Ae — AN EFFICIENT NEAR ADMISSIBLE HEURISTIC SEARCH ALGORITHM

Malik Ghallab and Dennis G. Allard

Laboratoire d'Automatique et d'Analyse des Systemes
du C.N.R.S.
7, avenue du Colonel Roche - 31400 TOULOUSE, France

ABSTRACT

The algorithm A* (Nilsson, 1979) presents two significant drawbacks. First, in seeking strict optimal solution paths it necessarily has high order of complexity. Second, the algorithm does not explicitly descriminate between the cost of a solution path and the cost of finding the solution path. To confront these problems we propose the algorithm AE, a generalization of A*. Instead of seeking an optimal solution, it seeks one which is within a factor (1+e) of optimum (e > 0). The basic idea is to avoid doing any search at all on most near optimal partial solutions by sticking to a small number of most fruitful paths. Various strategies for searching for near optimal partial solutions are discussed. Experimental results are presented indicating that A e has average complexity of lower order than A* and compares favorably to the related algorithm Af* (Pearl and Kim, 1982).

I NTRODUCTION — ADMISSIBLE SEARCH IS EXPENSIVE

This paper emphasizes the main ideas behind Ae and serves to introduce new empirical results concerning its behavior. A more extensive presentation of Ae and its properties appears in (Ghallab, 1982) and in (Ghallab and Allard, 1982). We assume familiarity with A* and notation of (Nilsson, 1979). We denote the set of successors of node n by X(n). The cost of arc <n,x> is k(n,x)>0. We denote a path between n and m by <n...m>. For start node s, the true costs of minimal cost paths <s...n>, <n...t>, and <s...n...t>, over all goal nodes t, are denoted by g*(n), h*(n), and f*(n) respectively. The estimators for these functions are g, h, and f respectively. OPEN is assumed ordered by increasing f (and decreasing g in the case of ties). At all times, n' denotes the first (least f) element of OPEN.

The complexity of A* has been studied in some detail. For perfect h = h*, A* is linear. But in general it has worst case complexity which is exponential as a function of the number of nodes in the found solution path (Pohl, 1970,77). For a large class of problems A* remains exponential in the average case unless h is very precise and remains unrealistically close to h* (Pearl, 1983).

Two drawbacks to A* explain its high complexity. First, A* tends to do much backtracking due to the invariable choice of n' as the node to be expanded next. A* tends to expand many nodes not in the final solution path since h (being a heuristic) fluctuates in quality and hence various near optimal paths take random turns appearing to be optimal. Such paths effectively "race" with one another to reach their goals. So we can trace the cause of backtracking to A*'s desire to fine tune an optimal cost solution.

The second drawback is that A* makes no explicit attempt to minimize search cost, i.e. the number of nodes expanded. Given two equally promising paths with respect to path cost, it would seem wise to develop that path which is the fewest arcs from completion. But for A*, it does not matter too much which promising path is developed next since eventually all promising paths must be. Thus A* does not have much need for an explicit heuristic to reduce search cost. Simply put, admissible search is inherently expensive.

Our basic premise is that a bounded loss of optimality can always be favorably traded for a gain in computational efficiency. The very notion of "cost" and "optimal cost" are imprecise in most applications involving modelling of the real world. Moreover, in heuristic search one often requires only a near optimal or even just decent solution. For example, in robotics, there is a tradeoff between the cost of generating a plan and the optimality of the plan when the plan is to be executed only once.

II A6 AND NEAR OPTIMAL PATH SEARCH STRATIGIES

Ae drops the strict optimality criteria and seeks instead a solution within a factor (l+f) of optimum for a user specified €>*,0,* This at least makes possible avoiding A*'s defect of having to investigate all optimal looking paths. The problem now becomes — which of the many near optimal paths merit attention?

A€ attempts to answer this question by generalizing A* in two primary ways. First, A€ performs a depth oriented search, preferring to stay on a single path as long as a successor to the frontier node (the last one expanded) of that path is "acceptable". An open node n is acceptable iff f(n) is bounded by (l +€)max f (n')} over all n' which have appeared as the first element (least f)

The second author is participating in an exchange program between LAAS du CNRS and the Electronics Research Laboratory, University of California, Berkeley.

in OPEN. To choose an acceptable successor to expand next or, in the case that no such successor exists, to choose which node in OPEN to backtrack to, A€ can make use of a second heuristic hc. hc(n) provides an explicit guess as to the computational cost involved in reaching a goal node by estimating the minimum number of arcs between n and a goal node.

