A COMPARISON OF THREE PROBLEM-SOLVING METHODS

Ranan B. Banerji
Temple University
Philadelphia, Pennsylvania 19122

and

George W. Ernst
Case Western Reserve University
Cleveland, Ohio 44106

Abstract

This paper is a comparison of ABSTRIPS, planning (as defined in Newell and Simon, 1972) and GPS. Each of these methods has parameters chat contain heuristic information which is problem dependent. These parameters are used to guide the methods[1] search and usually cause them to be incomplete in the sense that they cannot solve some problems that have solutions. We show that the parameters of the methods serve the same function in the following sense: Given the parameters for one method we can formulate the parameters for the other two such that all three can solve the same class of problem; i.e. those which have totally ordered solutions. This result is somewhat surprising because the search spaces of the methods are different. The Implications of this result to the efficiency of search Is discussed at some length.

1. Introduction

The purpose of this paper is to compare 3 different problem solving methods: ABSTRIPS (Sacerdoti, 1974); planning (Newell and Simon, 1972) and GPS (Ernst and Newell, 1969)+ . The relationship between planning and GPS is reasonably well understood, but their relationship to ABSTRIPS has been much less understood.

All three methods are incomplete in the sense that they cannot solve certain problems that have solutions. Hence, it makes sense to attempt to characterize the class of problems that they can solve. This should allow one to answer such questions as, "Under what conditions can GPS solve any problem that ABSTRIPS can solve?" To answer such questions we will build a formal model of ABSTRIPS and planning. The model deviates slightly from the way ABSTRIPS really works but it is a close approximation to ABSTRIPS. Using this model and the one for GPS (Ernst, 1969) we can answer the above question.

The formalisms are not in themselves particularly interesting. However, they do reveal the relationship among parameters of the three methods. That is, each method has certain heuristic information as parameters to guide its search. In GPS, the parameters are the differences and difference ordering; in ABSTRIPS they are the criticality levels of the various predicates. Our analysis clearly shows the relationship between these two kinds of parameters. In addition, since there are formal conditions of "good" differences for GPS (Ernst, 1969), these conditions should place constraints on "good" parameters for ABSTRIPS and planning. Sacerdoti (1974) gives some informal rules that he uses in assigning criticality levels to predicates which are consistent with the conditions of good differences but the latter are considerably stronger than his rules.

We start off with a brief description of ABSTRIPS and planning. This is followed by a formalization and analysis of ABSTRIPS and planning. The last section contains a discussion of the three methods.

2. Problem Specification

A problem state of ABSTRIPS (and its predecessor STRIPS (Fikes and Nilsson, 1971)) is a set of formulae in first-order logic. For example, if the formula Inroom (Robot, Rooml) were part of the state, that would indicate that the Robot was in Room 1 in the state. The set of desired states is also represented by a formula, W. Any state which implies that W is true is a desired state.

The operators are rules for transforming one state into another. For example, if the robot wanted to move Boxl to Box2 it would use the operator Pushb(Boxl, Box2) where Pushb is given in Figure 1. For this operator to be applicable, the preconditions must be true of the current state, i.e., the preconditions give the domain of the operator. The current state is modified by deleting all of the formulae listed under deletions and adding the formulae under additions. For

This research was supported by the National Science Foundation under grants GJ-1135 and MCS75-23412.

^Planning and GPS were conceived of by Al Newell, Cliff Shaw and Herb Simon in the 1950's. The references given here were chosen because they are more accessible than their original publications. This use of the word 'planning[1] is somewhat different than its use in robotics research.

example, if Nextto(Boxl, Doorl) was in the current state it would be one of the deletions and Nextto(Boxl, Box2) would be one of the additions in forming the new state resulting from applying Pushb(Boxl, Box2). The $1 in Figure 1 is just a variable.

```
Pushb (x,y)
   Preconditions:
      Type(y, Object)
      Pushable(x)
      Nextto(Robot,x)
      3r [Inroom(Robot,r) & Inroom(y,r)
                & Inroom(x,r)]
   Deletions:
      Nextto(Robot,$I)
      Nextto(x,$I)
      Nextto($I,x)
   Additions:
      Nextto(x,y)
      Nextto(y,x)
      Nextto(Robot,x)
```

Figure 1.

The ABSTRIPS operator that pushes box x to object y.

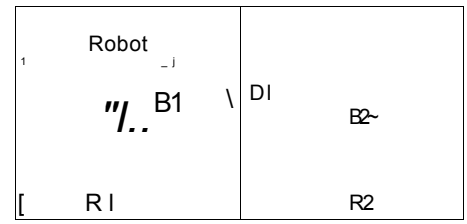
For the purposes of this report the details of the problem specification are not important. The important thing is that each problem has a state space, an initial state, a set of desired states, and a set of operators. Each operator is a partial function from states into states. The domain of the operator is given by its preconditions. It should be noted that Pushb is really a partial function schema; it becomes a partial function after values are specified for x and y, i.e., Pushb(Boxl, Box2) is a partial function. The point is that ABSTRIPS is not dependent on the problem specification language but its underlying algebraic structure. Hence, ABSTRIPS could be applied to problems specified in the other languages such as the one given in Ernst, et al (1974) in which states are represented by arrays.

3. Description of ABSTRIPS

ABSTRIPS assumes that each predicate in the preconditions of operators has a criticality level assigned to it. (Actually Sacerdoti (1974) has a, semi-automatic way of assigning criticality levels.) The intuitive idea behind criticality levels is that high level predicates are more difficult to change than low level predicates.

A simple problem is given in Figure 2 and ABSTRIPS[1] solution is depicted in Figure 3. It starts off by trying to transform the initial state in Figure 2 into one in which Bl is next to B2. (This is subproblem 1 in Figure 3.) ABSTRIPS notes that the operator Pushb(B1,B2) is relevant to this problem and applies it, which solves the problem.

Initial State:



Desired states: Nextto(Bl,B2)

Operators: Pushb(x,y)—Push box x to object y.
PTD(x,y)—Push box x through door y.
Gotob(x)—Go to box x.
Pushd(x,y)—Push box x to door y.
Other operators are not given here because they are not necessary to solve this problem.

Criticality: Level 6—Type, Pushable
Level 5—Inroom
Level 2—Nextto

Figure 2. A simple problem for ABSTRIPS.

The reader will quickly note that this is illegal because the initial state is not in the domain of the operator. However, this subproblem is being solved at criticality level 6 which means that all predicates of level 5 or less are ignored. Nextto has been designated as level 2 and Inroom is level 5. Hence, at level 6, ABSTRIPS considers the Preconditions of Pushb(Bl, B2) (see Figure 1) to be Type(B2, Object) and Pushable(Bl) which is true in the initial state and consequently the operator is applied. Normally, several operators will be necessary to solve the problem but in this case one suffices.

The next subproblem, #2 in Figure 3, is to get from the initial state to a state in which the level 5 and higher predicates of the preconditions of Pushb(Bl, B2) are true. In Figure 3 we have only listed the predicates which are false in the initial state for purposes of exposition. (We are assuming that the variable r is assigned the value R2.) ABSTRIPS notes that PTD(B1, Dl) is relevant to this subproblem and applies it which solves the subproblem. The level 2 predicates in the preconditions of PTD(B1, Dl) are ignored because this subproblem is being solved at criticality level 5.

Next, Pushb(Bl, B2) is applied to the result of this subproblem and this new state is transformed into a desired state. But is is already a desired state; i.e., applying (PTD(B1, Dl), Pushb(Bl, B2) to the initial state results in a state which satisfies Nextto(Bl, B2). Hence, no subproblem

	Criticality level	Initial state	Solution	Desired states
1.	6	Fig. 2	Pushb(B1,B2)	Nextto(B1,B2)
2.	5	Fig. 2	PTD(B1,D1)	Inroom(B1,R2) & Inroom(Robot,R2)
3.	2	Fig. 2	Gotob(B1), Pushd(B1,D1)	Nextto(B1,D1) & Nextto(Robot,D1)

Figure 3. Subproblems of the problem in Figure 2.

is created in this situation. This completes the activity at criticality level 5.

