first proof procedure, *Artificial Intelligence* **12** (1979) 23–40.
2. Hall, P.A.V., Branch-and-bound and beyond, *Proceedings Second International Joint Conference on Artificial Intelligence* (1971) 641–658.
3. Horowitz, E. and Sahni, S., *Fundamentals of Computer Algorithms* (Computer Science Press, Potomac, MD, 1978).
4. Ibaraki, T., On the optimality of algorithms for finite state sequential decision processes, *J. Math. Anal. Appl.* **53** (1976) 618–643.
5. Ibaraki, T., The power of dominance relations in branch and bound algorithms, *J. ACM* **24** (1977) 264–279.
6. Ibaraki, T., Branch-and-bound procedure and state-space representation of combinatorial optimization problems, *Inform. Control* **36** (1978) 1–27.
7. Knuth, D.E. and Moore, R.W., An analysis of alpha-beta pruning, *Artificial Intelligence* **6** (1975) 293–326.
8. Kohler, W.H. and Steiglitz, K., Characterization and theoretical comparison of branch and bound algorithms for permutation problems, *J. ACM* **21** (1974) 140–156.
9. Kohler, W.H. and Steiglitz, K., Enumerative and iterative computational approaches, in: E.G. Coffman, Jr. (Ed.), *Computer and Job-Shop Scheduling Theory* (Wiley, New York, 1976).

10. Kumar, V., A unified approach to problem solving search procedures, Ph.D. Dissertation, Dept. of Computer Science, University of Maryland, College Park, MD, 1982.
11. Kumar, V. and Kanal, L., Branch and bound formulations for and/or tree search with applications in pattern recognition and game playing, *1982 International Conference on Pattern Recognition and Image Processing*, Munich, 1982.
12. Kumar, V. and Kanal, L., A general branch and bound formulation for understanding and synthesizing and/or tree search procedures *Artificial Intelligence* **21** (1,2) (1983) 179–198.
13. Kumar, V., Nau, D.S. and Kanal, L.N., A general model for problem reduction and game search (1984) in preparation.
14. Lawler, E.L. and Wood, D.E., Branch-and-bound methods: A survey, *Oper. Res.* **14** (1966) 699–719.
15. Martelli, A. and Montanari, U., Additive AND/OR graphs, *Proceedings Third International Joint Conference on Artificial Intelligence* (1973) 1–11.
16. Martelli, A. and Montanari, U., Optimizing decision trees through heuristically guided search, *Comm. ACM* **21** (1978) 1025–1039.
17. Mitten, L.G., Branch and bound methods: General formulations and properties, *Oper. Res.* **18** (1970) 23–34. Errata in *Oper. Res.* **19** (1971) 550.
18. Nau, D.S., Kumar, V. and Kanal, L.N., A general paradigm for A.I. search procedures, *National Conference on Artificial Intelligence*, 1982.
19. Nilsson, N., Searching problem solving and game playing trees for minimum COST solutions, in: A.J.H. Morrel (Ed.), *Information Processing-68* (North-Holland, Amsterdam, 1968).
20. Nilsson, N.J., *Problem-Solving Methods in Artificial Intelligence* (McGraw-Hill, New York, 1971).
21. Nilsson, N.J., *Principles of Artificial Intelligence* (Tioga, Palo Alto, CA. 1980).
22. Pohl, I., Is heuristic search really branch and bound?, *Proceedings Sixth Annual Princeton Congress on Information Science and Systems* (1972) 370–373.
23. Reingold, E., Nievergelt, J. and Deo, N., *Combinatorial Optimization* (Prentice-Hall, Englewood Cliffs, NJ, 1977).
24. Smith, D.R., On the computational complexity of branch and bound search strategies, Ph.D. Dissertation, Duke Univ., Durham, NC, 1979; Tech. Rept. NPS 52-79-114, Naval Postgraduate School, Monterey, CA, 1979.
25. Stockman, G.C., A minimax algorithm better than alpha-beta?, *Artificial Intelligence* **12** (1979) 179–196.

Received May 1982; revised version received March 1983