ST. FRANCIS XAVIER UNIVERSITY

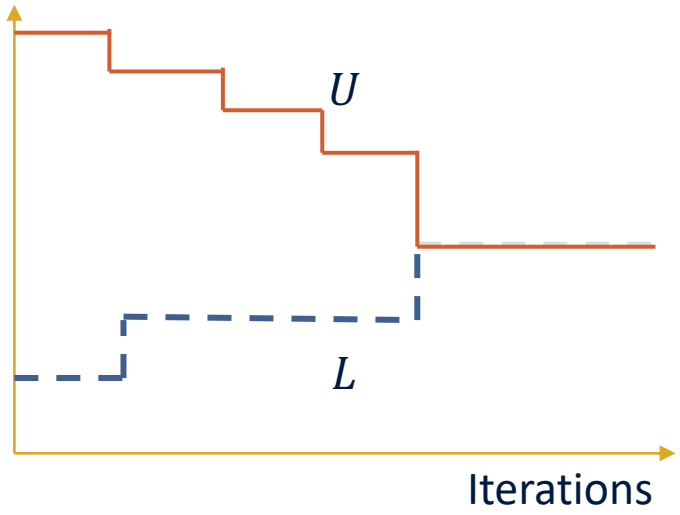
CSCI-564
CONSTRAINT PROCESSING AND HEURISTIC SEARCH

LECTURE 11 – LINEAR SPACE SEARCH (CONTINUED)

Dr. Jean-Alexis Delamer

Recap

- When the problem is large, we don't want (or can't) to maintain the entire graph in memory.
- We changed the approach from a graph search to a tree search.
- We can use a branch and bound approach:
  - It avoids exploring branch that are worst that our current solution.



Depth-first iterative deepening

- Explore the tree depth by depth.
- It maintain an upper bound to know which branch need to be explored at each iteration.
- The first solution returned is not always optimal.
- However, it doesn't use a heuristic to estimate the potential of a branch.

How is it impacting the algorithm?

Iterative-Deepening A*

- Extension of DFID including a heuristic estimate.
- First alternative when the memory constraint doesn't allow the use of A*.

- The algorithm is:
  - Most efficient when the problem is represented as a tree
  - Linear in space
  - No need to check for duplicates

IDA*

- Reminder on notations:
  - $w(u, v)$ is the weight of the edge (u, v).
  - $h(u)$ heuristic estimate
  - $g(u)$ cost from s to u
  - $f(u)$ combined cost

IDA*

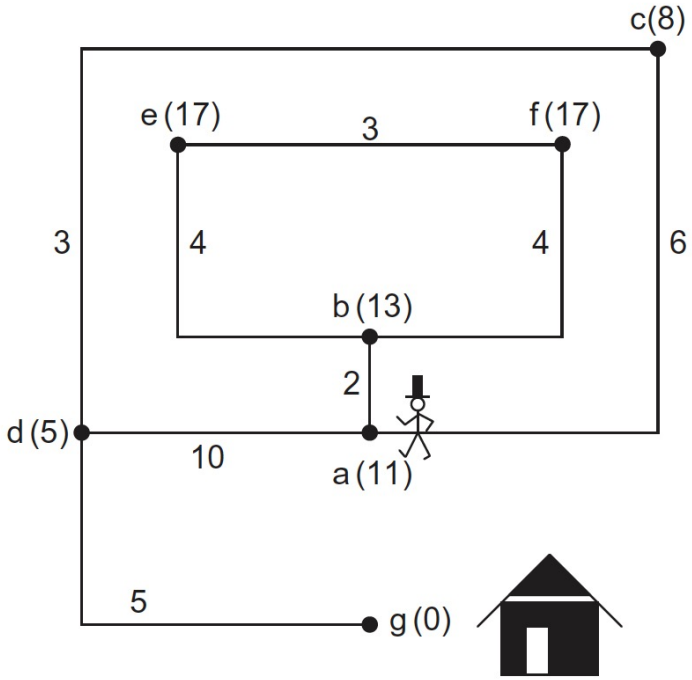
- How it's working?
  - During a depth-first search only nodes that have an f-value inferior or equal to U are expanded.
    - In DFID we checked only based on g.
  - The upper bound of the current iteration U', maintain the lowest f-value that is superior to U.
    - It ensure that one new node will be explored at the next iteration.
- The algorithm is optimal.
  - Why?