The next discrepancy shows up at level 2 because the preconditions of PTD at this level are not satisfied. ABSTRIPS sets up subproblem 3 in Figure 3, and notes that Pushb(BI, DI) is relevant. However, this operator is not directly applicable because the level 2 predicates in its precondition are not satisfied. Gotob(BI) rectifies this situation and hence (Gotob(BI),Pushd(BI, DI)) is applied to the initial state which results in a state s in the domain of PTD(DI). Since this subproblem is solved, ABSTRIPS applies PTD(DI) to s which yields a new state t and then attempts to transform t into the domain of Pushb(BI, B2). All of the level 2 predicates in the preconditions of Pushb(BI, B2) are considered because this subproblem is being solved at criticality level 2. But, since t is already in the domain of Pushb(BI, B2), this subproblem is trivially solved. Hence, Pushb(BI, B2) is applied to t and ABSTRIPS attempts to transform the result into the set of desired states. But the result is already a desired state and thus this subproblem also has a trivial, solution. Since criticality level 2 is the smallest level, the solutions of the subproblems comprise a solution of the main problem.

To summarize, ABSTRIPS sets out to solve a problem at a given criticality level by essentially removing from the preconditions of all operators, predicates whose level are less than the given level. After finding a solution it goes to the next level down and adds the predicates at this level back to the preconditions of the appropriate operators. This gives rise to subproblems that ABSTRIPS attempts to solve at the current criticality level, i.e., with lower level predicates removed from operator preconditions. This process is repeated until the subproblems at the smallest criticality level, which is 2 in the above example, are solved. Of course, there is search involved at all levels because at any given level a problem may have several different solutions some of which may generate unsolvable subproblems .

4. Planning

ABSTRIPS is quite similar to what Newell & Simon (1972) call planning. ABSTRIPS and planning differ in 2 major respects:

P1. Planning uses an entirely new problem solving space—both states and operators.

P2. Planning uses only 1 level of abstraction while ABSTRIPS has one for each criticality level except the lowest which is the problem space itself.

Although planning does not presuppose criticality levels, they can be used to define the planning space. We will apply planning to the example of the previous section to contrast the two methods.

In the planning space of the problem in Figure 2 all of the level 2 predicates are removed. (In this example Nextto is the only level 2 predicate.) Thus, a state in the planning space will be described exclusively by level 5 and level 6 predicates. All level 2 predicates are removed from operators to get the planning space operators. This includes the level 2 predicates in the addition and deletion lists as well as in the preconditions of the operators. Of course, removing these predicates causes some operators to become identity maps, e.g. Pushb, in which case they are not used in the planning space. All level 2 predicates are removed from the formula that describes the desired states to get its analogue in the planning space. Hence, the planning desired states must be given by Inroom(BI, R2) & Inroom (B2, R2). Removing the level 2 predicates from the description of the desired states in Figure 2, gives the null description which represents the set of all states.

The solution to the problem in Figure 2 in the planning space is the single operator PTD(B1, DI). This generates 2 subproblems in the original problem space: transforming the initial state into the domain of PTD (BI, DI), and transforming the result of applying PTD (BI, DI) to the solution of the first subproblem, into a desired state.

In general, planning is a two phase process: the first phase is to solve the problem in the planning space. The second phase is to elaborate the solution so that It works in the original problem space. This is done by inserting operators into the solution that was found in the planning space. The operators to be inserted are found by solving subproblems of transforming the appropriate states into the domains of the appropriate operators.

£. Formalization of ABSTRIPSLand„Plaiming

Our formal model of ABSTRIPS is based on two assumptions in addition to the above description.

A1. The set of desired states is processed in the same way as the preconditions of operators.

A2. When solving a subproblem at criticality level i, ABSTRIPS will reject any subsubproblems of criticality level greater than i.

By the first assumption we mean that at criticality level i, ABSTRIPS will ignore all predicates less than level i in the description of the desired states. This is what planning does. For example, the Nextto(B1, B2) in the goal states of Figure 3 would be considered at level 2 but not at level 6. We are not suggesting that this is a good idea, but without it and A2 we could not characterize the solutions that ABSTRIPS can find. This will be discussed further in section 7.

The second assumption has to do with efficiency of search. Suppose ABSTRIPS is working at criticalitv level 2 on a Nextto predicate and an incorrect Inroom predicate is generated by attempting to apply an operator. "Fixing" the Inroom predicate is a level 5 subproblem, but when it is considered at level 2 it is much more difficult bee ause all of the level 2 and level 5 predicates are considered to be part of the operator preconditions. Assumption 2 above specifies that all such subproblems are rejected as being too difficult.

At this point it is convenient to formalize the notion of the difficulty of a subproblem. A subproblem with initial state s and desired states T has difficulty i if there is a level i predicate in T which is not satisfied by s and all predicates of higher criticality level in T are satisfied by s. We define $D_i(s, T)$ to be true if and only if the subproblem defined by s and T has difficulty i.

Only certain operators will be used to solve the subproblems of difficulty i; these operators will be denoted by H_i. To define H_i we first define the notion of relevance and irrelevance. The operator f is irrelevant to D_i if for all s and T, $D_i(s, T)$ iff $D_i(f(s), T)$. That is, f is irrelevant to D_i if it never changes the difficulty level of a subproblem of difficulty i. All operators f that are not irrelevant to D_i are relevant to D_i. We can now define H_i to be the set of operators which are relevant to D_i but irrelevant to D_j for $j > i$.

The reason for these definitions is that all subproblems that ABSTRIPS attempts to solve at criticality level i have difficulty i; i.e., $D_i(s, T)$ where s is the initial state and T is the desired states. In fact the reason for A1 and A2 is to make this statement true. ABSTRIPS actually violates A1 and hence considers some level 2 problems at level 6 as shown by the example in section 3. It is not clear from the description in Sacerdoti (1974) whether or not ABSTRIPS adheres to A2. In any case our formal model assumes both A1 and A2. The only operators that ABSTRIPS uses at criticality level i are in H_i. Of course, this is true because all subproblems at this level have difficulty i.

We formalize ABSTRIPS as a process which searches exhaustively, for well stratified solutions as defined below. Let $(f_1,\ldots,f_k)$ be the solution of a subproblem whose initial state is s and whose desired states are given by T. That is, $f_k(\ldots f_1(s)\ldots) \in T$ and each intermediate state is in the domain of the appropriate operator. Below we will denote a subproblem by a pair (s,T) in which s is the current state and T is the set of desired states and S_f will denote the domain of the operator f. Define a non-empty subsequence $(f_{i_1}, f_{i_2}, \ldots, f_{i_t})$ of the solution and an interger m such that

i) $f_{i_q} \in H_m$ for all q $(1 \le q \le t)$;

ii) $f_j \notin H_p$ for $1 \le j \le k$ & $p > m$;

iii) $f_j \notin H_m$ unless $j = i_q$ where $1 \le q \le t$.

This subsequence defines the following set of subproblems and a corresponding solution for each.

1. If $i_1 > 1$, define $(f_1,\ldots,f_{i_1-1})$ to be the solution of $(s, S_{f_{i_1}})$.

2. For all j when $1 \le j \le t-1$ and $i_j + 1 < i_{j+1}$ define $(f_{i_j+1}, f_{i_j+2}, \ldots, f_{i_{j+1}-1})$ to be the solution of the problem $(f_{i_j}(\ldots f_1(s)\ldots), S_{f_{i_{j+1}}})$.

3. If $i_t < k$ then define $(f_{i_t+1}, f_{i_t+2}, \ldots, f_k)$ to be the solution of $(f_{i_t}(\ldots f_1(s)\ldots), T)$.

Such a subsequence of a solution is called the stratification of the solution and the subproblems it defines are called its stratified subproblems.

This definition is conceptually much simpler than it appears to be, as shown by the following example. Suppose that (f_3, f_5, f_6, f_{10}) is the stratification of the solution $(f_1,f_2,\ldots,f_{10})$ to the problem (s,W). This stratification "groups"

the solution as $(f_1f_2)f_3(f_4)f_5f_6(f_7f_8f_9)f_{10}$. Each of the parenthesized quantities are solutions corresponding to stratified subproblems. For example, (f_7,f_8,f_9) is the corresponding solution to $(f_6(\ldots f_1(s)\ldots), S_{f_{10}})$. Note that there is no subproblem between f_5 and f_6. In addition, f_3, f_5, f_6 and f_{10} are in some H_m and all the other f's are in "smaller" H's.

