UMIACS-TR-90-107
CS-TR-2518

August 1990

Performance of A* and IDA*— A Worst-Case Analysis

A. Mahanti* A.K. Pal† S. Ghosh‡
L.N. Kanal§ D.S. Nau¶

Abstract

This paper presents a detailed comparison between Algorithms A* and IDA*. A necessary and sufficient condition has been derived to show when IDA* can have the same worst-case time complexity as A* for tree searches. This condition is in apparent conflict with conditions stated in previous analyses of IDA*.

Under a monotone heuristic, A* has $O(N)$ time complexity in the worst case for any directed graph (either cyclic or acyclic), where N is the number of nodes n such that $f(n) \leq f^*(s)$. However, we show that the worst-case time complexity of IDA* is $O(N^2)$ for trees and $O(2^{2N})$ for acyclic graphs. These worst-case bounds hold even if the heuristic function is monotone and has constant relative error, and the heuristic branching factors is > 1.

We show that, in general, no heuristic search algorithm which runs in limited-memory (i.e., the same order of memory used by IDA*) can have the same $O(N)$ asymptotic time complexity as A* under a monotone heuristic. However, for trees, IDA* is asymptotically optimal amongst all limited-memory algorithms.

*Institute for Advanced Computer Studies, Department of Computer Science, and Systems Research Center, University of Maryland, College Park, MD 20742. Email: am@cs.umd.edu.

†Indian Institute of Management Calcutta, Diamond Harbour Road. P. Box No. 16757, Calcutta 700 027, India. Currently visiting Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.

‡Department of Computer Science, University of Maryland, College Park, MD 20742. Email: subrata@cs.umd.edu.

§Department of Computer Science, University of Maryland, College Park, MD 20742. Email: kanal@cs.umd.edu.

¶Department of Computer Science, Systems Research Center, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Email: nau@cs.umd.edu. This work was supported in part by an NSF Presidential Young Investigator award for Dr. Nau with matching funds from Texas Instruments and General Motors Research laboratories, NSF Grant NSFD CDR-88003012 to the University of Maryland Systems Research Center, NSF Equipment grant CDA-8811952, and NSF grant IRI-8907890.

1 Introduction

A network (or search graph) G is a directed graph with one start node and a nonempty set of goal nodes $t, t_1, t_2, \ldots$. There are other nodes m, n, p, q, r in G. Each directed arc (m, n) in G has a finite positive arc cost $c(m, n) \geq \delta > 0$. A path in G is a finite sequence of directed arcs starting at some node m and ending at some other node n. A solution path is a path that begins at the root node s and ends at a goal node. The cost of a path is the sum of the costs of its arcs. The objective of any heuristic search algorithm for networks is to find a solution path of minimum cost in G. To find such a path, an algorithm uses a nonnegative heuristic estimate $h(n)$ associated with every node n in G.

For every node n in G, a node evaluation function

$$f^*(n) = g^*(n) + h^*(n)$$

is defined. While $g^*(n)$ is the cost of a minimal cost path from s to n, $h^*(n)$ is the cost of a path of minimum cost from n to a goal node. If no goal node can be reached from n, then $h^*(n)$ is taken to be infinite. When a heuristic search algorithm is executed, let $g(n)$ be the cost of the path of least cost currently known from s to n, and $h(n)$ be the estimate of $h^*(n)$. Then $f(n)$ is viewed as an estimate of $f^*(n)$. The network G can have infinitely many nodes and arcs and G can have loops. For each node n in G, we assume that if $h^*(n)$ is finite then $h(n)$ is finite.

Under this given model Alg.A* [5] works in a best first manner. It maintains two lists OPEN and CLOSED. OPEN contains nodes which are to be expanded and CLOSED contains already expanded nodes. At any iteration, A* selects a node from OPEN with minimum f-value. Ties are resolved arbitrarily, but always in favour of a goal node. Newly generated successors are put in OPEN with their g, h, and f values. If an immediate successor of the expanded node is already present in CLOSED, and a better path to it is now found, the g value of the node is updated and the node is brought back to OPEN from CLOSED.

