On the Asymptotic Performance of IDA*

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

Address all correspondence to Dana S. Nau

Abstract

Since best-first search algorithms such as A* require large amounts of memory, they sometimes cannot run to completion, even on problem instances of moderate size. This problem has led to the development of limited-memory search algorithms, of which the best known is IDA* [9, 10]. This paper presents the following results about IDA* and related algorithms:

- The analysis of asymptotic optimality for IDA* in [10] is incorrect. There are trees satisfying the asymptotic optimality conditions given in [10] for which IDA* is not asymptotically optimal.
- To correct the above problem, we state and prove necessary and sufficient conditions for asymptotic optimality of IDA* on trees. On trees not satisfying our conditions, we show that no best-first limited-memory search algorithm can be asymptotically optimal.
- On graphs, IDA* can perform quite poorly. In particular, there are graphs on which IDA* does $\Omega(2^{2^N})$ node expansions where N is the number of nodes expanded by A*.

Keywords: Search, Asymptotic Optimality, A*, IDA*

This work was supported in part by NSF Grants IRI-8802419, NSFD CDR-88003012 and IRI-9306580 in the US, and the CMDS project (work order 019/7-148/CMDS-1039/90-91) in India.

*IIM, Calcutta, Diamond Harbour Road, PO. Box No. 16757, Calcutta 700 027, India.

†Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742. Email: subrata@cs.umd.edu.

‡Dept. of Computer Science, Institute for Systems Research, and Institute for Advanced Computer Studies, Univ. of Maryland, College Park, MD 20742. Email: nau@cs.umd.edu.

§IIM, Calcutta, Diamond Harbour Road, P. Box No. 16757, Calcutta 700 027, India.

¶Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742. Email: kanal@cs.umd.edu.

NOTATION

$G, G', G_1, G_2, \ldots$	Search graphs
$\mathcal{G}$	Sequence of graphs
T	Explicit search tree
s	Start node
$t, t_1, t_2, \ldots$	Goal nodes
$m, n, n_1, n_2 \ldots$	Other nodes
$c(m, n)$	Cost of the arc (m, n)
P	Path in G
$c(P)$	Cost of the path P
$c(P, m, n)$	Cost of the path P from m to n
$\mathrm{last}(P)$	Last node of P
$g^*(n)$	Cost of path of least cost from s to n
$h^*(n)$	Cost of path of least cost from n to a goal node
$f^*(n)$	$g^*(n) + h^*(n)$
$g(n)$	Cost of currently known path of least cost from s to n
$h(n)$	Underestimate of $h^*(n)$
$f(n)$	$g(n) + h(n)$
δ	A small positive real number
b	Node-branching factor
b_1	Constant > 1
OPEN, CLOSED	Lists
z, z'	IDA* thresholds
i, j	Integers
$W^G(z)$	Set of paths in G generable by IDA* with threshold z
$V^G(z)$	Set of nodes of G generable by IDA* with threshold z
$X^G(z)$	Set of nodes of G expandable by IDA* with threshold z
$x^G(z)$	Cardinality of $X^G(z)$
$i_1, i_2, \ldots$	Active iterations of IDA*
$j_{p1}, j_{p2}, \ldots$	Dummy iterations of IDA* following the active iteration i_p
u, u_0	Number of active iterations
$c_1, c_2, \ldots$	Number of dummy iterations
I, I'	Total number of iterations by IDA*
$X^G_{\mathrm{new}}(j)$	Set of new nodes expanded by IDA* during iteration j
$x^G_{\mathrm{new}}(j)$	Number of new nodes expanded by IDA* during iteration j
x^G_{tot}	Total number of node expansions done by IDA* on G
E^G	Surely expandable nodes of G
L	Maximum number of nodes on any path P s.t. $\mathrm{pathmax}(P) \leq h^*(s)$
S	Amount of Memory in number of nodes
$\mathcal{A}$, $\mathcal{A}_{bf}$	Search Algorithms

1 Introduction

The well known A* search algorithm [7, 17] is optimal in terms of number of node expansions (which is also a measure of its time complexity) in most cases [4]. However, since it requires exponential amount of memory to run, it runs out of memory even on problem instances of moderate size.

To overcome the storage problem, a variant of A* called IDA* (Iterative Deepening A*) was introduced by Korf [9, 10]. Basically, IDA* performs a depth-first search, backtracking whenever it finds a path whose cost exceeds a *threshold* value z. It repeats this search for larger and larger values of z, until it finds a solution. Since IDA* expands nodes more than once, the total number of node expansions for IDA* is more than for A*. But since IDA*'s memory requirement is only linear in the depth of the search, this enables IDA* to solve much larger problems than A* can solve in practice. For example, IDA* can solve randomly generated instances of the 15-puzzle [10].

This paper presents a detailed analysis of the properties of IDA*. Our results are as follows:[1]

1. **IDA* on Trees.** One of IDA*'s most important properties is that under certain conditions it is "asymptotically optimal in time and space over the class of best-first searches that find optimal solutions on a tree" [10, p. 236]; i.e., on these trees it expands $O(N)$ nodes, where N is the number of nodes eligible for expansion by A*.[2] In this paper, we describe and correct some difficulties with this claim. In particular, we show that:

   (a) IDA* is not asymptotically optimal in all of the cases where it was thought to be so. As shown in Section 4, there are trees satisfying all of asymptotic optimality conditions given in [10], such that IDA* will expand more than $O(N)$ nodes.[3]

   (b) The above trees appear to be common enough to affect IDA*'s average-case performance. Our experimental results, which are presented in Section 5, suggest that for certain kinds of Traveling Salesman Problems, IDA*'s average-case performance is not asymptotically optimal.

[1] Some of these results have also been summarized briefly in [12].

[2] A node n is eligible for expansion by A* under a given heuristic h, if there exists a tie-breaking rule such that if we run A* using h and the tie-breaking rule, A* will expand n.

[3] Previous papers have described trees on which IDA* expands more than $O(N)$ nodes [13, 18], but the trees described in these papers do not satisfy Korf's [10] requirements of finite precision and non-exponential node costs.

(c) In Section 6, we present correct sets of necessary conditions and sufficient conditions for IDA* to be asymptotically optimal on trees.

2. **IDA* on Graphs.** When the heuristic is monotone, A* will do $O(N)$ node expansions on graphs, because it will never expand a node more than once. However, IDA* cannot prevent the reexpansion of a node through a costlier path. Thus, although there are specific graph-search problems (such as the 15-puzzle) in which IDA* does well, in general the performance of IDA* for graphs is bound to become worse than that of A*.

   In particular, there are many graph and tree search problems where the node-branching factor grows with the problem size (examples include the traveling salesman problem, flow-shop scheduling, etc.). In Section 7, we show that on such graphs, IDA*'s worst-case complexity with a monotone heuristic is $\Omega(2^{2N})$, even if the graph is restricted to be acyclic.[4]

3. **Other Algorithms on Trees.** IDA* is not the only tree search algorithm that operates in limited memory. Following IDA*, several other algorithms have been developed that run in limited memory, such as MREC [21], MA* [3], RA* [5], SMA* [20], RBFS [8], and ITS [6]. In Section 8, we show that for trees that do not satisfy the asymptotic optimality conditions described in Section 6, no limited-memory best-first search algorithm is asymptotically optimal. More specifically, in the absence of any conditions (except that the heuristic is admissible and the maximum node-branching factor is constant) there does not exist any best-first admissible tree search algorithm which when running with $S = N/\psi(N)$ (where $\psi(N) \neq O(1)$) memory can have $O(N)$ worst-case time complexity like A*.

The paper is organized as follows. Section 2 contains background material, including brief descriptions of A* and IDA*. Section 3 presents some notation needed in our analysis of IDA*. Sections 4 through 8 present the results described above, and Section 9 contains concluding remarks.

[4]If cycles are allowed, then IDA*'s total number of node expansions can only increase.

2 Background

2.1 Basic Concepts and Terminology

A state space consists of a directed graph G that has one *start node* s and a non-empty set of goal nodes. Each arc (m, n) in G has a cost $c(m, n) \geq \delta > 0$. The number of children of a node is called its *node-branching factor*. If $P = (n_0, n_1, \ldots, n_k)$ is a sequence of nodes such that (n_{i-1}, n_i) is an arc of G for $i = 1, \ldots, k$, then we say that P is a *path from* n_0 *to* n_k. In this case, last(P) is the node n_k. P's *cost*, denoted by cost(P), is the sum of P's arc costs. A *solution path* in G is a path from s to a goal node. For each node n of G, $h^*(n)$ is the minimum cost of any path from n to a goal node. Thus $h^*(s)$ is the cost of the least costly solution path.

The objective of many heuristic search algorithms is to find a least costly solution path in G. To find such a path, many algorithms use a node evaluation function

$$f(n) = g(n) + h(n),$$

where $g(n)$ is the cost of the best path currently known from s to n, and h, the *heuristic function*, is an estimate of h^*. It is assumed that $h(n) \geq 0$ for every node n, with $h(n) = 0$ if n is a goal node.

Let P be any path from the start node s. Then the function pathmax(P) [1] is defined as follows:

$$\text{pathmax}(P) = \max_{n \in P}(c(P, s, n) + h(n)),$$

where $c(P, s, n)$ is the cost of the subpath of P that goes from s to n.

The heuristic function h is *admissible* if $\forall n \in G$, $h(n) \leq h^*(n)$. It is *monotone* if $\forall p \in G$, $h(p) \leq c(p, q) + h(q)$, where q is a child of p. Note that monotonicity implies admissibility. Moreover, if h is admissible and P is a minimum-cost solution path then pathmax$(P) = h^*(s) = cost(P)$. Algorithms which find an optimal (least-costly) solution when using an admissible heuristic are called admissible algorithms.

We use L to denote the maximum number of nodes on any path P for which pathmax$(P) \leq$

$h^*(s)$. Since the number of nodes on any path is one more than the length of the path [5],

$$L = 1 + \max\{length(P) : \text{pathmax}(P) \leq h^*(s)\}.$$

The set E^G of the *surely expandable* nodes of G is the set of all nodes n for which there exists a path P from s to n such that pathmax$(P) < h^*(s)$. A node is *surely generable* if it is in E^G or is a child of a node in E^G. It is well known [4] that if $\mathcal{A}$ is any admissible best-first tree-search algorithm and G is a graph, then $\mathcal{A}$ must generate every surely generable node of G.

Given a state space and a monotone heuristic function, the *heuristic branching factor* is the average ratio of the number of nodes of each f-value to the the number of nodes at the next smaller f-value, averaged over all f-values $\leq h^*(s)$ [10].

2.2 Algorithm A*

The best known admissible algorithm is A* [7, 17]. A* works in a best-first manner. It maintains two lists: OPEN, which contains nodes that are to be expanded, and CLOSED, which contains nodes that have already been expanded. At each iteration, A* selects a node n from OPEN with minimum f-value, generates all of its children, and puts these children into OPEN after setting their g, h, and f-values. If a child p of n is already present in OPEN and $g(p) > g(n) + c(n, p)$, then $g(p)$ is reset to $g(n) + c(n, p)$. If a child p of n is already present in CLOSED and a better path to it is now found, then $g(p)$ is reset to the newly found smaller path cost and p is brought back to OPEN from CLOSED. The process of node selection and expansion continues until a goal node is selected for expansion.

Besides its admissibility, A* has the property that if the heuristic function is monotone, then A* never expands any node more than once. Admissible heuristic functions designed by the commonly used method of "relaxation" [19] automatically satisfy the monotonicity condition.