A solution $(f_1,\ldots,f_k)$ with a stratification $(f_{i_1},\ldots,f_{i_t})$ is called <u>well stratified</u> if $D_m(s, T)$ and $D_m(f_{i_j}(\ldots f_1(s)\ldots), T)$ for $1 \leq j < t$, and the corresponding solution of each of the stratified subproblems is well stratified. That is, each subproblem is of difficulty m. (Note that i-iii above only refer to the H's and not the D's.) The definition of well stratified is recursive since it requires all of its subproblems to be well-stratified also. By the way the stratification subsequence is defined, m decreases with each level of recursion. Continuing with the above example, the solution is well stratified if all the difficulty of all top level subproblems is m; e.g., $D_m(f_3(f_2(f_1(s))),W)$. In addition all of the stratified subproblems must be well stratified. For example, (f_8) might be the stratification of (f_7,f_8,f_9) which groups it as $(f_7)f_8(f_9)$. This indicates that f_7 is the corresponding solution of $(f_6(\cdots f_1(s)\cdots),S_{f_8})$. The difficulty of the subproblems in this stratification is m' where $f_8 \in H_{m'}$, and $m' < m$; e.g., $D_{m'}(f_8(\cdots f_1(s)\cdots),S_{f_{10}})$.

To summarize then our formal model of ABSTRIPS is an exhaustive search procedure for well stratified solutions. Of course, ABSTRIPS uses a specific search strategy; i.e. first it looks for the stratification subsequence from "left to right" and then for subproblem solutions and backtracks in its own special way. We have not modeled this search order. In addition, our formal model is only approximate since it has A1 and A2 built into it.

Our formal model of planning is an exhaustive search procedure for well stratified solutions that have 2 levels. Again we do not model the order of search. Since planning uses A1, the only approximation used by our model is A2.

6. Totally-Ordered Solutions

GPS uses the strategy of working on difficult subproblems first, saving the easier subproblems for later. The difference ordering is used to specify the difficulty of a subproblem. This strategy is not complete in the sense that some problems that have solutions cannot be solved using this strategy. Ernst (1969) has characterized the class of solutions that GPS can find as totally-ordered solutions. Intuitively, a totally ordered solution is one which generates a sequence of states in which the difficulty is monotonically nonincreasing. In addition, all subproblem solutions are also totally ordered. The subproblems of a solution are defined in terms of m_o and i_o. Given a solution $(f_1,\ldots,f_k)$ of a subproblem (s, T), let m_o and i_o be defined as follows:

i) $f_j \notin H_p$ for $p > m_o$ and $1 \leq j \leq k$;

ii) $1 \leq i_o \leq k$

iii) $f_{i_o} \in H_{m_o}$

iv) $f_j \notin H_{m_o}$ for $1 \leq j < i_o$

That is m_o is the "highest level" of the operators in the solution and f_{i_o} is the first of such operators occuring in the solution. Let M(s, T) denote the difficulty of the subproblem. That is, M(s, T) = c if $D_c(s, T)$ and not $D_j(s, T)$ for $j > c$. Then $(f_1,\ldots,f_k)$ is a <u>totally ordered</u> solution to the subproblem (s, T) if

i) $M(f_j(\ldots f_1(s)\ldots), T) \geq M(f_{j+1}(\ldots f_1(s)\ldots),T)$ for $1 \leq j < k$;

ii) if $i_o > 1$ then $(f_1,\ldots,f_{i_o})$ is a totally ordered solution to the subproblem $(s,S_{f_{i_o}})$.

iii) if $k > i_o$ then $(f_{i_o+1},\ldots,f_k)$ is a totally ordered solution to the subproblem $(f_{i_o}(\ldots f_1(s)\ldots), T)$.

Totally-ordered is not only a characterization of the solutions that GPS can find but is also a characterization of the solutions that either ABSTRIPS or planning can find.

<u>Theorem</u>: Any well stratifies solution is totally ordered. The proof will not be given in this paper; the proof of this theorem, stated in a slightly different formalism, is given in Banerji and Ernst (1977). Although we have not proven the converse of the theorem we believe that it is true when "good" difference informations is used. If so, then the set of totally ordered solutions is a characterization of the class of solutions

that ABSTRIPS or planning or any other process that can find all and only well stratified solutions can find.

In Ernst (1969) a triangular table of connections was shown to be good difference information for GPS and it appears that this also specifies good parameters for ABSTRIPS and planning. Rather than giving a formal definition of a triangular table of connections we will give an example of it using the Fool's Disks problem. Figure 4 gives the initial state of the Fool's Disks problem, in which there are 4 concentric disks each containing eight numbers. These numbers line up so as to form 8 columns radiating from the center of the disks. A move consists of rotating one of the disks independent of the others. The desired state is one in which each of the 8 radial columns sums to 12.

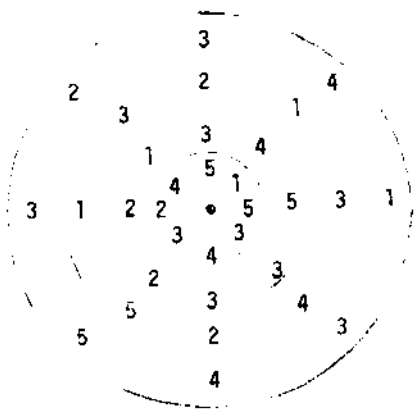


Figure 4.

The initial state in the Fool's Disks problem.

	1/8, 3/8, 5/8, 7/8 revolution	1/4, 3/4 revolution	1/2 revolution
The vertical and horizontal diameters does not sum to 48.	1	0	0
Some diameter does not sum to 24.	1	1	0
Some radius does not sum to 12.	1	1	1

Figure 5.

A triangular table of connections for Fool's Disks.

A triangular table of connections for this problem is given in Figure 5. A 0 entry in the table indicates that a difference is invariant over the operators. For example, the 0 in the upper right corner of Figure 5 indicates that turning a disk 1/2 revolution does not change the sum of the horizontal and vertical diameters. Note that Figure 3 has all O's above the main diagonal and no O's on it. The entries below the main diagonal are unimportant. The difference ordering is just the row order; i.e., the top row heading is the most difficult difference while the bottom is the easiest.

GPS only uses the entries on the main diagonal to solve the problem. Hence, first 1/8, 3/8, 5/8 and 7/8 revolution of disks are used to remove the most difficult difference, i.e. used to get the horizontal and vertical diameters to sum to 48. Next, 1/4 and 3/4 revolutions are used to get each diameters to sum to 24. And finally the 180^c moves are used to get a desired state. Notice that once a difference is removed it is never reintroduced because of the triangularity of the table in Figure 5.

7. Discussion

In the previous 2 sections we made a number of highly technical definitions ending with a somewhat cryptic theorem, but what can be concluded from all of this? There are 3 major points, listed below; they are followed by a more general discussion.

Point 1. Given the "same" heuristic parameters planning, ABSTRIPS and GPS (or at least our formal models which approximate them) can all solve precisely the same class of problems, i.e. those that have totally ordered solutions. The relationship of parameters of the different methods is described below but note that the Di of Section 5 are just differences for GPS. This result is somewhat surprising because tne search space of GPS is quite different than that of planning or ABSTRIPS. For example, for all states s generated by GPS there is a path from the initial state to s. This is not true for planning or ABSTRIPS because many of the subproblems may not be solvable. For any given problem probably one of the three methods is more efficient than the others, but none is best for all problems.

Point 2. Selecting heuristic parameters can have a much more drastic effect on the search efficiency than selecting one of the three methods, as shown by the Fool's Disks example above. The reason for this is that all of the methods are designed to search efficiently for totally-ordered solutions; they will not even consider subproblems that give rise to unordered solutions. Totally-ordered is defined in terms of the parameters which are problem dependent, and hence the parameters determine whether the class of totally ordered solutions is the same as the class of all solutions or whether the former is much smaller than the latter. In fact, one can view

the problem of selecting good parameters as the problem of making the class of totally ordered solutions as small as possible without eliminating all solutions to problems of interest.

Point 3. Any parameters which are good (in the sense of guiding search) for one method will have analogues which will be good for the other methods. A triangular table of connections (see Section 6) is a property of good difference information for GPS. Another property of good differences is given in Banerji and Ernst (1977). These properties, after suitable translation, are also properties of good parameters of planning and ABSTRIPS. Currently, we are implementing a program to discover difference information for GPS which satisfies these properties. Hence, this program effectively will discover good parameters for planning and ABSTRIPS also.

We feel that point 2 is the most important of the three points. Not much is known about the process of selecting good parameters. Some initial work on this topic is outlined under point 3 but more research is needed on this topic.

