IOSR Journal Of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727 Volume 6, Issue 3 (Sep-Oct. 2012), PP 01-08
Www.Iosrjournals.Org

Artificial Intelligence Algorithms

Sreekanth Reddy Kallem
Department of computer science, AMR Institute of Technology, Adilabad,JNTU,Hyderabad, A.P, India.

Abstract-*Artificial intelligence (AI) is the study of how to make computers do things which, at the moment, people do better. Thus Strong AI claims that in near future we will be surrounded by such kinds of machine which can completely works like human being and machine could have human level intelligence. One intention of this article is to excite a broader AI audience about abstract algorithmic information theory concepts, and conversely to inform theorists about exciting applications to AI.The science of Artificial Intelligence (AI) might be defined as the construction of intelligent systems and their analysis.*
Keywords: *Artificial Intelligence, Algorithms*

I. Introduction

Artificial intelligence (AI) is the intelligence of machines and the branch of computer science that aims to create it. AI textbooks define the field as "the study and design of intelligent agents"[1]where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success.[2] John McCarthy, who coined the term in 1955,[3] defines it as "the science and engineering of making intelligent machines."[4]

AI research is highly technical and specialized, deeply divided into subfields that often fail to communicate with each other.[5] Some of the division is due to social and cultural factors: subfields have grown up around particular institutions and the work of individual researchers. AI research is also divided by several technical issues. There are subfields which are focussed on the solution of specific problems, on one of several possible approaches, on the use of widely differing tools and towards the accomplishment of particular applications. The central problems of AI include such traits as reasoning,knowledge, planning, learning, communication, perception and the ability to move and manipulate objects.[6] General intelligence (or "strong AI") is still among the field's long term goals.[7] Currently popular approaches include statistical methods, computational intelligence and traditional symbolic AI. There are an enormous number of tools used in AI, including versions of search and mathematical optimization, logic, methods based on probability and economics, and many others.

The field was founded on the claim that a central property of humans, intelligence—the sapience of Homo sapiens—can be so precisely described that it can be simulated by a machine.[8] This raises philosophical issues about the nature of the mind and the ethics of creating artificial beings, issues which have been addressed by myth, fiction and philosophy since antiquity.[9] Artificial intelligence has been the subject of optimism,[10] but has also suffered setbacks[11] and, today, has become an essential part of the technology industry, providing the heavy lifting for many of the most difficult problems in computer science.[12]

1.1. Applications of AI:

1.1.1. R&D Plan for Army Applications of AI/Robotics:
Robotic Reconnaissance Vehicle with Terrain Analysis,

- Automated Ammunition Supply Point (ASP),
- Intelligent Integrated Vehicle Electronics,
- AI-Based Maintenance Tutor,
- AI-Based Medical System Development.

1.1.2. Expert system:
An expert system, is an interactive computer-based decision tool that use both facts and heuristics to solve difficult decision making problems based on knowledge acquired from an expert.

- An expert system compared with traditional computer:

Inference engine + Knowledge = expert system
(Algorithm + data structures = program in traditional computer)
1.1.3. Fuzzy Logic:
Fuzzy logic is more than thirty years old and has a long-lasting misunderstanding with artificial Intelligence, although the formalization of some forms of commonsense reasoning has motivated the development of fuzzy logic.

1.1.4. Mobile Robotics and Games (Path Planning):

Mobile robots often have to replan quickly as the world ortheir knowledge of it changes. Examples include both physical robots and computer-controlled robots (or, more generally, computer-controlled characters) in computer games. Efficient replanting is especially important for computer games since they often simulate a large number of characters and their other software components, such as the graphics generation, already place a high demand on the processor. In the following, we discuss two cases where the knowledge of a robot changes because its sensors acquire more information about the initially unknown terrain as it moves around.

II. Genetic Algorithm

This article describes how to solve a logic problem using a Genetic Algorithm. It assumes no prior knowledge of GAs.A genetic algorithm is a search technique used in computing, to find true or approximate solutions to optimization and search problems, and is often abbreviated as GA. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover(also called recombination).

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves towards better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals, and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly mutated) to form a new population. The new population is then used in the next iteration of the algorithm.

Follow that?? If not, let's try a diagram. (Note that this is a Microbial GA, there are lots of GA types, but I just happen to like this one, and it's the one this article uses.)