IDA*

```
Function IDA*(u, g, U):
        if (Goal(u))
                return path(u)
        Succ(u) ← Expand(u)
        foreach v in Succ(u)
                if (g + w(u,v) + h(v) > U)
                        if g + w(u,v) + h(v) < U'
                                U' ← g + w(u,v) + h(v)
                else
                        p ← IDA*(v, g + w(u,v) + h(v), U)
                        if p ≠ ∅ return (u,p)
End Function

U' ← h(s)
bestPath ← ∅;
While(bestPath=∅ and U' ≠ ∞)
        U ← U'
        U' ← ∞
        bestPath ← IDA*(s, 0, U)
return bestPath;
```

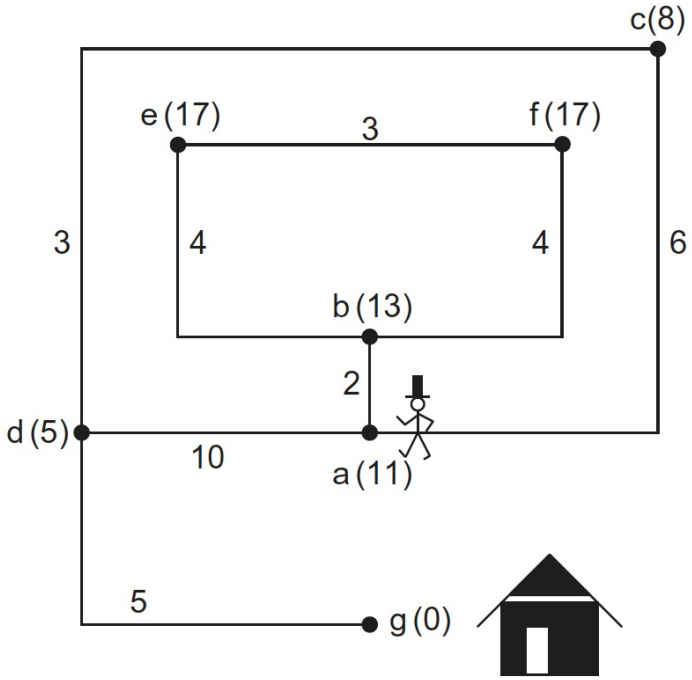
IDA*

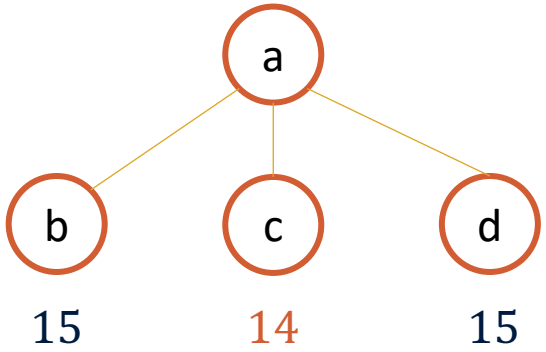


a

Iteration: 1
U:11
U':∞

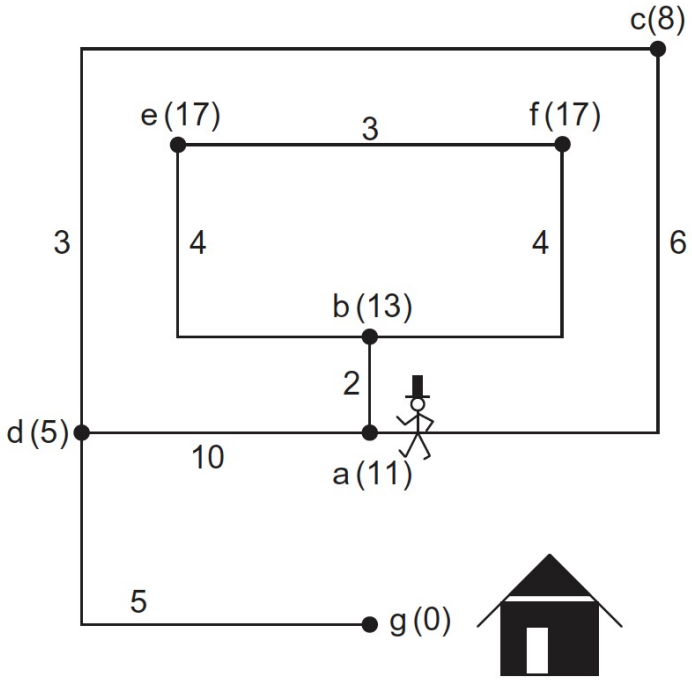
IDA*





Iteration: 1
U:11
U':14

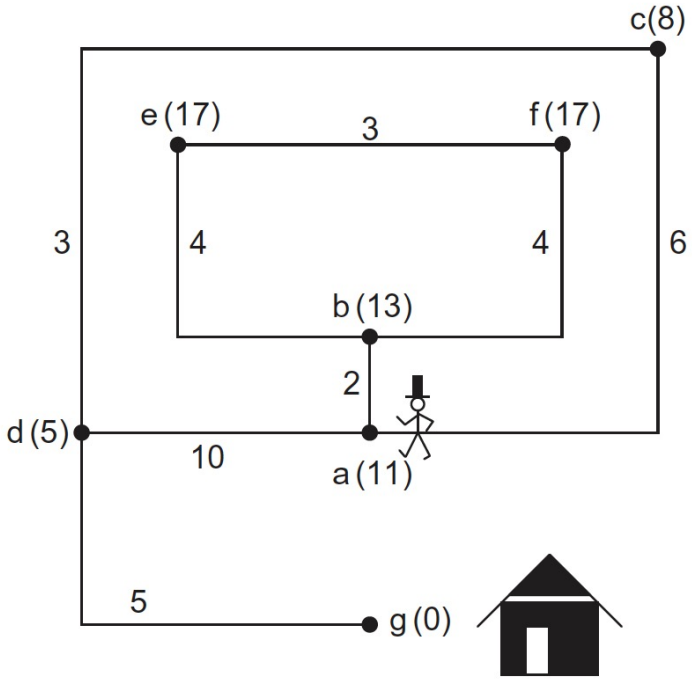
IDA*

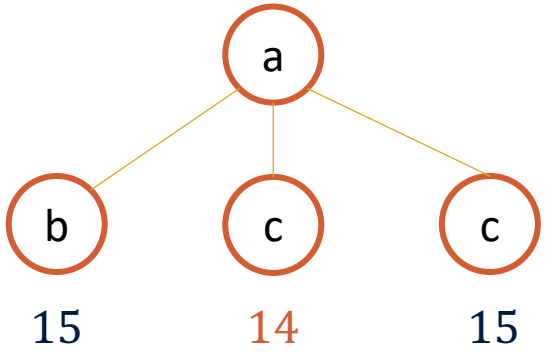


a