To understand the above points, we need to know the relationship among the parameters of the 3 methods. The D, of Section 5 are just the differences of GPS. That is, subproblem (s, T) possesses difference d if D,(s, T) where d corresponds to D.. The subscripts on the D's give the difference ordering, e.g., the difference corresponding to D„ in the example in Figure 2 is the easiest (smallest) difference because 2 is the smallest crlticality level. The table of connections indicate that the operators in H. should be used to reduce D.. Note that in our formal models of both ABSTRIPS and GPS an operator f may be relevant to D. but not in H,. This will happen when f e H for some j > 1 which indicates that f is relevant to the more difficult difference D* and hence is used solely for the purpose of reducing D.. Including f in H. would give rise to more search but not to more totally-ordered solutions.

Planning, in many respects, looks like ABSTRIPS with 2 criticality levels. In fact our formal model of the previous section is really a hybrid of ABSTRIPS and planning; it has many levels of abstraction like ABSTRIPS, and like planning assumes AI. The heuristic parameters of planning are basicly the difference information of GPS. Although planning requires a entire planning space as input, this information is really derived from the difference orderinc. In fact one of the innovations in ABSTRIPS is its ability to automatically generate the various planning or abstracted spaces from the criticality levels of the predicates.

Our formal model of ABSTRIPS uses assumptions AI and A2 (Section 5) which is a deviation from the real ABSTRIPS. Why might ABSTRIPS want to violate AI and/or A2? The answer lies in the fact that the class of totally-ordered solutions may be a little too restrictive because some solutions may not be totally-ordered. Violating AI and/or A2 allows ABSTRIPS to find solutions that are "almost but not quite" totally-ordered. We do not know how to measure the degree of unorderedness or what these violations due to the efficiency of search. To see how an unordered solution is generated note that ABSTRIPS' solution to the problem in Figure 2 is unordered because the difficulty of the original problem is level 2 whereas the difficulty of some of the subproblems is level 5. This unordered solution is a result of violating AI. To have planning solve the problem in Figure 2 the desired states had to be more completely specified as described in Section 4 because planning adheres to AI.

What is the relationship of the properties of good differences in (Ernst, 1969) to the parameters of ABSTRIPS? As mentioned above the criticality levels of predicates basically specify the differences and difference ordering. If the table of connections is generated as defined in Section 5 (i.e. use operators in H, to reduce D), then the properties of good differences are automatically satisfied except that some of the H, may be empty. What this means is that if all solutions of a problem contain subproblems of difficulty I, then the problem has no totally-ordered solution. In other words the criticality level assigned to predicates is not good. A better approach is to ask what assignment of criticality levels results in no empty H's. This is very similar to the approach in Eavarone and Ernst (1970) .

We feel that the biggest restriction in ABSTRIPS is its use of a very restrictive class of differences, i.e., those defined by the predicates in the problem specification. Most of the good ideas in ABSTRIPS apply to a more general class of differences; and we hope that the restriction is removed in the future. As an example of a difference not defined by a predicate in the problem specification, the logic task in Newell and Simon (1972) uses the number of different proposition letters in an expression as a difference. Figure 5 also contains some more complex differences. We believe that ABSTRIPS can be extended to a wider class of differences such as these.

References

1. Banerji, R. B. and Ernst, G. W., Some Properties of GPS-Type Problem Solvers, Report //1179, Computer Engineering and Information Sciences Dept., Case Western Reserve University, 1977.

2. Eavarone, D. S. and Ernst, G. W., A Program that Discovers Good Difference Orderings and Table of Connections for GPS, 1970 IEEE Systems Science and Cybernetics Conference Record. 1970.

3. Ernst, G. W., Banerji, R. B., Hookway, R. J., Oyen, R. A. and Shaffer, D. E., Mechanical Discovery of Certain Heuristics, Report #1136, Computer Engineering Department, Case Western Reserve University, 1974.

4. Ernst, G. W., "Sufficient Conditions for the Success of GPS," Journal of the ACM, October, 1969.

5. Ernst, G. W. and Newell, A., GPS: A Case Study in Generality and Problem Solving, Academic Press, 1969.

6. Fikes, R. E. and Nilsson, N. J., "STRIPS: A New approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, 1971, pp. 189-208.

7. Newell, A. and Simon, H. A., Human Problem Solving, Prentice-Hall, 1972.

8. Sacerdoti, E. D., "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence, 1974, pp. 115-135.

A THEORY FOR THE COMPLETE MECHANIZATION OF A GPS-TYPE PROBLEM SOLVER

R. B. Banerji
Temple University
Philadelphia, Pa. 19122

C. W. Ernst
Case Western Reserve University
Cleveland, Oh. 44106

Abstract

The data structure that drives the General Problem Solver is the Connection Table. This paper describes the theoretical basis for the automatic construction of this table by computer programs. The programs for this purpose have been developed at the Case Western Reserve University. They basically isolate certain attributes of the problem states which are invariant under certain moves and then put those attributes together to "triangularize" the Connection Table.

Descriptive Terms

Theory of Heuristics, General Problem Solver

1. Introduction

According to our view of mechanical problem solving, there are a number of different problem solving methods each of which has problem dependent parameters. For each method there is a condition which specifies the properties the parameters should have in order for the method to "work". Hence, we view problem solving as the two phase process shown in Figure 1. In the first phase, the method's condition is used to generate "good" parameters for the method. The input to this phase is the problem specification, since the parameters are usually problem dependent. The output is either good parameters or an indication that this method should not be used on the given problem. The second phase attempts to solve the problem (as specified at the input to the first phase) using the method with the parameters generated in the first phase. Of course, there is a Dicture like Figure 1 for each method, and if the first method is not applicable, we merely move on to the next method and attempt to use it.

So far, all the methods studied this way [Coray(1970), Ernst(1969), Banerj1(1971)] seem to depend on the recognition of certain attributes of the problem states which remain invariant under some of the moves. We have previously published two reports on the design and implementation of a program which would isolate some of these attributes on the basis of the problem description [Ernst e£a_l(1974), >Oyen(1975)].

Our present effort deals with the combination of the invariant properties to yield the Connection Table of GPS [Newell & Simon(1963), Ernst & Newell(1969)]. Our efforts in using our previous theory [Ernst(1969)] for the purpose of mechanizing the heuristic were not successful, because a difference (good or bad) was a binary relation between states and sets of states, i.e., a subset of S x 2s where S is the set of problem states, which is a complicated concept.

In an attempt to simplify matters we said, "What if a difference were just a set of states?" In this case, a state s possesses difference D, if s *i* D. With this simple view we can visualize GPS's strategy as follows (Figure 2).

S is the set of all states. W is the set of goal states: $W \subset D' \subset D \subset S$, and s_0 is the initial state. GPS would attempt to solve the problem as follows:

1. Find a path from SO to some state *s* D.
2. Find a path from s to some state s_2 e D^1 but the path must be entirely inside D.
3. Find a path from s_2 to some state s e W but the path must be entirely inside D'.

In step 1 GPS is removing the most difficult difference D. In step 2 the second most difficult difference is being removed without reintroducing D. The easiest difference W is being removed in step 3 without reintroducing either D or D'.

A point ought to be made here about the original GPS which was a somewhat more general device than the one we are describing here in that,

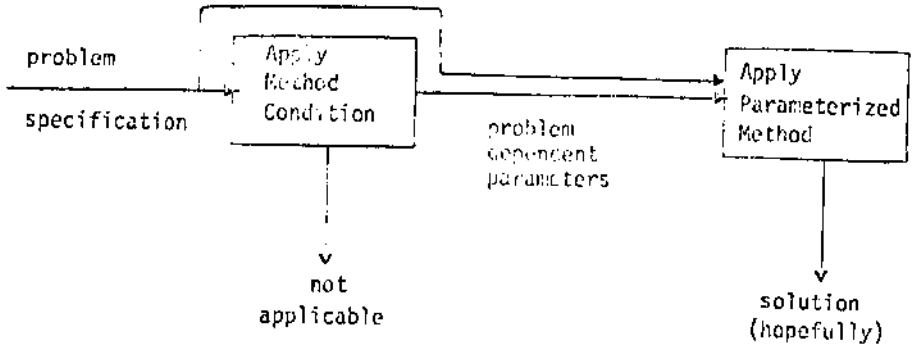
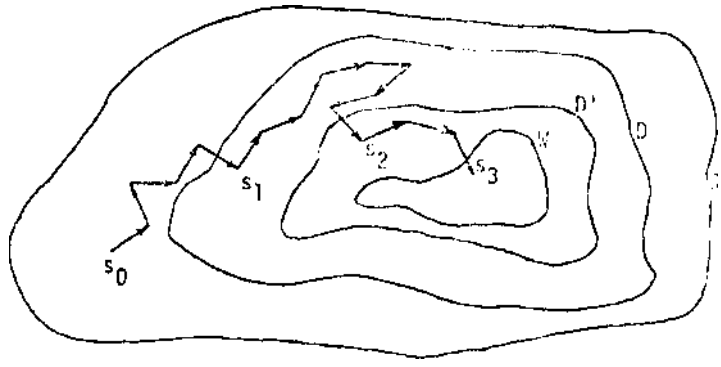


Figure 1. Our view of mechanical problem solving.