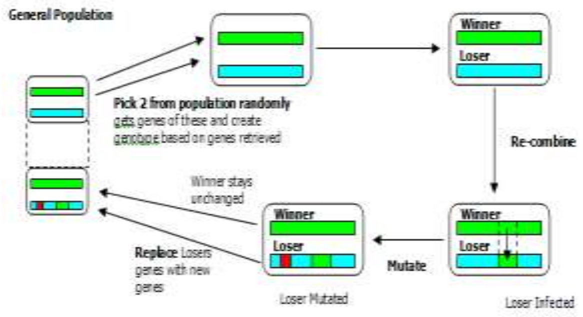

General Population
Pick 2 from population randomly
gets genes of these and create genotype based on genes retrieved
Winner
Loser
Re-combine
Winner
Loser
Loser Infected
Mutate
Winner
Loser
Loser Mutated
Winner stays unchanged
Replace Losers genes with new genes

Fig. 1 Genetic Algorithm

I prefer to think of a GA as a way of really quickly (well, may be quite slow, depending on the problem) trying out some evolutionary programming techniques, that mother nature has always had.

III. Path Finding Algorithms/Ai

The algorithms we shall discuss in this tutorial are the decisionbased algorithms as well as the simplest of the recursive type algorithms. This is because they are easy to comprehend as well aseasy toimplement. We will ignore BFS, Dijkstraand A* since these are very complex algorithms which may require special data structures and are harder to implement.

Path-finding consumes a significant amount of resources, especially in movement-intensive games such as (massively) multiplayer games. We investigate several path-finding techniques, and explore the impact on performance of workloads derived from real player movement sin a multiplayer game. We find that a map-conforming, hierarchical path-finding strategy performs best, and in combination with caching optimizations can greatly re-duce path-finding cost. Performance is dominated primarily by algorithm, and only to a lesser degree byworkload variation. Understanding the real impact of path-finding techniques allows for refined testing and optimization of game design.

Path-finding in computer games is commonly approached as a graph search problem. The world is de-composed, abstracted as a graph model, and searched, typically using some variant of IDA* (Korf 1985), basedon the well-known A* (Hart et al. 1968) algorithm. Underlying world decompositions can have a significantimpact on performance. Common approaches includethose based on uniformly shaped grids, such as squareor hexagonal tilings (Yap 2002), as well as the use ofquadtrees (Chen et al. 1995; Davis 2000) or variableshaped tiles (Niederberger et al. 2004) for adaptivityto more arbitrary terrain boundaries. Properties ofthe decomposition, its regularity, convexity, or Voronoiguarantees, as well as geometric computations, such asvisibility graphs, or even heuristic roadmap information (Kavrakiet al.1996) can then be used to improvesearch efficiency.

Hierarchical path-finding incorporates multiple graph orsearch-space decompositions of different granularity asa way of reducing search cost, perhaps with some lossof optimality. Hierarchical information has been used toimprove A* heuristics (Holte et al. 1996), and proposedin terms of using more abstract, meta-information al-ready available in a map, such as doors, rooms, floors,departments (Maio and Rizzi 1994). Less domain-specific are graph reduction techniques based on recursively combining nodes into clusters to form a hierarchical structure (Sturtevant and Buro 2005). Our approachhere is most closely based on the HPA* multi-level hierarchical design, where node clustering further considersthe presence of collision-free paths between nodes (Boteaet al. 2004).

Path-finding can also be based on the physics of dynamic character interaction. In strategies based on potential fields (Khatib 1985) or more complex steeringbehaviours (Reynolds 1999; Bayazit et al. 2002) a character's path is determined by its reaction to its environment. This reactive approach can be combined withsearch-based models to improve heuristic choices duringsearching (Pottinger 2000). There are many possibleheuristics to exploit; in our implementations we use a"diagonal distance" metric to approximate the cost of unknown movement (Patel 2003).

3.1. Heuristic Function:

A heuristic is a technique that improves the efficiency of search process, possibly by sacrificing claims of completeness. While the almost perfect heuristic is significant for theoretical analysis, it is not common to find such a heuristic in practice.heuristics play a major role in search strategies because of exponential nature of the most problems.Heuristic help to reduce the number of alternatives from an exponentialnumberto polynomial number. Heuristic search has been widely used in both deterministic and probabilistic planning. Yet, although bidirectional heuristic search has been applied broadly in deterministic planning problems, its function in the probabilistic domains is only sparsely studied.

