Making Judgments

Hans J. Berliner
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213

Abstract

Reasoning-based problem solving deals with discrete entities and manipulates these to derive new entities or produce branching behavior in order to discover a solution. This paradigm has some basic difficulties when applied to certain types of problems. Properly constructed arithmetic functions, such as those using our SNAC principles, can do such problems very well. SNAC constructions have considerable generality and robustness, and thus tend to outperform hand coded case statements as domains get larger. We show how a SNAC function can avoid getting stuck on a sub-optimal hill while hill-climbing. A clever move made by our backgammon program in defeating the World Champion is analyzed to show some aspects of the method.

1 Introduction

Problem solving research and examples usually deal with sequential reasoning toward a conclusion or required response. For such situations, criteria exist that make it possible to identify the correct response and possibly order other responses with respect to their goodness. However, in most domains such a paradigm is not possible because the number of states in the domain is so large that it is next to impossible to describe the properties of an arbitrary state with sufficient accuracy to be able to reason about it. Expertise in such domains appears to require judgment. We consider judgment to be the ability to produce graded responses to small changes in the stimulus environment. In judgment domains several responses may be considered adequate, while reasoned decisions would appear to only be correct or incorrect.

The ability to reliably judge small differences in chess positions is what separates the top players from their nearest competitors. Even though a decision procedure exists for determining whether one position is better than another, it is intractable. It is this intractability or the inability to isolate features that can be used in a clear reasoning process that distinguishes the judgment domain from the reasoning domain. The boundary between the two is certainly fuzzy, and undoubtedly changes as new information about any particular domain is developed. It seems that the larger the domain and the less precise the methods of making comparisons between elements of the domain, the less adequate are reasoning techniques.

2 The Problem

There are a number of techniques available to allow a program to make comparisons, i.e. to discriminate good from bad from indifferent in selecting among courses of action and among potential outcomes. However, while these techniques are fine for doing simple comparisons, most of them break down with even small additional complexity.

Consider the syllogism:

1) The more friends a person has, the happier he is.

2) John has more friends than Fred.

Therefore: John is happier than Fred.

So far so good. However, adding just a small amount of complexity with the two additional propositions:

3) The more money a person has, the happier he is.

4) Fred has more money than John.

makes it possible to derive two contradictory conclusions from the premises. This is a most unsatisfactory state of affairs. Especially so, since recoding the premises into first order predicate calculus does not help either. Neither will using productions or the branching logic of programming languages. For such representations, the most likely formulation would be that *X* will be happier than *Y iff* he is superior in

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

all applicable categories. Another formulation would have X happier than Y if he is superior in a majority of categories (with a tie being undefined). Such "voting" techniques can be shown to be deficient if we further increase the complexity of the decision that is to be made.

If premises 2 and 4 were restated as:

2a) John has 25 friends and Fred has 20.

4a) Fred has $20,000 and John has $500.

Most people would agree that Fred was happier according to our definitions of happiness. Yet, the only machinery available for coming to grips with problems such as this in systems that reason is to produce a large number of additional axioms that contain compound conditions, or to define degrees of difference so that degrees of happiness can be ascertained and summed.

The world of reasoning is a world of quantized perception and action. These systems are discrete and do business on a "case" basis. In order to achieve expertise it is necessary to react differentially to states that were formerly considered equivalent. Thus, the number of distinct perceptions and actions gets larger with expertise. This makes it more expensive to find the applicable rule or pattern, and creates difficulty in keeping new rules from interfering in unintended ways with the effects of older rules. Further, the possibility that more than one pattern will match grows as complexity grows, and default conditions are usually defined for states that fail to match specific rules or patterns. This makes adding new knowledge a formidable task for even moderate size domains [3]. So unless, a method is found for automatically appending viable rules to such a system, there seems to be a definite limit on the expertise it can achieve.

Because it is easier to pay attention to only a few things at one time, reasoning systems seem to have more of a sub-optimization nature than is necessary in sequential problem solving. The need to solve the top level goal can obscure the fact that it could possibly by solved later with greater facility. For instance, a plan for taking a trip by car could include:

1. Get suitcase
2. Pack clothes in suitcase
3. Put suitcase in car

If the raincoat is already in the car, this would involve getting it from the car only to bring it back later inside the suitcase. Conceivably, it would be simpler to bring the packed suitcase to the car and put the raincoat inside it at that time. This shows that goals need not have an immutable hierarchy. Further, there are times when achieving several low level goals is more desirable than achieving the top level goal.

