Al-Mustaqbal University
College of Science

جامـــــعة المـــــستـقبـل
AL MUSTAQBAL UNIVERSITY

كلية العلـوم

قســـم علوم الذكاء الاصطناعي

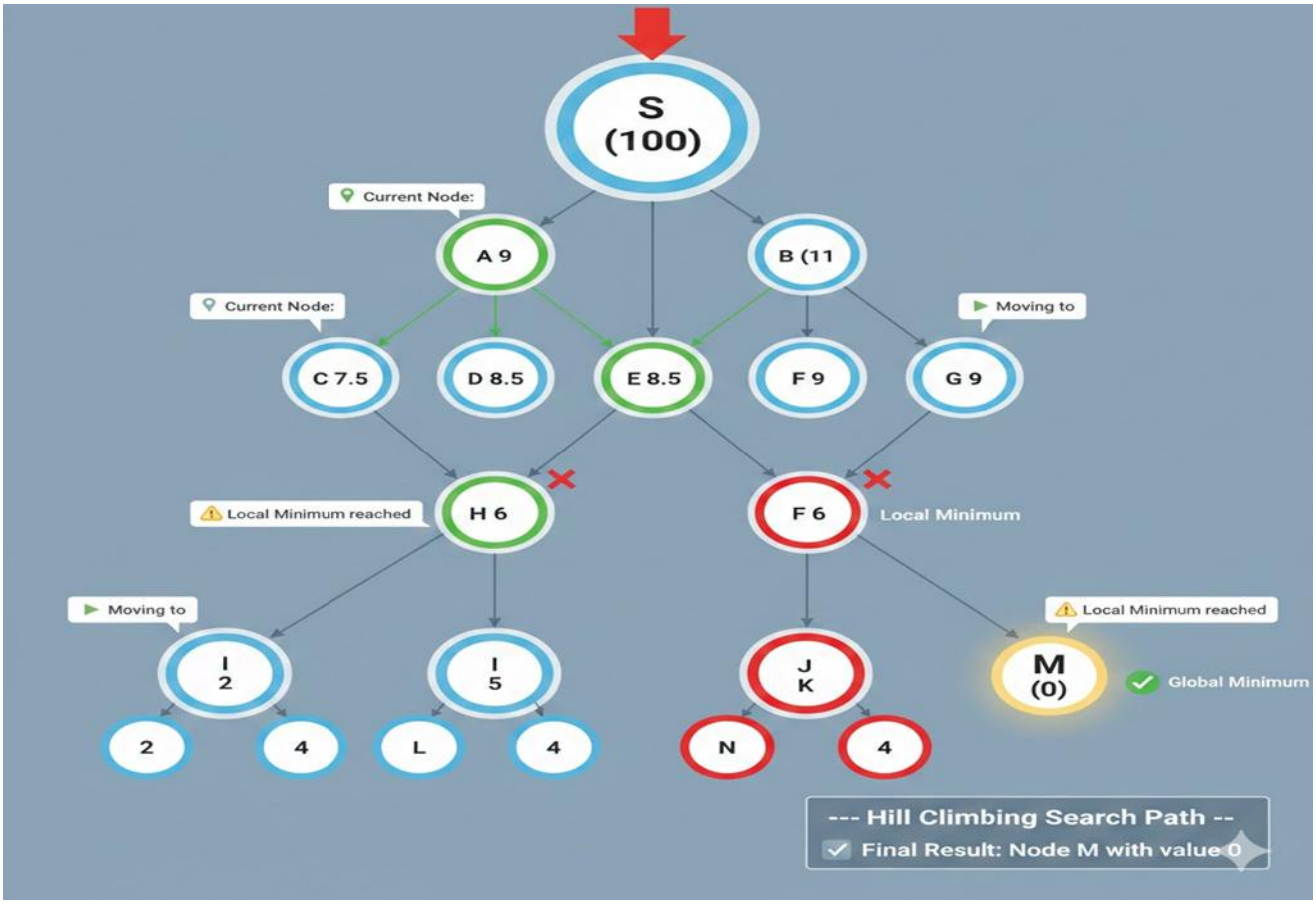
Lab 1-2-Heuristic-Search-Methods

Stage:Two
Lectures:
Msc.Hadi Salah Hadi
Programmer.Noor Hasan Obaid

Study Year: 2025-2026

Hillclimbing search algorithm

The Hill Climbing algorithm is one of the simplest local search algorithms in artificial intelligence. It is used to find the "best solution" from a set of available solutions.



Explaining the concept through a "tree" analysis

(Based on the attached code, the algorithm works as a minimization algorithm, searching for the smallest value). **Here's how the algorithm moves through the tree:**

1-Root: It starts from node **S** with a value of **100**.

2-Comparison: Node **S** looks at its immediate children **(A=9, B=11).**

3-Decision: Since **9 < 11** and less than **100**, the algorithm moves to **A**.

4-Continuation: At A, it looks at the children **(C=7.5, D=8.5, E=8)** and chooses the smallest,

which is **C (7.5)**.

Reaching the target: It continues in this way until it reaches node **M**, which has a value of **0**, the lowest .5 possible value (Global Minimum).

Code Explanation Step-by-Step

The code relies on recursion, and **here's how it works:**

Data Structure (Class Node): This defines each "point" in the tree, storing its name, value, and a list of its associated children.