One major problem with A* is that for storing nodes in OPEN and CLOSED, it requires

exponential amount of memory as the search goes deeper and deeper. Thus it becomes impossible to solve any practical problem of resonable size using A*. In a recent study Korf [4] presented a new algorithm namely IDA*. IDA* unfolds a graph to a tree. In every iteration, IDA* starts the search from the root node and makes a depth first search within the current threshold. Threshold for the next iteration is calculated as the minimum f-value of the discarded nodes of the previous iteration. IDA* has the property that it can overcome the storage limitation of A*. Assuming that the branching factor of a node is constant,IDA* requires $O(L)$ memory, where L is the maximum possible length of any minimal cost solution path. Thus, given sufficient execution time IDA* can solve relatively harder combinatorial optimisation problems. Following the idea of IDA*, several other limited-memory algorithms have been designed [1, 2, 3, 7].

It has been shown in [4] that IDA* is asymptotically optimal amongst all admissible best first search algorithms for tree searches. However, the optimality proof of IDA* assumes a very stringent condition that the number of new nodes grow exponentially from threshold to threshold. In this paper we present a necessary and sufficient condition for the $O(N)$ time complexity of IDA*, where N is the total number of nodes having f-value $\leq h^*(s)$. We show that IDA* can become $O(N^2)$ for tree searches in absence of any such conditions. When the heuristic is monotone, A* can handle a graph like a tree and it never expands a node more than once. But, for graphs IDA* can not prevent the reexpansion of a node through a costlier path. Thus the performance of IDA* is bound to become worse than A*. We show that for acyclic graphs the worst case time complexity of IDA* is $O(2^{2N})$. In the last iteration of IDA* there can be $O(2^N)$ node expansions. Total number of node expansions can increase in presence of cycles. These $O(N^2)$ and $O(2^{2N})$ upper bounds for trees and acyclic graphs hold under all possible assumptions such as monotone heuristic, constant relative error, heuristic branching factor > 1, and identical tie resolutions with A*.

Finally we show that there can not exist any limited-memory best first search algorithm which will have the linear time complexity as A* in trees, and in graphs with monotone heuristic. We conjecture, IDA* is asymptotically optimal among all limited-memory best

first search algorithms for trees.

2 IDA* on Trees

Definition 1: For any given tree (or network) G, by N we mean the number of nodes in G having f-value $\leq h^*(s)$.

Definition 2: L is the maximum number of nodes in any path generated by A*.

Definition 3: Let G be a tree and $s = n_1, n_2, \ldots, n_N$ be the nodes in G such that, $0 \leq f(n_i) \leq h^*(s)$. Since the branching factor of a node is assumed to be constant, there can be only finitely many such $n_i's$. Let $0 \leq f_1 < f_2 < f_3 < \ldots < f_X = h^*(s)$ be the (maximum possible) distinct f-values of all $n_i's$. Clearly if IDA* is run on G, there will be X thresholds and it will run for X iterations.

Definition 4: Let t_i be the number of new nodes at threshold i. Let $i_1, i_2, \ldots, i_u$ be the thresholds such that $t_{i_{j+1}} > t_{i_j}$, $1 \leq j \leq u$. Let b_1 be the average of the ratios $t_{i_{j+1}}/t_{i_j}$.

Definition 5: let $M = X - u$.

Example 1.

Consider the search graph given in Figure 1(a)

b = Node branching factor = 2.

Cost of each arc (m, n) is $c(m, n) = 1$, $\forall (m, n) \in G$.

$h(n) = 0$ $\forall n \in G$ (monotone heuristic)

Error in the heuristic function = 100% (constant relative)

heuristic branching factor = $[1 + 2 + 2 + 2 + 0.25 + 2 + 2] \div 7 = 1.607$

Every leaf node is nonterminal except the rightmost one, which is a goal node. Now, let us generalize the search tree G of figure 1(a), such that the total number of nodes $N = 2N'$, and the cost of the solution path = $2[\log_b(N' + 1) - 1]$ (which is not exponential in the depth

of the search tree).