]'i£i:re 2

while removing an easier difference, a more difficult difference could get reintroduced. The only search pruning involved in this general case was that involved in the relevance of moves to differences (vide ultra). The extra constraint we have introduced here (and one which also characterizes our previous work [Ernst(1969)]) constrains the search for greater efficiency, while at the same time it neglects a certain class of solutions. Our present analysis follows the same line.

It is probably somewhat counterintuitive that the most difficult difference contains all of the other differences as subsets. One would normally think that the larger the set of states, the easier it would be to "get" inside of it. Also, one does not normally think of W as a difference. But this somewhat unintuitive picture works quite well.

Consider, for example the Fool's Disk problem. Figure 3 gives the initial state of the Fool's Disk problem, in which there are 4 concentric disks each containing 8 numbers. These numbers line up so as to form 8 columns radiating from the center of the disks. A move consists of rotating one of the disks independently of the others. The desired state is one in which each of the 8 radial columns sums to 12.

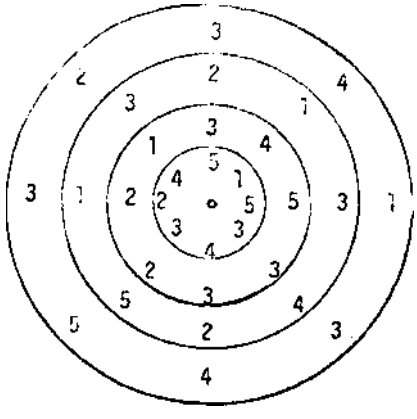


Figure 3
The initial state in the Fool's Disk problem

This problem fits the above picture exactly. D' is the set of states in which each diameter sums to 24, while D is the set of states in which the sum of the N, E, S, and W radii is 48. To keep the path from s_1 to s_2 in D, GPS only considers moves which rotate disks 90°. To get from s_1 to s_3, GPS rotates disks by 180° only.

One might be disturbed that each difference contains all of the easier differences. This is not a difficulty, because any set of differences not possessing this property can be converted to differences which have this property. (In fact, our theory [Banerji &. Ernst (1977)] does not require this "nesting" of differences.) Consider, for example, the 3 disk Tower of Hanoi in which we are trying to move all disks to peg P_3. Let D. be the set of states in which disk i is on P3 where disk 3 is the largest disk. Then, one might think of using D_1, D_2, and D_3 as differences for this problem. These are essentially the differences that were given to GPS for this problem. Certainly these sets are not perfectly nested. However, this set of differences can be converted to the above picture by intersecting them together, i.e., $D = D_1$, $D' = D_3 \cap D_2$, and $W = D_3 \cap D_2 \cap D_1$.

A more disturbing feature of this set of differences is that they are only useful when the set of desired states is W. In the original GPS (as well as In our previous work) the same set of differences served to characterize all subgoals - including "make such and such a move applicable." This is not the case anymore. If, for example, the set of desired states is the domain of the operator which moves disk 3 from P_1 to P_3, the D_1 seems to be a useless set of differences. The difficulty is that we have "built" W into the differences. We did this on purpose to simplify the differences to allow mechanization. Our original theory had differences as binary relations between states and sets of states. If we specify the latter to be W, then we are left with a monadic relation on states which is just a set of states. But how are we going to accommodate goals other than W?

The key to answering this question is that not only W but also the domains of operators can be the goals of subproblems. Since the number of operators is usually quite small, we will use a dif-

ferent set of differences for the domain of each operator being the goal of a subproblem.

These modifications were introduced in our theory of GPS to make it easier for person or machine to discover "good" differences. An added advantage of the modified GPS is that it can easily handle problems in which the sets of differences for subproblems with different desired states are truly different.

The above discussions will, we hope, serve as a motivation for the changes we have introduced in the theory. We do not plan to give a formal counterpart to these motivations or exhibit a formal connection between the old and the new theories. Instead, we shall exhibit and motivate the new theory ab initio so that readers unacquainted with our previous work will find the discussion self-contained. We shall, of course, assume that the reader has had former acquaintance with GPS [Ernst & Newell(1969)1.

In the next section we give a formal definition of good differences. This is followed by an example of good differences and how they are used by GPS. Section 4 characterizes the class of solutions that GPS can find given the kind of differences described in Section 2.

2• Definition of Good Differences

Since GPS builds its solution to a problem by setting up subproblems, we cannot build this theory by defining what a problem is but rather by defining a larger structure in which a class of subproblems can be embedded. Also, this structure should contain the concepts which reflect the idea of differences and the connection table. We shall call this structure the problem domain, "domain" for short. As in the previous models, we start with a set S of states and a subset W of S, consisting of winning states. We also have a set C of partial functions (mapping subsets of S into S) which we shall call moves or operators. If f e G is a move, we shall denote by Sf its domain of definition, i.e., states where f is an applicable move. Since subgoals in GPS have the form "make move f applicable," these Sf, for various members f of G, serve as winning sets for subproblems just as W serves for a problem. The class of all these sets (W and Sf for various f) we shall call X. For each set in this class we also define the differences which allow GPS to work on them. That is, for each T e X (T being either W or S_f for some f e G) we define a class of sets Tj, $T_?$, T3 ..,T_n with the property that Tj n T n...T_n"= T. The actual number n of specified differences of course depends on the set T chosen. So, instead of writing n we shall write n(T) when there is any doubt as to which subproblem we are talking about. Also, for reasons of convenience of discussion we shall often give the name TQ to T and call S itself, $T_{n(T)+1}$.

It may be appropriate at this point to point out that the T^ catches the idea of difference in that when a state s *i* $T_{i\$}$ a difference exists between s and T. The higher i is, the larger the difference is.

The next important concept in GPS, of course, is that of relevance of a move to a difference. The major assumption on which GPS theory is based is that a solution can be obtained by removing the higher ("more difficult") differences before the lower differences and never reintroducing higher differences once they are removed. A difference Is considered higher than others, if fewer moves are available to remove it. Of course, S or $^{T}{}_{n}('T)+i^{are}$ the most difficult differences to remove, since no move changes a state to a non-state. Let H1 c G be the set of moves which, when applicable, affects the position of the state with respect to T1. Instead of making the very strong assumption that moves in H1 bring all states outside T1 into T,, we shall make the more realistic assumption that these moves remove the states from *Ti* when applied. This assumption seems "backward" to many, in spite of the fact that in most real problems, relevance of moves does appear that way and was used that way even in the original GPS. In our difference-finding program, a state is characterized by giving the values of certain attributes for the state. A winning state is characterized by specifying that some of the attributes should have specific unique values. To find mechanically that a certain move is relevant to a certain difference T., we test whether the move changes the values of those attributes which characterize T..

It is this "property-changing" characterization for moves which gives relevance the backward appearance. Of the various values to which the attribute can change, only one characterizes the win states. Hence, it is not to be expected that merely changing the value of a property yields a win value. On the other hand, if it already has a win value, changing it certainly changes it to a non-win value.

Another important characteristic we demand of the moves in $W_{\pm}$ (called triangularity of the difference table in the previous theory) is that Hi does not affect the differences higher than Ti, i.e., is irrelevant to Tj for j > I. Thus, once a state is in T_i, as long as we use moves in Hj with j < i, $T_{\pm}$ will not be reintroduced.

This effort shows up nicely in the difference transformation tables of GPS. If we arrange the Ti's from top to bottom in decreasing order of i and the H. from left to right in decreasing order, and mark the (i,j) cell with a 1 if moves in Hj are relevant to $T_{\pm t}$ then the upper right half of the table will be blank. Tables of this nature we call triangular tables, and differences which give rise to triangular tables we call good differences .