The second difference between A€ and A* is that A€ possesses an inner loop invoked when the frontier node has no acceptable successor. The inner loop attempts to render some successors acceptable by expanding a certain number of times the first node of OPEN, n'. Doing so often (if h is monotone, always) increases f(n') and thus may turn unacceptable nodes into acceptable ones. We refer to this idea as the perserverant strategy of A€. A more precise specification of A€ is as follows.

Algorithm A€:

```
1.   OPEN:= {s}   CLOSED:= nil  SOLVED:= nil
     g(s):= 0  f(s):= h(s)  €threshold:= (1+€)f(s)
     Expand(s)
     AX:= { x ∈ X(s) : Acceptable(x) }

2.   do until ((∃ t ∈ SOLVED)Acceptable(t)
              or  OPEN = nil)
2.1     if AX ≠ nil then
           n:= SelectAX
        else
           n:= SelectOPEN  fi
2.2     Expand(n)
2.3     do until ((∃ x ∈ X(n))Acceptable(x)
                 or OPEN = nil or not Persevere)
           Expand(n')
        od
2.4     AX:= { x ∈ X(n) : Acceptable(x) }
     od

3.   if OPEN = nil then failure
     else
        set t to node of least f(t) in SOLVED
        output path <s...t>
        output €' = ((1+€) + €threshold)f(t)  - 1
     fi

where,

Acceptable(n) iff n∈ OPEN  and  f(n) ≤ €threshold


Expand(n):

OPEN:= OPEN - {n}    CLOSED:= CLOSED union {n}
do (∀ x ∈ X(n))
  if x ∉ OPEN union CLOSED or g(n)+k(n,x)<g(x) then
     OPEN:= OPEN union {x}
     g(x):= g(n) + k(n,x)
     f(x):= g(x) + h(x)
     set a back pointer to keep track of path to x
     if x is a goal node then
        SOLVED:= SOLVED union {x}
     fi
  fi
od
€threshold:= max {€threshold,(1+€)f(n')}
```

Notice output €' which gives the actual relative cost of the found solution with respect to max {f(n')'}. It is always true that €'<:€ since the output solution (if any) satisfies Acceptable.

SelectAX selects which acceptable successor of n is to become the new n. A general approach is to minimize a weighted sum Af(x) + Bhc(x). Since the main purpose here is to stay on a path previously seen to be near optimal, we believe that A should be larger than B. The extreme approach of taking A = 1 and B - 0 leads to a depth oriented best first search and has the advantage of simplifying the main loop -- statement 2.1 can be replaced by

```
if Acceptable(xmin)  then
   n:= xmin
else
   n:= SelectOPEN  fi
```

where xmin is the open successor of n having least f. And statement 2.4 can be removed entirely.

SelectOPEN has the more difficult task of deciding which acceptable n in OPEN to backtrack to. Again we can minimize some weighted function Cf(n) + Dhc(n) over all n in OPEN. Minimizing fn) gives us precisely n'. This would lead after expansion to raising fthreshold as much as possible hence increasing the likelihood that the developed path will remain acceptable. Minimizing hc(n) moves us closer to a solution but increases the risk that the inner loop may have to abandon the selected path soon thus provoking backtracking. Nevertheless, we argue that SelectOPEN should place importance on hc since the inner loop will be capable of raising €threshold. In general, experimentation should help determine weights for f and hc in any specific application.

The perserverant strategy of A6 is embodied in the predicate Persevere, the heuristic element of the inner loop. Persevere should return true as long as it seems wise to try to render at least one of the successors of n acceptable, i.e. to persevere on the current path. We list below factors which would indicate that doing so is worthwhile.

- n has a successor which is "almost" acceptable.

- n has a successor close to a goal (hc small).

- The second or third best f(n) over n in OPEN is significantly greater than f(n').

- The inner loop has iterated few times.

- A node already in SOLVED is almost acceptable (n' should be expanded to try rendering a found goal acceptable).

- (l+f)f(n') = fthreshold. If h is monotone this will always be true so this condition would not be useful in that case.

Notice that A* halts when it runs into a goal node as first element of OPEN whereas A€ continually surveys all generated goal nodes and explicitly attempts to render one or more of them acceptable.

For search in a δ-graph (one in which no infinite path has finite cost) where a path from s to a goal node exists, the following properties of A_ϵ hold. See (Ghallab, 1982) and (Ghallab and Allard, 1982) for proofs and discussion.

1. A_ϵ halts after finding a goal node t satisfying Acceptable(t).

2. If for some fixed constant e, $0 \leq h(n) \leq (1+e)h^*(n)$ for all nodes n, then the found goal node t satisfies $f(t) \leq (1+e)(1+\epsilon')h^*(s)$.

