4. Continuous Space Search

Course: Introduction to AI
Instructor: Saumya Jetley
Teaching Assistant(s): Raghav Awasty & Subhrajit Roy

October 14, 2022

What is a continuous space?*

- Space with *uncountably* infinite states
- Connected path from one state to the next

What is a continuous space?*

(Background on problem spaces)

- Problem statements are associated with a *spaces of states* e.g geographical space or parameter space

- Discrete state spaces have discrete number of states (can be infinite)
- Continuous state spaces have an uncountably infinite number of states

What is a continuous space?*

- Is the 3D space we live in discrete or continuous?
- So, a maze (in a park in the real world) is a continuous space?

- NOTE: We can approximate a continuous space with a discrete space by binning parts of it.

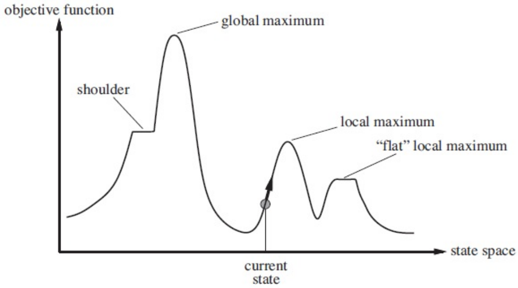
Search landscape

- We have a continuous manifold of states
- Keeping the terminology from before, we have:
  - Non-goal states
  - A goal state
- **How do we differentiate the goal state from non-goal states?**

Search landscape

- **How do we differentiate the goal state from non-goal states?**

- Let's say we put a score on states, given by a score function $f(x)$
- A landscape has both "location" (x) (defined by the state) and "elevation" ($f(x)$), and looks like below:



Goal of search

- **How do we differentiate the goal state from non-goal states?**

- If elevation/score corresponds to cost, then the goal would be to find the lowest point — a **global minimum**.
- If elevation/score corresponds to performance function, then the goal would be to find the highest peak—a **global maximum**.

Convex vs non-convex functions

- A *set of points* S *is convex* if the line joining any two points in S is also contained within S
- A *convex function* is one for which the space "above" it forms a *convex set*
- By definition, convex functions have no local (as opposed to global) minima

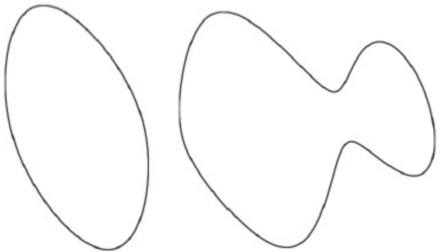


Figure: Example of a convex set (left) and a non-convex set (right)

Optimization in continuous spaces

To reach the goal state - let's say **Maximise an Objective Function** $f(x)$

The path to the goal is not important anymore - reaching the goal state is - in reasonable time

State function, not a path function

Different Approaches:

1. Solve as a discrete space search problem
   - Binning
2. Using derivative of $f(x)$ - Solve for $f'(x) = 0$
   - Gradient descent
   - Newton's Method

1. Discrete space search

- Perform binning
  - For each continuous variable in the state space, divide its domain into fixed number of equally spaced bins.
- Use standardised discrete search methods

Related discrete search algorithms

We could make use of search algorithms such as,

- Hill Climbing
- Simulated Annealing
- Local beam search
- Genetic algorithms

1.1. Hill climbing?

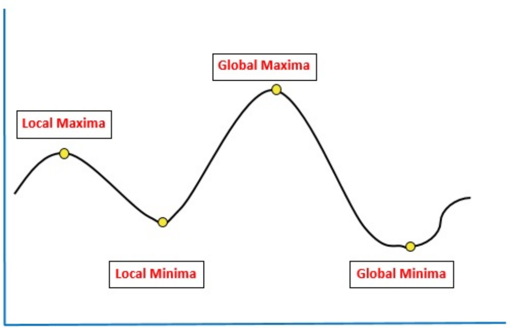
- The search algorithm (**steepest-ascent** version) is a loop that continually moves in the direction of steepest increasing value i.e. *fastest uphill*.
- Terminates when reaches a “peak” - no neighbor has higher value.