Algorithm Logic (hill_climbing_minimization):

Finding the Best: The line `**min(current_node.children, key=lambda x: x.value)**` checks all available options under the current node and selects the lowest value.

Optimization Condition: `if **best_child.value < current_node.value**`: This is the core of the algorithm. If a node finds a child with a lower value, it moves to that child.

Local Minimum: If all neighboring nodes (children) have a value higher than the current node, the algorithm stops and declares that it has reached the Local Minimum, the best local solution, even if there is a better solution elsewhere in the tree.

The path to the final solution in your example

Based on the code you provided, the path would be:

S(100) ➔ A(9) ➔ C(7.5) ➔ I(5) ➔ M(0)

Code:

```
#تعريف العقدة
class Node:
    #name: اسم العقدة
    #value: القيمة الرقمية المرتبطة بها (التي نريد تقليلها)
    #children: قائمة بالعقد المرتبطة بها (الأماكن التي يمكنك الذهاب إليها من هذه النقطة).
    def __init__(self, name, value):
        self.name = name
        self.value = value
        self.children = []

    #  تعريف دالة داخل الكلاس، تستقبل قائمة (List) من العقد الأخرى وتسميها nodes.
    def add_children(self, nodes):
        #.extend(nodes): هي دالة في لغة Python تُستخدم لإضافة عدة عناصر مرة واحدة إلى القائمة
        self.children.extend(nodes)

def hill_climbing_minimization(current_node):
    print(f"  Current Node: {current_node.name} (Value: {current_node.value})")
#التحقق من نقطة النهاية (Leaf Node)
#تعمل الدالة بشكل تكراري (Recursive)
    if not current_node.children:
        print("  Reached a leaf node. Search complete.")
        return current_node

    # ان كان لم يكن هناك جار افضل حتى لو كان هنالك ابناء وقيمتهم جميعا اعلى من القيمة الحالية يتوقف الكود ويعلن انه وصل للنقطة الصغرى المحلية
    # إذا كان الجار أفضل (قيمته أقل): ينتقل الكود إليه فوراً ويكرر العملية
    #لايجاد اقل قيمة minدالة
    # Find the child with the lowest value
    best_child = min(current_node.children, key=lambda x: x.value)
    #فيعني اننا نتحسن للاسفل  قيمة الابن الي اخترناه اصغر من قيمة العقدة التي نقف عليها
    #  يقوم البرنامج بطباعة رسالة توكد الانتقال ثم استدعاء الدالة مرة اخرى لتبدا نفس العملية من العقدة الجديدة وتسمى العملية عملية استدعاء الداتي
# Check if the best child is an improvement (lower value)
    if best_child.value < current_node.value:
        print(f"  Moving to {best_child.name} (Value {best_child.value} is lower than {current_node.value})")
        return hill_climbing_minimization(best_child)
    #إذا لم نجد أي ابن قيمته أقل من قيمتنا الحالية، فهذا يعني أننا وصلنا لأقل نقطة ممكنة
    else:
        print(f"  Local Minimum reached at {current_node.name}. No child has a lower value.")
        return current_node
```

```
# --- Building the Tree from your Image ---
# Root
S = Node("S", 100) # Assuming a high starting value for S to initiate movement
#تعريف كائنات (Objects) من نوع Node.
#الوسيط الأول ("A" أو "B"): هو اسم العقدة للتعريف بها عند الطباعة.
#الوسيط الثاني (9 أو 11): هو القيمة (Heuristic Value) التي سيعتمد عليها الكود للمقارنة
#S.add_children([A, B])يخبر البرنامج أن العقدتين A و B هما "أبناء" للعقدة S.
# Level 1
A = Node("A", 9)
B = Node("B", 11)
S.add_children([A, B])
```

```
# Level 2 (under A)
C = Node("C", 7.5)
D = Node("D", 8.5)
E = Node("E", 8)
A.add_children([C, D, E])
```

```
# Level 2 (under B)
F = Node("F", 9)
G = Node("G", 9)
B.add_children([F, G])
```

```
# Level 3 (under C)
H = Node("H", 6)
I = Node("I", 5)
C.add_children([H, I])
```

```
# Level 3 (under F)
J = Node("J", 7)
K = Node("K", 6)
F.add_children([J, K])
```

```
# Level 4 (under I)
L = Node("L", 2)
M = Node("M", 0)
I.add_children([L, M])
```

```
# Level 4 (under J)
N = Node("N", 4)
O = Node("O", 4)
J.add_children([N, O])
```

```
# Execution
print("--- Hill Climbing Search Path ---")
result = hill_climbing_minimization(S)
print(f"\n✅ Final Result: Node {result.name} with value {result.value}")
```

Output:

--- Hill Climbing Search Path ---

Current Node: S (Value: 100)

Moving to A (Value 9 is lower than 100)

Current Node: A (Value: 9)

Moving to C (Value 7.5 is lower than 9)

Current Node: C (Value: 7.5)

Moving to I (Value 5 is lower than 7.5)

Current Node: I (Value: 5)

Moving to M (Value 0 is lower than 5)

Current Node: M (Value: 0)

Reached a leaf node. Search complete.

✓ Final Result: Node M with value 0