Iteration: 2
U:14
U':∞

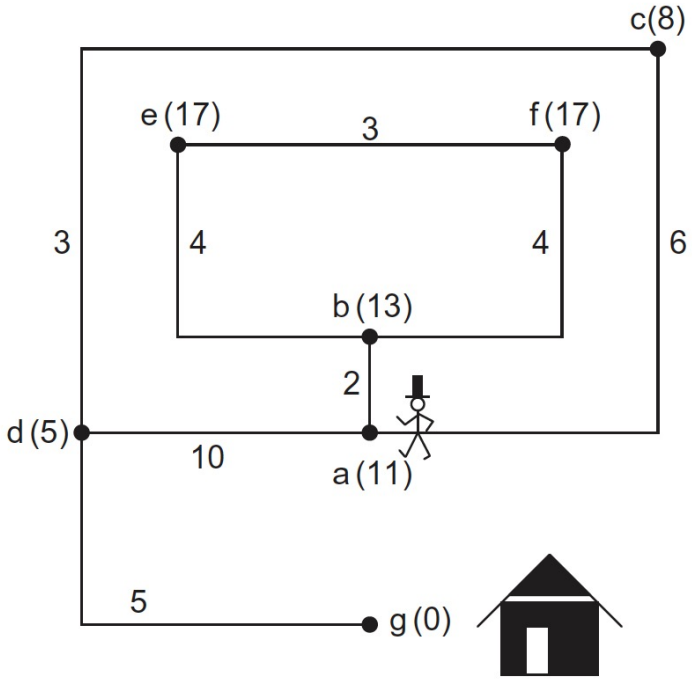
IDA*

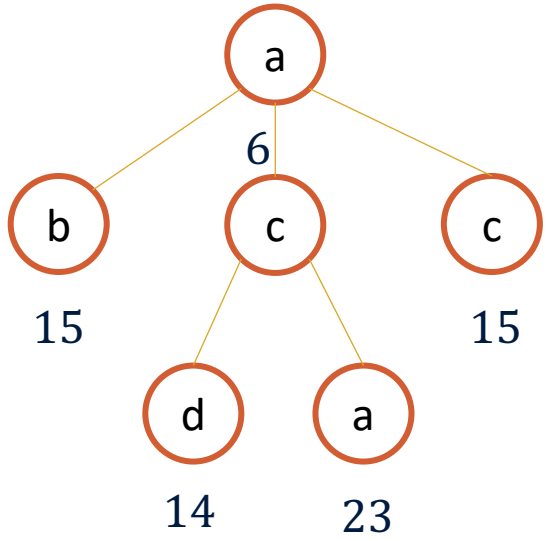




Iteration: 2
U:14
U':15

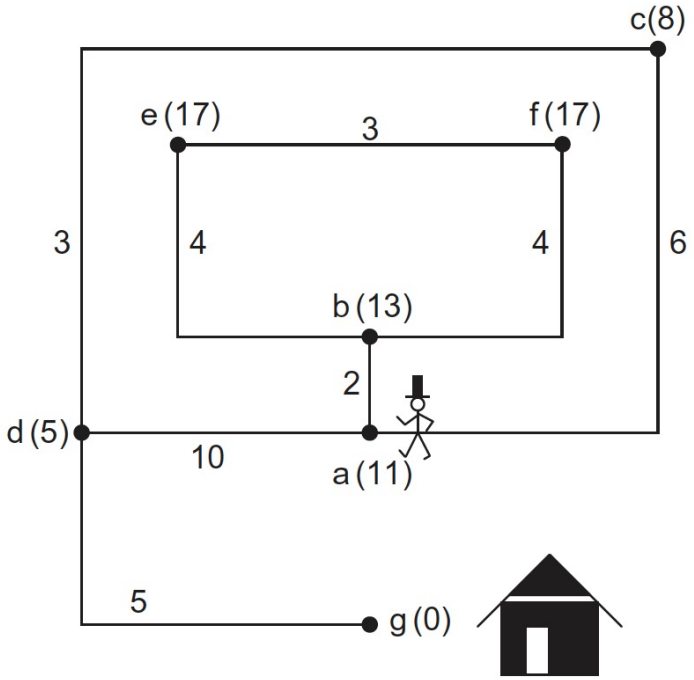
IDA*

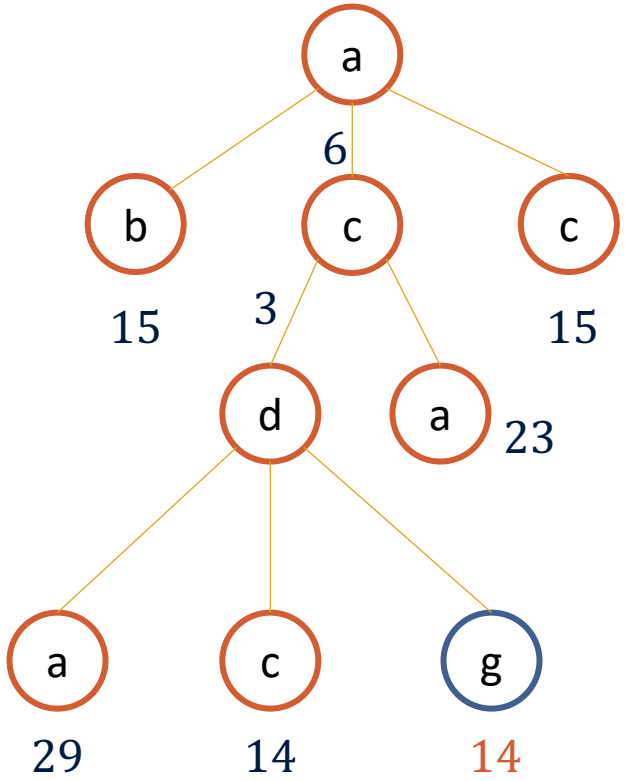




Iteration: 2
U:14
U':15

IDA*





Iteration: 2
U:14
U':15

IDA*

- Using the heuristic drastically reduces the search effort:
  - DFID: 57 steps for the example
  - IDA*: 7 steps for the example