We define the maximum difference between T and s, M(s, T), to be i if s *4* $T_{\pm}$ and s t Tj for all j greater than i.

Implicit in the above definitions is an ordering of the $T_{\pm}$ (and the Hi) which corresponds to the difference ordering of GPS. The most difficult difference is T_n, while the easiest difference is $T_{\pm}$. GPS's basic problem solving strategy is to work on hard differences first and easy differences last. GPS accomplishes this (as discussed in Section 4) by using the following to guide its search:

51 To reduce the maximum difference T-j, use only operators in H^.

52 Suppose a subproblem were generated to reduce difference T_{i4} Then do not use the operators in Hj, i < j < n to solve the subproblem.

Rule SI was in our previous theory. Note that there may be many other operators besides H. which are relevant to T^ because we have placed no conditions on Hj for j > i. SI causes GPS to ignore such Hj even though some of its operators may be relevant to Tj_.

The purpose of S2 is to require subproblems to be easier than the problem for which they are created. In our previous theory this was accomplished by requiring the differences of a problem to be harder than the differences of its subproblems. This is no longer possible, because we cannot compare subproblem differences to problem differences because they will have different goals and hence different differences. However, S2 can be used, because all differences are reduced by the same operators. Note that S2 is applied recursively. That is, suppose Fl and F2 are the sets of operators according to S2 that cannot be used on subproblems SP1 and SP2, respectively. If SP2 is a subproblem of SP1, then GPS will not use any operator in Fl u F2 to solve SP2, because, the restrictions on SPI are passed down to all of its subproblems.

3. An Example of Good Differences

The definitions above appear quite formidable and somewhat unlike GPS. A simple example will clarify things. For our example we have chosen that old chestnut about the monkey and the bananas, a formulation of which is given in Figure 4. We have chosen this example because it has (non-trivial) good differences, subproblems are created in solving it, and it is simple.

One way to formalize the differences above is by positing that there is a separate table of connections for each goal which is either W or the domain of an operator. Figure 5 illustrates Monkey and Bananas this way. The I's indicate which operators are relevant to which differences. The O's indicate irrelevance. A move is neither relevant nor irrelevant - we use a question mark. Note that the bottom row heading of each table is just the goal and that each row is a subset of the row above it. Although our theory does not require these properties, they make things easier to visualize as discussed at the beginning of

A state is a 3 place vector whose components are the monkey's position, the box's position, and the contents of the monkey's hand.

A win is a state in which the bananas are in the monkey's hand.

(Walk, Climb, Push, Grab)

Walk The monkey walks to someplace in the room.

Climb The monkey climbs onto the box, i.e., the monkey's position becomes ONBOX. Climb is applicable only when the monkey's position equals the box's position.

Push The monkey pushes the box to some in the room. Push is applicable only when the monkey's position equals the box's position.

Grab The monkey grabs the bananas. Grab is applicable only when the monkey is on the box, and the box is under the bananas.

Figure 4

A Formulation of the Monkey and Bananas Problem

Section 2.

The row headings are the Tj_ in the definitions of Section 2, and the column headings are the Hi. The definitions of the T_A and the Hi require that the tables of connection are triangular in the sense that the main diagonal and all entires above it are 0. In addition, the subdiagonal (the diagonal immediately below the main diagonal) contains all l's.

Walk is a total function on S, hence its domain is S. We do not need a table of connections for such an operator, because a subproblem of getting into its domain will never be created. We included the table of connection for Walk in Figure 4, because the degenerate case of a definition often helps one understand the definition.

If a column of an operator is all 0's, then that operator will never remove a state from the goal set and will never transform a state outside the goal set into the goal set. An all 0 row indicates that no operator will add or remove a state to the T^ which labels the row.

The above is an example of "difference information" which satisfies our definition of good in Section 2. The most important feature of the tables in Figure 5 is that the triangularity constraint orders the rows (and the columns). This row ordering is the difference ordering - difficult differences are at the top of a table, and easy differences are at the bottom. Of course, there may be several different row orderings

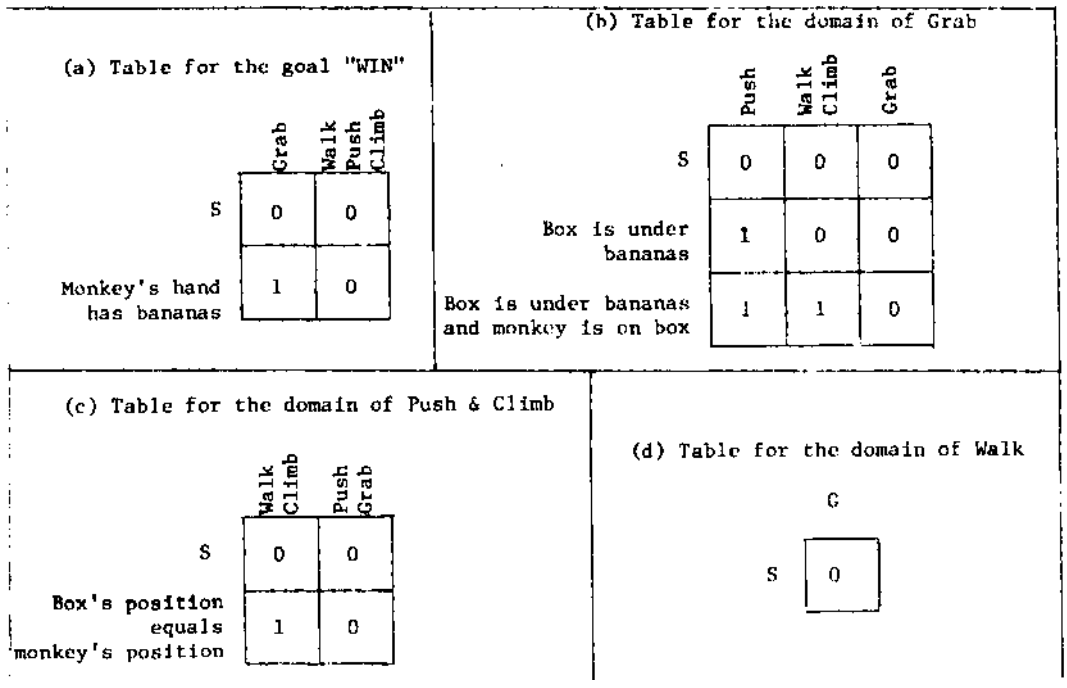


Figure 5
The Table of Connections for
each goal in Monkey and Bananas

which gives rise to a triangular table, in which case any one of them will satisfy our formal definition of good.

Now we can describe how GPS solves Monkey and Bananas using the difference information in Figure 5. Suppose that in the initial state S0 the monkey's hand is empty and the box is not under the bananas. Then the largest difference, M(S0,W), is that the monkey's hand is empty, hence GPS attempts to apply Grab. But so *i* Sgrab, hence GPS sets up the subprobiem of transforming S0 into Sgrab, b^{ut} Grab cannot be used in solving the subproblem because of rule S2.

To solve the subprobiem, GPS attempts to reduce the difference that the box is not under the bananas since this is M(so, SG_{rat}>)• Hence, GPS attempts to apply Push which is not applicable, and the subprobiem of transforming SQ into Sp_{ush} is generated, but S2 restricts the solution of this subprobiem to the operators Walk & Climb. The remaining part of solving this problem is quite straightforward and similar to the way the usual GPS works.

4. Totally-Ordered Solutions

The above discussion raises the question, "Can GPS solve all problems which have a solution?" The answer is no (which can be shown quite easily), because'the differences, together with rules SI and S2, prevent GPS from looking at sequences of operators that may be necessary to find a solution. Hence, the question becomes, "Can we somehow characterize the class of problems which GPS can solve?" The purpose of this section is to show that GPS can solve any problem that has a totally-ordered solution which is our characterization of the class of problems that GPS can solve. Intuitively a totally-ordered solution is one in which one never introduces a difference more difficult than the current differences at any point in solving the problem. This applies to all subproblems at all levels. We will indicate in Theorem 1 that using rules Sl and S2 with good differences produces the class of totally-ordered solutions.

