(Cambridge: MIT Press; 1987).

Rumelhart, D.E., McClelland, J.L., and the PDP Research Group. *Parallel Distributed Processing, Volumes 1 and 2.* (Cambridge, MA: MIT Press, 1986)

THE INTERNATIONAL NEURAL NETWORK SOCIETY

Write to:
Dr. Harold Szu
Naval Research Lab
Code 5756
Washington, D.C. 20375-5000

Dr. Maren is a Research Associate with the University of Tennessee Space Institute, in Tullahoma, TN. Her research interests include image understanding, AI, and models of human and machine information processing. She is currently developing neural network and symbolic processing systems for image understanding and scene description and matching.

On the Consistency Assumption, Monotone Criterion and the Monotone Restriction

James B. H. Kwa
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland, U.K.

Abstract

Hitherto, the consistency assumption has been deemed to be strong and it suffices for the purposes of proving certain important properties of the A* algorithm to use either of two weaker versions, viz. monotone criterion and monotone restriction. Here, we show that the opposite is true -- the consistency assumption is not only weaker, but sufficient. Moreover, it implies admissibility.

1. Introduction

In the literature describing the A* algorithm, one comes across the terms "consistency assumption," "monotone criterion" and "monotone restriction," which are constraints on the estimates of the remaining path cost. Their definitions differ, but each leads to the same two main effects i.e. when a node is closed, the optimal path to it has been found, and the estimate f, of the cost of the optimal path via a closed node is not less than that of any previously closed nodes. The current view is that the consistency assumption is a strong constraint and it suffices to use either of the monotone terms. The purpose of this article is to clarify the meanings and implications of these terms, and to show the relationship between these terms and search admissibility.

2. Consistency Assumption

Historically, the terms appeared in the order of mention in the preceding paragraph. In [HNR68] the consistency assumption constraint was introduced to prove the optimality of the A* algorithm. Stated formally, the consistency assumption is:

> $\forall$ m,nεG [h(m)-h(n)<=h*(m,n)] [C1] where m and n are nodes in the problem graph G. m is an ancestor of n, h*(m,n) is the *true* minimal cost from m to n, h(m) is the *estimate* of the remaining path cost from m to the goal, etc.

According to Hart et al. [HNR68], it means that "any estimate h(n) calculated from data available in the 'physical' situation represented by node n alone would not be improved by using corresponding data from the situations represented by the other nodes."

Originally, the optimality theorem stated that given the consistency assumption, an A* algorithm will expand less nodes than any other admissible algorithm which is less informed (in the sense of having a lower h estimate always). Later, a correction was published [HNR72] recanting the dependence of the optimality proof on the consistency assumption. It showed that the consistency assumption was only a useful constraint to have since it then meant that when a node is closed, the best path to it from the start node is known. Consequently, no updating and propagation of changes are required when subsequent expansions lead to it again. We shall call this the "once closed always closed" theorem. It is the same as Result 7 in [Nil80]. A practical consequence of this theorem is that the A* algorithm can be simplified significantly. A more recent clarification can be found in [DP82] in which it is stated that the optimality theorem is true only if algorithms being compared belong to the A* class i.e. subject to the constraint: $\forall n \varepsilon G$, [h(n)<=h*(n)] AND [f(n)=g(n)+h(n)].

3. Monotone Criterion and Monotone Restriction

The "monotone criterion" term was introduced in [Poh77] with the view that the consistency assumption is a strong constraint and probably more difficult to ascertain in real problems. Stated formally, the monotone criterion is a conjunction of the following three constraints:

$\forall$m,nεG, [h(m) >= h(n)] [C2]
$\forall$m,nεG, [h(m) - h(n) <= c(m,n)] [C3]
h(t)=0 [C4]

where m and n are nodes in the problem graph G, m is the *parent* of n, and c(m,n) is the arc cost from m to n, and t is the goal node.

The monotone criterion *inter alia* was used by Pohl to show that the "once closed always closed" theorem holds in his HPA algorithm.

Taking the same view that the consistency assumption is a strong constraint and hence implying that it is unnecessarily strong, the current "monotone restriction" term was introduced ([Nil80] pg. 95). It differs from the monotone criterion only in not requiring constraint [C2].

