**Chapter IX Analysis of Adaptive Algorithms**

**Introduction**

An **adaptive algorithm** changes its behaviour at the run time dynamically, based on data available and on *a priori* defined criterion. Such information may be the story of received data, information on the available resources, or other run-time acquired (or *a priori* known) information related to the environment in which it operates.

Among the most used adaptive algorithms, which represents a class of stochastic gradient-descent algorithms used in adaptive filtering and machine learning. In adaptive filtering the LMS is used to mimic a desired filter by finding the filter coefficients that relate to producing the least mean square of the error signal (difference between the desired and the actual signal).

For example, stable partition, without additional memory is *O*(*n* lg *n*) but given *O*(*n*) memory, it can be *O*(*n*) in time. As implemented by the C++ Standard Library, stable partition is adaptive and so it acquires as much memory as it can get (up to what it would need at most) and applies the algorithm using that available memory. Another example is adaptive sort, whose behaviour changes upon its input.

An example of an adaptive algorithm in radar systems is the constant false alarm rate detector.

In machine learning, many algorithms are optimized and adaptive or have adaptive variants, which usually means that the algorithm parameters are automatically adjusted according to statistics about the optimisation thus far (e.g. the rate of convergence). Examples include [adaptive simulated annealing](https://en.wikipedia.org/wiki/Adaptive_simulated_annealing), [adaptive coordinate descent](https://en.wikipedia.org/wiki/Adaptive_coordinate_descent), and [adaptive quadrature](https://en.wikipedia.org/wiki/Adaptive_quadrature).

In [data compression](https://en.wikipedia.org/wiki/Data_compression), adaptive coding algorithms such as Adaptive Huffman coding or Prediction by partial matching can take a stream of data as input, and adapt their compression technique based on the symbols that they have already encountered.

In [signal processing](https://en.wikipedia.org/wiki/Signal_processing), the Adaptive Transform Acoustic Coding  is called "adaptive" because the window length (the size of an audio "chunk") can change according to the nature of the sound being compressed, to try to achieve the best-sounding compression strategy.

In cybernetic feedback the neuro-signature of the conscious unit melds with the intelligence of computers. As intelligence decouples from consciousness, Artificial Intelligence evolves and such algorithms become known as adaptive algorithms to signify the origin of the source code. These are unique, distinctive and diverse conscious algorithms from organics that are capable of computing functions beyond the biological component.

**Breadth-First Search**

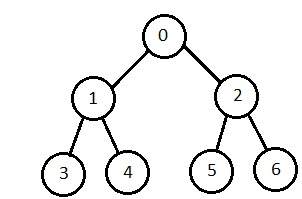
It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

If **branching factor** (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = bd.

The total no of nodes created in worst case is b + b2 + b3 + … + bd.

**Disadvantage** − since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.



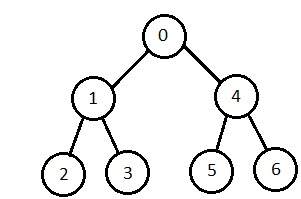
**Depth-First Search**

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor *b*and depth as *m*, the storage space is *bm.*

**Disadvantage** − This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is *d*, and if chosen cut-off is lesser than *d*, then this algorithm may fail. If chosen cut-off is more than *d*, then execution time increases.

Its complexity depends on the number of paths. It cannot check duplicate nodes.



**Bidirectional Search**

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

**Uniform Cost Search**

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

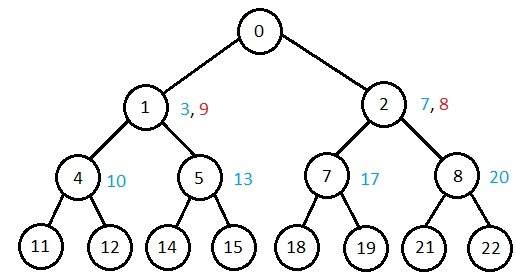
It explores paths in the increasing order of cost.

**Disadvantage** − There can be multiple long paths with the cost ≤ C\*. Uniform Cost search must explore them all.

**Iterative Deepening Depth-First Search**

It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found.

It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth *d*. The number of nodes created at depth *d* is bd and at depth *d-1* is bd-1.



**Comparison of Various Algorithms Complexities**

Let us see the performance of algorithms based on various criteria −

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterion** | **Breadth First** | **Depth First** | **Bidirectional** | **Uniform Cost** | **Interactive Deepening** |
| Time | bd | bm | bd/2 | bd | bd |
| Space | bd | bm | bd/2 | bd | bd |
| Optimality | Yes | No | Yes | Yes | Yes |
| Completeness | Yes | No | Yes | Yes | Yes |

**Informed (Heuristic) Search Strategies**

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

**Heuristic Evaluation Functions**

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

**Pure Heuristic Search**

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

**A \* Search**

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

* g(n) the cost (so far) to reach the node
* h(n) estimated cost to get from the node to the goal
* f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

**Greedy Best First Search**

It expands the node that is estimated to be closest to goal. It expands nodes based on f(n) = h(n). It is implemented using priority queue.

