FROM DYNAMIC PROGRAMMING TO SEARCH ALGORITHMS WITH FUNCTIONAL COSTS

A. Martelli and U Montanari
Istituto di Elaborazione dell'Informazione
Cunsiglio Nazionalc delle Ricerche
Pisa, Italy

Abstract

In this paper we approach, using artificial intelligence methods, the problem of finding a minimal-cost path in a functionally weighted graph, i.e a graph with monotone cost functions associated with the arcs This problem is important since solving any system of functional equations in a general dynamic programming formulation can be shown equivalent to it. A general heuristic search algorithm with estimate is given, which is a nontrivial extension of algorithm A* by *Hart,* Nilsson and Raphael. Putting some constraints on cost functions and on the estimate, this algorithm can be simplified until the classical version, with additive cost functions, is reached.

I. Introduction

Dynamic programming is a well-known methodology for representing optimization problems in terms of the so-called functional equations and for solving them. While in the artificial intelligence literature dynamic programming is often described reductively as a static, breadth-first technique for searching cycle-free graphs, a general model for the discrete, deterministic case has been given by Karp and Held [1]. This model, called *sequential decision process* consists essentially of a finite automaton with *cust functions* associated with the transitions. If the cost functions arc monotone, and if a limit condition is satisfied, searching the transition graph of the finite automaton is equivalent to solving a suitable set of functional equations. The authors have shown how both the functional equations model and the Karp and Held model can be derived from a quite primitive problem-reduction schema [2,3] which directly interprets the Bellman's concept *of problem embedding* and shows how the *optima lity principle* holds if and only if the cost functions are monotone.

In this paper, we approach the problem of searching a functionally weighted graph with methods proper of artificial intelligence, in particular we extend the heuristically guided search method by Hart, Nilsson and Raphael [4] Since here the cost of an arc is represented by a generic monotone function, the extension is nontrivial, and it may happen that no *simple* or no *finite* optimal path exists. Putting further constraints on the cost functions and on the estimate (strict monotomcity, positive mo notonicity) it is then possible to simplify the general search algorithm until the classical version is reached.

The relevance of bridging the gap between dynamic programming and heuristic search methods is twofold. On one hand, the Karp and Held model provides a framework where search problems can be stated in quite general terms and (heuristic) search algorithms can be proved effective under well specified constraints. On the other hand, if good estimates can be derived from the problem domain, the solution of problems amenable to a dynamic programming representation can be greatly sped up.

2- Djynarmic Programming

Dynamic programming [5[is an optimization technique based on the representation of a problem as a process which evolves from state to state. When the cost structure is appropriate, the determination of an optimal solution may be reduced to the solution of a system of *functional equations.* A general form of such equations is the following [1]

$$(2.1) \qquad y_i \leq \min\left(\min_{j=1,\ldots,n} f_{ij}(y_j),\ a_i\right) \qquad i = 1, \ldots, n$$

where variables y_i can assume any value in the complete lattice* $L = R \cup (+\infty) \cup (-\infty)$, consisting of the real numbers together with a bottom element $-\infty$ and a top element $+\infty$; the functions $f_{ij} : L \to L$ are *monotone* (i.e. $x_1 \leq x_2 \Rightarrow f(x_1) \leq f(x_2)$), continuous from above; the values a_i are real constants.

The solution of system (2.1) is the maximal n-tuple $(\bar{y}_1, \ldots, \bar{y}_n)$ satisfying the given inequalities. Although for general functions such a maximum does not exist, it is possible to prove, by using the fixpoint theorem of lattice theory, that in the monotone case the maximum exists and satisfies (2.1) with the equal sign. Furthermore, the fixpoint theory provides an iterative algorithm for computing the solution.

The relationships between dynamic programming and search algorithms can be made clear by introducing a model of dynamic programming, due to Karp and Held [1], called *sequential decision process.* This model is a finite automaton with functions associated to its transitions, but it can also be presented as a graph (the transition graph of the automaton).

We have a directed graph G with n+1 nodes $N_1, \ldots, N_n, N_{n+1}$. A *monotone* function continuous from above $f_{ij}(x)$, called *cost function,* is associated with each arc $(N_i\ N_j)$. The node N_{n+1}, called the *solved node,* has no outgoing arcs and has a constant c_{n+1} (its *cost*) associated with it. The graph G with cost functions is called a *functionally weighted graph.* An example is given in Fig. 1.

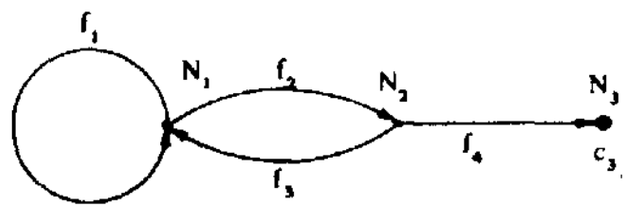

Fig. 1

Given a functionally weighted graph G, we can obtain from it a system of functional equations by associating a variable y_i to each node N_i ($i = 1, \ldots, n$) and by writing a system of the type (2.1) where the functions f_{ij} are the cost functions of the arcs $(N_i\ N_j)$ in G. Furthermore, we put $a_i = f_{i,n+1}(c_{n+1})$, for each i such that arc $(N_i\ N_{n+1})$ exists. Conversely, we can obtain a functionally weighted graph from a system of functional equations. For istance, the functionally weighted graph of Fig. 1 corresponds to the following system of functional equations, if $a = f_4(c_3)$:

$$y_1 = \min(f_1(y_1),\ f_2(y_2))$$
$$y_2 = \min(f_3(y_1),\ a)$$

Given a path in a functionally weighted graph from node N_i to the solved node N_{n+1}, its *cost* is obtained by applying in the reverse order to constant c_{n+1} the cost functions of the arcs of the path. The next theorem shows how the solution of the functional equations is related to the costs of the paths in the corresponding functionally weighted graph. (See [3] for a proof).