A heuristic function is a function that maps from problem state descriptions to measures of desirability, usually represented as number. Heuristic functions generally have different errors in different states. Heuristic functions play a crucial rule in optimal planning, and the theoretical limitations of algorithms using such functions are therefore of interest. Much work has focused on finding bounds on the behavior of heuristic search algorithms, using heuristics with specific attributes.

3.2. Depth-First Search:

The first strategy is depth-first search. In depth-first search, the frontier acts like a last-in first out stack. The elements are added to the stack one at a time. The one selected and taken off the frontier at any time is the last element that was added. This Algorithm is also called as Recursive Algorithm.

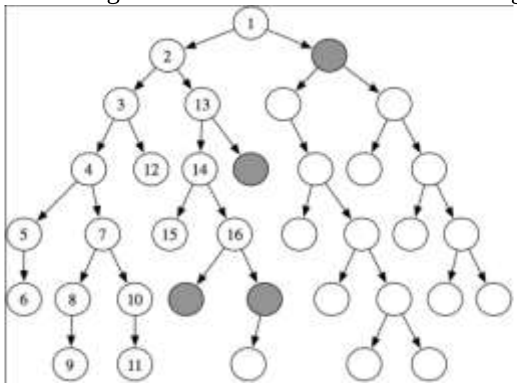


Fig. 2The order nodes are expanded in depth-first search.

Recursive algorithms are popular because they are the most powerful and are used in association with tile based games such as rpgs. The thing with these set of algorithms is that they require a TONof computing power. This is because these algorithms generally run in exponential time which to put simply means that for a small increase in map size, running time will increase a lot. If you're serious about making games, i suggest you learn a high level language such asc++ or java. If you're just here to fool around then i suggest you stick to small map sizes.

In order to design a recursive path finding algorithm, one mustunderstand the concept of recursion. Basically a recursive functionis a function which calls itself. To illustrate this concept, lets look at a simple function which calculates fibonacci numbers.(fibonacci numbers are a sequence of numbers where each successivenumber is the sum of the previous two ie. 1, 1, 2, 3, 5, 8, 13).

Code:
```
Function fib (int n) {
If (n == 0) return 0
If (n == 1) return 1
Return fib (n-1) + fib(n-2)
}
```
If we look carefully at the code we see that the function fib() callsupon itself to calculate the previous two numbers. This process keepsrepeating until it either calculates the value of 0 or 1 which wealready know the value of.

Lets go step by step how this functionworks:

1) If we try to find the 6th element in the sequence we call fib(6)
2) Since n is not equal to 0 or 1, we must calculate the valuerecursively.
3) The value of fib(6) is equal to fib(5)+fib(4) which are the twoprevious numbers in the series. So we call the fib function to calculate these two numbers.
4) This process keeps repeating itself untill it encounters the twopreconditions which return a constant value. If it weren't for thesepreconditions, the function would keep calling itself untill yourcomputer runs out of RAM!!!

3.3. Breadth-First Search:

In breadth-first search the frontier is implemented as a FIFO (first-in, first-out) queue. Thus, the path that is selected from the frontier is the one that was added earliest.

This approach implies that the paths from the start node are generated in order of the number of arcs in the path. One of the paths with the fewest arcs is selected at each stage.

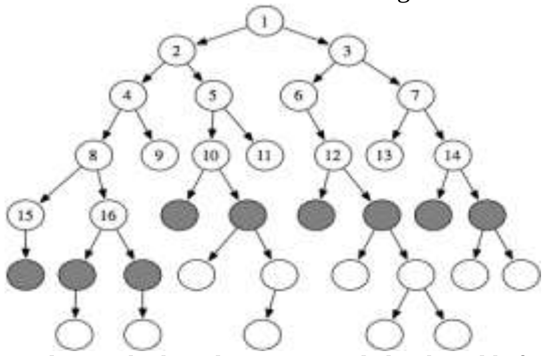


Fig. 3The order in which nodes are expanded in breadth-first search

Depth-first search is appropriate when either

- space is restricted;
- many solutions exist, perhaps with long path lengths, particularly for the case where nearly all paths lead to a solution; or
- The order of the neighbors of a node are added to the stack can be tuned so that solutions are found on the first try.
- It is a poor method when
- it is possible to get caught in infinite paths; this occurs when the graph is infinite or when there are cycles in the graph; or
- Solutions exist at shallow depth, because in this case the search may look at many long paths before finding the short solutions.