One major problem with A* is the amount of memory required to store nodes in OPEN and CLOSED. As shown in [4], every admissible search algorithm must expand all *surely expandable*

[5] The length of a path P is the number of arcs in P.

nodes before finding a solution. The number of such nodes often grows exponentially with some measure of the problem size—and since A* keeps track of all of these nodes, it needs an exponential amount of memory in which to store these nodes.

2.3 Performance Measures of Search Algorithms

Generally the following criteria are used to measure the efficiency of an algorithm: (a) solution quality, (b) time complexity, and (c) storage requirement. Any best-first search algorithm running with an admissible heuristic always returns an optimal solution. Therefore, in measuring performance of admissible algorithms, storage and time are the two dominant factors. Since the total running time is closely related to the total number of node expansions, the number of node expansions is often used as a measure of time complexity. Similarly, the maximum number of nodes stored at any one time is often used as a measure of the storage requirement.

Since the admissible heuristics designed by the commonly used method of relaxation are also monotone, and A* expands each node at most once if the heuristic is monotone, one question that naturally arises is whether A* is optimal in terms of the number of node expansions. Dechter and Pearl [4] and Mero [16] have shown that out of the set of all admissible best-first search algorithms guided by the same admissible heuristic function h, no algorithm can be (in their terminology) *0-optimal*; i.e., no algorithm can guarantee fewer node expansions than all other algorithms for all problem instances. However, A* is *1-optimal*; i.e., for every problem instance p there exists a tie-breaking rule R for selecting nodes from OPEN, such that if we run A* on p using R, then A* will do fewer node expansions on p than all other algorithms. Since there does not exist any *0-optimal* algorithm, we will use *optimal* to mean 1-optimal.

An algorithm $\mathcal{A}$ *asymptotically dominates* an algorithm $\mathcal{B}$ over a class of problems $\mathcal{P}$ if for all problems in $\mathcal{P}$, $\mathcal{A}$ does $O(N_B)$ node expansions, where N_B is the number of node expansions done by B. An algorithm is *asymptotically* optimal if it asymptotically dominates an optimal algorithm. Thus since A* is an optimal algorithm, an admissible state-space search algorithm algorithm is asymptotically optimal if and only if it does $O(N)$ node expansions, where N is the number of nodes eligible for expansion by A*.

```
procedure IDA*:
    Let z, z' be global variables.
    Set z := h(s), where s is the start node. Set z' := ∞.
    Do the following steps repeatedly:
        set P := the path containing only s;
        call Depth-First(P);
        set z := z'.

procedure Depth-First(P):
    Set f := cost(P) + h(last(P)).
    If f > z, then set z' := min(z', f).
    Otherwise, if last(P) is a goal node, then exit from IDA*, returning P.
    Otherwise, do the following steps for every child n of last(P):
        set P' := the path formed by appending n to P;
        call Depth-First(P').
```

Figure 1: Pseudocode for IDA*.

Researchers have recently realized that in addition to considering a search algorithm's time complexity, it is just as important to consider its storage complexity. For example, A*'s major bottleneck in practical use is its storage requirement. For most problems of non-trivial size, A* runs out of memory before any significant amount of execution time. This limitation also prevails in the case of other known variants of A* (B [15], GRAPHSEARCH [17], C [1], PropC [2], MarkA [2], B' [16], D [14], etc.), which are not tailored to run with limited memory. Therefore, design of search algorithms that run with limited memory has significant practical importance.

2.4 Algorithm IDA*

To overcome the storage problem, a variant of A* called IDA* (Iterative Deepening A*) was introduced by Korf [9, 10]. Basically, IDA* performs a depth-first search, backtracking whenever it finds a path whose cost exceeds a *threshold* value z. It repeats this search for larger and larger values of z, until it finds a solution. Figure 1 shows a pseudocode version of IDA*.

Unlike A*, IDA* is a tree search algorithm—in other words, it does not keep track of alternate paths to each node. Thus if the search space is not a tree, IDA* can do many re-expansions of each node as it finds new paths to that node. But since it only keeps track of the nodes on the path it is currently exploring, IDA* requires only $O(L)$ memory. In other words, IDA*'s memory requirement grows only linearly with the depth of the search. This enables IDA* to solve much larger problems than A* can solve, such as the 15-puzzle [10]. Thus IDA* has drawn significant attention from the AI research community.

3 Definitions

As we will see later, the quantities defined below correspond closely to the behavior of A* and IDA*.

If $P = (n_0, \ldots, n_k)$ is any path, then all-but-last(P) is the path $(n_0, \ldots, n_{k-1})$ and *last*(P) is the node n_k. For each $z \geq 0$, we define

$$
\begin{aligned}
W^G(z) &= \left\{\begin{array}{l}\text{all paths } P \text{ from } s \text{ such that } \mathrm{cost}(P) + h(\mathrm{last}(P)) > z,\\ \text{pathmax(all-but-last}(P)) \leq z, \text{ and all-but-last}(P) \text{ con-}\\ \text{tains no goal nodes}\end{array}\right\};\\
V^G(z) &= \{\text{all nodes of all paths in } W^G(z)\};\\
X^G(z) &= \{\text{all non-tip nodes of all paths in } W^G(z)\};\\
x^G(z) &= |X^G(z)|;\\
N(G) &= x^G(h^*(s)).
\end{aligned}
$$

In the terms defined above, we will usually omit G if its identity is clear.

From these definitions, it follows immediately that if we run A* on a tree, then every path generated by A* will be a subpath of a path in $W(h^*(s))$, every node generated by A* will be in $V(h^*(s))$, and every node expanded by A* will be in $X(h^*(s))$. Furthermore, in the worst case (which occurs if A* expands all nodes on OPEN that have f-values $\leq h^*(s)$ before it selects a goal node), A* will generate all paths in $W(h^*(s))$ and all nodes in $V(h^*(s))$, and will expand all nodes in $X(h^*(s))$, for a total of $x(h^*(s))$ node expansions. Thus $X(h^*(s))$ is the set of all nodes eligible for expansion by A*, and $N = x(h^*(s))$ is the number of nodes eligible for expansion by A*. The quantity N is known as the number of *possibly expandable* nodes.

For $j = 1, \ldots,$ we inductively define

$$
\begin{aligned}
z^G(j) &= \begin{cases} h(s), \text{ where } s \text{ is } G\text{'s start node}, & \text{if } j = 1, \\ \min\{\text{cost}(P) + h(\text{last}(P)) : P \text{ is a path in } W^G(z^G(j-1))\}, & \text{otherwise;} \end{cases} \\
X^G_{\text{new}}(j) &= \begin{cases} X^G(z^G(1)), & \text{if } j = 1, \\ X^G(z^G(j)) - X^G(z^G(j-1)), & \text{otherwise;} \end{cases} \\
x^G_{\text{new}}(j) &= |X^G_{\text{new}}(j)|; \\
I^G &= \min\{j : \text{there is a solution path } P \text{ such that pathmax}(P) \le z^G(j)\}; \\
x^G_{\text{tot}} &= \sum_{j=1}^{I^G} x^G(z^G(j)) \;=\; \sum_{j=1}^{I} |X^G(z^G(j))|.
\end{aligned}
$$

As before, we will usually omit the superscript G if the identity of G is clear.

Suppose we run IDA* on a tree. The definitions of $V(z(j))$ and $z(j)$ presented above correspond precisely to the way that IDA* generates nodes and sets thresholds. Thus it follows that IDA* does I iterations, and that for $j = 1, 2, \ldots, I-1$,

$$
\begin{aligned}
\text{the threshold used during IDA*'s } j\text{'th iteration} &= z(j); & (1)\\
\{\text{nodes generated during IDA*'s } j\text{'th iteration}\} &= V(z(j)); & (2)\\
\{\text{nodes expanded during IDA*'s } j\text{'th iteration}\} &= X(z(j)); & (3)\\
\{\text{new nodes expanded during IDA*'s } j\text{'th iteration}\} &= X_{\text{new}}(j). & (4)
\end{aligned}
$$

Since IDA* generates each node at most once during each iteration, it follows that

$$
\begin{aligned}
\text{the number of nodes expanded during IDA*'s } j\text{'th iteration} &= x(z(j)); & (5)\\
\text{the number of new nodes expanded during IDA*'s } j\text{'th iteration} &= x_{\text{new}}(j). & (6)
\end{aligned}
$$

During the final iteration (i.e., $j = I$), IDA* can find a goal node and terminate before every node of $V(z(I))$ has been generated. Thus

$$
\text{the threshold used during IDA*'s } I\text{'th iteration} = z(I); \qquad (7)
$$

$$
\begin{aligned}
\{\text{nodes generated during IDA*'s } I\text{'th iteration}\} &\subseteq V((z(I)); && (8)\\
\{\text{nodes expanded during IDA*'s } I\text{'th iteration}\} &\subseteq X(z(I)); && (9)\\
\{\text{new nodes expanded during IDA*'s } I\text{'th iteration}\} &\subseteq X_{\text{new}}(I); && (10)\\
\text{the number of nodes expanded during IDA*'s } I\text{'th iteration} &\leq x(z(I)); && (11)\\
\text{the number of new nodes expanded during IDA*'s } I\text{'th iteration} &\leq x_{\text{new}}(I); && (12)
\end{aligned}
$$

with equality in the worst case. Furthermore, from the correctness of IDA*, it follows that

$$z(I) = h^*(s). \tag{13}$$

The *heuristic branching factor* defined in the previous section is the average, over $j = 2, \ldots, I$, of the quantity

$$\frac{x_{\text{new}}(j)}{x_{\text{new}}(j-1)}.$$

Finally, it is straightforward to see that the set of surely generable nodes is precisely $V(z(I-1))$, i.e., the set of nodes generated by IDA* in its second-to-last iteration and $X(z(I-1))$ is the set of surely expandable nodes.

4 IDA* on Trees: Examination of Optimality Conditions

In his analysis of IDA*, Korf [10] introduces the following conditions that might or might not be satisfied by various search problems:

Condition 1: h has a heuristic branching factor > 1;

Condition 2: the problem space grows exponentially with depth;

Condition 3: representation of costs (f-values) is with finite precision;

Condition 4: cost (f) values do not grow exponentially with depth.

For the case where G is a tree and h is an admissible heuristic function, it has been shown that IDA* will do no more than $(N + N^2)/2$ node expansions in the worst case [18]. In addition, Korf

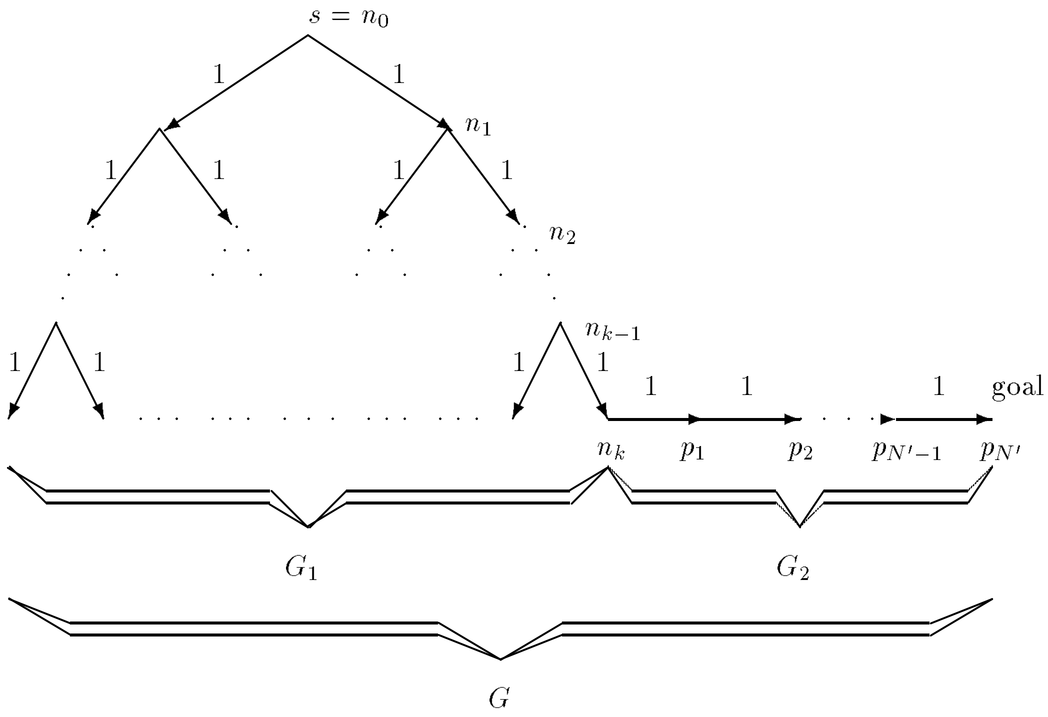


Figure 2: IDA*: $\Omega(N^2)$

[10] states that if Condition 1 holds, then IDA* is *asymptotically optimal*, i.e., it will do only $O(N)$ node expansions. In order to argue that Condition 1 is a reasonable one for most best-first tree searches, he states that Conditions 2, 3, and 4 are sufficient to guarantee Condition 1, and that Conditions 2, 3, and 4 are satisfied in most realistic tree search problems. Thus, we call Condition 1 the *primary condition*, and Conditions 2, 3, and 4 the *secondary conditions*.