Theorem 2.1. The solution $\bar{y}_i$ of functional equations for node N_i is equal to the greatest lower bound (glb) of the costs of all paths in the

* The presentation of functional equations given here is based on a model of dynamic programming developed by the authors in [2].

functionally weighted graph from node N_i to the solved node N

This theorem shows that a dynamic programming problem can be reduced to a minimal-cost path problem, if the cost of a path is suitably defined Therefore such a problem could be solved by using algorithms for finding minimal-cost paths in a functionally weighted graph. These algorithms are well-known in the case of additive costs, i,e. where the cost functions have the form f(x) = x+a [4,6] In the next section we give a general search algorithm which uses heuristic information to guide the search

Conversely, this theorem says that a minimal-cost path problem can be solved by solving a system of functional equations with, for example, the iterative algorithm. Such a result is well-known in the additive case, and several iterative algorithms have been given for finding the shortest path in a graph [7].

We point out that the model of dynamic programming presented in this section is general as far as *monadic* functions are considered. However there are dynamic programming problems which are better modelled by assuming that the functional equations contain *polyadic* functions [8] In this case our model can be generalized to an AND/OR graph (9).

3. A General Propagation Algonthmwith Estimate

In the previous section we have described a general model for dynamic programming and an iterative solution algorithm. No better method is available, unless special assumptions are made on the functions f and/or an *estimate* is supplied for the cost of all intermediate problems based on information from the problem domain This line of development corresponds to merging the branch-and-bound idea with dynamic programming and has been originally pursued by Hart, Nilsson and Raphael [4] in the special case of additive functions with positive weights. Here we give a general algorithm for searching functionally weighted graphs and we show its validity when the estimates arc lower bounds

Let G be a functionally weighted graph with n+1 nodes N_i, ... N , N * and *monotone* cost functions f , Given a path p from node N to node N_{ik} ($p = N_{i1} N_{j2} \ldots N_{tk}$) *c define the cost function of path p as the (monotone) function f (x) obtained by composing the cost functions of the arcs of p

$$f_p(x) = f_{i1,i2}(f_{i2,3}(\ldots(f_{ik-1,ik}(x))\ldots))$$

Furthermore we define the *distance function* d, (x) from node $N_($ to node N_j as

$$d_{ij}(x) = \operatorname*{glb}_{p \in P_{ij}} f_p(x) \quad \text{for every } x$$

where P.. is the (possibly infinite) set of paths from N to N . Notice that the distance is defined, for every x. as the greatest lower bound and not simply as the minimum, because, in some cases, it can only be obtained as a limit, i.e. with an infinite path

Finally wc can define the *solution cost c* of a node N^ (i= 1,.... n) as the gib of all values which can be achieved by applying the cost function of every path from $N_$ to N_{n+1} to the cost c_{n+1} of the solved node N_{n+1} :

$$c_i = d_{i,n+1}(c_{n+1})$$

We give now an algorithm for finding the solution cost c of a given node $N_($ of G. called the *top node,* and a path for which this cost is achieved, called *solution path* Note that, when the solution cost $c_($ is obtained as a limit, no (finite) solution path exists. This algorithm is a *bottom-up* propagation algorithm, i.e. it propagates toward the top node starting with the solved node N which, from now on, will also be called *bottom node* and denoted with N_t No restriction is made on the generality of the algorithm by having only one bottom node and one top node. In fact, if more than one top node were present, we could add to the graph a new node N_0 connected to each top node through an arc with the identity function associated with it, and we could consider N_0 as the top node of the new graph

* Usually, in *the* artificial intelligence literature, aearcti algorithm* are given for infinite graphi Must results of thes paper can he extended to this casc.