Depth-first search is the basis for a number of other algorithms, such as " iterative deepening".

Breadth-first search is useful when

- space is not a problem;
- you want to find the solution containing the fewest arcs;
- few solutions may exist, and at least one has a short path length; and
- infinite paths may exist, because it explores all of the search space, even with infinite paths.

It is a poor method when all solutions have a long path length or there is some heuristic knowledge available. It is not used very often because of its space complexity.

3.4. A search algorithm:*

The A* algorithm combines features of uniform-cost search and pure heuristic search to efficiently compute optimal solutions. A* algorithm is a best-first search algorithm in which the cost associated with a node is f(n)=g(n)+h(n), where g(n) is the cost of the path from the initial state to node n and h(n) is the heuristic estimate or the cost or a path from node n to a goal. Thus, f(n) estimates the lowest total cost of any solution path going through node n. At each point a node with lowest f value is chosen for expansion. Ties among nodes of equal f value should be broken in favour of nodes with lower h values. The algorithm terminates when a goal is chosen for expansion.

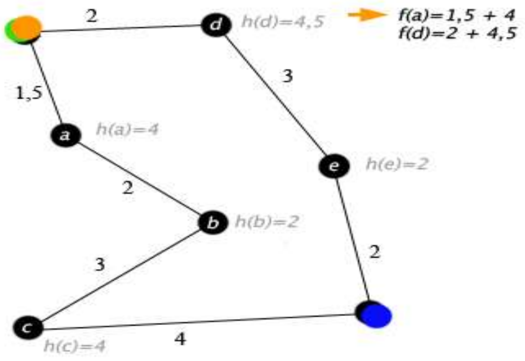
A* algorithm guides an optimal path to a foal if the heuristic function h(n) is admissible, meaning it never overestimates actual cost. For example, since airline distance never overestimates actual highway distance, and manhatten distance never overestimates actual moves in the gliding tile.

A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest known heuristic cost, keeping a sorted priority queue of alternate path segments along the way.

I t uses a distance-plus-cost heuristic function (usually denoted $f(x)$) to determine the order in which the search visits nodes in the tree. The distance-plus-cost heuristic is a sum of two functions:
the path-cost function, which is the cost from the starting node to the current node (usually denoted $g(x)$)
an admissible "heuristic estimate" of the distance to the goal (usually denoted $h(x)$).
The $h(x)$ part of the $f(x)$ function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for an application like routing, $h(x)$ might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points or nodes.

If the heuristic *h* satisfies the additional condition $h(x) \le d(x,y) + h(y)$ for every edge *x*, *y* of the graph (where *d* denotes the length of that edge), then *h* is called monotone, or consistent. In such a case, A* can be implemented more efficiently—roughly speaking, no node needs to be processed more than once (see closed set below)—and A* is equivalent to runningDijkstra's algorithm with the reduced cost $d'(x,y) := d(x,y) - h(x) + h(y)$.
Example:

An example of an A star (A*) algorithm in action where nodes are cities connected with roads and h(x) is the straight-line distance to target point:



Key: green: start; blue: goal; orange: visited
Note: This example uses a comma as the decimal separator.

3.4.1. Complexity*:*
The time complexity of A* depends on the heuristic. In the worst case, the number of nodes expanded is exponential in the length of the solution (the shortest path), but it is polynomial when the search space is a tree, there is a single goal state, and the heuristic function *h* meets the following condition:
$|h(x) - h^*(x)| = O(\log h^*(x))$
where h^* is the optimal heuristic, the exact cost to get from x to the goal. In other words, the error of *h* will not grow faster than the logarithm of the “perfect heuristic” h^* that returns the true distance from x to the goal (see Pearl 1984[11] and also Russell and Norvig 2003, p. 101[12])

The main drawback of A*, and indeed of any best-first search,is its memory requirement.Since at list the entire Open list must be saved, A* is severely space-limited in practice, and is no morePractical than breadth-first search on current machines. For example, while it can be Run successfully in the Eight Puzzle, it exhausts available memory in a matter of minutes on the Fifteen Puzzle.