In this section we show the following:

- The above conditions are not sufficient to ensure the $O(N)$ time complexity of IDA*. Even when all of them are satisfied, IDA* fails to achieve $O(N)$ time complexity in the worst case.
- These conditions are not necessary either; i.e. IDA* can have $O(N)$ complexity without satisfying them.
- Conditions 2, 3, and 4 are not sufficient to guarantee Condition 1.

Example 1 Consider the search tree G in Figure 2 where there are $N = 2N' - 1$ non-goal nodes which are all possibly expandable, and one goal node. G consists of two subtrees G_1 and G_2. While

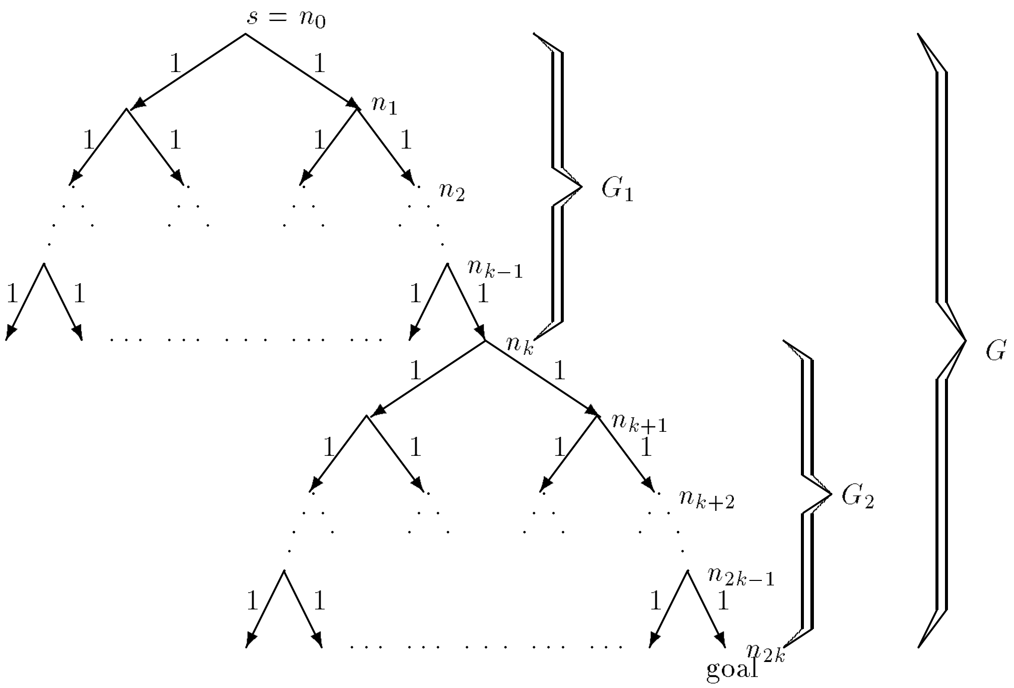


Figure 3: IDA*: $\Omega(N \lg N)$

G_1 is a full binary tree, G_2 contains N' nodes in the form of a degenerate tree. Every leaf node in G is a non-terminal node except the rightmost one ($p_{N'}$), which is a goal node. Cost of the solution path $= N' + \lfloor \lg N' \rfloor$, maximum node branching factor is 2, and heuristic branching factor > 1. Each arc has cost 1, and heuristic value is zero at every node. Thus the heuristic is monotone. When IDA* expands nodes from G_1, it expands an exponentially increasing number of new nodes threshold by threshold. But when it begins expanding nodes from G_2, it expands only one new node at each iteration and for each new node p_i, $1 \leq i \leq N'$, IDA* reexpands all nodes of G_1. Since there are N' nodes in G_2 and G_1 also contains N' nodes, the total number of node expansions by IDA* is clearly $\Omega(N'^2) = \Omega(N^2)$. This result also holds for several other types of trees [13].

The example above satisfies the primary condition, and all secondary conditions except Condition 2: the search space is not exponential in the solution depth. In the next two examples, we show that the $O(N)$ worst-case time complexity for IDA* does not hold even when all the conditions are satisfied.

Example 2 In the search tree G given in Figure 3, each non-leaf node has a branching factor $b = 2$, and each arc has unit cost. G consists of two subtrees (called G_1 and G_2) where each one is a full binary tree of height k. G_2 is rooted at the right most node of G_1. Every leaf node, except the one labeled as goal, is a non-terminal leaf node. For each node n in G, we assume $h(n) = 0$. Then h is monotone. The heuristic branching factor is

$$\frac{2k + \frac{1}{2^{k-1}} + 2(k-1)}{(2k)} = 2 + \frac{1}{k2^k} - \frac{1}{k} \approx 2.$$

Note that the goal node is at a depth of $2k = O(\lg N)$, where N is the total number of possibly expandable nodes in G. Therefore the search space is exponential. The maximum cost value is $2k$ which grows only linearly with depth. The precision constraint is vacuously satisfied because the cost values are not fractions. Thus, the primary condition and all of the secondary conditions are satisfied. Now we calculate the total number of node expansions by IDA* on the tree G.

Clearly G_1 and G_2 each contain $N' = \lceil N/2 \rceil$ nodes. The cost of the solution path is $2k = 2[\lg(N'+1) - 1]$. Let

$$N_0 = b^k + 2b^{k-1} + 3b^{k-2} + \ldots + kb.$$

Then the total number of node expansions by IDA* in the worst-case is

$$x_{tot} = N_0 + kN' + N_0 \geq kN' + N' = k(N'+1) = \Omega(N \lg N).$$

Example 3 In the example tree G shown in Figure 4 a more interesting situation is illustrated. The root node n_0 has only one child n_1, which roots a complete b-ary tree ($b = 2$ in this case). Every leaf node of G is a goal node. However, there is only one minimum-cost solution path of cost $2k - 1$, which in Figure 4 is the rightmost path in G. Every other solution path has cost $2k$. Each arc has cost 1, except for the ones which are incident on the goal nodes, in that case the cost is k. Heuristic values are shown beside the nodes. Since every non-goal node up to and including level k (assuming that the root node is at level 0) has f-value less than $h^*(s)(= 2k-1)$, IDA* must expand all of them before finding a goal node and terminating. IDA* will expand one new node in iteration 1. Thereafter, in iteration i, $2 \leq i \leq k$, IDA* will expand 2^{k-i+1} new nodes. It will make a total of $k+1$ iterations before termination corresponding to thresholds $k-1, k, \ldots, 2k-1$. The

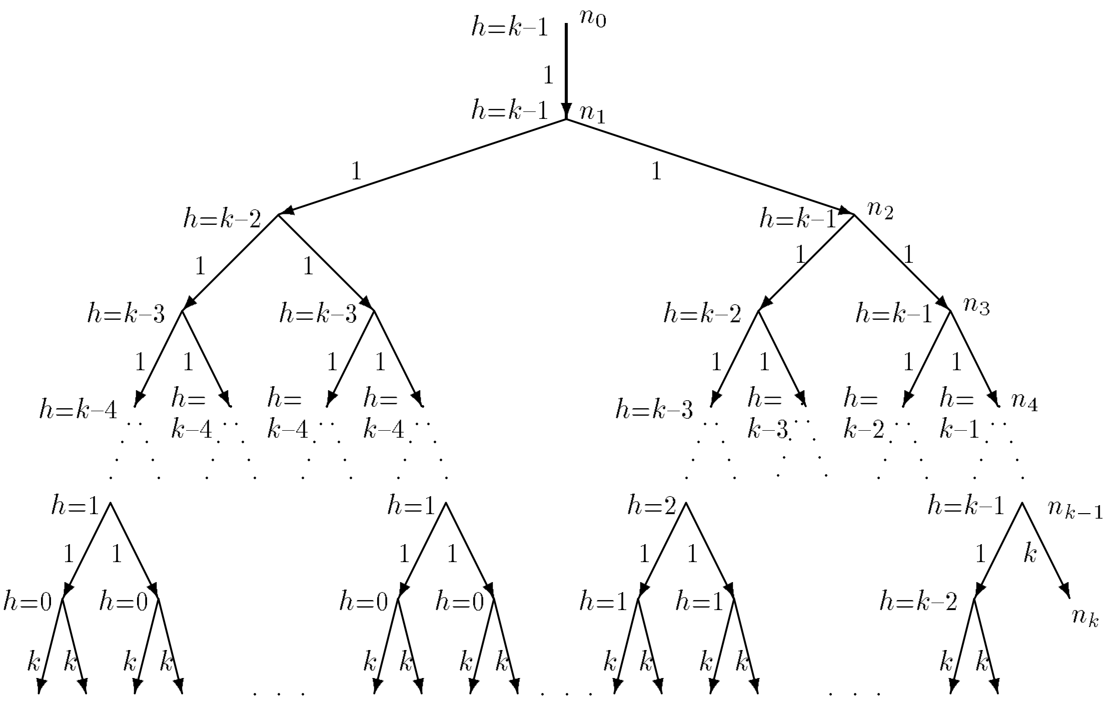


Figure 4: IDA*: $\Omega(N \lg N)$

heuristic branching factor is

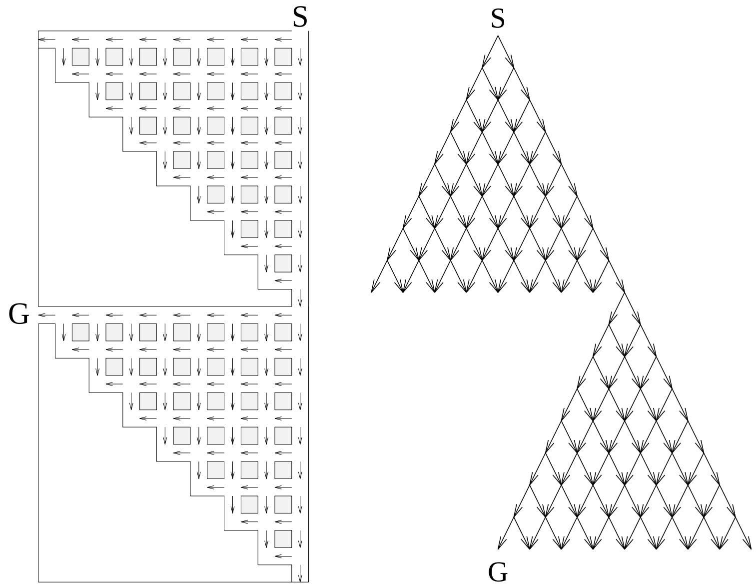
$$\frac{\frac{2^{k-1}}{1} + (k-1) * \frac{1}{2}}{k} > 1,$$

and thus the primary condition is satisfied. It can be easily seen that the secondary conditions are also satisfied. The total number of node expansions by IDA* is

$$\begin{aligned} x_{\text{tot}} &= (k+1) * 1 + \sum_{i=2}^{k} (k-i+2) * 2^{k-i+1} \\ &= (k+1) + 2^k(k-1). \end{aligned}$$

But $k = \lg(N+1)$, where N is the number of possibly expandable nodes in G. Therefore, $x_{\text{tot}} = \Omega(N \lg N)$.