**Disadvantage** − It can get stuck in loops. It is not optimal.

**Local Search Algorithms**

They start from a prospective solution and then move to a neighboring solution. They can return a valid solution even if it is interrupted at any time before they end.

**Hill-Climbing Search**

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements.

function Hill-Climbing (problem), returns a state that is a local maximum.

inputs: problem, a problem

local variables: current, a node

neighbor, a node

current <-Make\_Node(Initial-State[problem])

loop

do neighbor <- a highest\_valued successor of *current*

if Value[neighbor] ≤ Value[current] then

return State[current]

current <- neighbor

end

**Disadvantage** − this algorithm is neither complete, nor optimal.

**Local Beam Search**

In this algorithm, it holds k number of states at any given time. At the start, these states are generated randomly. The successors of these k states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool. The pool is then sorted numerically. The highest k states are selected as new initial states. This process continues until a maximum value is reached.

function Beam Search(problem*, k*), returns a solution state.

start with k randomly generated states

loop

generate all successors of all k states

if any of the states = solution, then return the state

else select the k best successors

end

**Simulated Annealing**

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable.

We initially set the temperature high and then allow it to ‘cool' slowly as the algorithm proceeds. When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

Start

* Initialize k = 0; L = integer number of variables;
* From i → j, search the performance difference Δ.
* If Δ <= 0 then accept else if exp(-Δ/T(k)) > random(0,1) then accept;
* Repeat steps 1 and 2 for L(k) steps.
* k = k + 1;

Repeat steps 1 through 4 till the criteria is met.

End

**Travelling Salesman Problem**

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

Start

Find out all (n -1)! Possible solutions, where n is the total number of cities.

Determine the minimum cost by finding out the cost of each of these (n -1)! solutions.

Finally, keep the one with the minimum cost.

end

Fuzzy Logic Systems (FLS) produce acceptable but definite output in response to incomplete, ambiguous, distorted, or inaccurate (fuzzy) input.

**Fuzzy Logic**

**What is Fuzzy Logic?**

Fuzzy Logic (FL) is a method of reasoning that resembles human reasoning. The approach of FL imitates the way of decision making in humans that involves all intermediate possibilities between digital values YES and NO.

The conventional logic block that a computer can understand takes precise input and produces a definite output as TRUE or FALSE, which is equivalent to human’s YES or NO.

The inventor of fuzzy logic, Lotfi Zadeh, observed that unlike computers, the human decision making includes a range of possibilities between YES and NO, such as −

|  |
| --- |
| CERTAINLY YES |
| POSSIBLY YES |
| CANNOT SAY |
| POSSIBLY NO |
| CERTAINLY NO |

The fuzzy logic works on the levels of possibilities of input to achieve the definite output.

**Implementation**

* It can be implemented in systems with various sizes and capabilities ranging from small micro-controllers to large, networked, workstation-based control systems.
* It can be implemented in hardware, software, or a combination of both.

**Why Fuzzy Logic?**

Fuzzy logic is useful for commercial and practical purposes.

* It can control machines and consumer products.
* It may not give accurate reasoning, but acceptable reasoning.
* Fuzzy logic helps to deal with the uncertainty in engineering.

**Fuzzy Logic Systems Architecture**

It has four main parts as shown −

* **Fuzzification Module** − It transforms the system inputs, which are crisp numbers, into fuzzy sets. It splits the input signal into five steps such as −

|  |  |
| --- | --- |
| **LP** | x is Large Positive |
| **MP** | x is Medium Positive |
| **S** | x is Small |
| **MN** | x is Medium Negative |
| **LN** | x is Large Negative |

* **Knowledge Base** − It stores IF-THEN rules provided by experts.
* **Inference Engine** − It simulates the human reasoning process by making fuzzy inference on the inputs and IF-THEN rules.
* **Defuzzification Module** − It transforms the fuzzy set obtained by the inference engine into a crisp value.



The **membership functions work on** fuzzy sets of variables.

**Membership Function**

Membership functions allow you to quantify linguistic term and represent a fuzzy set graphically. A **membership function** for a fuzzy *set A* on the universe of discourse X is defined as μA:X → [0,1].

Here, each element of *X* is mapped to a value between 0 and 1. It is called **membership value** or **degree of membership**. It quantifies the degree of membership of the element in *X* to the fuzzy set *A*.

* x axis represents the universe of discourse.
* y axis represents the degrees of membership in the [0, 1] interval.