3.5. A Generic Searching Algorithm:

This section describes a generic algorithm to search for a solution path in a graph. The algorithm is independent of any particular search strategy and any particular graph.

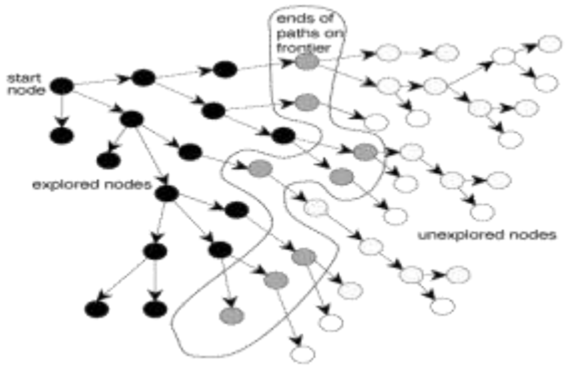


Fig.4 Problem solving by graph searching

The intuitive idea behind the generic search algorithm, given a graph, a set of start nodes, and a set of goal nodes, is to incrementally explore paths from the start nodes. This is done by maintaining a frontier(or fringe) of paths from the start node that have been explored. The frontier contains all of the paths that could form initial segments of paths from a start node to a goal node. (See Figure 3.3, where the frontier is the set of paths to the gray shaded nodes.) Initially, the frontier contains trivial paths containing no arcs from the start nodes. As the search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered. To expand the frontier, the searcher selects and removes a path from the frontier, extends the path with each arc leaving the last node, and adds these new paths to the frontier. A search strategy defines which element of the frontier is selected at each step.

```
1: **Procedure** Search($G,S,goal$)
2:  **Inputs**
3:  $G$: graph with nodes $N$ and arcs $A$
4: $S$: set of start nodes
5: $goal$: Boolean function of states
6: **Output**
7: path from a member of $S$ to a node for which $goal$ is true
8: or $\perp$ if there are no solution paths
9: **Local**
10: $Frontier$: set of paths
11: $Frontier \leftarrow \{\langle s \rangle : s \in S\}$
12: **while** ($Frontier \neq \{\}$)
13: **select** and **remove** $\langle s_0,\ldots,s_k \rangle$ from $Frontier$
14: **if** ( $goal(s_k)$) **then**
15: **return** $\langle s_0,\ldots,s_k \rangle$
16: $Frontier \leftarrow Frontier \cup \{\langle s_0,\ldots,s_k,s \rangle : \langle s_k,s \rangle \in A\}$
17: **return** $\perp$
```

Fig.5. Generic graph searching algorithm.

The generic search algorithm is shown in Figure 3.4. Initially, the frontier is the set of empty paths from start nodes. At each step, the algorithm advances the frontier by removing a path $\langle s_0,\ldots,s_k \rangle$ from the frontier. If $goal(s_k)$ is true (i.e., s_k is a goal node), it has found a solution and returns the path that was found, namely $\langle s_0,\ldots,s_k \rangle$. Otherwise, the path is extended by one more arc by finding the neighbors of s_k. For every neighbor s of s_k, the path $\langle s_0,\ldots,s_k,s \rangle$ is added to the frontier. This step is known as expandingthe node s_k.

This algorithm has a few features that should be noted:

- The selection of a path at line 13 is non-deterministic. The choice of path that is selected can affect the efficiency; see the box for more details on our use of "select". A particular search strategy will determine which path is selected.
- It is useful to think of the return at line 15 as a temporary return; another path to a goal can be searched for by continuing to line 16.
- If the procedure returns $\perp$, no solutions exist (or there are no remaining solutions if the proof has been retried).
- The algorithm only tests if a path ends in a goal node *after* the path has been selected from the frontier, not when it is added to the frontier. There are two main reasons for this. Sometimes a very costly arc exists from a node on the frontier to a goal node. The search should not always return the path with this arc, because a lower-cost solution may exist. This is crucial when the least-cost path is required. The second reason is that it may be expensive to determine whether a node is a goal node.

If the path chosen does not end at a goal node and the node at the end has no neighbors, extending the path means removing the path. This outcome is reasonable because this path could not be part of a path from a start node to a goal node.

IV. Problem Reduction Algorithms

Problem reduction search can be planning how best to solve a problem that can be recursively decomposed into sub-problems in multiple ways.