Example 4 In the examples above, we have shown that the conditions stated in [10] for the asymptotic optimality of IDA* are not sufficient. In this example, we show that these conditions are not necessary either. Consider a search tree which is identical to the tree in Figure 2, except



(a) the maze (b) the search graph

Figure 5: An example maze and its search graph.

for two changes:

- $h(s) = k$;
- in the degenerate tree G_2, there is only a constant number, say c, of nodes from n_k to the goal node.

G contains a total of $N = N' + c - 1$ non-goal nodes, and one goal node. All the nodes of G_1 will be expanded by IDA* in the first iteration. Thereafter, in each iteration only one new node will be expanded. The heuristic branching factor is :

$$\frac{(c-1) * 1 + \frac{1}{N'}}{c} < 1$$

Since IDA* does only a constant number of iterations in this example, it will clearly do only $O(N)$ node expansions. Note that the primary condition stated previously is not satisfied in this case. This example also shows that the secondary conditions do not imply the primary condition.

Example 5 Above, we have constructed trees on which the previous analysis of IDA* fails. Below is an example of a search problem that gives rise to such a tree. Consider the maze shown in Figure 5(a). The objective is to find a least-cost path through the maze from S to G, obeying the one-way markings shown in the maze. To represent the maze as a search graph, we use nodes to represent junction points and dead ends, and use arcs to represent the hallways between them. Thus, the search graph for this maze is the graph shown in Figure 5(b). To search this graph, IDA* will unfold [6] the space into the exact same tree shown in Figure 3.

5 IDA* on Trees: Experimental Studies

In Section 4, we showed that there are trees satisfying the asymptotic optimality conditions stated for IDA* in [10], for which IDA* is not actually asymptotically optimal. In this section, we examine the practical impact of the existence of such trees, by studying IDA*'s performance on randomly generated instances of the Traveling Salesman Problem (TSP).

The TSP is a combinatorial optimization problem and is defined as follows: given a set of cities with nonnegative cost between each pair of cities, find the cheapest tour. A tour is a path that starting at some initial city visits every city once and only once, and returns to the initial city. Our results suggest that on randomly generated instances of the TSP, the number of node generations by IDA* will grow exponentially faster than the number of node generations by A* if the cost of going from one city to another is allowed to grow in proportion to the square of the number of cities (which satisfies the asymptotic optimality conditions in [10]).

To represent the search space and the lower bound heuristic for the Traveling Salesman Problem, we chose the well known method of Little et al. [11]. The search space in this formulation is a binary tree.

We generated two sets of data, which we will call TSP Set 1 and TSP Set 2, and ran both IDA* and A* on each set. For both sets we selected the number of cities equal to 5, 10, 15, 20, 25, 30, and 35. For each value of the number of cities, one hundred cost matrices were generated. For TSP Set 1 the cost values $c(i,j)$ were taken at random from the interval [0,100] using a uniform distribution except for $i = j$, in which case $c(i,j)$ was set to ∞. For TSP Set 2 the cost values $c(i,j)$ were taken at random from the interval [0,(10 $\times$ number of cities^2)] using a uniform distribution except for $i = j$, in which case $c(i,j)$ was set to ∞. In general the cost matrices of TSP Set 1 and TSP

[6] What we mean by "unfold" should be intuitively clear. We give the precise definition in Section 7.

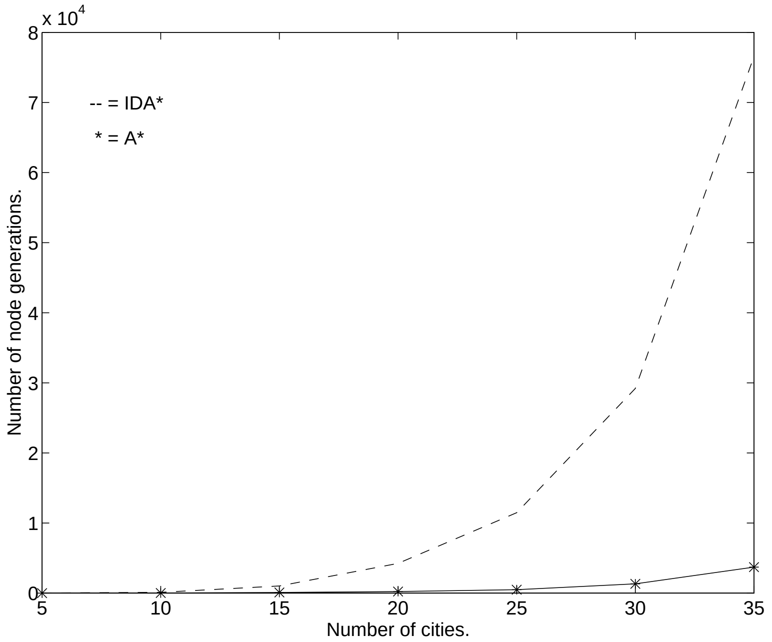


Figure 6: Node generations versus number of cities for IDA* and A* on TSP Set 1.

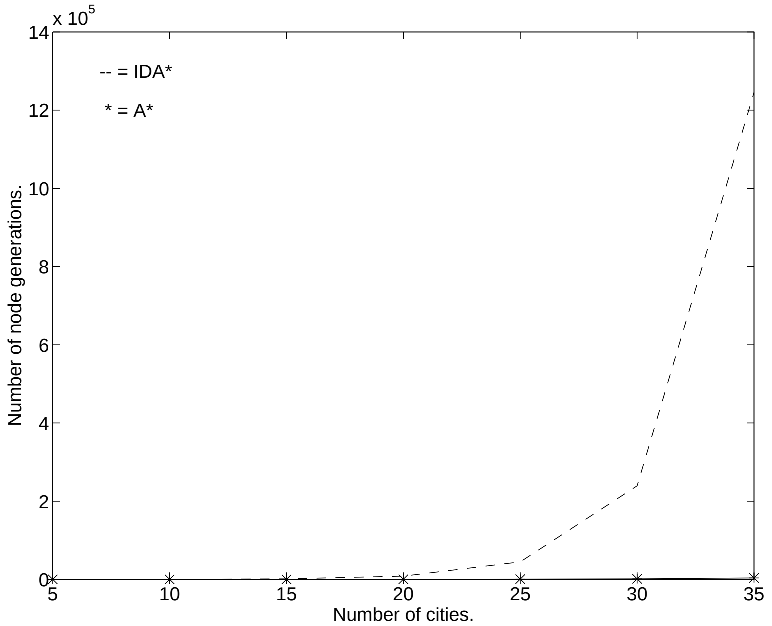


Figure 7: Node generations versus number of cities for IDA* and A* on TSP Set 2.

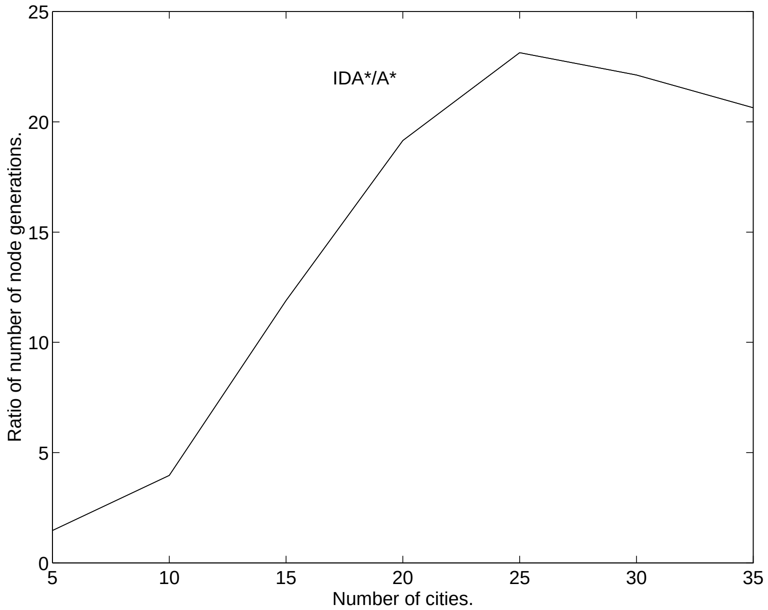


Figure 8: Ratio of IDA* node generations to A* node generations, versus number of cities on TSP Set 1.

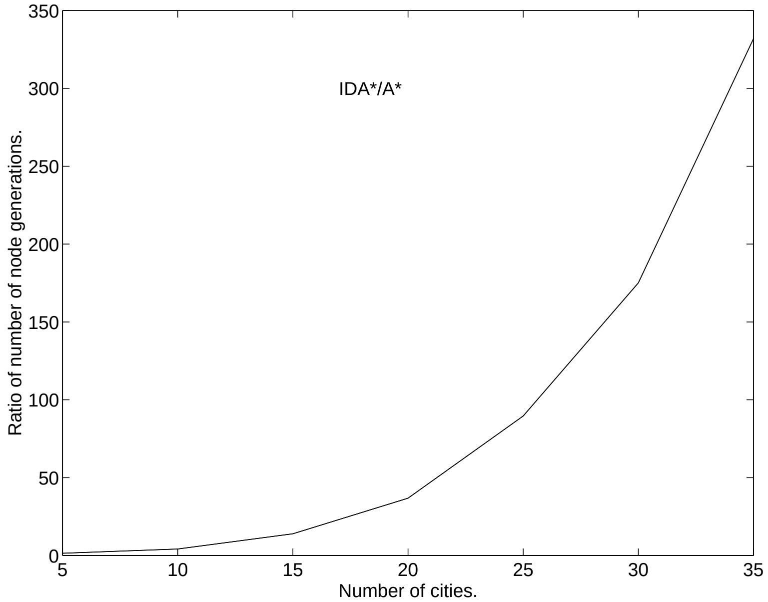


Figure 9: Ratio of IDA* node generations to A* node generations, versus number of cities on TSP Set 2.

Set 2 were not symmetric and did not satisfy the triangle inequality. It is easy to verify that the trees generated by the above technique satisfy all the conditions stated previously for asymptotic optimality of IDA*. The only difference between TSP Set 1 and TSP Set 2 is that the costs between the cities in TSP Set 1 are drawn from a fixed range whereas in TSP Set 2 the range grows in proportion to the cost matrix which is of the size of square of the number of cities.

The results of our experiments are summarized in Figures 6 through 9, which graph the performance of IDA* and A* for TSP Set 1 and TSP Set 2. Each data point in Figures 6 through 9 is the average over the one hundred problem instances.