Let

$$d = [\log_b(M+1) - 1]$$

and

$$N_0 = b^d + 2b^{d-1} + 3b^{d-2} + \ldots + db.$$

Now the total number of node expansions by IDA* is

$$dN_0 + (b^d + 2b^{d-1} + 3b^{d-2} + \ldots + db) = N_0(d+1) = O(N \log_b N).$$

Similarly, if we consider the search tree in figure 1(b) where there are $N = 2N'$ nodes as in figure 1(a) and the last N' nodes form a degenrate tree. Every leaf node is a nonterminal except the rightmost one, which is a goal node. Cost of the solution path $= N' + \log_b N'$, node branching factor > 1, and heuristic branching factor > 1. All other conditions are same as in Figure 1(a). Clearly, IDA* will make $O(N^2)$ node expansions for this search tree in the worst case.

Remark: It is easy to construct an example tree with varying arc cost so that the heuristic function satistfies proportional error [6] and the heuristic branching factor will be greater than 1 but the number of thresholds will be proportional to N. And, in that case IDA* will have $O(N^2)$ asymptotic complexity again.

Theorem 1. The asymptotic time complexity of IDA* is O(N) iff the following conditions hold:

$$(i) \lim_{N \to \infty} u = \infty$$

and

$$(ii) \lim_{N \to \infty} M < \infty$$

Proof:

Case 1: (If part) By assumption the conditions hold. Let $Y = b_1^u + 2b_1^{u-1} + \ldots + ub_1$. Therefore, the total number of node expansions by IDA* is $\leq Y + MY = Y(1+M)$. Since M is finite, the upperbound for the total number of node expansions by IDA* is $O(N)$.

Case 2: (only if Part) Given that IDA* has the asymptotic complexity of $O(N)$, we need to show that the two conditions hold. We prove it by contradiction. As in case 1, suppose the total number of node expansions by IDA* is $Y(1+M)$. Now we consider the three subcases below:

(a) Condition 1 is true but condition 2 is false.
Clearly, then $(1+M)Y = O(N^2)$ - a contradiction.

(b) Condition 1 is false and condition 2 is true.
Then the total number of node expansions by IDA* will be bounded by a constant, which is impossible. This is because, in the worst case, every node must be expanded at least once. Thus a contradiction is immediate.

(c) Condition 1 and condition 2 are both false.
Clearly, the total number of node expansions by IDA* will be greater than or equal to $\frac{M(M+1)}{2}$, which will be $O(N^2)$ - a contradiction.

Theorem 2. In case of trees with heuristic branching factor > 1, there does not exist any limited-memory best-first search algorithm which uses $O(L)$ memory and has worst-case time complexity $O(N)$.

proof: (By contradiction.) Let $n_1, n_2, \ldots, n_N$ (with $n_1 = s$) be the nodes in the search tree having f-value less than or equal to $h^*(s)$. Let P_i be the path from s to n_i. In the worst case, we can assume that for each n_i, $f(n_i) = \text{cost}(P_i) + h(n_i)$ is distinct. The n_i's can be placed in the tree in such a way that for $i = 2, \ldots, N-1$, P_i and P_{i+1} have no nodes in common except the root node s (see Figure 2). Now, an algorithm under consideration will have the following situation. After the expansion of n_i, it realizes that the next node to be expanded is n_{i+1}. However, it might so happen that after the expansion of n_{i+1}, it has to expand one, say immediate successor of n_i. Thus the algorithm will have two choices: to re-expand the nodes from s to n_i, or to remember the children of n_i. Each case is discussed below:

Case 1: remember the children of n_i. Now we can extend this logic further, and find that after the expansion of n_{i+1}, the algorithm has to expand a child of the child of the

immediate parent of n_i and so on. Thus after the expansion of n_{i+1}, if the algorihtm has to find the next node to be expanded which belongs to a different subtree below s and then the algorithm needs to store that subtree (to avoid re-expansion), then a contradiction is immediate that it cannot run with $O(L)$ memory.

Case 2: re-expand the nodes from s to n_i. Since no nodes (except possibly only the root node s) of the subtree below s to which n_i belongs have been stored, to find n_i's child say n_{i+2}, the algorithm has to expand all nodes of the subtree having f-value less than $f(n_{i+2})$ (because no other identification of n_{i+2} has been stored). Now there can be $O(N)$ nodes in the subtree with f-value less than $f(n_{i+2})$. Again, in the worst case there can be $O(N)$ such n_i's. Thus a contradiction is immediate and the algorithm cannot run in $O(N)$ time.