To exhibit this result we have to give definitions of what a problem is and what a solution is. Given a domain as defined in Section 2, a problem is defined by a state $s \in S$, a subset F of G (the set of moves), and a goal $T \subset X$. A solution of a problem defined by the triple <s, F, T> is a sequence of moves $(f_1,\ldots,f_k)$, each f_i $(1 \le i \le k)$ being an element of F, and such that $sf_1,\ldots f_i$ is defined for all i $(1 \le i \le k)$ and $sf_1\ldots f_k \in T$.

At this point we invoke the partition $H_0(T)$, $H_1(T),\ldots,H_{n(T)}(T)$ and recall that each f_i in the above solution is an element of some $H_j(T)$. This yields a sequence $j_1, j_2,\ldots,j_k$ of integers such that for each i, $f_i \in H_{j_i}(T)$. Among them will now occur an integer which is greater than all the integers before it and no less than any element after it, i.e., the "leftmost peak" below, where we have plotted j_i against i.

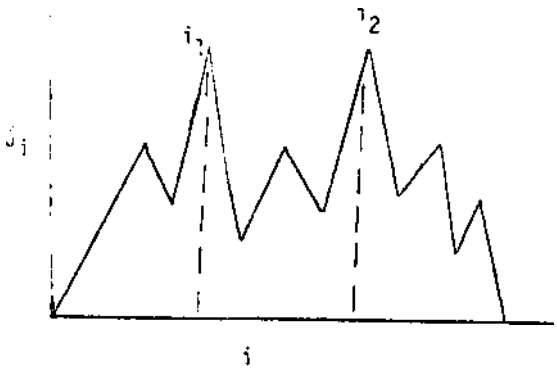


Figure 6

We will initially show that the solution can be interpreted to mean that, at this point, the first effort was made to remove the highest difference between the goal and the initial state s. The moves before this peak are attempts to make it possible to apply the move used at the peak. This sequence (from the start to the peak i_1) will be called the solution to the subproblem, and the sequence between it and the end will be called the solution to the pseudoproblem. The rest of the definitions follow from these considerations.

The triple $<(f_1,\ldots f_{i_1-1}),\ f_{i_1},\ (f_{i_1+1},\ldots f_k)>$ is called the parse of the solution $f_1,\ldots f_k$. It is obvious that given a domain and a solution of any problem <s, F, T>, the parse exists and is unique.

Given the parse as above, we define two problems. The first, which is defined only in the case that $i_1 > 1$, is

$$<s,\ F\text{-}DR(p),\ S_{\bigcup_{p=1}^{n(T)} i_1}>$$

and will be called the subproblem of <s, F, T> corresponding to the solution $(f_1,\ldots f_k)$. The second, which is defined only when $i_1 < k$, is

$$<sf,\ldots f_{i_1},\ F,\ T>$$

and will be called the pseudo-problem of <s, F, T> corresponding to the solution.

Once again, it is obvious that $f_1,\ldots f_{i_1-1}$ is a solution of the subproblem corresponding to the original solution.

Thus, one can say that any solution can be interpreted, i.e., parsed, to be one in which one seeks to apply moves in $H_m(T)$ with "the highest m," making such a move applicable by using moves only in $H_p(T)$ with $p < m$. However, such an interpretation could be given with any ordered partition, having nothing to do with a difference ordering. The reader will recall that in our original definition of a domain the partition on F was so done as to be in keeping with the differences.

To bring our discussion back to the difference orderings rather than with the partitions on the moves, we introduce another definition. Given a problem <s, F, T> and a solution $(f_1,\ldots,f_k)$ and a parse, the solution is called totally ordered, if

ti) $M(sf_1\ldots f_i, T) \geq M(sf_1\ldots f_{i+1}, T)$ for all i $(1 \leq i < k)$, and

tii) each subproblem and pseudoproblem has its corresponding solution totally ordered.

Totally ordered solutions are of importance in that they characterize the kind of solution that can be found by the technique used by GPS. So far we have not formally defined this technique. The statement of the following theorem formalizes the technique as well as characterizes solutions that can be found by it. However, the statement of the theorem needs the following definition.

Given a problem <s, F, T> in a domain and a solution $f_1,\ldots,f_k$ for it, the problem-set for this solution consists of the problem and the members of the problem set for the solutions to the sub- and pseudo-problem of the original problem, if they exist.

We are now ready to state the major theorem of this paper.

Theorem 1: Given a solution $(f_1, f_2,\ldots f_k)$ of a problem <s, F, T> in a domain, the solution is totally ordered, if and only if for each i $(1 \leq i < k)$, f_i is the second element of the parse of the solution of some problem <s', F', T'> in the problem set, and $f_i \in H_m(T')$ implies $M(s', T') = m$.

Intuitively, this theorem says that, if the solution $(f_1,\ldots f_k)$ were found by a search process guided by rules S1 and S2, then it is totally ordered. In addition, such a search process is capable of finding any totally ordered solution.

We shall not include the proof, because it is a long "walk along the definitions" given above and needs some more inessential pedantry like transfinite induction (on a finite space at that!).

5. Conclusion

We now have a working program [Goldstein (1977)] which, using invariant attributes of the problem [Oyen(1975)], would construct a set of properties T_i and partitions H_i as given in Section 2. These would yield what we have previously called triangular connection tables.

It will be noticed that, as previously warned, Theorem 1 of Section 4 does not assure us of

a solution whenever a triangular difference table exists; one has to be blessed with a totally ordered solution - totally ordered by the ordering mechanically or otherwise chosen in the connection table. We have had various problems in which more than one triangular connection table exist, and yet one can prove that some of the connection tables would not yield a solution. This problem has appeared in other, seemingly closely related, garbs in planning programs for Robots, leading to the work on Non Linear Plans [Sacerdoti(1975)]. The analogous problem in our formalization would be the detection of the nonexistence of totally ordered solutions. One approach, that of the detection of "factorable subproblems" [Goldstein 1977)] will be reported on at a future date.

6. Acknowledgements

The work described in this paper was supported by the National Science Foundation under grant MCS 75-23412 to the Case Western Reserve University. The preparation of the paper was partially supported by them under grant MCS76-0-200 to Temple University.

References

1. Banerji, R. B. "Similarities in Games and their use on Strategy Construction," Proceedings of the Symposium on Computers and Automata, pp. 337-359, Polytechnic Press of the Polytechnic Institute of Brooklyn (1971).

2. Banerji, R. B. and Ernst, G. W., "Strategy Construction Using Homomorphisms Between Games," Artificial Intelligence, Vol. 3, pp. 223-248 (1972).

3. Banerji, R. B. and Ernst, G. W. Ernst, "Some Properties of GPS-type Problem Solvers," Report #1179, Jennings Computing Center, Case Western Reserve University, (January, 1971).

4. Coray, G., "An Algebraic Representation of a Puzzle Solving Heuristic," Publication 67, Dept. d'Informatique, University of Montreal (1970).

5. Ernst, G. W., "Sufficient Conditions for the Success of GPS," Journal of the ACM (October, 1969).

6. Ernst, G. W. and Newell, A. GPS: A Case S udy in Generality and Problem Solving, Academic Press (1969).

7. Ernst, G. W. et. al., "Mechanical Discovery of Certain Heuristics," Report #1136-A, Jennings Computer Center, Case Western Reserve University (January, 1974).

8. Goldstein, M., Unpublished Research (1977).

9. Newell, A. and Simon, H. A., "GPS, A Program that Simulates Human Thought," Computers and Thought, E. Feigenbaum & J. Feldman, eds., pp. 279-293 (McGraw-Hill, 1963).

10. Oyen, R. A., "Mechanical Discovery of Invariances for Problem Solving," Report //1168 Jennings Computing Center, Case Western Reserve University (June 1975).

11. Sacerdoti, E. D., "The Non Linear Nature of Plans," Proceedings of the Fourth International Joint Conference on Artificial Intelligence, pp. 206-214 (1975).