For TSP Set 1, from Figure 8, the ratio of node generations by IDA* to node generations by A* first goes up and then goes down. If A*'s asymptotic performance were strictly better than IDA*'s, we would have expected the ratio to keep going up. Therefore, Figure 8 suggests that both algorithms have the same asymptotic performance on this problem on TSP Set 1, i.e. IDA* is asymptotically optimal. However, for TSP Set 2, the ratio of node generations by IDA* to node generations by A* continues to grow with the number of cities. Although it is difficult to make a conclusive statement about asymptotic complexity on the basis of a finite number of tests, our results clearly suggest that IDA* is not asymptotically optimal on TSP Set 2.

6 IDA* on Trees: New Optimality Conditions

In this section, we derive two new conditions, one of which is necessary for asymptotic optimality of IDA*, and the other of which is sufficient to guarantee asymptotic optimality of IDA*.

Korf stated that IDA* is asymptotically optimal if the heuristic branching factor exceeds 1. As we have seen, this condition is not sufficient to guarantee asymptotic optimality—but we can guarantee asymptotic optimality if we make the condition stricter.

For each i, consider the ratio of the number of new nodes at iteration $i+1$ to the number of new nodes at iteration i. Even though the *average* ratio may exceed 1 (in which case the heuristic branching factor exceeds 1), there may be a number of "dummy iterations" in which the ratio is not "large enough." If there are too many of these dummy iterations in the wrong places—then IDA* will perform badly.

Below, we formalize the idea of a dummy iteration, and what it means for there to be "too many" of them in the "wrong places." This enables us to state two new conditions, one of which is necessary for asymptotic optimality of IDA*, and the other of which is sufficient to guarantee asymptotic optimality of IDA*.

Let $b_1 > 1$ be a fixed constant. Then IDA*'s *active iterations* on G are the iterations $i_1^G, i_2^G, \ldots, i_{u^G}^G$ defined inductively as follows:

$i_1^G = 1$.

For $p = 2, \ldots, u$, i_p^G is the smallest integer such that $x_{\text{new}}(i_p^G)/x_{\text{new}}(i_{p-1}^G) \geq b_1$.

As usual, we omit the superscript G when the identity of G is obvious.

Intuitively the active iterations are the iterations in which the number of new nodes expanded by IDA* grows exponentially. We call the remaining iterations *dummy iterations*. For each i_p, let $j_{p1}, j_{p2}, \ldots, j_{pc_p}$ be the dummy iterations immediately following i_p.

Dummy iterations can occur anywhere after the first active iteration i_1. For $q = 1, \ldots, u$, let c_q be the number of dummy iterations that occur between iterations i_q and i_{q+1}. Note that $c_q \geq 0$, and $c_1 + c_2 + \ldots + c_u = I - u$. We define $M(G) = \max_q c_q$, i.e., $M(G)$ is the maximum number of adjacent dummy iterations.

Using the concept of dummy iterations, Sections 6.1 and 6.2 provide sufficient and necessary conditions, respectively, for IDA* to be asymptotically optimal on trees. In addition, both sections give examples in which these conditions are used to tell whether IDA* is asymptotically optimal on the trees presented in Section 4.

6.1 Sufficient Conditions for Optimality

Theorem 1 gives a sufficient condition for IDA* to be asymptotically optimal.

Theorem 1 Let $\mathcal{G} = (G_1, G_2, \ldots)$ be any sequence of trees such that $M = O(1)$, i.e., the maximum number of adjacent dummy iterations is bounded by a constant. Then on $\mathcal{G}$, the number of node expansions by IDA* is $x_{\text{tot}} = O(N)$, where $N(G_i)$ is the number of possibly expandable nodes in G_i.

Proof.

Let $\mathcal{G}$ be as defined above, $G \in \mathcal{G}$, and

$$Y = ux_{\text{new}}(i_1) + (u-1)x_{\text{new}}(i_2) + \ldots + x_{\text{new}}(i_u).$$

By definition, $x_{\text{new}}(i_{q+1})/x_{\text{new}}(i_q) \geq b_1$, $1 \leq q \leq u-1$. Thus $x_{\text{new}}(i_{u-p})/x_{\text{new}}(i_u) \leq \epsilon^p$, $1 \leq p \leq u-1$, where $\epsilon = 1/b_1 < 1$. Therefore,

$$Y \leq x_{\text{new}}(i_u)\{u\epsilon^{u-1} + (u-1)\epsilon^{u-2} + \ldots + 2\epsilon + 1\} \leq \frac{x_{\text{new}}(i_u)}{(1-\epsilon)^2} \leq \frac{N}{(1-\epsilon)^2} = O(N).$$

For each node n in G, let $t(n)$ be the total number of times n is expanded by IDA*. Then

$$t(n) \leq \begin{cases} u + Mu = (u)(M+1) & \text{if } n \in X_{\text{new}}(i_1); \\ (u-1) + Mu = (u)(M+1) - 1 & \text{if } n \in X_{\text{new}}(j_{11}); \\ \ldots & \\ (u-1) + (u-1)M + 1 = (u)(M+1) - M & \text{if } n \in X_{\text{new}}(j_{1c_1}); \\ & \\ (u-1) + M(u-1) = (u-1)(M+1) & \text{if } n \in X_{\text{new}}(i_2); \\ (u-2) + M(u-1) = (u-1)(M+1) - 1 & \text{if } n \in X_{\text{new}}(j_{21}); \\ \ldots & \\ (u-2) + (u-2)M + 1 = (u-1)(M+1) - M & \text{if } n \in X_{\text{new}}(j_{2c_2}); \\ & \\ \ldots & \\ & \\ (u-(u-1)) + M(u-(u-1)) = M+1 & \text{if } n \in X_{\text{new}}(i_u); \\ (u-u) + M(u-(u-1)) = (M+1) - 1 & \text{if } n \in X_{\text{new}}(j_{u1}); \\ \ldots & \\ (u-u) + (u-u)M + 1 = (M+1) - M & \text{if } n \in X_{\text{new}}(j_{uc_u}). \end{cases}$$

Therefore, the total number of node expansions is

$$
\begin{aligned}
x_{\text{tot}} \quad = \quad & \sum_{n \in X_{\text{new}}(i_1)} t(n) + \left(\sum_{n \in X_{\text{new}}(j_{11})} t(n) + \ldots + \sum_{n \in X_{\text{new}}(j_{1c_1})} t(n) \right) \\
& + \sum_{n \in X_{\text{new}}(i_2)} t(n) + \left(\sum_{n \in X_{\text{new}}(j_{21})} t(n) + \ldots + \sum_{n \in X_{\text{new}}(j_{2c_2})} t(n) \right) \\
& + \cdots \\
& + \sum_{n \in X_{\text{new}}(i_u)} t(n) + \left(\sum_{n \in X_{\text{new}}(j_{u1})} t(n) + \ldots + \sum_{n \in X_{\text{new}}(j_{uc_u})} t(n) \right) \\
\\
\leq \quad & \sum_{n \in X_{\text{new}}(i_1)} (u)(M+1) \\
& + \sum_{n \in X_{\text{new}}(j_{11})} ((u)(M+1) - 1) + \ldots + \sum_{n \in X_{\text{new}}(j_{1c_1})} ((u)(M+1) - M) \\
& + \sum_{n \in X_{\text{new}}(i_2)} (u-1)(M+1) \\
& + \sum_{n \in X_{\text{new}}(j_{21})} ((u-1)(M+1) - 1) + \ldots + \sum_{n \in X_{\text{new}}(j_{2c_2})} ((u-1)(M+1) - M) \\
& + \cdots \\
& + \sum_{n \in X_{\text{new}}(i_u)} (M+1) \\
& + \sum_{n \in X_{\text{new}}(j_{u1})} ((M+1) - 1) + \ldots + \sum_{n \in X_{\text{new}}(j_{uc_u})} ((M+1) - M) \\
\\
\leq \quad & x_{\text{new}}(i_1)(u)(M+1) \\
& + (b_1)x_{\text{new}}(i_1)((u)(M+1) - 1) + \ldots + (b_1)x_{\text{new}}(i_1)((u)(M+1) - M) \\
& + x_{\text{new}}(i_2)(u-1)(M+1) \\
& + (b_1)x_{\text{new}}(i_2)((u)(M+1) - 1) + \ldots + (b_1)x_{\text{new}}(i_2)((u-1)(M+1) - M) \\
& + \cdots \\
& + x_{\text{new}}(i_u)(M+1) \\
& + (b_1)x_{\text{new}}(i_u)((M+1) - 1) + \ldots + (b_1)x_{\text{new}}(i_u)((M+1) - M) \\
\\
\leq \quad & (b_1)x_{\text{new}}(i_1)(u)(M+1)(M+1)
\end{aligned}
$$

$$+ (b_1)x_{\text{new}}(i_2)(u-1)(M+1)(M+1)$$

$$+ \cdots$$

$$+ (b_1)x_{\text{new}}(i_u)(M+1)(M+1)$$

$$\leq \quad (b_1)(M+1)(M+1)(ux_{\text{new}}(i_1) + (u-1)x_{\text{new}}(i_2) + \ldots + x_{\text{new}}(i_u))$$

$$\leq \quad (M+1)(M+1)(b_1)Y.$$

Thus since $Y = O(N)$, b_1 and $M = O(1)$, $x_{\text{tot}} = O(N)$. ■

Example 6 In Example 4 of Section 4, the total number iterations I is a constant. Therefore, M = the maximum number of adjacent dummy iterations = $O(1)$, so by Theorem 1, $x_{tot} = O(N)$.

6.2 Necessary Conditions for Optimality

In order to present necessary conditions for IDA* to be optimal, we first need the following lemma. This lemma shows that if a tree G' is constructed from G in such a way that G' is identical to G except that one node n in G' has a higher f-value than in G, i.e. $f^{G'}(n) > f^{G}(n)$, then the total number of node expansions by IDA* on G' will be less than the number of node expansions by IDA* on G. What this means is that if a new problem instance is created from an old problem instance of IDA* by pushing a new node of iteration j to the iteration k, such that $k > j$, then x_{tot} in the new problem instance will be less than in the old problem instance.

Lemma 1 Let G be any tree on which IDA* does $I \geq 2$ iterations, and let $1 \leq j < k \leq I$. If $x_{\text{new}}(j) = 1$, then let G' be any tree on which IDA* does $I' = I - 1$ iterations and

$$\begin{aligned}
x'_{\text{new}}(i) &= x_{\text{new}}(i), & i &= 1, \ldots, j-1; \\
x'_{\text{new}}(i) &= x_{\text{new}}(i+1), & i &= j, \ldots, k-2; \\
x'_{\text{new}}(k-1) &= x_{\text{new}}(k) + 1; & & \\
x'_{\text{new}}(i) &= x_{\text{new}}(i+1), & i &= k, \ldots, I-1.
\end{aligned}$$

Otherwise, let G' be any tree on which IDA* does $I' = I$ iterations and

$$\begin{aligned}
x'_{\text{new}}(i) &= x_{\text{new}}(i), && i = 1, \ldots, j-1; \\
x'_{\text{new}}(j) &= x_{\text{new}}(j) - 1; && \\
x'_{\text{new}}(i) &= x_{\text{new}}(i), && i = j+1, \ldots, k-1; \\
x'_{\text{new}}(k) &= x_{\text{new}}(k) + 1; && \\
x'_{\text{new}}(i) &= x_{\text{new}}(i+1), && i = k+1, \ldots, I.
\end{aligned}$$

Then $x'_{\text{tot}} < x_{\text{tot}}$.