4. Discussion

The omission of [C2] in the monotone restriction begs the question of whether [C2] is a necessary condition for the "once closed always closed" theorem. Evidently not, and [C4] likewise, since no where in the published proof ([Nil80] pp. 82-83) for the "once closed always closed" theorem were they used. Neither [C2] nor [C4] was also required to prove that the f estimates of the sequence of closed nodes are monotonically nondecreasing. Perhaps [C2] was introduced to show explicitly that h estimates along an optimal path are monotonically nonincreasing. Although unnecessary, it agrees with intuition--as the goal is approached, the estimate of the remaining path cost should diminish. [C4] then ensures that h estimates are positive (again intuitive but unnecessary). A more important point is that [C4] says that when the goal node is being considered, one knows that it is the goal node. However, this is assumed in the A* algorithm: in particular, the termination condition embodies it. [C2] and [C4] thus seem superfluous.

5. Further Observations

There are two other interesting observations which can be made. Although these have no immediate practical

significance, they do offer further insight to the properties associated with heuristic search.

Firstly, while it is *apparently* true that both the monotone criterion and the monotone restriction are easier to establish than the consistency assumption, it is not true that the consistency assumption is a stronger constraint compared to either. It can be shown that the converse is true. Furthermore, if the consideration is restricted to [C1] and [C3] then the three terms are in fact *equivalent*.

Since the term "strong constraint" is sometimes vague, it is helpful to be specific in its interpretation. It can be better understood from two alternative viewpoints.

One is the material implicaiton viewpoint. If wherever constraint A is satisfied implies that constraint B is satisfied, but not vice-versa, then one can say that A is stronger than B. Taking this approach, we can establish that [C1] implies [C3].

Theorem 1: [C1] ==> [C3].
Proof:

By definition, the set X of descendants of a node m must contain the set Y of successors of m. Furthermore, if n is a successor of m, then $h^*(m,n)=c(m,n)$. It follows that when [C1] is *universally* true in X, [C3] is *universally* true in Y. Therefore [C1] ==> [C3].

The other viewpoint is based on the notion of difficulty in satisfaction. A weak constraint should be easier to satisfy than a strong constraint. Assuming that all situations are equally likely to occur, and if constraint A can be satisfied in any one of a set X of situations, and constraint B in any one of a set Y of situations, then it is reasonable and intuitive to say that A is more difficult (stronger) to satisfy than B if $|(X)|<|(Y)|$. Note that in this viewpoint we have used *existential* satisfaction. The converse would be true if *universal* satisfaction is used; it is easier to satisfy all situations in X than all in Y if $|(X)|<|(Y)|$. Since a node has at least as many descendants as successors, with universal satisfaction, [C1] can be seen to be more difficult to satisfy. Both viewpoints are in accord and *at this juncture*, [C1] is stronger. But the whole truth is yet to be completely revealed.

To be consistent, the material implication viewpoint will be used to complete the proof that the consistency assumption is not the stronger constraint. It rests on the fact that [C3] implies [C1].

Theorem 2: [C3] ==> [C1].
Proof:

For any descendant n_r of a node n_1, let $P=(n_1,n_2,...,n_r)$ be the shortest path from n_1 to n_r.

Using [C3],

$\forall n_{i-1},n_i \varepsilon P, [h(n_{i-1})-h(n_i)<=c(n_{i-1},n_i)]$.

Letting i run from 2 to r gives:

$h(n_1) - h(n_2) <= c(n_1,n_2)$ [1]
$h(n_2) - h(n_3) <= c(n_2,n_3)$ [2]
...
$h(n_{r-1}) - h(n_r) <= c(n_{r-1},n_r)$ [r-1].

Summing equations [1] to [r-1] gives:

$h(n_1)-h(n_r)<=c(n_1,n_2)+c(n_2,n_3)+...+c(n_{r-1},n_r)=h^*(n_1,n_r)$.
(Q.E.D.)

Corollary 2:
From Theorem 1, [C1]==>[C3]. Hence [C1]<==>[C3].