A GENERAL BACKTRACK ALGORITHM THAT ELIMINATES MOST REDUNDANT TESTS

John Gaschnig
Dept. of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

We define a faster algorithm functionally equivalent to the classical backtrack algorithm for assignment problems, of which the Eight Queens puzzle is an elementary example (Fillmore & Williamson 1974, Knuth 1975]. Experimental measurements (figure 1) reveal reduction by a factor of 2.5 for the 8-queens puzzle (factor of 8.7 for 16 queens) in T, the number of pair-tests performed before finding a solution (i.e., first solution). A pair-test in this case determines whether a queen on square (ij, jj) attacks a queen on square (12, j2^ *i*^n ^C^P^U" seconds, net speedup is by a factor of 2.0 and 6.0 for 8- and 16-queens, respectively. 16-queens was solved in 0.14 seconds on a PDP KL/10. The speedup can be attributed to the elimination of almost all redundant tests otherwise recomputed in many parts of the search tree, as indicated in figure 2, which shows the mean number of times, D, an arbitrary pair-test is executed. If D = 1 then all tests are distinct (no recomputatiOn). Note that each data point in the figures represents the mean over 30 or 70 problem instances that differ as follows: instead of instantiating queen 3, say, on square (3,1), then on (3,2),..., then (3,8), these 8 squares *are* ordered randomly. A problem instance is defined by choosing a "legal squares ordering" for each queen. Random ordering generally gives a smaller value of T, on the average, than the "natural" 1,2,3,...,N ordering (for 20-queens, a factor of 5 00 smaller!).

The algorithm exploits an advantageous time-space tradeoff and is defined below in general form by recursive SAIL procedure BKMARK [Swinehart & Sproull 1971]. The classical backtrack algorithm is defined the same, minus the underlined portions (except that "NEW[VAR]" in line 7 is replaced by "1"). The algorithm applies to any problem having NVARS variables (8, for 8-queens), each variable Xj having NVALSH] a priori possible values (8 squares for each queen (- one row of board), except 4 for queen 1 for symmetry reasons). An assignment vector ASSIGN[1:NVARS] of values to variables is a solution iff PAIRTEST0, ASSIGN[i], j, ASSIGN[j]) is true for all 0 < i < j < NVARS (iff no queen can take any other queen). Below, ASSIGN contains indices to the actual values. Top level invocation for 8-queens takes the form *tmp* <- BKMARKO, 8, A, B, C, D) with *array* dimensions D[1:NVARS] and C[1:NVARS, I:k], where k is the maximum of the B[i] values *(-8* for 8-queens). Initial values of A are irrelevant; C and D values are initially 1. BKMARK returns 1, with solution in ASSIGN, or returns 0 if no solution exists. Define PAIRTEST for 8-queens and trace the execution (new vs. old versions) to see how it works. (Suggestion: define an array VALUES with same dimensions as MARK, so that an element of VALUES encodes a board location.) For brevity, the symbol st.ands for "comment".

```
recursive integer procedure bkmark(integer var, nvars;
        integer array assign, nvals, mark, new);
begin integer i, val;       boolean testflg;
for val ← 1 step 1 until nvals[var] do
  if mark[var,val] geq new[var] then
    begin testflg ← true;
    for i ← new[var] step 1 while i < var and testflg do
      testflg ← pairtest(i, assign[i], var, val);
    mark[var,val] ← i - 1;                ! # of successful tests;
    if testflg then                       ! if passed all tests...;
      begin assign[var] ← val;
      if var = nvars then return(1)  ! done, so unwind;
      else if bkmark(var+1, nvars, assign, nvals, mark, new)
            = 1 then return(1)
      end
    end;
new[var] ← var - 1;       ! reset state of this var...;
for i ← var + 1 step 1 until nvars do ! ...and others;
    if new[i] > new[var] then new[i] ← new[var];
return(0)                 ! backtrack and continue search;
end;
```

REFERENCES

1. Fillmore, J., and S. Williamson, "On Backtracking: A Combinatorial Description of the Algorithm", S1AM J. Computing (3), No. 1 (March 1974), pp. 41-55.
2. Knuth, D., "Estimating the Efficiency of Backtrack Programs", Mathematics of Computation (29), No. 129 (Jan. 1975), pp. 121-136.
3. Swinehart, D., and B. Sproull, SAIL, Stanford AI Project Operating Note No. 57.2, January 1971.

This research was supported by the Defense Advanced Research Projects Agency under Contract no. F44620-73-C-0074 and monitored by the Air Force Office of Scientific Research.

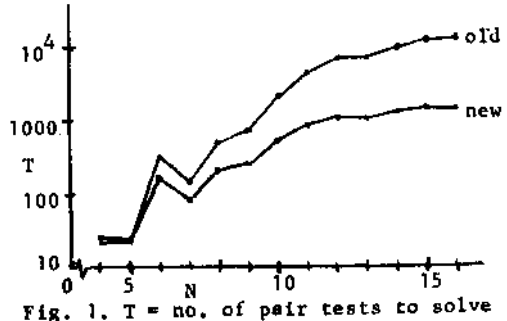


Fig. 1. T = no. of pair tests to solve N-queens puzzle
old -- classical algorithm, new -- BKMARK

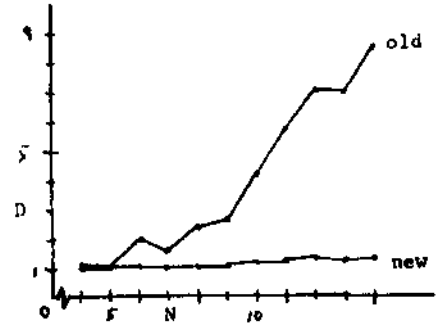


Fig. 2. D = total no. of pair tests / no. of distinct pair tests

GENERALITY AND COMPUTATIONAL COST

Azriel Rosenfeld
Computer Science Center
University of Maryland
College Park, Maryland 20742
U.S.A.

The purpose of this note is pedagogical. It discusses how one can reduce the computational cost of applying a set of operators (or predicates) by breaking them up into combinations of commonly occurring, simpler ones. This can be thought of as a process of generalization, in the sense that the common, simple operators are more "general" than the original, more complex ones. We are thus suggesting that even when one has a priori knowledge of a specialized nature (i.e., that the complex operators are applicable), it may still be desirable to use generalized operators in order to reduce computational cost.

To illustrate this idea, suppose that we want to apply a set of predicates P-,,...,P to an input I, and suppose that the cost of applying predicate Pi is (proportional to) the cardinality $|S_i|$ of its set of support $S_i \subseteq I$. Thus the total cost of applying the P's is $\sum_{i=1}^{n} |S_i|$. For example, applying P. might involve a template-matching process, where Pi is true iff. a perfect match to the template is found in I. Here I could be an image, or a string (where the "template" is the right-hand side of a rule in a grammar), or a graph (where the "template" is a subgraph). In what follows, we will use the image/template metaphor.

Suppose now that there exists a set of subtemplates Qj,____,Q_m such that, for $1 \le i \le n$, P_i is a concatenation of n. of the Q_j's. The cost of applying the Q_j's to I is $\sum_{j=1}^{m} |T_j|$, where T_j is Q_j's set of support. If we store the match positions in a new array I', then to test for P., we need only apply a template of cardinality n_i to I'. Thus testing for all the P_i's costs $\sum_{i=1}^{n} n_i$, and the total cost of the two-step matching process is $\sum |T_j| + \sum n_i$.

Under what circumstances is the two-step cost less than the brute-force cost $\sum |S_i|$ of applying the P_i's directly? We claim that this depends on the degree to which the Q's "generalize" the P's — i.e., on how few Q's are needed to construct all the P's. For concreteness, suppose that all the Q.'s have the same support size $|T_j| = r$, and that each P. consists of the same number $n_i = s$ of Q_j's. Thus each P. has support size $|S_i| = rs$, and the costs of the brute-force and two-step approaches are nrs and mr+ns, respectively. If there are few Q's, they must be used repeatedly, and we have m << ns (m=ns would mean that each Q is used only once); thus mr+ns will be much smaller than nrs. The fewer Q's we need, the greater a saving mr+ns is over nrs. Thus the more we can generalize the P's, the lower the computational cost.

This template example is certainly not a universal one. It would be desirable to extend this type of analysis to other situations. (On the advantages of hierarchical matching in the graph/subgraph case see Barrow et al. (1972).) However, our example does illustrate the idea that it may be advantageous to use generalized rather than specialized knowledge (see Zucker et al. (1975)), because this can lead to savings in computational cost.

References

Barrow, H. R., A. P. Ambler, and R. M. Burstall, Some techniques for recogniz ing structure in pictures, in S. Watanabe, ed., Frontiers of Pattern Recognition, Academic Press, N. Y., 1972, 1-29.

Zucker, S. W., A. Rosenfeld, and L. S. Davis, General-purpose models: expectations about the unexpected, Proc. 4IJCAI, 1975, 716-721.

———

The support of the National Science Foundation under Grant MCS-76-23763 is gratefully acknowledged, as is the help of Mrs. Shelly Rowe in preparing this note.