Proof. There are two cases:

1. $x_{\text{new}}(j) = 1$. Then the number of node expansions on G' is

$$\begin{aligned}
x'_{\text{tot}} &= [(I')x'_{\text{new}}(1) + \ldots + (I' - j + 2)x'_{\text{new}}(j-1)] \\
&\quad + [(I' - j + 1)x'_{\text{new}}(j) + \ldots + (I' - k + 3)x'_{\text{new}}(k-2)] \\
&\quad + (I' - k + 2)x'_{\text{new}}(k-1) \\
&\quad + (I' - k + 1)x'_{\text{new}}(k) + \ldots + x'_{\text{new}}(I') \\
&= [(I-1)x_{\text{new}}(1) + \ldots + (I - j + 1)x_{\text{new}}(j-1)] \\
&\quad + [(I - j)x_{\text{new}}(j+1) + \ldots + (I - k + 2)x_{\text{new}}(k-1)] + (I - k + 1)(x_{\text{new}}(k) + 1) \\
&\quad + (I - k)x_{\text{new}}(k+1) + \ldots + x_{\text{new}}(I) \\
&< [(I)x_{\text{new}}(1) + \ldots + (I - j + 2)x_{\text{new}}(j-1)] \\
&\quad + [(I - j)x_{\text{new}}(j+1) + \ldots + (I - k + 2)x_{\text{new}}(k-1)] + (I - k + 1)x_{\text{new}}(k) \\
&\quad + (I - k)x_{\text{new}}(k+1) + \ldots + x_{\text{new}}(I) \\
&< [(I)x_{\text{new}}(1) + \ldots + (I - j + 2)x_{\text{new}}(j-1)] + (I - j + 1)x_{\text{new}}(j) \\
&\quad + [(I - j)x_{\text{new}}(j+1) + \ldots + (I - k + 2)x_{\text{new}}(k-1)] + (I - k + 1)x_{\text{new}}(k) \\
&\quad + (I - k)x_{\text{new}}(k+1) + \ldots + x_{\text{new}}(I) \\
&= x_{\text{tot}}.
\end{aligned}$$

2. $x_{\text{new}}(j) \neq 1$. Then the number of node expansions on G' is

$$\begin{aligned}
x'_{\text{tot}} &= [I'x'_{\text{new}}(1) + \ldots + (I'-j+2)x'_{\text{new}}(j-1)] + (I'-j+1)x'_{\text{new}}(j) \\
&\quad + [(I'-j)x'_{\text{new}}(j+1) + \ldots + (I'-k+2)x'_{\text{new}}(k-1)] \\
&\quad + (I'-k+1)x'_{\text{new}}(k) \\
&\quad + (I'-k)x'_{\text{new}}(k+1) + \ldots + x'_{\text{new}}(I') \\
&= [Ix_{\text{new}}(1) + \ldots + (I-j+2)x_{\text{new}}(j-1)] + (I-j+1)(x_{\text{new}}(j)-1) \\
&\quad + [(I-j)x_{\text{new}}(j+1) + \ldots + (I-k+2)x_{\text{new}}(k-1)] + (I-k+1)(x_{\text{new}}(k)+1) \\
&\quad + (I-k)x_{\text{new}}(k+1) + \ldots + x_{\text{new}}(I) \\
&< [Ix_{\text{new}}(1) + \ldots + (I-j+2)x_{\text{new}}(j-1)] + (I-j+1)x_{\text{new}}(j) \\
&\quad + [(I-j)x_{\text{new}}(j+1) + \ldots + (I-k+2)x_{\text{new}}(k-1)] + (I-k+1)x_{\text{new}}(k) \\
&\quad + (I-k)x_{\text{new}}(k+1) + \ldots + x_{\text{new}}(I) \\
&= x_{\text{tot}}.
\end{aligned}$$

■

Note that if G is any state space, then for $j = 1, \ldots, I$, the total number of node expansions by IDA* at iteration j denoted by x_j is

$$x_j = \sum_{i=1}^{j} x_{\text{new}}(i).$$

Thus, if I is the total number of iterations done by IDA* on G, then the total number of node expansions done by IDA* on G is

$$\begin{aligned}
x_{\text{tot}} &= \sum_{j=1}^{I} x_j \\
&= \sum_{j=1}^{I}\sum_{i=1}^{j} x_{\text{new}}(i) \\
&= Ix_{\text{new}}(1) + (I-1)x_{\text{new}}(2) + \ldots + x_{\text{new}}(I). \qquad (14)
\end{aligned}$$

Furthermore, since i_u is the last active iteration on G, $I = i_u + c_u$. Thus from Eq. (14),

$$\begin{aligned} x_{\text{tot}} &= I x_{\text{new}}(1) + (I-1)x_{\text{new}}(2) + \ldots + x_{\text{new}}(I) \\ &> (c_u)x_{\text{new}}(i_u+1) + (c_u-1)x_{\text{new}}(i_u+2) + \ldots + x_{\text{new}}(i_u+c_u). \end{aligned} \tag{15}$$

Theorem 2 Let $\mathcal{G} = (G_1, G_2, \ldots)$ be any sequence of trees. Then in $\mathcal{G}$, IDA*'s total number of node expansions is $x_{\text{tot}} = O(N)$ only if $c_u = O(1)$, i.e., only if the number of dummy iterations after the last active iteration is bounded by a constant.

Proof. Suppose that $x_{\text{tot}} = O(N)$ and $c_u \neq O(1)$. Then there are two cases:

1. $x_{i_u} = \Omega(N)$. Clearly, $x_{\text{tot}} > c_u x_{i_u}$. Thus

$$c_u < \frac{x_{\text{tot}}}{x_{i_u}} = \frac{O(N)}{\Omega(N)} = O(1),$$

   which is a contradiction.

2. $x_{i_u} \neq \Omega(N)$. To show that $x_{tot} \neq O(N)$, we have to show that there does not exist any constant $c > 0$, such that $x_{tot} \leq cN$. Assume the opposite, i.e. there exists a constant $c > 0$ such that $x_{tot} \leq cN$. Let $d = (2 + b_1(1+2c) - \sqrt{(2 + b_1(1+2c))^2 - 4})/2$. Since $x_{i_u} \neq \Omega(N)$, therefore it must be the case that $x_{i_u} < dN$.

   Now, by repeated application of Lemma 1, it follows that $x_{\text{tot}} \geq x'_{\text{tot}}$, where G' is a search tree having the following properties:

$$\begin{aligned} N' &= N; \\ u' &= u; \\ i'_u &= i_u; \\ x'_{\text{new}}(i) &= x_{\text{new}}(i), && i = 1, \ldots, i_u; \\ x'_{\text{new}}(i) &\leq (b_1)x_{\text{new}}(i_u) - 1, && i = i_u + 1; \\ x'_{\text{new}}(i) &= (b_1)x_{\text{new}}(i_u) - 1, && i = i_u + 2, \ldots, I'; \end{aligned}$$

$$c'_u \quad = \quad I' - i_u = \left\lceil \frac{N - x_{i_u}}{(b_1)x_{\text{new}}(i_u) - 1} \right\rceil .$$

Thus from Eq. (15),

$$\begin{aligned}
x_{\text{tot}} &\geq x'_{\text{tot}} \\
&> (c'_u)x'_{\text{new}}(i_u + 1) \\
&\quad + [(c'_u - 1)x'_{\text{new}}(i_u + 2) + (c'_u - 2)x'_{\text{new}}(i_u + 3) + \ldots + x'_{\text{new}}(i_u + c'_u)] \\
&\geq 0 + [(c'_u - 1)x'_{\text{new}}(i_u + 2) + (c'_u - 2)x'_{\text{new}}(i_u + 3) + \ldots + x'_{\text{new}}(i_u + c'_u)] \\
&= (c'_u - 1)(b_1 x_{\text{new}}(i_u) - 1) + (c'_u - 2)(b_1 x_{\text{new}}(i_u) - 1) + \ldots + (b_1 x_{\text{new}}(i_u) - 1) \\
&= \frac{c'_u(c'_u - 1)}{2}(b_1 x_{\text{new}}(i_u) - 1) \\
&= \frac{(N - x_{i_u})^2}{2(b_1 x_{\text{new}}(i_u) - 1)} - \frac{(N - x_{i_u})}{2} \\
&> \frac{(N - x_{i_u})^2}{2b_1 x_{\text{new}}(i_u)} - \frac{N}{2} \\
&> \frac{(N - dN)^2}{2b_1 dN} - \frac{N}{2} \\
&= \frac{N}{2}[\frac{(1-d)^2}{b_1 d} - 1] \\
&= cN
\end{aligned}$$

which contradicts our assumption that $x_{tot} \leq cN$. ■

Example 7 Consider Example 1 of Section 4. For any b_1, the last N' iterations are dummy iterations, i.e. $c_u = \Omega(N') = \Omega(N)$. Therefore by Theorem 2 $x_{tot} \neq O(N)$.

Example 8 Consider Example 2 of Section 4. For any b_1, the last k iterations are dummy iterations, i.e. $c_u = \Omega(k) = \Omega(\lg N)$. Therefore by Theorem 2 $x_{tot} \neq O(N)$.

Example 9 Consider Example 3 of Section 4. For any b_1, the last $k - 1$ iterations are dummy iterations, i.e. $c_u = \Omega(k - 1) = \Omega(\lg N)$. Therefore by Theorem 2 $x_{tot} \neq O(N)$.

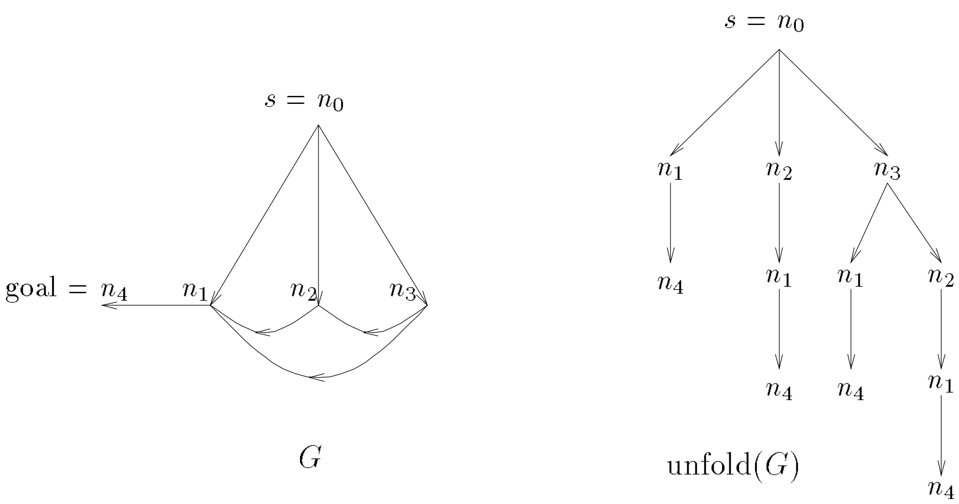


Figure 10: A graph and the corresponding unfolded tree.

7 IDA* on Acyclic Graphs

When A* is run with a monotone heuristic on a directed graph G, it expands no node more than once, and thus does only $O(N)$ node expansions. However, depth-first search will expand a node n many times, once for each path it finds from s to n. Thus, there are cases where depth-first search can do 2^N node expansions on a directed acyclic graph with N possibly expandable nodes [10]. Since IDA* does a depth-first search at each iteration, this means that on graphs it can also do exponentially many node expansions. More specifically, this section shows that in the worst case, IDA* does $\Theta(2^{2N})$ node expansions.

In the previous sections, all of our terminology for talking about IDA*'s behavior was developed for trees rather than graphs. If we run a procedure like IDA* on a directed graph G, it behaves exactly the same as if it were searching the *unfolded* tree unfold(G), which contains several duplicates of n: one for for each path from s to n in G (for an example, see Figure 10). Thus, to extend our terminology to handle graphs as well, we can refer to the unfolded tree unfold(G). For example, the set of new nodes expanded by IDA* on G during iteration j is $X^G_{\text{new}}(j) = X^{\text{unfold}(G)}_{\text{new}}(j)$. Since $X^{\text{unfold}(G)}_{\text{new}}(j)$ is the set of new nodes in unfold(G), it may contain several duplicates of each node n of G.

