**Iterative Deepening Search (IDS)**

**Abstract.**

 The memory requirements of best-ﬁrst graph search algorithms such as A\* often prevent them from solving large problems. The best-known approach for coping with this issue is iterative deepening, which performs a series of bounded depth-ﬁrst searches. Unfortunately, iterative deepening only performs well when successive cost bounds visit a geometrically increasing number of nodes. While it happens to work acceptably for the classic sliding tile puzzle, IDA\* fails for many other domains. In this paper, we present an algorithm that adaptively chooses appropriate cost bounds on-line during search. During each iteration, it learns a model of the search tree that helps it to predict the bound to use next. Our search tree model has three main beneﬁts over previous approaches: 1) it will work in domains with real-valued heuristic estimates,2) it can be trained on-line, and 3) it is able to make predictions with only a small number of training examples. We demonstrate the power of our improved model by using it to control an iterative-deepening A\*search on-line. While our technique has more overhead than previous methods for controlling iterative-deepening A\*, it can give more robust-performance by using its experience to accurately double the amount of search eﬀort between iterations.

**Introduction to the Problem Structure**

Each link in the tree has a **weight that reflects the distance between a pair of divisions**. Navigation is done by moving from the current position in the map (say the robot is at division B) to other map locations (say D) in order to service customer orders.

**Part A**

In this part we are concerned with minimizing the amount of inter-divisional movement. A customer always orders items from only **one** division in the company. Answer the following two questions. Note that the answers to the questions do not need any programming effort.

Q1) From the map above what data structure needs to be derived to support the minimization of movement between divisions that is required to support consecutive orders that involve different divisions?

Q2) How will we populate the data structure that you proposed in Q1 above? Outline the procedure involved. You only need to describe the procedure, not implement it at this stage.

**Part B**

In Part A above we only considered movement to divisions. In this part we will consider the effort involved in servicing a customer order by moving to the divisions required as well as navigating to the shelf location that contains the item ordered.

Customer orders are generated by a 3-step process. First, a random number is generated that specifies the division number that contains the items ordered. This will be thus a random number in the range 1 to 15. Next, a random number is generated that represents the number of items ordered k. We will assume that no customer orders more than 3 items. In the 3rd step, k random numbers, each in the range 1 to 63, are generated that represent the shelf numbers where the items are located. For example, a customer may have ordered two different items at shelf locations 17 and 61 respectively. Shelves across all divisions have the same numbering scheme, which is in the range 1 to 63.

**Algorithm**

Depth first search is incomplete if there is an infinite branch in the search tree.

Infinite branches can happen if:

* paths contain loops
* infinite number of states and/or operators.

For problems with infinite (or just very large) state spaces, several variants of depth-first search have been developed:

* depth limited search
* iterative deepening search

Depth limited search (DLS) is a form of depth-first search.

It expands the search tree depth-first up to a maximum depth *l*

The nodes at depth *l* are treated as if they had no successors

If the search reaches a node at depth *l where the path is not a solution, we backtrack to the next choice point at depth < l*

Depth-first search can be viewed as a special case of DLS with *l* = ∞

The depth bound can sometimes be chosen based on knowledge of the problem

For e.g., in the route planning problem, the longest route has length s - 1, where s is the number of cities (states), so we can set *l =*s - 1

For the most problems, d is unknown.

Iterative deepening (depth-first) search (IDS) is a form of depth limited search which progressively increases the bound.

It first tries *l = 1,*then *l*= 2, then *l* = 3, etc. until a solution is found

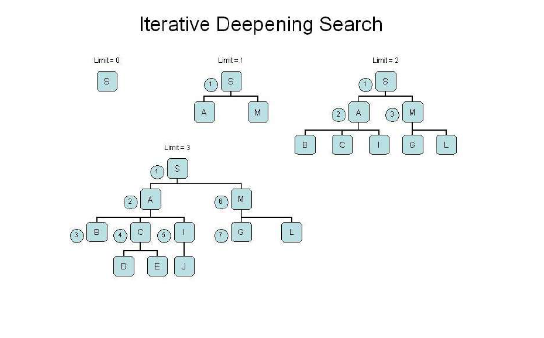
Solution will be found when *l = d*

IDDFS combines depth-first search’s space-efficiency and breadth-first search’s fast search (for nodes closer to root).

IDDFS calls DFS for different depths starting from an initial value. In every call, DFS is restricted from going beyond given depth. So basically we do DFS in a BFS fashion.

The example of Iterative-deepening depth-first search as given below, with the current depth-limit (*l)*starting at 1 and incrementing each time:





[dpthegrey](https://medium.com/@dpthegrey?source=post_sidebar--------------------------post_sidebar-----------)

Computer Engineer working as a DevOps Engineer. AI Researcher. Builder. Creator. Developer. Influencer. Innovator. Learner. Motivator. Writer. Human.

Follow

**Results**

Input into the Algorithm can be transalted as

11 19

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 1

1 2 1 1 2 1 2 1 1 2 1 2 1 1 1 1 1 2 1

1 2 1 4 2 1 2 2 1 2 2 2 1 4 1 2 2 2 1

1 2 1 1 1 1 1 2 1 1 1 2 1 2 1 2 1 1 1

1 2 2 1 2 2 1 3 2 2 1 2 2 2 1 2 1 4 1

1 1 2 2 2 1 1 2 1 2 1 1 1 1 1 2 2 2 1

1 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 1 1

1 2 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1

1 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2 4

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

-The following is the output of the system

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 2 2 2 2 1 5 5 5 5 1 2 2 2 2 2 2 2 1

1 2 1 1 2 1 5 1 1 5 1 2 1 1 1 1 1 2 1

1 2 1 4 2 1 5 5 1 5 5 5 1 4 1 2 2 2 1

1 2 1 1 1 1 1 5 1 1 1 5 1 5 1 2 1 1 1

1 2 2 1 2 2 1 3 2 2 1 5 5 5 1 2 1 4 1

1 1 2 2 2 1 1 2 1 2 1 1 1 1 1 2 2 2 1

1 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 1 1

1 2 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1

1 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2 4

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

---

length=17

time=157

**Conclusion**

References

1. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions of Systems Science and Cybernet-ics SSC-4(2), 100–107 (July 1968)

2. Haslum, P., Botea, A., Helmert, M., Bonte, B., Koenig, S.: Domain-independent construction of pattern database heuristics for cost-optimal planning. In: Proceed-ings of the Twenty-Second Conference on Artiﬁcial Intelligence (AAAI-07) (Jul2007)

3. Korf, R.E.: Iterative-deepening-A\*: An optimal admissible tree search. In: Proceedings of the International Joint Conference on Artiﬁcial Intelligence (IJCAI-85). pp.1034–1036 (1985)

4. Korf, R.E., Reid, M., Edelkamp, S.: Time complexity of iterative-deepening-A\*.Artiﬁcial Intelligence 129, 199–218 (2001)

5. Korf, R.E., Zhang, W., Thayer, I., Hohwald, H.: Frontier search. Journal of the ACM 52(5), 715–748 (2005)

6. M´er˜o, L.: A heuristic search algorithm with modiﬁable estimate. Artiﬁcial Intelligence pp. 13–27 (1984)7. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving.Addison-Wesley (1984)8. Rose, K., Burns, E., Ruml, W.: Best-ﬁrst search for bounded-depth trees. In: The