At each step, the algorithm selects a node N_i and expands it generating all of *its parents,* i.e. all nodes N. such that (N N) is an arc of G. The selection of the node to be expanded is made according to a *total estimate* e associated with each node N , which is an estimate of the solution cost c_t of node N_t, with the constraint (hat the solution path passes through node N The total estimate e can be expressed as follows

$$e_i = h_i(g_i)$$

where g_i is an estimate of the solution cost c_i of node N and h. (x) is a *monotone* function which gives an estimate of the distance function c . (x) fiom N_i to N_j . This estimate can be obtained using the heuristic information available from the problem domain, whereas g_i will be constructed step by step by the algorithm

Our algorithm is an extension to the case of monotone cost functions of algorithm A* by Hart, Nilsson and Raphael (4.10), which considers the case of additive cost functions with positive weights, i.e.

$$f_{ij}(x) = x + c_{ij} \qquad (c_{ij} > 0)$$

for every arc (N N). Furthermore, algorithm A* assumes that the bottom node N_b. has a cost $c_b = 0$ and that the estimate h (x) is also an additive function, i.e. $h_i(x) = x + h_i$ Thus, the total estimate becomes, in this case, $e_i = g_i + h_i$

Since our algorithm deals with the general case of monotone cost functions, the extension is nontrivial and some new features have to be added First of all we point out that the solution path can contain the top node N more than once This happens when the distance function from the top node to itself is smaller than the identity function ($d_{tt}(x) < x$, for soir.c x), i.e. the top node belongs to a cycle which can reduce its cost This fact is taken into account by adding a new node N_0 to the graph and an arc (N_oN) with the identity function lix) associated with it. The algorithm will stop only when this node will be encountered.

Furthermore, a solution path can contain the same node several times. Therefore, if we want to obtain the solution path by tracing back through the pointers from the top node, we must have more than one pointer for each node. This is achieved by associating with each node a *stack* of pointers, to which a new element is added every time the node is reopened.

Algorithm SEARCH

1) Add a new node N_0 to the graph and an arc (N_0 N_t). Let the cost function of this arc be $f_{01}(x) = 1(x) = x$ and $h_0(x) = 1(x)$.

2) Put the bottom node N_b on a list called OPEN and let $g_b \leftarrow c_b$ and $e_b \leftarrow h_b(c_b)$.

3) If OPEN is empty, exit with failure, otherwise continue.

4) Remove from OPEN that node N_i whose total estimate e_i is smallest and put it on a list called CLOSED. In case of tie on the e values, choose the node N_i whose g_i value is smallest. Resolve further ties arbitrarily, but always in favor of node N_0.

5) If N_i is node N_0, exit with the solution path obtained by tracing back through the pointers; otherwise continue.

6) Expand N_i, generating all of its parents. For each parent N_j compute $g_j' \leftarrow f_{ji}(g_i)$.

7) Associate with the parents N_j not already on either OPEN or CLOSED the values $g_j \leftarrow g_j'$ and $e_j \leftarrow h_j(g_j)$. Put these nodes on OPEN and put on top of their stacks a pointer back to the top of the stack of N_i.

8) If a parent N_j is on OPEN and $g_j' < g_j$ let

$$g_j \leftarrow g_j' \text{ and } e_j \leftarrow h_j(g_j).$$

Replace the pointer on top of the stack of N_j with a pointer to the top of the stack of N_i.

9) If a parent N_j is on CLOSED and $g_j' < g_j$, put it on OPEN and let

$$g_j \leftarrow g_j' \text{ and } e_j \leftarrow h_j(g_j).$$

Add on top of the stack of N_j a pointer to the top of the stack of N_i.

10) Go to 3.

It is possible to prove that the above algorithm is *admissible (i.e. if it terminates cleanly* it finds an optimal solution), if the estimate $h_i(x)$ is a lower bound on the distance $d_{ij}(x)$ for each node ($h_i(x) \leq d_{ij}(x)$ for each x and for each i). The proof is contained in the Appendix.

The proof is based on a lemma which states that, if the estimate $h_i(x)$ is a lower bound on $d_{ij}(x)$, then at any iteration of algorithm SEARCH there is an open node N_j with $c_j \leq c_i$. This result is not trivial since the solution cost c_i might be achieved only as a limit, and thus infinite paths have to be considered.

Notice that, in general, there is no way of deciding whether the algorithm will stop. Special classes of functions and estimates for which such a decision can be made will be presented later on.

As a further remark we point out that the bottom-up structure of our algorithm is the most natural choice for the general case. In fact, when no estimate is available, with our algorithm we propagate *costs*, i.e. constants, from the solved node toward the top node, whereas, by using a top-down approach, we should propagate *functions*. Such a difference between bottom-up and top-down is not present in the case of additive cost functions, where functions can be propagated very easily. In fact, bidirectional search algorithms have been given for this case [12].

Algorithm SEARCH can be easily extended to the case where the estimate $h_i(x)$ is not a lower bound on $d_{ij}(x)$, but the maximal amount by which $h_i(x)$ can exceed $d_{ij}(x)$ is known. This case has been studied in [13] for additive cost functions.

Assume that a strictly monotone* *error function* $b(x)$ is given such that $h_i(x) \leq b(d_{ij}(x))$ for every x and for every node. Algorithm SEARCH can be modified by associating with the arc $(N_0\ N_i)$ the cost function b(x). It is easy to see that the new algorithm finds the solution cost of node N_i. In fact, the distance function from N_0 to every node N_i is $d_{0i}(x) = b(d_{ii}(x))$ and the estimate $h_i(x)$ can be considered as a lower bound on $d_{0i}(x)$. Therefore the algorithm finds the solution cost of node N_0, but, since every solution path for N_0 is a solution path for N_i as well, for the strict monotonicity of b, it finds also the solution cost of N_i.