The following theorem establishes an upper bound on the number of node expansions done by

IDA*:

Theorem 3 IDA* does no more than $(2^N + 2^{2N})/2$ node expansions on acyclic graphs with N possibly expandable nodes.

Proof. In an acyclic graph with N possibly expandable nodes, there can be at most 2^N distinct paths. From Theorem 5, in the worst case IDA* expands exactly one new path (i.e. $x_{\text{new}}(k) = 1$) at each iteration $k, 1 \leq k \leq I = 2^N$. From Eq. (14) the total number of node expansions by IDA*

$$= \quad I x_{\text{new}}(1) + (I-1) x_{\text{new}}(2) + \ldots + x_{\text{new}}(I).$$

Since the maximum occurs at $I = 2^N$ and $x_{\text{new}}(k) = 1$ for each $k, 1 \leq k \leq I$. Therefore the total number of node expansions by IDA*

$$\begin{aligned} &\leq \quad 2^N + 2^{N-1} + \ldots + 1. \\ &\leq \quad (2^N + 2^{2N})/2. \end{aligned}$$

■

The next step is to show that in the worst case, this upper bound is actually achieved. Below, we present theorems showing how to construct worst-case examples for IDA*. By using these theorems to construct worst-case examples, and analyzing how IDA* performs in these examples, we show that there are cases in which IDA* can do $\Omega(2^{2N})$ node expansions on directed acyclic graphs with N possibly expandable nodes.

The total number of node expansions by IDA* depends not only on the number of dummy iterations but also on their positions. In the following theorem we show that, keeping the total number of iterations I and the number of active iterations u fixed, the total number of node expansions by IDA* increases as the dummy iterations are moved to the right, i.e. a dummy iteration j is moved to k where $k > j$. In particular, the theorem shows that IDA*'s total number of node expansions x_{tot} attains its maximum when all the dummy iterations appear after the last active iteration.

Theorem 4 Let N, u, I, be positive integers, and let $\mathcal{G}$ be the set of all graphs G for which there are N possibly expandable nodes, u active iterations, and I iterations. Then over all members of $\mathcal{G}$, the maximum value of $x_{\rm tot}$ (the number of node expansions by IDA*) occurs in a graph G for which all dummy iterations occur after the last active iteration, i.e., $c_1 = c_2 = \ldots = c_{u-1} = 0$.

Proof.

Case 1: $u = 1$ or $u = I$. Then either all dummy iterations occur after the first active iteration or there are no dummy iterations, so the proof is vacuous.

Case 2: $u = 2$ and $I = 3$. Then there are two possibilities. $c_1 = 1$ and $c_2 = 0$, or $c_1 = 0$ and $c_2 = 1$. Below, we show that the first possibility produces a smaller value for $x_{\rm tot}$.

Let G be any graph such that $c_1 = 1$ and $c_2 = 0$. Then $i_1 = 1$, $j_{11} = 2$, and $i_2 = 3$, and

$$x_{\rm tot} = 3x_{\rm new}(i_1) + 2x_{\rm new}(j_{11}) + x_{\rm new}(i_2).$$

There is a graph G' such that $c'_1 = 0$ and $c'_2 = 1$ and

$$\begin{aligned} x'_{\rm new}(i'_1) &= x_{\rm new}(i_1); \\ x'_{\rm new}(i'_2) &= x_{\rm new}(i_2); \\ x'_{\rm new}(j'_{21}) &= x_{\rm new}(j_{11}). \end{aligned}$$

Thus $i'_1 = 1$, $i'_2 = 2$, and $j'_{21} = 3$, and

$$x'_{\rm tot} = 3x'_{\rm new}(i'_1) + 2x'_{\rm new}(i'_2) + x'_{\rm new}(j'_{21}).$$

But $x_{\rm new}(j_{11}) < x_{\rm new}(i_2)$, for otherwise j_{11} would be an active iteration. Thus,

$$\begin{aligned} x_{\rm tot} &= 3x_{\rm new}(i_1) + 2x_{\rm new}(j_{11}) + x_{\rm new}(i_2) \\ &< 3x_{\rm new}(i_1) + 2x_{\rm new}(i_2) + x_{\rm new}(j_{11}) \end{aligned}$$

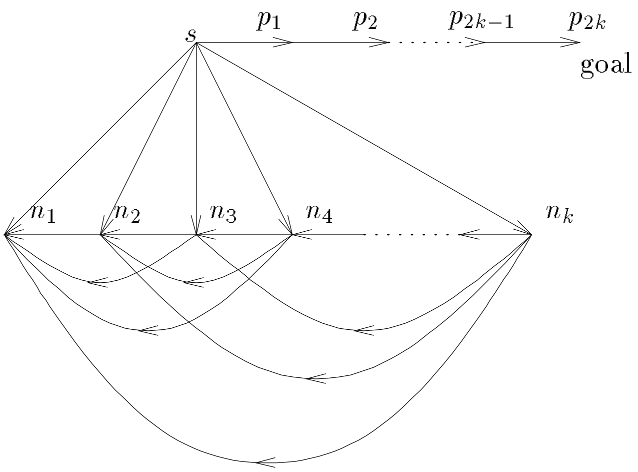


Figure 11: IDA*: $\Omega(N2^N)$

$$\begin{aligned} &= 3x'_{\text{new}}(i'_1) + 2x'_{\text{new}}(i'_2) + x'_{\text{new}}(j'_{21}) \\ &= x'_{\text{tot}}. \end{aligned}$$

Case 3: *all other cases.* In general, the total number of node expansions by IDA* will be

$$\begin{aligned} x_{\text{tot}} &= Ix_{\text{new}}(i_1) + \sum_{q=1}^{c_1}(I - j_{1q} + 1)x_{\text{new}}(j_{1q}) \\ &\quad + (I - i_2 + 1)x_{\text{new}}(i_2) + \sum_{q=1}^{c_2}(I - j_{2q} + 1)x_{\text{new}}(j_{2q}) \\ &\quad + \ldots \\ &\quad + (I - i_u + 1)x_{\text{new}}(i_u) + \sum_{q=1}^{c_u}(I - j_{uq} + 1)x_{\text{new}}(j_{uq}). \end{aligned}$$

From the definition of an active iteration, it follows that if i is any active iteration and j is any dummy iteration preceding i, then $x_{\text{new}}(j) < x_{\text{new}}(i)$. Thus, using the same argument as in the case $u = 2, I = 3$ above, it can be shown that if we shift a dummy iteration from its current position to a position after the next active iteration then the number of nodes generated by IDA* will increase. Thus, the maximum value for x_{tot} occurs in a graph for which all dummy iterations occur after the last active iteration, i.e., $c_1 = c_2 = \ldots = c_{u-1} = 0$.

■

Example 10 Theorem 4 says that IDA* will perform badly in cases where all dummy iterations occur after the last active iteration. As an example, consider the search graph shown in Figure 11. There are $N = 3k$ non-goal nodes, all of which are possibly expandable; and one goal node. Each arc has unit cost, and $h(n) = 0$ for every node n. If we run A* on G, it will do N node expansions. If we run IDA* on G, there will be $2k + 1$ iterations, with $2k + 1$ thresholds $0, 1, 2, \ldots 2k$.

Iterations 0 through $\lfloor (k-1)/2 \rfloor$ are active iterations, and the remaining iterations are dummy iterations. More specifically, for each threshold i, IDA* expands $1 + \binom{k}{i}$ new nodes if $0 \leq i \leq k$, and makes $i + 2^k$ node expansions if $k + 1 \leq i \leq 2k$. Thus the total number of node expansions by IDA* is:

$$\begin{aligned}
x_{\rm tot} &= \frac{(2k)(2k+1)}{2} + \sum_{i=0}^{k} \binom{(k+1-i)}{i} + k2^k \\
&= \frac{(2k)(2k+1)}{2} + (k+1)2^{k-1} + 2^k + k2^k \\
&= (k+1)2^{k-1} + k2^k + 2^k + k(2k+1) \\
&= \Omega(k2^k) \\
&= \Omega(N2^N).
\end{aligned}$$

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

The following theorem tells how to construct worst-case examples for IDA* if we keep the number of possibly expandable nodes fixed without requiring that the number of active iterations and total number of iterations be fixed.

Theorem 5 Let N be a positive integer, and $\mathcal{G}$ be the set of all graphs for which there are N possibly expandable nodes. Then over all members of $\mathcal{G}$, the maximum value of $x_{\rm tot}$ occurs in a graph for which $x_{\rm new}(k) = 1$ for each iteration k.

Proof. Immediate from Lemma 1. ∎

Example 11 Theorem 5 says that IDA*'s worst case occurs if only one new node is expanded in each iteration. As an example, consider the search graph G shown in Figure 12 (which is a

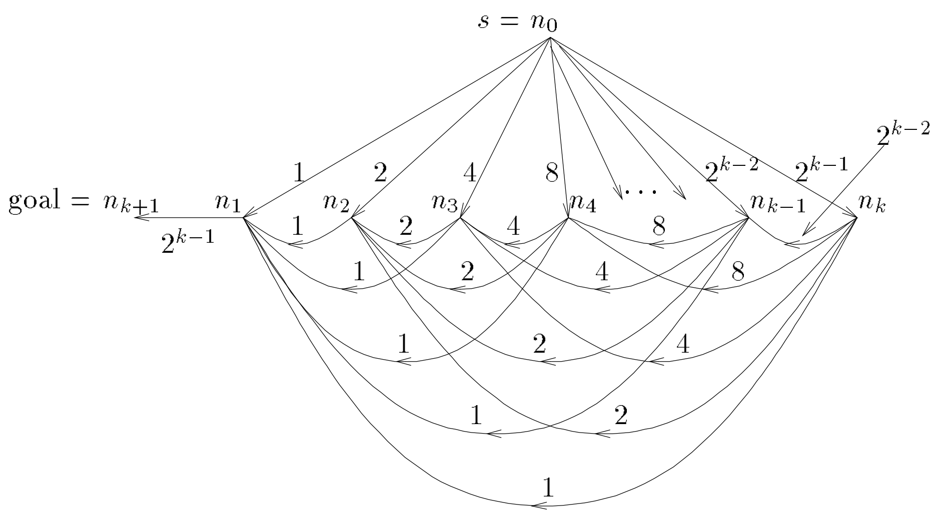


Figure 12: IDA*: $\Omega(2^{2N})$.

generalization of the graph shown in Figure 10). Let n_0 be the start node and n_{k+1} be the goal node. The cost structure is defined as follows:

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

It can be easily seen that unfold(G) will contain nodes of all f-values from 0 through $2^k (= 2^{N-1})$. Therefore the total number of node expansions will be $O(N)$ for A*, and $\Omega(2^{2N})$ for IDA*.

8 Limited-memory Search on Trees

In this section, we show that in general, limited-memory best-first search algorithms cannot always perform as well as A*, even on trees.

Let G be a tree, and $\mathcal{A}$ be any search algorithm used to search G. $\mathcal{A}$ *stores* a node n if during the current state of $\mathcal{A}$'s execution, $\mathcal{A}$ contains information about the identity of node n (plus possibly some other information about n). $\mathcal{A}$ *properly stores* node n if it stores not only n, but also at least one of the parents of n. $\mathcal{A}$ *properly runs* in storage $S \geq 0$ if at all times during its operation, it properly stores no more than S nodes. $\mathcal{A}$ is said to run in limited-memory if it properly runs in storage S.