- Remember that the heuristic needs to give enough information.
  - Limited when the heuristic have few distinct values.

- If no solution exists, the algorithm will not terminate.

IDA*

- Theorem:
  - Algorithm IDA* for graphs with admissible weight function is optimal.

- To prove it, we need to show that IDA* is ordered.
  - We will do a proof by induction over k iterations
  - E_k the set of newly encountered path
  - R_k set of generated, but not expanded paths

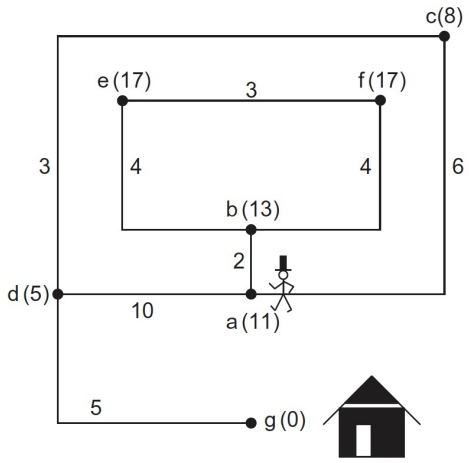
IDA*

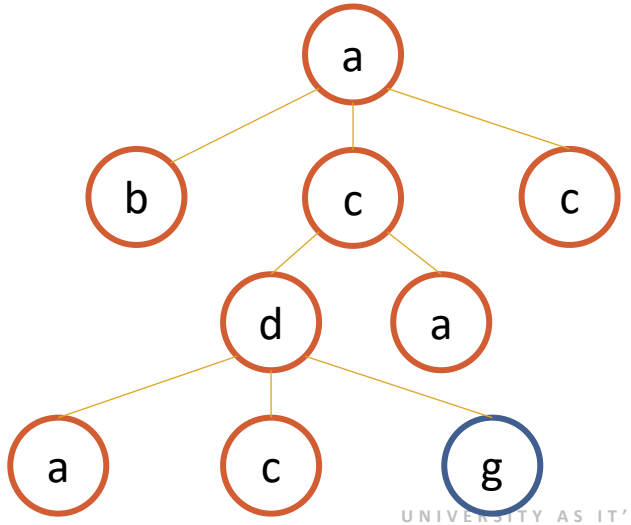
- Proof by induction:
  - Base case $k = 1$:
    - $\forall p \in E_1, w_{\max}(p) = U_1$
    - $\forall q \in R_1, w_{\max}(q) > U_1$
  - We admit that it's true for k
    - $\forall p \in E_k, w_{\max}(p) = U_k$
    - $\forall q \in R_k, w_{\max}(q) > U_k$
  - We show that it's true for $k + 1$:
    - $U_{k+1} = \min_{q \in R_k} \{w_{\max}(q)\}$
    - $\forall p \in E_{k+1}, w_{\max}(p) = U_{k+1}$
      - The contrary contradicts the monotonicity of $w_{\max}$, since only paths p with $w_{\max}(p) \leq U_{k+1}$ are newly expanded
    - Therefore, $\forall p \in E_{k+1}$, the condition $U_k < w_{\max}(p) = U_{k+1}$
  - Hence IDA* is ordered ■

IDA*

- The algorithms works very well with search trees.
  - No need to manage duplicates

- What happen, if we have a search graph?
  - The number of path is exponentially larger than the number of nodes.
  - Creates duplicates





Space complexity

- The asymptotic space complexity is the maximum depth of the recursion stack .
- The optimal solution cost is c, and the minimum edge cost is e.
- The maximum length of any path with total cost less than or equal c is c/e.
- The maximum search depth is $c/e + 1$.
- Since e is a constant, the asymptotic space complexity of IDA* is $O(c)$.