In order to see how algorithm SEARCH works, let us consider the following problem. A functionally weighted graph is given with n + 1 nodes, where each node N_i (i = 1, ..., n) corresponds to a different currency C_i. A directed arc $(N_i\ N_j)$ denotes the possibility of exchanging currency C_i into C_j. To each arc $(N_i\ N_j)$ (i, j = 1, ..., n) we associate a cost function $f_{ij}(x) = a_{ij}\,x + b_{ij}$ ($a_{ij} > 0$, $b_{ij} > 0$), which means that, if we want to have the amount x of currency C_j, we have to change an amount of currency C_i proportional to x plus a fixed amount for taxes. Furthermore, the solved node N_{n+1} corresponds to some goods and, for each arc $(N_i\ N_{n+1})$, the value $f_{i,n+1}(c_{n+1})$ gives the cost of these goods in the currency C_i.

Our problem is that of buying the given goods with the smallest amount of currency C_1. Notice that, if we are quite optimistic, we can hope of finding a cycle which gives us some gain, and thus the solution path may contain some cycles.

An example is given in Fig. 2. The operator/in the cost functions denotes integer division. The given estimates imply that very little heuristic information is assumed. However it is clear that they are lower bounds if the domain of the cost functions are the nonnegative integers.

In Fig. 3 we give some steps of the execution of algorithm SEARCH. In Fig. 3a we show the stacks of pointers after node N_2 has been expanded for the first time. Each element of a stack has a number associated with a pointer. This number is the g value of the node, and it gives the cost of the path obtained by following the pointers. Furthermore, the encircled numbers are the total estimate e of the open nodes. Notice that every time a node is reopened, a new element is added to the stack of pointers.

Finally, in Fig. 3b we give the last step of the algorithm. We see that the solution cost of node N_1 is 8 and the solution path, obtained by tracing through the pointers, is $N_1 N_2 N_3 N_2 N_3 N_5$. Thus we have obtained a solution path with a cycle.

We want to see now how algorithm SEARCH behaves when no estimate is available. This means that, if we do not put any restriction on the cost functions, we must have $h_i(x) = -\infty$ for each node. At each step

* A function f is strictly monotone if $x_1 < x_2 \rightarrow f(x_1) < f(x_2)$.