If [C2] and [C4] are indeed superfluous, then the consistency assumption, monotone criterion and monotone restriction are equivalent. Otherwise, the monotone criterion or monotone restriction is more constrained (stronger) than the consistency assumption.

The second observation is that it is superfluous to explicitly include the admissibility constraint whenever the monotone criterion or monotone restriction holds since either one of the monotone terms implies admissibility.

Theorem 3: [C3] AND [C4] ==> $\forall n \varepsilon G$, $h(n)<=h^*(n)$.
Proof:

Let n_1 be the goal node t, and the shortest path from a node n_r to n_1 be $P = (n_r,n_{r-1},...,n_1)$.

$h^*(n_1) = h^*(t) = 0$ by definition.
$h(n_1) = h(t) = 0$ from [C4].
Hence $h(n_1) <= h^*(n_1)$ is true.
Assume that $h(n_i) <= h^*(n_i)$. [2-1]
$h(n_i + 1) - h(n-(i)) <= c(n_i + 1,n_i)$ from [C3].
$h(n_i + 1) <= c(n_i + 1,n_i) + h(n_i)$
$<= c(n_i + 1,n_i) + h^*(n_i)$ from [2-1].

Since $c(n_i + 1,n_i + h^*(n_i) = h^*(n_i + 1)$,
therefore $h(n_i + 1) <= h^*(n_i + 1)$. [2-2]

Since [2-2] is true for i = 0, by induction,
it is true for i =1,2,3, ...r-1.
Hence [C3] AND [C4] ==> $\forall n \varepsilon G$, $h(n) <= h^*(n)$.

(Note: This theorem holds even if [C4] is relaxed to $h(t)<=0$.)

(Q.E.D.) i

Corollary 3:
Since [C1]<==>[C3],
therefore [C1] AND [C4] ==> $\forall n \varepsilon G$, $h(n) <= h^*(n)$.

6. Conclusion

Contrary to earlier claim that the consistency assumption is relatively stronger than the monotone criterion or monotone restriction, we have shown that it is the weaker since it is equivalent to [C3] which exists as a member of the conjunctive set of constraints defined by the monotone criterion and monotone restriction. In fact, [C2] and [C4] seem unnecessary to establish the important properties associated with the two monotone terms. If they are unnecessary, then the monotone criterion and monotone restriction are unnecessarily strong and the consistency assumption is vindicated. Furthermore, since [C4] is fundamental to the A* algorithm in the first instance, so long as [C1] or [C3] is in any of the three terms, an optimal solution (if it exists) is guaranteed.

References

[DP82] Dechter,R.& Pearl, J. the optimality of A* revisited. In *The Proceedings of AAAI-82*. pp 95-99, August 1982.

[HNR68] Hart, P.E.,Nilsson, N.J. & Raphael, B.A. formal basis for the heuristic determinationof minimum cost paths. *IEEE Trans. Sys. Sci. Cybernetics*, Vol. SSC-4, No. 2, pp. 100-107, 1968.

[HNR72] Hart, P.E., Nilsson, N.J. & Raphael, B. Correction to: A formal basis for the heuristic determination of minimum cost paths. *SIGART Newsletter*, Vol. 37, pp. 28-29, 1972.

[Nil80] Nilsson N.J. *Principles of Artificial Intelligence*. Tioga Publishing Co., Palo Alto, Califor-

nia, U.S.A. 1980.

[Poh77] Pohl, I. Practical and theoretical considerations in heuristic search algorithms. *machine Intelligence* 8. (Elcock, E.W. & Michie, D., eds), Edinburgh University Press, Edinburgh, pp. 55-72, 1977.