3 IDA* on Acyclic Graphs

What happens if we run IDA* on directed acyclic graphs ? For graphs with monotone heuristic, Algorithm A* runs in $O(N)$ time. When a node is expanded $g(n) = g^*(n)$ and no node is expanded more than once. Since IDA* runs in linear memory (linear in L), it can not store all expanded nodes for future duplicate checking as in A*. Thus IDA* can expand a node several times due to unfolding of a graph into tree. The following two examples will explain the worst case time complexity of IDA* on directed acyclic graphs.

Example 2:

Consider the search graph shown in Figure 3. Here N =3K +1. Each arc has unit cost and $h(n) = 0$ for every node n. If we run Alg. A* on G, it will make total N number of node expansions. But the total number of node expansions by IDA* will be as follows:

$$\frac{(2k)(2k+1)}{2} + K\sum_{i=0}^{K}(K+1-i)K_{C_i}$$

Now

$$\sum_{i=0}^{K}(K+1-i)K_{C_i} = (K+2)2^{K-1}$$

Thus the total number of node expansions by IDA* is

$$K(K+2)2^{K-1} + K(2K+1) = O(K^2 2^K) = O(N^2 2^N).$$

In the last iteration of IDA* there will be $O(2^N)$ node expansions. Note that the heuristic branching factor is greater than 1 in this case.

In Example 2, we have considered uniform arc cost. What happens if we allow variable arc cost in a directed acyclic graph? We consider that in the next example.

Example 3:

Consider the search graph G shown in Figure 4. We can generalize the graph for $N = K + 2$ nodes where n_0 is the start node and n_{K+1} is the goal node. The cost structure can be described as follows:

$$\begin{aligned} c(n_0, n_i) &= 2^{i-1}, & 1 \leq i < K; \\ c(n_1, n_{k+1}) &= 2^K - 1; & \\ c(n_i, n_{i-1}) &= 2^{i-2}, & 1 < i \leq K \\ c(n_i, n_j) &= 2^{j-1}, & 1 \leq j < i, 1 < i \leq K; \\ h(n_i) &= 0, & 0 \leq i \leq K+1. \end{aligned}$$

The total number of node expansions by A* and IDA* will be as follows:

A*: $O(N)$

IDA*: $O(2^{2N})$

Conjecture 1. If a limited-memory algorithm uses $O(L)$ memory (where L is the maximum number of nodes in any path generated by A*), then it will make $\Omega(N^2)$ node expansions in the worst case for trees.

Theorem 3. IDA* does not make more than $O(N^2)$ and $O(2^{2N})$ node expansions at the worst, in the case of trees and acyclic graphs, respectively.

proof: There can be N and 2^N thresholds for trees and acyclic graphs, respectively.

Thus IDA* has optimal asymptotic time complexity amongst all limited-memory best-first search algorithms which find optimal solutions under admissible heuristics for tree searches.

Remarks.

1. Although in most theoretical analysis the error of the heuristic function is assumed to be constant relative, it is hard to find any practical example where this holds. Constant relative error is as good as perfect heuristic if we consider unit arc cost. Because if the heuristic error is constant relative and we ignore g (or may be by multiplying h by a constant $\geq h^*(s)$) in f then A* can find an optimal solution path without expanding a single extraneous node. If we consider an example say 15-puzzle we will find when the manhattan distance estimate is 35 the actual cost could be 45, 47, 49 or 57 etc. (see [4]). This means the heuristic error in 15-puzzle is not constant relative. Therefore, if the heuristic branching factor value depends on the constant relative error assumption, we can infer that IDA* may not be asymptotically optimal with A* on 15-puzzle problem.

2. Quite often to gauge the time complexity of A* on a problem, the time complexity of IDA* in its last iteration is considered. This estimate would be fine if the problem space is a tree. However, if the problem space is a graph, then this estimate may be grossly erroneous.

3. To reduce the number of iterations of IDA*, often the threshold value for the next iteration is increased. On the average this may be helpful but any arbitrary increase of threshold would affect the asymptotic optimality of IDA*.