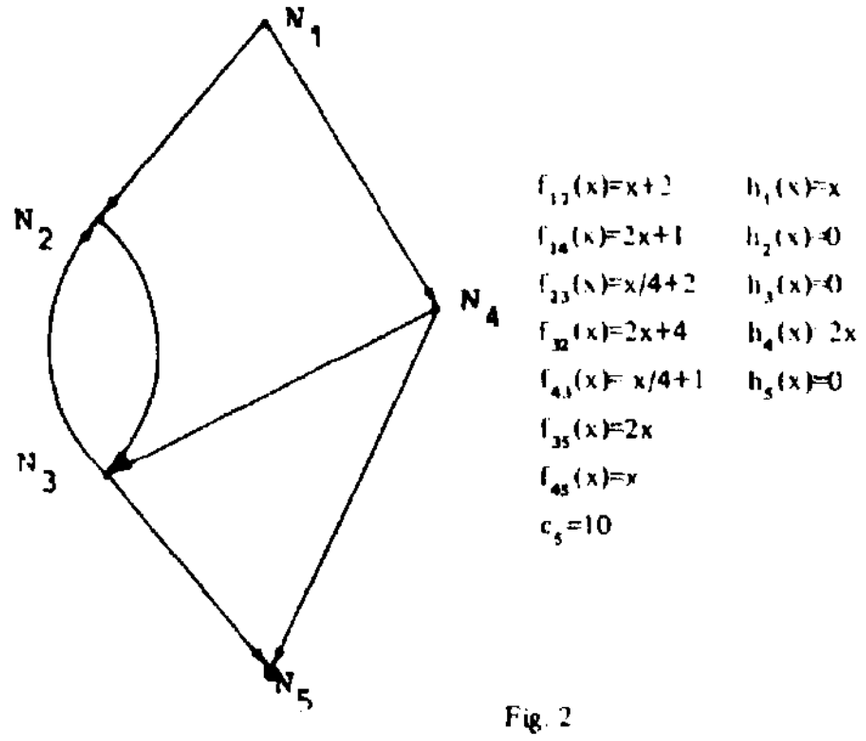

Fig. 2

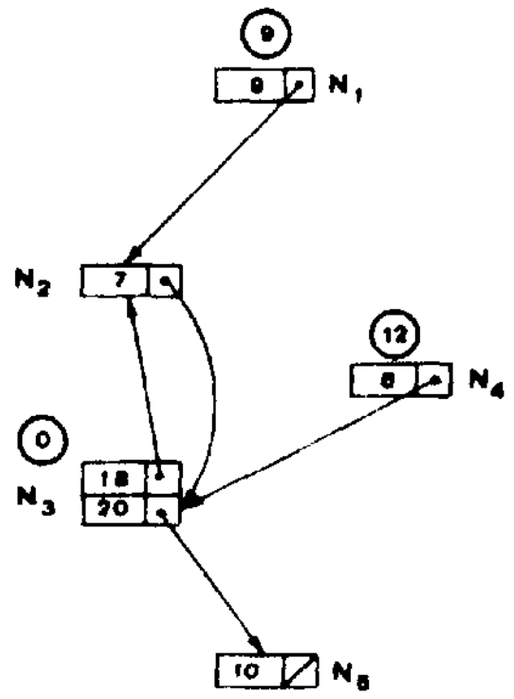

Fig. 3a

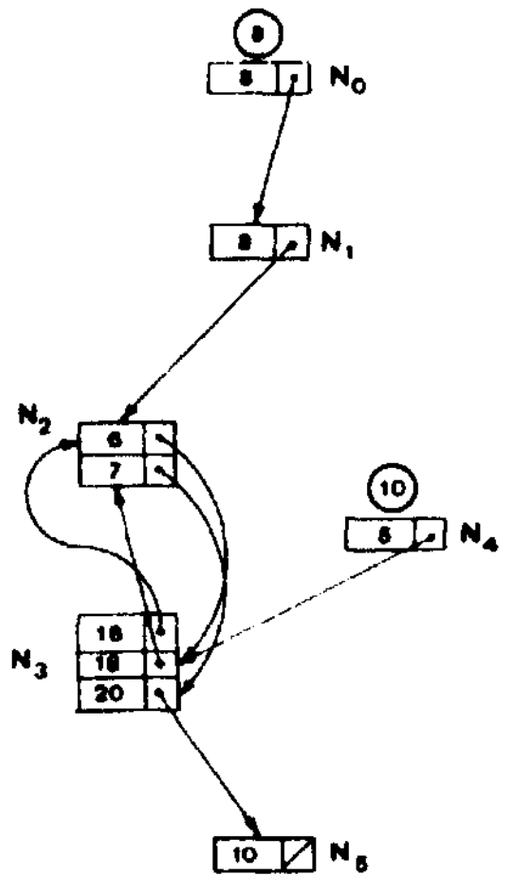

Fig. 3b

of the algorithm, each node N_i has a total estimate $e_i = -\infty$, except for node N_0 for which we have $e_0 = g_0$. According to the condition on ties in step 4 of the algorithm, the node whose g_i value is smallest is expanded at each step Since N_0 is the only node with total estimate $\neq -\infty$, the algorithm will stop only after all other nodes have been closed. In this case it is clear that the solution cost of *all* nodes is determined simultaneously. Thus the estimate $h(x) = -\infty$ for each node can be always assumed when the above more general problem must be solved.

Note that this algorithm may be used in the case of additive cost functions, if negative weights are allowed and no estimate is available.

4 Positively Monotone Cost Functions, and Consistent and Strictly Monotone Estimates

The first case we examine is when all cost functions are positively monotone. A monotone function is called *positively monotone* iff for all arguments x we have

$$f(x) \geq x$$

Examples of positively monotone functions are as follows

$f(x) = x + c \qquad c \geq 0$
$f(x) = ax \qquad a \geq 1 \qquad x \geq 0$
$f(x) = \max(x,c)$

It is easy to sec that the composition of two positively monotone functions is still a positively monotone function, and thus the cost function of every path is also a positively monotone function As a consequence, the estimates can also be assumed positively monotone without restrictions

For positively monotone cost functions we have the following theorem, (For the proofs of all theorems in this section, see the Appendix).

Theorem 4.1. Let C be a functionally weighted graph with positively monotone cost functions Then there always exists a simple solution path from the top node.

From the above theorem it lollows that algorithm SEARCH cm take into account only temporary solution paths without cycles Thus the algorithm can be simplified by eliminating the slacks of pointers and by replacing them with only one pointer for each node (as in algorithm A*).

When the estimate h is strictly monotone, looking at g values in step 4 of algorithm SEARCH for solving tics is no longer useful since $h_i(g_i) = h_j(g_j)$ implies $g_i = g_j$ hus in this case algorithm SEARCH behaves exactly as A^*.

Note that positive monotonicity alone, while insuring termination, does not guarantee that all closed nodes are never reopened. This case is well-known in the special, classical case of a graph weighted with positive constants, with estimate |4|. Following |4|. we thus introduce consistent estimates An estimate $h_k(x)$, $k = 1, \ldots, n$ is *ctinsistent* iff for all arcs $(N_j\ N_i)$ and lor all x we have

$$h_i(x) \leq h_j(f_{ji}(x))$$

In the case of consistent estimate, but not necessarily for positively monotone cost functions, we can prove the following theorem

Theorem 4.2 Let (i be a funchonally weighted graph with consistent estimate $h_k(x)$, $k = 1, \ldots, n$. Then the values of the total estimate e of the nodes closed in the successive iterations of algorithm SEARCH form a (possibly infinite) nondecrcasing sequence

The above result does not mean that closed nodes are never reopened. In fact a node N_i could he closed a second time with the same estimate c_i but with an improved cost g (Since $\bar{g}_i < g_i$, otherwise N would not be reopened, the monolonicily of h does not allow $\bar{c}_i > c_i$).