Time complexity

- The number of nodes generated by an iteration of IDA* with the cost threshold U is :

$$IDA^*(U) = \sum_{i=1}^{U} N(U)$$

- If $N(U)$ grows exponentially with U, with branching factor b then :

$$\frac{IDA^*(U)}{IDA^*(U-1)} = b$$

- This means that in each iteration the number of nodes developed also grows exponentially with the branching factor b.
- So according to what we saw the asymptotic time complexity of IDA* is the same as that of A*.

Recursive Best-First Search

- RBFS improve the time complexity of IDA*.
- It's a recursive algorithm.
- It expands nodes in best-first order and backing up heuristics values.
  - Even when the cost function is nonmonotone.
- RBFS use only a local upper bound for each recursive call.
- RBFS stores the nodes on the current search path and their siblings.
  - Consume slightly more memory than IDA*.

Recursive Best-First Search

- The idea:
  - With an admissible heuristic, the backed-up heuristic can only increase.
  - When exploring the children, the child with the lowest f-value should be explored.
    - Until the it exceed the f-value of the second-best child.

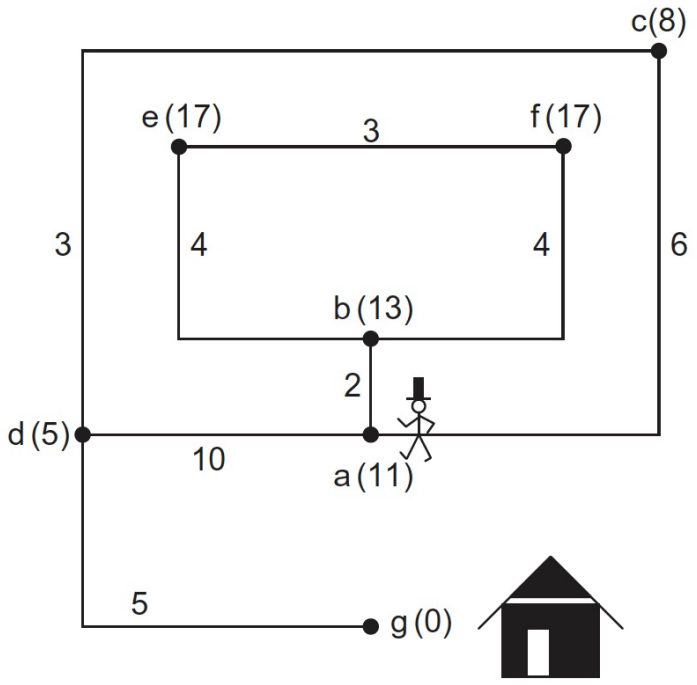
Recursive Best-First Search

```
Function RBFS(u, U):
        if (f(u) > U)              return f(u)
        if (Goal(u))               exit with Path(u)
        Succ(u) ← Expand(u)
        if (Succ(u) = ∅)           return ∞
        else if (|Succ(u)| = 1)    return RBFS(v_0, U)
        else
                foreach v in Succ(u)
                        backup(v) = max{f(u) + w(u, v), backup(u)}
                Let {v_0, ..., v_n} be Succ(u), sorted according to backup
                while (backup(v_0) < U)
                        backup(v_0) ← RBFS(v_0, min{U, f(v_1)})
                        Let {v_0, ..., v_n} be Succ(u), resorted according to backup
                return backup(v_0)
End Function

RBFS(s, ∞)
```

Recursive Best-First Search

- Exercise:



Linear space search

- Divide-and-Conquer methods can find optimal solutions with a memory consumption that is only logarithmic in the number of states.
  - These search methods are impractical due to their large runtime
- Depth-first search has a memory consumption that is linear.
  - It stores only the path from the root to the states that it currently expands.
  - It has a reasonable runtime.
- Branch and bound methods allow us to reduce the search effort.
- IDA* and RBFS return only the optimal solution.

Linear space search

- IDA* and RBFS has a low runtime only if each depth-first search expands several nodes.
- They work best for problems that have a search tree.
  - We can mitigate the issue by not expanding any nodes already in the path.