There can be multiple membership functions applicable to fuzzify a numerical value. Simple membership functions are used as use of complex functions does not add more precision in the output.

All membership functions for **LP, MP, S, MN,** and **LN** are shown as below −

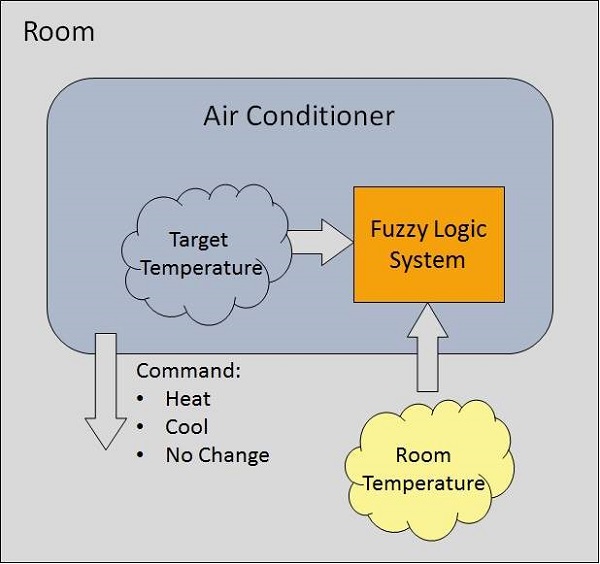


The triangular membership function shapes are most common among various other membership function shapes such as trapezoidal, singleton, and Gaussian.

Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

**Example of a Fuzzy Logic System**

Let us consider an air conditioning system with 5-level fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.



**Algorithm**

* Define linguistic variables and terms.
* Construct membership functions for them.
* Construct knowledge base of rules.
* Convert crisp data into fuzzy data sets using membership functions. (fuzzification)
* Evaluate rules in the rule base. (Inference Engine)
* Combine results from each rule. (Inference Engine)
* Convert output data into non-fuzzy values. (defuzzification)

**Logic Development**

**Step 1: Define linguistic variables and terms**

Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.

Temperature (t) = {very-cold, cold, warm, very-warm, hot}

Every member of this set is a linguistic term and it can cover some portion of overall temperature values.

**Step 2: Construct membership functions for them**

The membership functions of temperature variable are as shown −



**Step3: Construct knowledge base rules**

Create a matrix of room temperature values versus target temperature values that an air conditioning system is expected to provide.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **RoomTemp. /Target** | **Very\_Cold** | **Cold** | **Warm** | **Hot** | **Very\_Hot** |
| Very\_Cold | No\_Change | Heat | Heat | Heat | Heat |
| Cold | Cool | No\_Change | Heat | Heat | Heat |
| Warm | Cool | Cool | No\_Change | Heat | Heat |
| Hot | Cool | Cool | Cool | No\_Change | Heat |
| Very\_Hot | Cool | Cool | Cool | Cool | No\_Change |

Build a set of rules into the knowledge base in the form of IF-THEN-ELSE structures.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Condition** | **Action** |
| 1 | IF temperature=(Cold OR Very\_Cold) AND target=Warm THEN | Heat |
| 2 | IF temperature=(Hot OR Very\_Hot) AND target=Warm THEN | Cool |
| 3 | IF (temperature=Warm) AND (target=Warm) THEN | No\_Change |

**Step 4: Obtain fuzzy value**

Fuzzy set operations perform evaluation of rules. The operations used for OR and AND are Max and Min respectively. Combine all results of evaluation to form a final result. This result is a fuzzy value.

**Step 5: Perform defuzzification**

Defuzzification is then performed according to membership function for output variable.



**Application Areas of Fuzzy Logic**

The key application areas of fuzzy logic are as given −

**Automotive Systems**

* Automatic Gearboxes
* Four-Wheel Steering
* Vehicle environment control

**Consumer Electronic Goods**

* Hi-Fi Systems
* Photocopiers
* Still and Video Cameras
* Television

**Domestic Goods**

* Microwave Ovens
* Refrigerators
* Toasters
* Vacuum Cleaners
* Washing Machines

**Environment Control**

* Air Conditioners/Dryers/Heaters
* Humidifiers

**Advantages of FLSs**

* Mathematical concepts within fuzzy reasoning are very simple.
* You can modify a FLS by just adding or deleting rules due to flexibility of fuzzy logic.
* Fuzzy logic Systems can take imprecise, distorted, noisy input information.
* FLSs are easy to construct and understand.
* Fuzzy logic is a solution to complex problems in all fields of life, including medicine, as it resembles human reasoning and decision making.

**Disadvantages of FLSs**

* There is no systematic approach to fuzzy system designing.
* They are understandable only when simple.
* They are suitable for the problems which do not need high accuracy.