For instance, let us consider the graph in Fig 4 The exact distances arc as follows

$$d_{tt}(x) = \min(x, 4)$$
$$d_{ta}(x) = \min(x+4, 4)$$
$$d_{tb}(x) = \min(x+3, 4)$$

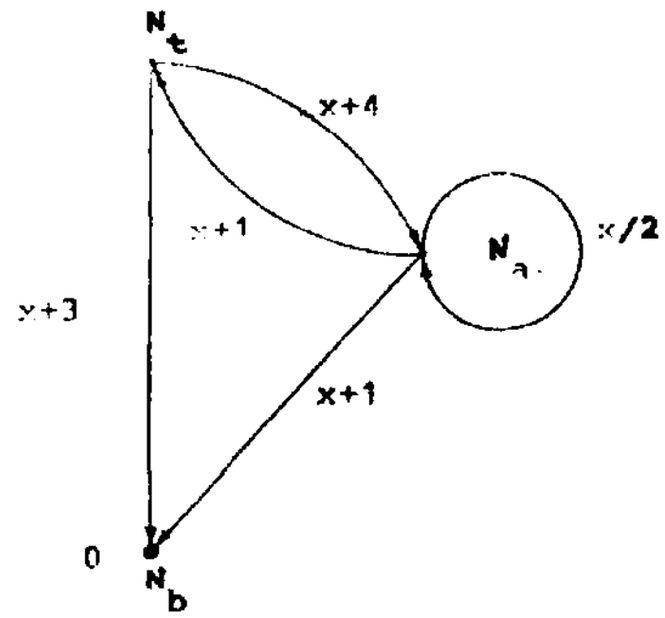

Fig. 4

Let us take as estimate the following functions

$$h_t(x) = \min(x-1, 5/2)$$
$$h_a(x) = \min(x+3, 5/2)$$
$$h_b(x) = \min(x+1, 2)$$

is easy to see that this estimate is consistent. For instance, for the loop node N_a we have

$$\min(x+3, 5/2) \leq \min(x/2+3, 5/2)$$

owever algorithm SEARCH does not stop, even if a solution path exists, id reopens node N_a infinitely many times. The values of the costs and the total estimates for nodes closed in successive iterations are as llows.

) node N_b $g_b = 0$ $e_b = 1$
) node N_t $g_t = 3$ $e_t = 2$
) node N_a $g_a = 1$ $e_a = 5/2$ we do not stop since $e_0 = g_t = 3 > e_a = 5/2$
) node N_a $g_a = 1/2$ $e_a = 5/2$ "
) node N_a $g_a = 1/4$ $e_a = 5/2$ "
id so on.

The following result allows to characterize the set of closed nodes hich could possibly be reopened.

heorem 4.3. Let G be a functionally weighted graph with consistent timate $h_k(x)$, $k = 1, \ldots, n$. At some stage of algorithm SEARCH, let N_i : a closed node with cost g_i and total estimate e_i *smaller* than the total timate e_j of all open nodes. Then node N_i will never be reopened, and s solution cost is g_i.

If the estimate is consistent, the above theorem allows to add a new termination condition to algorithm SEARCH: "If the top node in closed and has a total estimate *snwllcr* than the node currently under closure, stop with success". This condition allows for instance to slop at the third iteration of the previous example.

If the two conditions of positive monotonicity of the cost functions and of consistency of the estimate are taken jointly, we can prove the following theorem, which is an extension of a result bv Hart. Niluon and Raphael |4|. <

Theorem 4.4. Let G be a functionally weighted graph with positively monotone cost functions and with a consistent estimate Then when algorithm SEARCH closes a node N.. its cost g_j is the solution cost of N . Thus algorithm SEARCH never reopens a node.

The above theorem provides an obvious termination condition for algorithm SEARCH "Stop when the top node is closed".

The above results apply, as pointed out before, to the well-known

case of additive cost functions. However their generality allows also the application to cost functions of different form, where the standard A* algorithm cannot be used An example is provided by the so called *secondary optimization problem* which arises when we want to find the best ordering of vanahlc elimination in non-serial dynamic programming [14| The problem can be stated as that of finding the minimal cost path in a weighted graph, but where the cost of a path is the maximum among all the weights of its arcs Therefore the cost functions of this graph are of the form

$$f_{ij}(x) = \max(x, c_{ij})$$

i.e. they are positively monotone functions and algorithm SEARCH can be used for finding the solution path, if suitable estimates are provided*

A further example of positively monotone but nonadditive cost I unctions is the so called *I-dimensional stock cutting problem [*1], where we want to determine the minimum number of standard pieces of material of length L from which it is possible to cut a_t pieces of length 1_1, a_2 pieces of length 1_2, ... and a_n pieces of length I In the graph representation (if this problem, the meaning of an arc is "cut a piece of length I ' and the cost (unction associated with it is

$$f(x) = \begin{cases} x + l_j & \text{if } x + l_j \leq L\lceil x/L \rceil \\ L\lceil x/L \rceil + l_j & \text{otherwise} \end{cases}$$

The second case in the definition of f(x) corresponds to the situation in which the current standard piece does not have enough material remaining for the next cut, and a new standard piece is required

According to Theorem 4.2 and 4.3, if the estimate in consistent, then a closed node N can be reopened only with the same total estimate c - c and with a better cost g < g. Thus it is clear that, if the estimate is strictly monotone, then no closed node can be reopened Thus the same result proved in Theorem 4 4 holds also in this case

As a conclusion, we point out that in this paper we have given algorithm SEARCH for searching functionally weighted graphs which is an extension of algorithm A*, and we have shown that, for generic monotone functions, this algorithm requires some new features such as the addition of node N_o, the stack of pointers and the tie handling procedure in Step 4. Furthermore, we have shown that, by considering restricted classes of cost functions and estimates, simpler versions of algorithm SEARCH can be given In particular, the validity of the trivial generalization of A* is extended cither to the case of strictly monotone estimates and positively monotone cost functions or to the case of strictly monotone and consistent estimates Properties of algorithm SEARCH for another special class of cost functions (stricly monotone cost functions) arc described and proved in [11].