Algorithm 1 Algorithm for Hill climbing (Steepest Ascent)

$i \leftarrow$ initialstate
repeat
    $BestScore = f(i)$, $BestState = i$
    while $N(i)$ is non empty **do**
        generate $s \in N(i)$, remove s from $N(i)$
        if $f(s) > BestScore$ **then**
            $BestScore := f(s)$
            $BestState := s$
        end if
    end while
    $i := BestState$

State-space diagram of Hill Climbing

- **Local maxima**: Higher than its neighbours but not highest
- **Global maxima:** Here, the value of the objective function is highest
- **Current State:** Current State is the state where the agent is present



Key challenges for hill climbing

- **Greedy Approach:** The search moves in the direction of optimizing the cost i.e. finding local maxima/minima
- **No Backtracking:** It cannot remember the previous state of the system so backtracking to the previous state is not possible

  - Local Maxima: The algorithm terminates when the current node is a local maximum but there are better solutions.
  - Ridge: Imagine walking along the ridge of a mountain to reach the top - many local maxima along the way!
  - Plateau: Plateau is the region where all the neighbouring nodes have the same value - possibility of getting lost!

Performance stats for Hill Climbing

- Very useful in routing-related problems - Travelling Salesmen Problem, Job Scheduling, Chip Designing, Portfolio Management
- Good in solving the optimization problem using limited computation power

- 14% success rate with vanilla version
- 94% success rate when lateral moves are permitted*

- Extension to hill climbing
  - Stochastic hill climbing
  - First-choice hill climbing
  - Random restart hill climbing

1.1.a. Stochastic Hill Climbing

- **Stochastic hill climbing** chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move.
- Usually converges more slowly, but in some state landscapes, it finds better solutions.

Algorithm 2 Algorithm for Stochastic Hill climbing

1: $i \leftarrow initialstate$
2: **while** $t \leq maxIter$ **do**
3: Generate all $s \in N(i)$
4: Select s with probability

$$p(s) = \frac{1}{1 + e^{(score(i) - score(s))}}$$

5: $i \leftarrow s$ & $t \leftarrow t + 1$

Reference: A. Mazidi M. Fakhrahmad M. Sadreddini. (2016). A meta-heuristic approach to CVRP problem: local search optimization based on GA and ant colony. JACR.

1.1.b. First-Choice Hill Climbing

- **First-choice hill climbing** implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state.
- Choose the first randomly generated *uphill* move!
- This is a good strategy when a state has many (e.g., thousands) of successors.
- Greedy, incomplete, and sub-optimal

1.1.c. Random-restart Hill Climbing*

- It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.
- It is trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state.
- Stops each search at local maxima - or better yet, stops after a fixed amount of time.

1.2. Simulated annealing

- A hill-climbing algorithm that never makes “downhill” moves toward states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum
- Escapes local maxima by allowing bad moves (by gradually decreasing their size and frequency)
- **Simulated Annealing** combine hill climbing with a random walk in some way that yields both efficiency and completeness

1.2. Simulated annealing - Algorithm

Algorithm 3 Algorithm for Simulated annealing

$i \leftarrow initialstate$
while $t \leq maxIter$ **do**
  Generate $s \in N(i)$
  if $score(i) < score(s)$ **then**
    $i := s$
  else
    Select s with probability

$$p(s) = \frac{1}{1 + e^{\frac{(score(i) - score(s))}{T}}}$$

    $i \leftarrow s$ & $t \leftarrow t + 1$

1.3. Local Beam Search

- The **local beam search** algorithm keeps track of k states rather than just one.
- *Here, useful information is passed among the parallel search threads.* (main difference with random-restart search)

Algorithm 4 Algorithm for Local Beam Search

function: BEAM-SEARCH(problem, k) **return** a solution state with
k randomly generated states
repeat
  generate all successors of k states
  if any of them is a solution **then** return it
  else
    select the k best successors

1.3. Local Beam Search - Algorithm

Characteristics:

- Draws all threads to a common region of goodness
- Can suffer from lack of diversity (get concentrated in small region of state-space)
- A variant called **stochastic beam search**, analogous to stochastic hill climbing, chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value.