4. It can be observed that under monotone heuristic for tree search problems if we consider A* and IDA* under best and worst possible threshold distributions then the number of node expansions under the best and the worst situations by these two algorithms would be as follows:

Distribution of Thresholds	Total Number of node Expansions	IDA*	A*
Best	Best	$O(L)$	$O(L)$
Best	Worst	$O(N)$	$O(N)$
Worst	Best	$O(L^2)$	$O(L)$
Worst	Worst	$O(N^2)$	$O(N)$

5. Because of the very different natures of A* and any limited-memory algorithms, it is not possible for any limited-memory algorithm to follow the tie resolution strategy of A* even if the complete tie resolution information is available during the search. Moreover, in case of a tie, A* always favours a goal node over a nongoal node. This simple strategy would be quite hard to implement in any limited-memory algorithm.

4 Conclusion

IDA* and several other heuristic search algorithms are "limited-memory" algorithms, in the sense that they run in $O(L)$ memory, where L is the maximum number of nodes in any path generated by A*. Because of the limited memory, these algorithms must in most cases make more node expansions than A*. When the heuristic is monotone, A* runs in time $O(N)$, where N is the number of nodes n such that $f(n) \leq f^*(s)$. But no limited-memory algorithm can exploit this monotone property of the heuristic function.

In [4], it was stated that for tree searching, if certain conditions are satisfied, then IDA* achieves the same $O(N)$ time complexity as A*, and thus is asymptotically optimal among all best-first algorithms for tree searching which find optimal solutions using non-overestimating heuristic functions. The specific set of conditions used in [4] is that the heuristic function is underestimating and has constant relative error, and the heuristic branching factor is > 1. However, in the current paper, we have shown that these conditions are neither sufficient nor necessary for the result stated in [4] to be true—but if a different set of conditions is satisfied, then the result is true. If these conditions are not satisfied, then the worst-case time complexity of IDA* is $\Omega(N^2)$ for trees and $\Omega(2^{2N})$ for acyclic graphs.

Although IDA* has higher worst-case time complexity than A*, It is currently unknown whether there exists any limited-memory heuristic search algorithm which has better worst-case time complexity than IDA^*.

References

[1] Chakrabarti P., Ghosh S., Acharya A. and De Sarkar S., Heuristic Search in Restricted Memory, *Artificial Intelligence* **41**, 1989.

[2] Evett M., Hendler J., Mahanti A., and Nau D., PRA*: A Memory-Limited Heuristic Search Procedure for the Connection Machine, *Frontiers '90: Frontiers of Massively Parallel Computation*, College Park, MD, 1990 (to appear).