References

1. Karp. R.M. and Held, M., finite slate processes and dynamic prog ramminp *SIAM J Appl Math, 1*5. 1967, pp. 693-718
2. Martelli, A. and Montanari, U., On the foundations of dynamic programming, N.I B73-11,1.fc.I , Pisa, Oct 1973, to appear in *Topics in CombinatorialOptimization*. S Rinuidi ed.. CISM, Springer Verlag.
3. Martelli.A, and Montanari U., Dynamic programming schemata, *Proc. second Coll. on Automata, Languages and Programming.* Saarbrucken, July 1974, Springer Verlag. pp 66-80
4 Hart, P., Nilsson, N. and Raphael, H , A formal basis for the heuristic determination of minimum cost paths, *It'.t'E Trans, on Svs. Sci Cybernetics,SSC-4,N* 2, July 1968, pp 100-107
5. bellman, R.E., *Dynamic Programming*, Princeton University Press, Princeton N J. 1957.
6 Dijkstra, E., A note on two problems in connection with graphs, *NumenscheMath.. I,* 1959. pp. 269-271
7 Johnson. D.B., Algorithms for shortest paths. Ph. 0. Thesis. Cornell University, Ithaca, NY., May 1973
8. Martelli, A and Montanari, U., Additive AND/OR graphs, *Proc Third Int. Joint Conf. on Art. int.,* Stanford. 1973. pp. 1-11
9. Martelli.A and Montanari, U.. Programma/ione dinamicac punto f*isto,AttiConv. dilnformatica Teorica,* Mantova, Nov. 1974, pp. 1-19 (in Italian)
10 Nilsson, N J , *Problem-solving methods in artificial intelligence,* McGraw-Hill, NY.. 197 |
11. Martelli, A. and Montanari, U., From dynamic programming to search algorithms with functional costs, IEI Internal Report B75-I, January 19'75
12. Pohl, I , Bi-directional and heuristic search in path problems, Ph.D. Thesis, Stanford University, Stanford, 1969.
13. Harris, L.R., The heuristic search under conditions of error. *Artificial Intelligence,* 5.N. 3, Fall 1974, pp. 217-234
14 Bnoschi, F and Even, S., Minimizing the number of operations in certain discrete-variable optimization problems, *Operations Research, 18,* N. 1, 1970, pp. 66-81

Appendix

Lemma 3.1. If $h_j(x) \leq d_{tj}(x)$ for every x and for every node, then at any step of the algorithm and for any path p from N_0 to N_b, there exists an open node N_j on p with $e_j \leq f_p(c_b)$.

Proof. Let N_j be the first open node we encounter by following path p backwards from N_b to N_0. There must be such a node because otherwise N_0 is closed and the algorithm has terminated. Let p' be the part of path p from N_j to N_b. Since all nodes on p' are closed, we must have

$$g_j \leq f_{p'}(c_b) .$$

Since the distance function $d_{tj}(x)$ is a monotone function and since, in general, p is not a solution path, we have

$$d_{tj}(g_j) \leq d_{tj}(f_{p'}(c_b)) \leq f_p(c_b)$$

Finally, since $h_j(x)$ is a lower bound on $d_{tj}(x)$, we have

$$e_j = h_j(g_j) \leq d_{tj}(g_j) \leq f_p(c_b) \qquad \text{Q.E.D.}$$

Lemma 3.2. If $h_j(x) \leq d_{ti}(x)$ for every x and for every node, then at any step of the algorithm there exists an open node N_j with $c_j \leq c_t$.

Proof. If c_t is achieved with a finite path p, then $c_t = f_p(c_b)$ and the lemma is proved by Lemma 3.1. Assume now that c_t is obtained as a limit, and let $p_1, p_2, p_3, \ldots$ be an infinite sequence of paths from N_0 to N_b such that

$$f_{p1}(c_b) \geq f_{p2}(c_b) \geq f_{p3}(c_b) \geq \cdots \geq c_t$$

and $\lim_{i \to \infty} f_{pi}(c_b) = c_t$.