1.4. Genetic Algorithms

- A **genetic algorithm** (or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function) $\mapsto$ Higher $=$ better

1.4. Genetic Algorithms

- **Pros:**
  - Random exploration can find solutions that local search cannot (via crossover primarily)
  - Appealing connection to human evolution
- **Cons:**
  - Large number of "tunable" parameters
  - Difficult to replicate performance from one problem to another
  - Lack of good empirical studies comparing to simpler methods

1.4. Genetic Algorithm

Algorithm 5 Genetic Algorithm

function: GENETIC-ALGORITHM(population, fitness-function)
return an individual
repeat
  initialize with new population with $\emptyset$
  for i = 1 to size(population) **do**
    x = random-select(population, fitness-function)
    child = cross-over(x, y)
    mutate (child) with small random probability
    add child to new population
  end for
  population = new-population
until some individual is fit enough or enough time has elapsed **return** the best individual in population w.r.t. the fitness function =0

Discrete to Continuous

1. How to choose the factor for binning?
2. Combinatorially complex when state-space is multidimensional?

3. What happens when branching factor b is in thousands?
4. What happens when branching factor is infinite?

Does it serve us better to work in the continuous space or find a continuous space approximation?

2. Using the derivative

- Remember, when f'(x) = 0, the f(x) is
  - at a maximum
  - at a minimum
  - at a point of inflection
- Solve f'(x) = 0
- So really we have to find the roots of f'(x)
- Then, check each x in f(x) for a maximum (or check f"(x) for each x)
  - f"(x) = 0, then point of inflection
  - f"(x) = positive, then minimum
  - f"(x) = negative, then maximum

2. Using the derivative

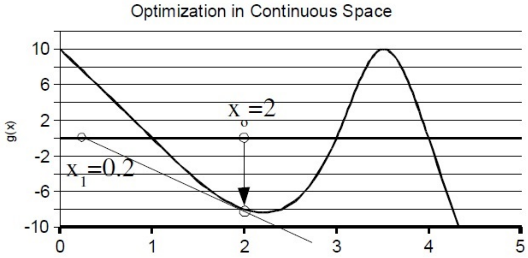
What if you can't solve f'(x) = 0 but you can still figure out the derivative at each point?

I. Use Newton's Method

II. Use Gradient descent, or

2.i. Newton's Method of Root Finding

- We are trying to find the root of some function g(x)
  - Remember, this is the derivative of the function that we want to optimise (g(x) = f'(x))
- Assume g(x) is linear, and
- You know the value of g(x) and g'(x) for your initial guess x_0



2.i. Newton's Method of Root Finding

Algorithm 6 Algorithm for Newton's root finding

Require: $g(x), g'(x)$

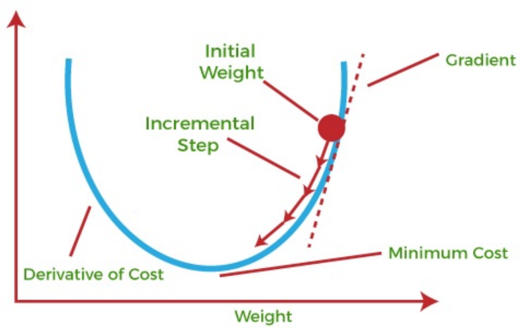
1: $x_0 \leftarrow initialguess$

2: **while** $(x_{i+1} - x_i) \geq \epsilon$ **do**

3: $\quad x_{i+1} = x_i - g(x_i)/g'(x_i)$

4: **end while**

2.ii. Gradient Descent (Ascent)



2.ii. Gradient Descent (Ascent)

- Pick an initial value for x_0
- Pick a step size (also called - learning rate lr)

$$x_{i+1} = x_i - lr\frac{d(f(x))}{dx}$$

- Step size too small (takes forever to reach maximum)
- Step size too large ("step over" the maximum)

- Repeat until the sign of f'(x) changes or f'(x) = 0

Gradient Descent: Practicalities

How to fix the step size?

- Variable step size scheme - I
  - Start with large step size and find range within which maximum lies
  - Repeat using smaller step size inside that range
  - Continue until you have enough precision
- Variable step size scheme - II
  - Start by using a small step size
  - As long as the sign of f'(x) doesn't change, double the step size
  - When sign changes, drop back to initial step size and continue search

References

- **Chapter 4: Artificial Intelligence, A Modern Approach (3rd Edition)** - Stuart Russell and Peter Norvig

Overview