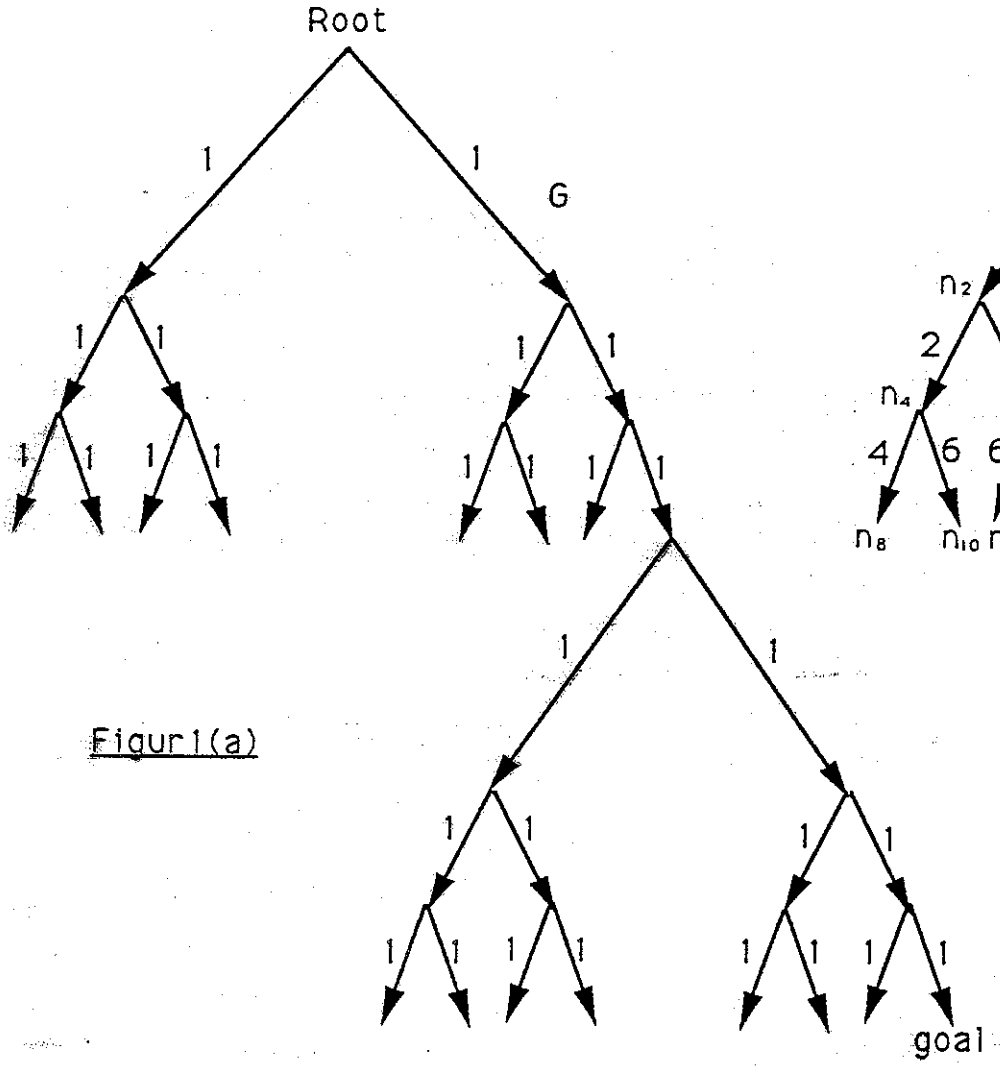
[3] Huang, S., *Design and Analysis of Some High-Performance Heuristic Search Algorithms.* Ph.D. Dissertation, University of Maryland, 1990.

[4] Korf, R. Depth First Iterative Deepening: An Optimal Admissible Tree Search, *Artificial Intelligence* **27**, 1985, pp. 97-109.

[5] Nilsson, N. J. *Principles of Artificial Intelligence*, Tioga Publications Co., Palo Alto, CA, 1980.

[6] Pearl J., Heuristics, *Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1984.

[7] Sen A. and Bagchi A., Fast Recursive Formulations for Best-First Search That Allow Controlled Use of Memory, *Proc. IJCAI-89*, 1989.



Figur1(a)

