CSL020U4E: Artificial Intelligence
Lecture 03 *(Heuristic Search)*

Sumit Kumar Pandey

Agust, 2025

Heuristic Search

- The algorithms introduced in the last lecture were **blind**, or **uninformed**, taking no cognizance at all of the actual problem instance to be solved and behaving in the same manner wherever the goal might be.
- In this lecture, we look at approaches to **informed search**.
- In this lecture we introduce the idea of **heuristic search**, which uses domain specific knowledge to guide exploration.
- This is done by devising a **heuristic function** that **estimates** the distance to the goal for each candidate in *OPEN*.
- When heuristic functions are not very accurate, search complexity is still exponential, as revealed by experiments.
- We then investigate local search methods that do not maintain an *OPEN* list, and study gradient based methods to optimize the heuristic value.

Heuristic Functions

- With every node N in our search space, we associate a function that estimates the distance from N to the goal node.
- Traditionally, we denote the heuristic function as $h(n)$ or $h(N)$.
- To begin with, we assign the responsibility of defining this function to the user. This is the third domain function we expect from the user, in addition to the MoveGen and GoalTest functions.

Heuristic Functions for Some Problems

Map Colouring:

- The heuristic function could be the number of colours already used up in a partial assignment of colours to regions.
- The idea is that the fewer the colours used so far, the more the likelihood of extending the partial assignment to other regions and completing the task.

SAT:

- For a candidate solution, the value of $h(n)$ can be the number of clauses satisfied.
- One simple way to convert this into such a function is to redefine it as follows:

$$h(n) = \text{Total clauses} - \text{the number of clauses satisfied.}$$

Heuristic Functions for Some Problems

<u>8-Puzzle:</u>

- We can consider two heuristic functions - $h_{Hamming}(n)$ and $h_{Manhattan}(n)$.
    1. $h_{Hamming}(n) =$ The number of tiles out of place.
    2. $h_{Manhattan}(n) =$ This adds up the Manhattan distances for each from its destination.

Heuristic Functions for Some Problems

Route Finding:

- Exercise!!

Travelling Salesman Problem

- Exercise!!

Best First Search

- Heuristic search is search guided by a heuristic function.
- Every node has an estimate of goodness, which most of the time means closeness to the goal, expressed as a distance function.
- We assume that the user supplies this heuristic function $h(n)$ along with the MoveGen and GoalTest functions.
- When invoked, $h(n)$ returns the heuristic value for that node, often referred to as the h-value.
- We augment the representation of *nodePair*, the node in the search tree, to include the heuristic value.

Best First Search

```
BestFirstSearch($S$)
  $OPEN \leftarrow (S, \textbf{null}, \textbf{h}(S)) : [\,]$
  $CLOSED \leftarrow [\,]$
  while $OPEN$ is not empty
    $nodePair \leftarrow$ head $OPEN$
    $(N, \_, \_) \leftarrow nodePair$
    if GoalTest($N$) $=$ TRUE
      return ReconstructPath($nodePair$, $CLOSED$)
    else $CLOSED \leftarrow nodePair : CLOSED$
      $children \leftarrow$ MoveGen($N$)
      $newNodes \leftarrow$ RemoveSeen($children$, $OPEN$, $CLOSED$)
      $newPairs \leftarrow$ MakePairs($newNodes$, $N$)
      $OPEN \leftarrow \textsc{Sort}_h(newPairs \mathbin{++}$ tail $OPEN)$
  return empty list
```

Performance of BESTFIRSTSEARCH

Completeness:

- It is complete for finite graphs.
- When the graph is infinite, we cannot make the claim of completeness.

Quality of Solution:

- The path found by BESTFIRSTSEARCH may not be optimal.
- When edge or move costs are equal, BESTFIRSTSEARCH can still find non-optimal paths.

Performance of BESTFIRSTSEARCH

Space Complexity:

- With a good heuristic function, BESTFIRSTSEARCH has characteristics similar to DFS.
- So, we might hope that the space complexity might be linear. But it is possible that the algorithm may change its mind and sprout new branches in the search tree.
- This goes against the grain of linear space, and it has been empirically observed that the space required is often exponential.

Time Complexity:

- If the heuristic function were to be perfect, it would drive the search directly towards the goal. Then the time complexity would be linear.
- However, in practice, this is not seen to be the case and, more often than not, the time complexity is exponential.

Local Search Method

Local Search Method

- We can prune the search space by not retaining in *OPEN* the nodes generated earlier.
- We consider ony the new nodes for choosing the one to inspect.
- We can modify the algorithm as follows:

$$\mathit{OPEN} \leftarrow \textsc{Sort}_h(\mathit{newPairs})$$

instead of

$$\mathit{OPEN} \leftarrow \textsc{Sort}_h(\mathit{newPairs} \mathbin{++} \textbf{tail}\ \mathit{OPEN})$$

- The first observation is that it commits to moving to one of the neighbours of the current node, the best one.
- Second, because we have not made any other change, and the filtering of nodes continues, the search will never turn back.
- It will terminate if it finds the goal node, or if no new neighbour can be generated.