In addition to the above there is another problem that exists in domains that interface to the real world, where sensed parameters that have a quasi-continuous character may have to be quantized. Premature quantization of variables loses information and can cause problems when the variable is to be used later for making decisions. For instance, if day/night is a binary variable and it is advantageous to be in day, a program may arrange its problem solving behavior so that it samples the environment just before day turns to night (by the system definition), and, being satisfied with what it finds, pronounces this branch of the solution search as favorable. If it had been forced to continue the branch even a few steps, if would have come to a different conclusion as night was closing in. However, quantization of the relatively continuous day/night variable causes the *blemish effect* [2], a behavior anomaly similar to the horizon effect [1], but with the step size of the variable rather than the fixed depth of the search being the culprit. This problem can be prevented by retaining a variable in its quasi-continuous state as long as possible. However, if a variable has a very large range it is impractical to create tests for each value in the range. Resorting to the testing of sub-ranges merely recreates the problem. Thus, discrete treatment of such a variable can cause problems, no matter how it is done.

3 A Better Way

Arithmetic functions can do all the above things easily and cheaply if they are constructed in the right way. A polynomial of terms that represent important features in the domain is constructed. We have described our SNAC method of constructing such polynomials and shown [2, 4] that:

- It is important that the values of terms vary smoothly.
- Non-linearity of terms is extremely important for expertise.
- Some method must exist for determining the degree to which each feature is applicable in the present situation. This is done with slowly varying variables that we call application coefficients.

The SNAC method also makes it possible to avoid the previously vexing problem of getting stuck on a sub-optimal hill while hill-climbing.

Figure 1 shows how getting stuck on a hill is avoided. With non-linear functions, the peaks of hills can be rounded so that retaining the peak becomes less desirable, especially if some other high ground is in view of the searching process. Further, with application coefficients it is possible to change the contour of the hill even as it is being climbed. This is shown in a - c; the arrow showing the location of the current state. As the hill is being climbed, one or more application coefficients that sense the global environment cause the goal of achieving the hilltop to become less important since it is very near being

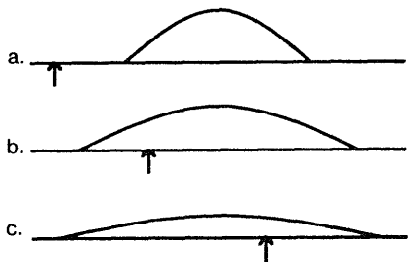

Figure 1: Effect of SNAC on Hill Shape

achieved. The change in value of the application coefficients causes the contour of the hill to begin to flatten, making the achievement of the summit less desirable, and resulting in the program looking for the next set of goals before even fully achieving the current set. Thus application coefficients can direct progress by reducing the importance of goals that are near being achieved, have already been achieved, or are no longer important.

The above is achieved mathematically as follows: The function $X^2+Y^2=C^2$ for $-C \leq X \leq C$ and $Y \geq 0$ will produce a semi-circle similar to Figure 1a. If we now change the function to be $X^2+A^*Y^2=C^2$ where $A \geq 1$ is an application coefficient (a variable), we can flatten the semi-circle into a semi-elipse of arbitrary flatness. Here, let *OLDX* be the value of X before the analysis of the current move starts. Then $A=(OLDX+K1)/K2$ for suitable constants $K1$ and $K2$ can be constructed so that it has a value near 1 for $X < -C$ and grows in value as *OLDX* increases in value. The construction is finalized by only recognizing values of A while *OLDX* is in the range of (say) $-2C$ to $+2C$. This construction has the desired properties. It should be noted that it is extremely important to use *OLDX* and not X in the construction of A, because if X were used then achieving the vicinity of the hill would never seem a desirable thing to do because the program could not tell the difference between getting there when it was far away or already very close.

4 An Example of SNAC Sensitivity

The backgammon position in Figure 2 occurred in the final game of the match in which my program, BKG 9.8, beat World Champion Luigi Villa in July, 1979. In this position, BKG 9.8 had to play a 5,1. There are four factors that must be considered here:

1. Black has established a strong defensive backgame position with points made on the 20 and 22 points.

2. In backgame positions timing is very important. Black is counting on hitting White when he brings his men around and home. At such time he must be prepared to contain the man sent back. This can only be done if the rest of his army is not too far advanced so it can form a containing pocket in front of the sent-back man. At the moment Black would not mind having additional men sent back in order to delay himself further and improve his timing.

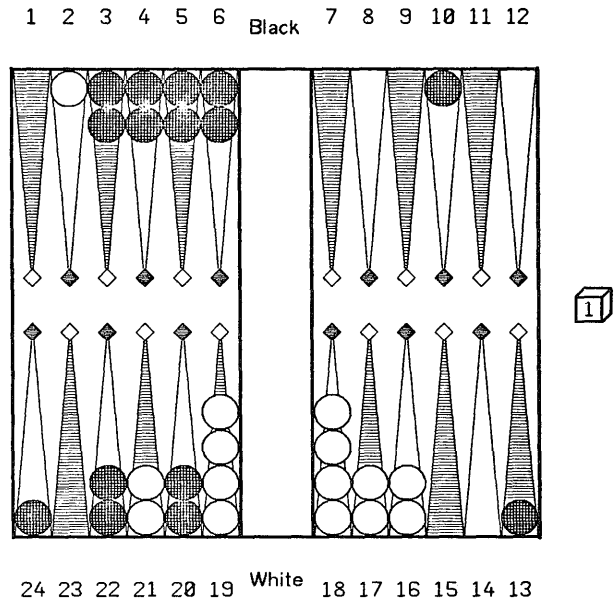

Figure 2: Black to Play a 5,1

3. There is also a possibility that Black could win by containing the man that is already back, but this is rather slim since White can escape with any 5 or 6. However, blockading this man is of some value in case White rolls no 5's or 6's in the near future.