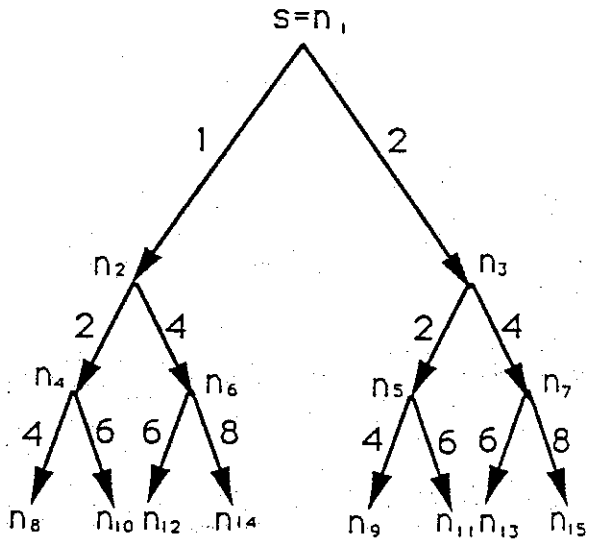


Figure2

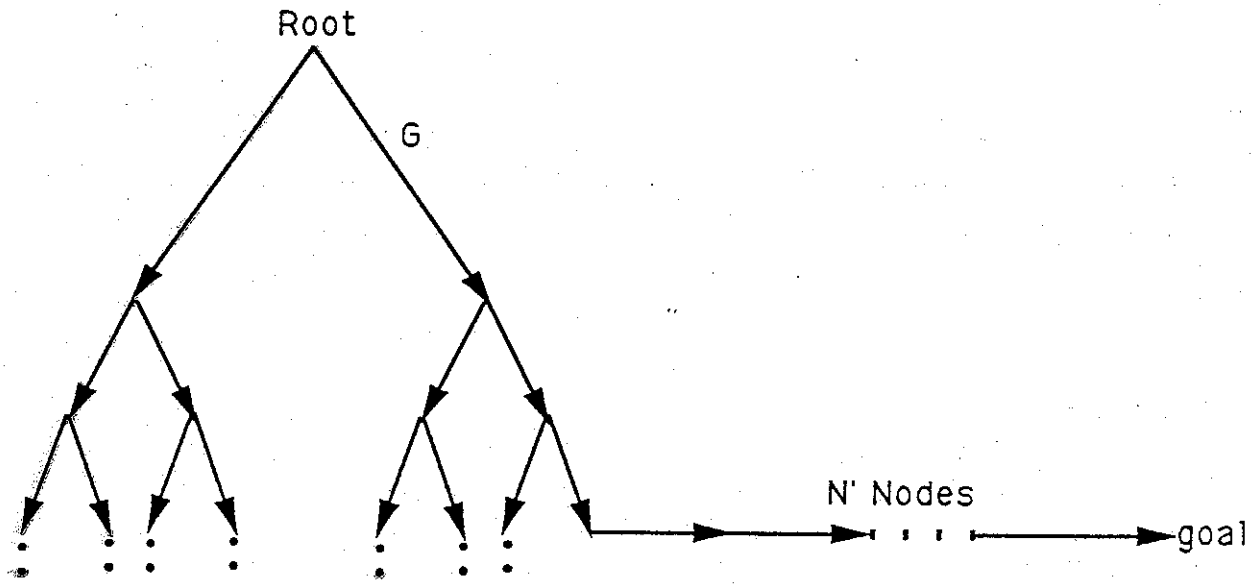


Figure1(b)