Hill Climbing

- HillClimbing looks at the immediate neighbourhood and moves to the best neighbour *nextNode*, but only if it is better.
- Otherwise, it stops. At all times it keeps *bestNode*, the current best.

```
HillClimbing($S$)
  $bestNode \leftarrow S$
  $nextNode \leftarrow$ head Sort$_h$(MoveGen($bestNode$))
  while $h(nextNode)$ is better than $h(bestNode)$
    $bestNode \leftarrow nextNode$
    $nextNode \leftarrow$ head Sort$_h$(MoveGen($bestNode$))
  return $bestNode$
```

- Replacing Line 2 with the following would be more elegant:

$$nextNode \leftarrow \textsc{Best}(\textsc{MoveGen}(bestNode))$$

Performance of HILLCLIMIBING

Completeness:

- The algorithm terminates on all finite domains.
- It may also terminate on all infinite domains if the heuristic values are bounded. At some point, no neighbour will be better than the current node, and the algorithm may terminate.
- However, the algorithm may not find a solution node, or a path to the solution node, because it may halt at a local optimum.
- Hence, it is **not complete**.

Quality of Solution:

- Solution may not be found.

Performance of HillClimibing

Time Complexity:

- The algorithm is guided by the heuristic function to the nearest optimum. And then it stops.
- The time required is linear with depth, since it never looks back or sideways in its search.

Space Complexity:

- Since it does not maintain an *OPEN*, and only needs enough space to generate the neighbours, the space needed is constant.
- This is because the branching factor of the MoveGen function is constant or bounded.

Variable Neighbourhood Descent

- While the heuristic **defines** the terrain, the neighbourhood function dictates **how** we traverse the terrain.
- We may have at our disposal a set of neighbourhood functions with a varying number of neighbours.
- A **sparse** neighbourhood function has fewer choices at each point, while a dense function would have more choices. But the **denser** neighbourhoods are also more expensive to inspect.

Variable Neighbourhood Descent

- Neighbourhood functions that are sparse lead to quicker movement during search, because the algorithm must inspect fewer neighbours. But there is a greater probability of getting stuck on a local optimum.
- This probability becomes lower as neighbourhood functions become denser, but then search progress also slows down because the algorithm must inspect more neighbours before each move.
- Algorithm VARIABLENEIGHBOURHOODDESCENT tries to get the best of both worlds.
- It starts searching with a sparse neighbourhood function.
- When it reaches an optimum, it switches to a denser function.
- The hope is that most of the movement would be done in the earlier rounds, and that the overall time performance will be better.

Variable Neighbourhood Descent

- Algorithm VariableNeighbourhoodDescent begins with a sparse neighbourhood function and moves to a denser function on reaching an optimum.
- The algorithm assumes that the function MoveGen can be passed as a parameter.
- It assumes that there are *n* MoveGen functions sorted on the density of the neighbourhoods produced.

```
VariableNeighbourhoodDescent()
  node ← start
  for i ← 1 to n
    MoveGen ← MoveGen(i)
    node ← HillClimbing(node, MoveGen)
  return node
```

Beam Search

- HillClimbing keeps only the best neighbour.
- Algorithm BeamSearch keeps the "*b*" best options instead of just one, where *b* is known as the beam width.
- With memory becoming abundant, the algorithm has greater potential as beam widths can increase.

Exploration vs Exploitation

- A heuristic function guides search.
- It creates a landscape over which search follows the gradient.
- We say that the algorithm is **exploiting** the gradient.
- HILLCLIMBING and BEAMSEARCH do this faithfully and diligently.
- However, this fixation with gradient leads to the possibility of being stranded on the first optimum that comes along.
- This can be lethal for local search.

Exploration vs Exploitation

- Local search may need to stray a little from the path indicated by the steepest gradient.
- Or learn to go past a (local) optimum.
- For this, search needs another driver, **exploration**, the tendency to venture into newer areas.
- This is right up the alley of randomized algorithms, and we study them later in this course.

Tabu Search

- The algorithm described here derives from the Tongan word **tapu** or Fijian **tabu**, which roughly translates to ‘prohibited’ or ‘forbidden’.
- The algorithm allows search to proceed beyond a local optimum and prohibits it from going back to where it came from.
- The main idea in tabu search is to augment the **exploitative** strategy of heuristic search with an explorative tendency that continues to look for new areas in the search space.
- That is, tabu search follows the diktat of the heuristic function only as long as better choices present themselves.
- But when there are no better choices, instead of terminating as HILLCLIMBING would, it gives in to its **explorative** tendency to continue searching.

Tabu Search

- TabuSearch moves to the best allowed neighbour till some termination criterion.

```
TabuSearch(Start)
  N ← Start
  bestSeen ← N
  Until some termination criterion
    N ← Best(Allowed(MoveGen(N)))
    If N better than bestSeen
      bestSeen ← N
  return bestSeen
```

Iterated Hill Climbing (IHC)

- IHC does HillClimbing N times starting each time from N different randomly selected starting points.

```
IHC($N$)
  $bestNode$ ← random candidate solution
  repeat $N$ times
    $currentBest$ ← HillClimbing(new random candidate solution)
    if $h(currentBest)$ is better than $h(bestNode)$
      $bestNode$ ← $currentBest$
  return $bestNode$
```

Thank You