4. In case the sole White back man does not escape, there is a possibility of bringing up the remainder of Black's men not used for the backgame and trying to win with an attack against the back man.

In view of the above it is very difficult to determine the right move, and none of the watching experts succeeded in finding it. The most frequently mentioned move was 13-7, which brings a man into the attack and hopes he is hit so as to gain timing (delay one's inevitable advance). However, BKG 9.8 made the better move of playing 13-8, 3-2, breaking up its blockade somewhat in order to get more attack, and attempting to actively contain the White back man. It did not worry about the increased chance of being hit, as this only helps with later defense. This gives the program two chances to win: If the attack succeeds, and by getting more men sent back, if the attack fails it improves the likelihood of success of its backgame.

I have not seen this concept in this form before in books or games. Humans tend to not want to break up the blockade that they have painstakingly built up, even though it is now the least valuable asset that Black has.

It is instructive to see how the program arrived at the judgment it made; one that it had never been tested for. Black has 28 legal moves. The top choices of the program were (points of the scoring polynomial in parentheses): 13-8, 3-2 (687); 10-5, 3-2 (682); 13-8, 10-9 (672); and

13-7 (667). The third and fourth choices were the ones most frequently mentioned by watching experts, thus showing they missed the idea of breaking up the blockade; the thing common to the program's top two choices.

Let us see why it judged the move actually played as better than the third choice (12-17, 15-16). The program considers many factors (polynomial terms) in its judgments and quite a few of these are non-linear. The six factors on which the two moves differed were (points for each and difference in parentheses):

1. Containment of enemy man (177, 131, +46). The move made does hinder his escape more. Containment is always desirable unless one is far ahead and the end of the game is nearing.

2. Condition of our home board (96, 110, -14). It breaks up one home board point. Breaking up the board (points 1 thru 6) is never considered desirable.

3. Attack (37, 21, +16). It is the best attacking move. Attack is desirable unless we are jeopardizing a sure win in the process.

4. Defensive situation (246, 260, -14). The move slows White down, thus could reduce the effectiveness of the backgame.

5. Long-term positional (-2, 11, -13). It puts a man on the 2 point, which is undesirable when the game still has a long way to go because it is too far advanced to be able to influence enemy men from there.

6. Safety of men (-12, -4, -8). The move made is dangerous. The program realizes this, but also understands that with a secure defensive position such danger is not serious. However, all other things being equal, it would prefer the least dangerous move

Thus the better containment and attack are considered to be more important than the weakening of the homeboard, the temporary slowing down of White, the long-term positional weakness, and the safety of the men. The difference between the first and second choice was that in the first choice the attack is slightly stronger.

The importance of each of the above terms varies with the situation. In the example, a backgame is established; else the safety term would outweigh the attack term, and BKG 9.8 would not leave two blots in its home board. It does recognize the degree of danger, however, and will not make a more dangerous move unless it has compensating benefits. This is typical of the influence that application coefficients exert in getting a term to respond to the global situation.

5 Perspective

We have been employing the SNAC method of making judgments for over two years now, and are struck with its simplicity and power. The happiness example posed earlier is solved trivially in all its forms with SNAC. If the above travel planning problem were solved as a search problem using SNAC functions that measure the economy of effort of the steps used, then undoubtedly SNAC would also do better than sequential planning based on rules, with no evaluation of outcome other than success or failure.

At the moment it is difficult to determine what role, if any, SNAC like mechanisms have in human thinking. We have constructed them to simulate lower level "intuitive" type of behavior, and they appear to work admirably in capturing good judgment in the large domain of backgammon. We conjecture that as variables become more and more discrete in character and as criteria for success become more obvious, reasoning gradually replaces such judgment making.

At present our backgammon program is being modified to be able to interpret its own functions with the aim of being able to explain its actions, and ultimately being able to identify its failures by type and modifying the culprit functions.

References

[1] Berliner, H. J.
Some Necessary Conditions for a Master Chess Program.
In *Third International Joint Conference on Artificial Intelligence*, pages 77-85. IJCAI, 1973.

[2] Berliner, H.
On the Construction of Evaluation Functions for Large Domains.
In *Sixth International Joint Conference on Artificial Intelligence*, pages 53-55. IJCAI, 1979.

[3] Berliner, H.
Some Observations on Problem Solving.
In *Proceeding of the Third CSCSI Conference.* Canadian Society for Computational Studies of Intelligence, 1980.

[4] Berliner, H. J.
Backgammon Computer Program beats World Champion.
Artificial Intelligence 14(1), 1980.