The usual definition of "best first" is inappropriate for limited memory algorithms, because they may expand nodes more than once. Instead, a limited-memory algorithm $\mathcal{A}_{bf}$ is said to perform a best-first search on tree G if for every $j \geq 1$, it does not expand any node of $X^G_{\mathrm{new}}(j+1)$ before expanding every node of $X^G(z(j))$ at least once.

The following theorem shows that there exists no best-first tree search algorithm, which while using less than a constant fraction of the memory used by A*, can have the same worst-case asymptotic time complexity as A* on all trees. The following lemma is used to prove the theorem.

Lemma 2 Let G be a b-ary tree that is complete to depth k for some $k > 0$, and $\mathcal{A}$ be a search algorithm that properly runs in storage S on G. Let d be the smallest integer such that $S \leq \frac{b^{d+1}-1}{b-1}$. If $d < k$, then $\mathcal{A}$ properly stores no more than b^d of the nodes at depth $d+1$ of G.

Proof. Suppose the lemma is false, i.e., that its conditions are satisfied but $\mathcal{A}$ properly stores more than b^d of the nodes of depth $d+1$ of G. Then $\mathcal{A}$ must properly store at least the following number of nodes at each depth:

Depth	Nodes	
$d+1$	$> b^d$	(by assumption)
d	$> b^{d-1}$	(since the branching factor is b)
$d-1$	$> b^{d-2}$	
$\vdots$	$\vdots$	
2	$> b$	
1	> 1	
0	1	

Thus the total number of nodes properly stored by $\mathcal{A}$ exceeds $1 + 1 + b + b^2 + \ldots + b^d = 1 + \frac{b^{d+1}-1}{b-1}$. But from the statement of the lemma $S \leq \frac{b^{d+1}-1}{b-1}$. This is a contradiction. ■

Theorem 6 There is no best-first algorithm $\mathcal{A}_{bf}$ such that for every sequence of trees $\mathcal{G} = \{G_i\}_{i=1}^{\infty}$, $\mathcal{A}_{bf}$ has $O(N)$ complexity and properly runs in $S = \frac{N}{\psi(N)}$ memory, where $\psi(N)$ is a function that is $\neq O(1)$. The same is true even if $\mathcal{G}$ is restricted to be a sequence of trees such that the number of possibly expandable nodes grows exponentially as a function of i.[7]

Proof. By contradiction. Let us assume that there exists an algorithm $\mathcal{A}_{bf}$ which when running with memory S has $O(N)$ complexity. Now we construct an infinite sequence of trees $\mathcal{G} = (G_1, G_2, \ldots)$ and show that the assumption is false. From now on in this proof, by G we mean a tree of this sequence. Let G be a uniform $b-$ary tree of height $H = \lceil \log_b(N) \rceil$. At level H there is only one goal node which is the left most leaf node and every other leaf node of level H is a non-terminal leaf node.

Let $d \geq 0$ be such that

$$\frac{b^d - 1}{b - 1} < S \leq \frac{b^{d+1} - 1}{b - 1}$$

Let $d_0 = d + 1$. Now by lemma 2, at any time during its execution, $\mathcal{A}_{bf}$ cannot properly store more than b^d nodes of level $d \geq d_0$. Let the nodes of level d_0 be named as

$$m_1, m_2, \ldots, m_q, q = b^{d_0}$$

Below each $m_i, 1 \leq i \leq q$, there will be b^{H-d_0-1} nodes of level $H - 1$. Let $k = b^{H-d_0-1}$. Let us call the descendents at level $H - 1$ of node m_i as $m_{i,1}, m_{i,2}, \ldots, m_{i,k}$. Out of the total b^{H-1} nodes at level $H - 1$, we construct k disjoint sets of nodes $S_1, S_2, \ldots, S_k$ in a manner such that every node belonging to set $S_j, 1 \leq j \leq k - 1$ has f-value smaller than every node belonging to set S_{j+1}. Now we present a detailed construction of G by giving the arc-costs and heuristic estimates as follows:

[7]In the case where we do not restrict $\mathcal{G}$ to have this property, the proof is relatively trivial—it suffices to consider the case where each G_i is just a path of length i.

$$
\begin{aligned}
c(s,n) &= 1 && \text{for every child } n \text{ of the root node } s; \\
c(n,n_i) &= 1 && \text{for every pair of nodes } n, n_i \text{ such that } n_i \text{ is} \\
& && \text{a child of } n, \text{ and } n_i \text{ is not a goal node.} \\
c(n,n_i) &= 1+k && \text{if } n_i \text{ is a child of } n, \text{ and } n_i \text{ is a goal node.} \\
h(s) &= H-1 && \\
h(n) &= H-1-d && \text{if } n \text{ is a node of level } d < H-1 \\
h(n) &= 0 && \text{if } n \text{ is a node of level } H \\
h(m_{i,j}) &= j && 1 \le i \le q, 1 \le j \le k
\end{aligned}
$$

Following the construction given above, we get for $1 \le j \le k$,

$$S_j = \{m_{i,j} \mid 1 \le i \le q\}$$

and

$$f(m_{i,j}) = H - 1 + j, 1 \le i \le q$$

Clearly, if $n_1 \in S_j$ and $n_2 \in S_{j+1}$ then

$$f(n_2) = f(m_{i,j+1}) = H + j > H - 1 + j = f(m_{i,j}) = f(n_1)$$

The nodes of level $H-1$ have f-values $H, H+1, \ldots, H+k-1$. The only goal node at level H has f-value $= H-1+k+1 = H+k$, which is greater than the f-value of any node of level $H-1$. Therefore, $\mathcal{A}_{bf}$ must expand every node of level $H-1$ prior to selecting the goal node for expansion and terminating. In other words $\mathcal{A}_{bf}$ must expand all nodes of each set $S_j, 1 \le j \le k$, containing nodes of equal f-values. Note that starting from their ancestors at level d_0, no two nodes of any set S_j belong to a common path.

Let $l_1, l_2, \ldots, l_k$ be the time instants when $\mathcal{A}_{bf}$ begins expanding nodes of sets $S_1, S_2, \ldots, S_k$ respectively. At time instant $l_j, 1 \le j \le k$, $\mathcal{A}_{bf}$ begins expanding nodes of S_j. We know all nodes $m_{i,j} \in S_j$ are from disjoint paths from level d_0 to level $H-1$. There are a total of b^{d_0} disjoint paths (from level d_0 to level $H-1$) for b^{d_0} nodes in S_j, of which at most b^{d_0-1} paths or parts of them can be in memory at instant l_j, and no portion of the remaining $(b-1)b^{d_0-1}$ paths (from level d_0 to level $H-1$) have been stored. Prior to the instant l_{j+1} all the nodes of S_j must be expanded. Therefore, all nodes of these $(b-1)b^{d_0-1}$ paths must be expanded prior to the instant l_{j+1}. Now expanding this logic for each $j, 1 \leq j \leq k$ we get the total number of node expansions by $\mathcal{A}_{bf}$

$$
\begin{aligned}
&\geq (b^{H-d_0-1})(b-1)b^{d_0-1}(H-d_0-1) \\
&= (b-1)(b^{H-2})(H-d) \qquad (16)
\end{aligned}
$$

From the construction of the tree, we know that

$$
\begin{aligned}
N &= N^G = \frac{b^H-1}{b-1} \\
H &= \lceil \log_b N \rceil
\end{aligned}
$$

and since

$$
S > \frac{b^d-1}{b-1}
$$

therefore

$$
d < \log_b S + 1 = log_b(N) - \log_b(\psi(N)) + 1
$$

Now substituting the values of H, b^{H-2} and maximum value of d in Eq. (16), we get the total node expansions by $\mathcal{A}_{bf}$

$$
\begin{aligned}
&\geq \left(\frac{b-1}{b^2}\right)[(b-1)N+1][\log_b \psi(N) - 1] \\
&= \Omega(N \lg \psi(N)) \\
&\neq O(N)
\end{aligned}
$$

since $\psi(N) \neq O(1)$. ■

9 Conclusion

Although the A* search algorithm is optimal in terms of number of node expansions, its exponential memory requirement makes it impractical on a number of problems. This difficulty with A* and similar state-space search algorithms has led to the development of limited-memory algorithms such as IDA*. Basically, limited-memory algorithms save space at the expense of time: they save space by removing nodes from their memory storage, but this means that they will need to re-generate and re-expand these nodes later.

In this paper, our goal has been to do an in-depth analysis of the nature of this tradeoff. In particular, we have presented the following results:

1. We have presented necessary and sufficient conditions for IDA* to be asymptotically optimal. Our conditions show that IDA* is asymptotically optimal in a somewhat different range of problems than was originally believed. For example, the conditions stated in [10] are not sufficient to guarantee asymptotic optimality of IDA*; i.e., IDA* will perform badly in some of the trees on which it was thought to be asymptotically optimal. However, this failing is not unique to IDA*, for we have shown that in general, no best-first limited-memory heuristic search algorithm can be asymptotically optimal.

2. It is well known that the amount of memory needed by A* can be exponential in the amount of memory needed by IDA*. But on graphs, the amount of time take by IDA* can be exponential in the amount of time take by A*—even if IDA* is using a monotone heuristic. Thus, if sufficient memory is available, on graphs it is much preferable to use a graph search algorithm rather than using IDA*.

Acknowledgement: We thank Richard Korf for many helpful discussions.

References

[1] A. Bagchi and A. Mahanti. Search algorithms under different kinds of heuristics-a comparative study. *JACM*, 30(1):1–21, 1983.

[2] A. Bagchi and A. Mahanti. Three approaches to heuristic search in networks. *JACM*, 32(1):1–27, 1985.

[3] P. P. Chakrabarti, S. Ghosh, A. Acharya, and S. C. De Sarkar. Heuristic search in restricted memory. *Artificial Intelligence*, 47:197–221, 1989.

[4] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *JACM*, 32(3):505–536, 1985.

[5] M. Evett, J. Hendler, A. Mahanti, and D. Nau. PRA*: A memory-limited heuristic search procedure for the connection machine. In *Frontiers'90: Frontiers of Massively Parallel Computation*, 1990.

[6] S. Ghosh, A. Mahanti, and D. S. Nau. ITS: An efficient limited-memory heuristic tree search algorithm. In *AAAI 1994*, pages 1353–1358, 1994.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Sciences and Cybernetics*, pages 1556–1562, 1968.

[8] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[9] R. E. Korf. Depth first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[10] R. E. Korf. Optimal path finding algorithms. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 200–222. Springer Verlag, 1988.

[11] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11:972–989, 1963.

[12] A. Mahanti, S. Ghosh, D. S. Nau, A. K. Pal, and L. N. Kanal. Performance of IDA* on trees and graphs. In *AAAI 1992*, pages 539–544, 1992.

[13] A. Mahanti and A. K. Pal. Worst-case time complexity of IDA*. In *Tenth International Conference in Computer Science*, Santiago, Chile, July 1990.

[14] A. Mahanti and K. Ray. Network search algorithms with modifiable heuristics. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 200–222. Springer Verlag, 1988.

[15] A. Martelli. On the complexity of admissible search algorithms. *Artificial Intelligence*, 8:1–13, 1977.

[16] L. Mero. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23:13–27, 1984.

[17] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, 1980.

[18] B. G. Patrick, M. Almulla, and M. M. Newborn. An upper bound on the complexity of iterative-deepening-A*. *Annals of Mathematics and Artificial Intelligence*, 5:265–278, 1992.

[19] J. Pearl. *Heuristics.* Addison-Wesley, Reading, MA, 1984.

[20] S. Russell. Efficient memory-bounded search methods. In *ECAI-1992*, Vienna, Austria, 1992.

[21] A. Sen and A. Bagchi. Fast recursive formulations for best-first search that allow controlled use of memory. In *IJCAI-89*, pages 274–277, 1989.