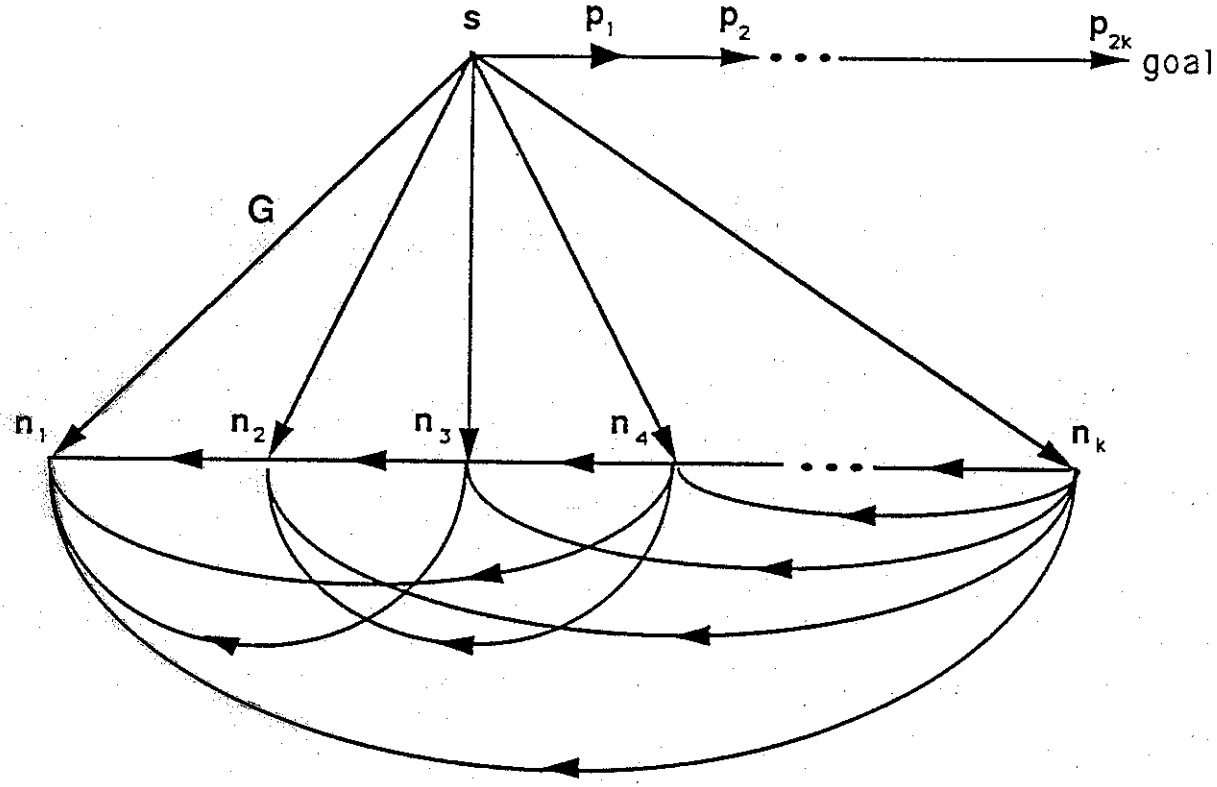


Figure3

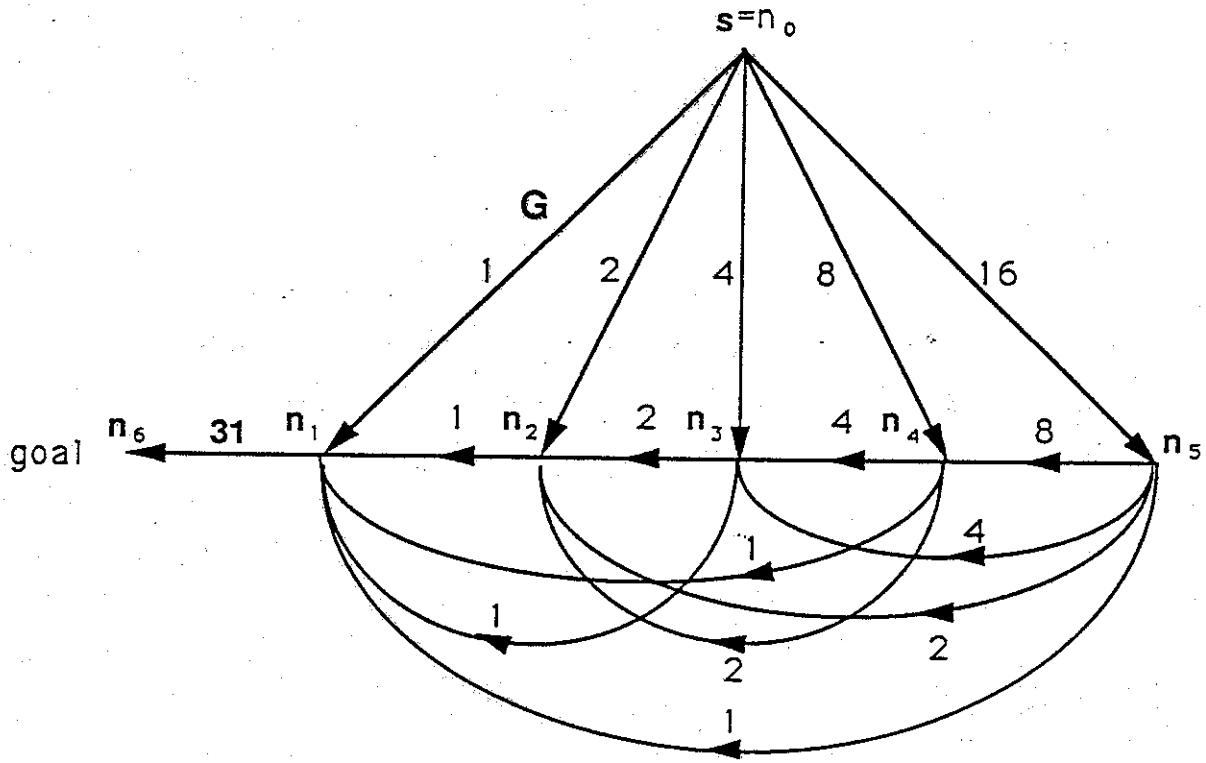


Figure4