Taking e = 0 in the above result and noticing that $\epsilon' \leq \epsilon$ gives:

3. If h is admissible then A_ϵ is ε-admissible.

4. If h is monotone then f(n') increases monotonically and, for all expanded n, $g^*(n) \leq g(n) \leq (1+\epsilon)g^*(n) + \epsilon h(n)$.

Let us briefly mention how A_ϵ differs from A_ϵ^* (Pearl and Kim, 1982). A_ϵ^* is like A* except that A_ϵ^* expands the node with least h_c among all n in OPEN satisfying $f(n) \leq (1+\epsilon)f(n')$. This last condition is more restrictive than our Acceptable predicate. But most importantly, A_ϵ^* uses neither depth oriented search nor a perserverant strategy (no inner loop) and hence may perform unnecessary backtracking. Also, if a good h_c is not available, A_ϵ is only partially handicapped (see SelectOPEN) and can still perform depth oriented search, whereas A_ϵ^* simply reduces to A*.

III EXPERIMENTAL RESULTS AND CONCLUSION

We have tested the hehavior of A€ on TSP (the travelling salesman problem) under the same test conditions (N = 9 cities randomly distributed uniformly in the unit square) and using the same heuristics h and hc as (Pearl and Kim, I982). Tests were made on sixty different distributions of cities. For each test A, was run with € set successively to 0, 0.0I, 0.05, 0.10, 0.15, 0.25, and 00 (€ = 0 yields the optimum solution; € = oo yields a solution in the minimum number of steps -- exactly N-2 nodes are expanded).

Table I indicates the computational effort exerted by A,. Three complexity indicators were employed: E, the number of nodes expanded, G the number of nodes generated, and B ,the number of times A, had to backtrack (switch paths). Table 2 gives the actual cost of the solutions obtained and f', the guarantee returned by A€ of the degree of optimality of the solutions. All figures (€ excepted) are given relative to their values for *f* = 0 (corresponding to what A* would do). The numbers in parentheses give the standard deviation *(* for the corresponding mesures.

Notice that for all indicators (Table I), computational cost decreases rapidly withincreasing € (from exponential complexity for *t* = 0 down to linear complexity for (=00). The obtained results compare favorably with those of (Pearl and Kim, 1982, Fig. 4*)* shown as E1 in Table I below.

As Table 2 shows, the actual solution costs remain quite near optimal. It is interesting that with (= 00, the found solution is guaranteed to be within 27 % of optimal.

100xε	0	1	5	10	15	25	∞
E (σ)	100	92 (8.8)	77 (22)	54 (24)	42 (25)	23 (16)	15 (13)
G (σ)	100	92 (11)	79 (20)	58 (23)	46 (27)	28 (20)	21 (19)
B (σ)	100	83 (11)	48 (17)	21 (15)	13 (13)	2.8 (5.1)	0
E'	100	97	85	70	65	65	65

Table I

100xε	0	1	5	10	15	25	∞
Cost (σ)	100	100.1 (.6)	100.4 (.8)	101.1 (1.7)	101.9 (2.5)	103.0 (4.1)	107.0 (5.7)
100xε' (σ)	0	.36 (.39)	2.7 (1.3)	7.5 (2.7)	11 (3.5)	18 (5.3)	27 (17)

Table 2

To obtain completeness with respect to Pearl's results we also tested A, on "difficult problems" in which intercity distances are confined to the interval (0.75,1.25). In 25 test cases, as soon as *(* exceeded .05, no backtracking was ever performed. Our results for number of nodes developed are similar to those of Pearl (c.f. his Fig. 6); E = 81 and 28 for < = .0 1 and .05 respectively.

These results compare A€ favorably to A* and to A€ *. We are currently conducting further tests on TSP and other search problems in order to experiment with the search strategies discussed in section I I.

REFERENCES

Ghallab M. Optimisation de processus decisionnels pour la robotique. These d'Etat. Universite Paul Sabatier de Toulouse. October 1982.

Ghallab M., Allard D. Near admissible heuristic search algorithms. Second World Conference on Mathematics at the Service of Man, Las Palmas (Canary Islands), Spain, June 1982.

Nilsson N.J. Principles of Artificial Intelligence. Tioga Publishing Co., 1979.

Pearl J. Knowledge versus search: a quantitative analysis using A*. Artificial Intelligence 20(1), January 1983.

Pearl J., Kim J. Studies in semi-admissible heuristics. IEEE Transactions on Pattern Analysis and Machine Intelligence, July 1982.

Pohl I. First results on the effect of error in heuristic search. In Machine Intelligence 5, Metzer and Michie eds., Edinburgh U. Press, 1970.

Pohl I, Practical and theoritical considerations in heuristic search algorithms. In Machine Intelligence 8, Elcock and Michie eds., 1977.