Let us apply now the construction of Lemma 3.1 to each path of the sequence and let us consider the subsequences of paths for which the same node is obtained. Among these subsequences, there must be an infinite subsequence $p_{j1}, p_{j2}, p_{j3}, \ldots$ such that

$$\lim_{i \to \infty} f_{pji}(c_b) = c_t .$$

Let N_j be the node obtained with the construction of Lemma 3.1 from these paths. We have

---

* Note that if estimates of the same type $h_j(x) = \max(x,c)$ are considered, the tie handling procedure in Step 4 of algorithm SEARCH is essential for Theorem 4.4 to hold. The same happens whenever the estimates are not strictly monotone.

$$e_j \leq f_{pji}(c_b) \qquad i = 1, 2, 3, \ldots$$

Therefore we have

$$e_j \leq c_t \qquad \text{Q.E.D.}$$

Theorem 3.1. If $h_i(x) \leq d_{ti}(x)$ for every x and for every node, and if the algorithm terminates cleanly (i.e. without failure), then it finds the solution cost c_t of node N_t.

Proof. When the algorithm stops we have $e_0 = h_0(g_0) = f_{0t}(g_t) = g_t$. Assume now that e_0 is not the solution cost of N_t, i.e. $e_0 = g_t > c_t$. By lemma 3.2 we know that there existed just before termination an open node N_j with $e_j \leq c_t$. Thus, N_j, which must be different from N_0, would have been selected for expansion rather than N_0, contradicting the hypothesis that the algorithm terminates. Q.E.D.

Theorem 4.1. Let G be a functionally weighted graph with positively monotone cost functions. Then there always exists a simple solution path from the top node.

Proof. Given a nonsimple path P_{tb} of cost

$$p_{tb}(c_b) = p_{ti}(p_{ii}(p_{ib}(c_b)))$$

there always exists a *simple* path P'_{tb} of not larger cost

$$p'_{tb}(c_b) = p_{ti}(p_{ib}(c_b))$$

In fact, $p_{ii}(x)$, being the composition of positively monotone functions, is positively monotone and thus

$$p_{ib}(c_b) \leq p_{ii}(p_{ib}(c_b))$$

But $p_{ti}(x)$ is monotone, so we have

$$p'_{tb}(c_b) = p_{ti}(p_{ib}(c_b)) \leq p_{ti}(p_{ii}(p_{ib}(c_b))) = p_{tb}(c_b)$$

As a consequence, any path of minimum cost in the *finite* set of simple paths is optimal. Q.E.D.

Theorem 4.2. Let G be a functionally weighted graph with consistent estimate $h_k(x)$, $k = 1, \ldots, n$. Then the values of the total estimate e_i of the nodes closed in the successive iterations of algorithm SEARCH form a (possibly infinite) nondecreasing sequence.

Proof. When a node N_i is closed by algorithm SEARCH its total estimate $e_i = h_i(g_i)$ by construction is not larger than the total estimates e_k of all open nodes. In the updating phase, the total estimate of some node N_j, adjacent to N_i, may be decreased as follows:

$$\bar{e}_j = h_j(f_{ji}(g_i))$$

and, if N_j was closed, it will be reopened. However, the new value $\bar{e}_j$ is not smaller than e_i by the consistency assumption. Thus, the total estimate of the next closed node, which is the minimal among the total estimates of all open nodes after updating, will not be smaller than e_i.

Q.E.D.

Theorem 4.3. Let G be a functionally weighted graph with consistent estimate $h_k(x)$, $k = 1, \ldots, n$. At some stage of algorithm SEARCH, let N_i be a closed node with cost g_i and total estimate e_i *smaller* than the total estimate e_j of all open nodes. Then node N_i will never be reopened, and its solution cost is g_i.

Proof. The cost g_i is clearly not worse than the costs of all paths visiting only closed nodes. Let now P be any path from N_i to N_b which passes through at least one open node, and let N_j be the last open node on P. We have

$$p_{ib}(c_b) = p_{ij}(p_{jb}(c_b))$$

and

$$p_{jb}(c_b) \geq g_j .$$

From the consistency assumption, extended to the part of P from N_i to N_j we have

$$h_j(g_j) \leq h_i(p_{ij}(g_j)) \leq h_i(p_{ib}(c_b)) .$$

But since

$$h_i(g_i) < h_j(g_j)$$

by assumption, we have

$$h_i(g_i) < h_i(p_{ib}(c_b))$$

and finally, since h_i is monotone,

$$g_i < p_{ib}(c_b) .$$

Clearly, since g_i is already the optimal cost, node N_i will never be reopened.

Q.E.D.

Theorem 4.4. Let G be a functionally weighted graph with positively monotone cost functions and with a consistent estimate. Then when algorithm SEARCH closes a node N_i, its cost g_i is the solution cost of N_i. Thus algorithm SEARCH never reopens a node.

Proof. The proof is quite similar to that of Theorem 4.3. We can write again

$$p_{ib}(c_b) = p_{ij}(p_{jb}(c_b))$$

and

$$p_{jb}(c_b) \geq g_j .$$

Now if

$$h_i(g_i) < h_j(g_j)$$

the result of the proof of Theorem 4.3 is valid and the thesis follows. On the other hand, if

$$h_i(g_i) = h_j(g_j)$$

then we have

$$g_i \leq g_j$$

since N_i was closed before N_j. Thus we have

$$g_i \leq p_{jb}(c_b) .$$

But since $p_{ij}(x)$ is now positively monotone we have

$$g_i \leq p_{ij}(p_{ib}(c_b)) = p_{ib}(c_b) \qquad \text{Q.E.D.}$$