A CONSTRAINT APPROACH TO MASTERMIND IN LOGIC PROGRAMMING

Pascal Van Hentenryck
European Computer-Industry Research Centre
Arabellastr. 17
8000 Muenchen 81 (FRG)

ABSTRACT

Logic programming is a convenient tool for stating combinatorial problems due to its nondeterminism and its relational form. It is not surprising that simple and declarative programs can be written for problems like mastermind. However, due to their search strategy, logic languages are also very inefficient for solving the natural formulation of problems. Moving away from this natural formulation leads to much programming effort and to less modifable and extensible programs. This paper shows, on the mastermind example, that it is possible to write very declarative programs which will be executed efficiently by an extended logic programming language. The key idea is to embed consistency techniques inside logic programming. Within this approach, constraints are used actively to prune the search space in an "a priori" way instead of the passive way (for testing values) of usual languages. The resulting program outperforms all the proposed algorithms and achieves a speed-up of 40 over Shapiro's program in the average and a speed-up of 72 on Powers' benchmark.

1. Introduction.

Some time ago, Shapiro proposed in SIGART a program for playing mastermind [10]. His program[1] illustrates the adequacy programs to solve discrete combinatorial problems. However, Shapiro's program is very inefficient due to its *generate and test* strategy which only uses constraints in a *passive* way for testing values. Subsequently, many papers in SIGART (including [9]) proposed modifications of the original algorithm in order to improve its efficiency. A common feature of all these programs is that they substantially move away from the natural formulation[2].

This simple example reveals a profound limitation of current logic programming languages wrt discrete combinatorial problems. While being convenient for stating these problems, they are also very inefficient for solving the natural (generate & test) formulation of the problems. Also, moving away to a more efficient formulation leads to much more programming effort and to poorly modifiable and extensible programs.

Starting with the recognition of this state of affairs, we have proposed to embed consistency techniques in logic programming [13, 14, 15, 16, 4]. Consistency techniques is an emerging paradigm from AI to solve discrete combinatorial problems [6, 7]. Contrary to "generate and test" and depth-first search with chronological backtracking, consistency techniques use the constraints *actively* to prune the search space in an "a priori" way by removing combinations of values that cannot appear together in a solution. By embedding consistency techniques inside logic programming, it is possible to preserve the descriptive facilities of logic programming while bringing into it the efficiency of consistency techniques. More precisely, it is possible to write "generate & test" programs which will be efficiently handled by the system.

This approach has already been applied to the solving of several real-life combinatorial programs that are out of scope for any logic languages we know of. Moreover, it turns out that these problems can be solved inside logic programming with an efficiency comparable to the one of specific codes written in a procedural language. Of course, the interest of using logic programming enhanced with consistency techniques comes from the "generate & test" formulation of the problems which provides the programmer with a short development time for, and a great modifiability and extensibility of, the programs. Examples of real-life problems solved within the system include graph-coloring, warehouse location problems, scheduling with disjunctive constraints, cutting stock and microcode label assignment [16].

This paper illustrates this approach on the mastermind example. We first describe the only feature necessary for this specific problem. The reader interested in a more comprehensive description can consult the above references. Next, the program will be presented and compared to existing approaches.

2. Consistency techniques in logic programming.

The basis for embedding consistency techniques inside logic programming is the *domain* concept [13]. From the user point of view, it provides a way to specify the range of variable. But, once the domain concept is embedded inside logic programming, new *inference rules* can be defined which embody the idea of "a priori" pruning present in consistency technique [14, 15]. These inference rules are the theoretical foundations of new control mechanisms and novel implementations of the primitive constraints. We now describe the only two features that are necessary for the mastermind program, i.e. the domain declarations and the unequality constraints.

2.1. Domain declarations.

Domain declarations are introduced to provide the user with a way to specify the range of a predicate.

Definition 1: Given an n-ary predicate p, a *domain declaration* is an information of the following form:

$$\text{domain } p(d_1,\ldots,d_n).$$

where d_1 is either the constant h, a set $\{a_1,\ldots,a_n\}$ of constants or strings or an expression l..u where l and u are natural numbers such that u>l.

In the first case, it means that the ith argument ranges over the Herbrand universe. In the second case, it means that the ith argument is either a variable or a list of variables which range over the set $\{a_1,\ldots,a_n\}$. The last case

---

[1] Note that, in 1978, van Emden had already illustrated the use of logic programming with a mastermind program [12].

[2] In fact, Shapiro's program has already made some concessions to the declarative expression by using assert combined with the negation as failure in order to gain some efficiency.