1) Matrix multiplication problem
2) Tower of Hanoi
3) Blocks world problems
4) Theorem proving

4.1. AO Algorithm:*

1. Let G consists only to the node representing the initial state call this node INTT.Computeh' (INIT).
2. Until INIT is labeled SOLVED or hi (INIT) becomes greater than FUTILITY, repeat the following procedure.

(I) Trace the marked arcs from INIT and select an unbounded node NODE.

(II) Generate the successors of NODE. if there are no successors then assign FUTILITY as h' (NODE). This means that NODE is not solvable. If there are successors then for each one called SUCCESSOR, that is not also an ancester of NODE do the following

(a) add SUCCESSOR to graph G

(b) if successor is not a terminal node, mark it solved and assign zero to its h ' value.

(c) If successor is not a terminal node, compute it h' value.

(III) propagate the newly discovered information up the graph by doing the following . letS be a set of nodes that have been marked SOLVED. Initialize S to NODE. Until S is empty repeat the following procedure;

(a) select a node from S call if CURRENT and remove it from S.

(b) compute h' of each of the arcs emerging from CURRENT , Assign minimum h' to CURRENT.

(c) Mark the minimum cost path a s the best out of CURRENT.

(d) Mark CURRENT SOLVED if all of the nodes connected to it through the new marked are have been labeled SOLVED.

(e) If CURRENT has been marked SOLVED or its h' has just changed, its new status must be propagate backwards up the graph. Hence all the ancestors of CURRENT are added to S.

4.1.1.AO* Search Procedure:

1. Place the start node on open.
2. Using the search tree, compute the most promising solution tree TP .
3. Select node n that is both on open and a part of tp, remove n from open and place it no closed.
4. If n is a goal node, label n as solved. If the start node is solved, exit with success where tp is the solution tree, remove all nodes from open with a solved ancestor.
5. If n is not solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. Remove all nodes from open ,with unsolvable ancestors.
6. Otherwise, expand node n generating all of its successor compute the cost of for each newly generated node and place all such nodes on open.
7. Go back to step (2)

Note: AO* will always find minimum cost solution.

Properties of AO*:

• AO* is a generalization of A* for AND-OR graphs

• AO*, like A*, is admissible if the heuristic function isadmissible and the usual assumptions (finite branchingfactor etc) hold

• AO*, like A* is also optimal among the class of heuristicsearch algorithms that use an additive cost / evaluation function.

V. Conclusion

We conclude that by using this algorithm we can solve AI problems easily.AI algorithms are also called as a problem solving algorithms. Artificial Intelligence will surpass human intelligence. Although it has proven itself to be similar to the human brain, computers do not think in the same way. In this report, we have discussed Algorithms of AI and their impact on problem solving and our lives.

References

[1] http://www.hutter1.net/ai/uaibook.html.
[2] Elaine Rich, Kevin Knight, " Artificial Intelligence", Second Edition, page no.3.
[3] " Universal Artificial Intelligence"Subtitle: Sequential Decisions based on Algorithmic Probability.
[4] http://en.wikipedia.org/wiki/Artificial_intelligence.
[5] http://www.codeproject.com/Articles/16286/AI-Simple-Genetic-Algorithm-GA-to-solve-a-card-pro.
[6] http://artint.info/html/ArtInt_51.html.
[7] David Poole,AlanMackworth, "Artificial Intelligence: Foundations of Computational Agents",2010.
[8] http://www.kirupa.com/forum/showthread.php?72863-Tutorial-Path-Finding-Algorithms-AI.
[9] http://webdocs.cs.ualberta.ca/~lanctot/files/papers/pathfinding-orbius-2006.pdf.
[10] http://en.wikipedia.org/wiki/A*_search_algorithm.
[11] Richards E.Korf* , "Artificial Intelligence Search Algorithms", Computer Science Department University of California, Los Angeles,Ca.90095 June 23,1998.
[12] http://www.synergy.ac.in/intranet/e-book/(Ebook%20-%20Paper)%20Artificial%20Intelligence%20Search%20Algorithms.pdf.
[13] Robin ,"A star algorithm"December 18th, 2009 |
[14] http://intelligence.worldofcomputing.net/ai-search/a-star-algorithm.html.
[15] http://artificialintelligence-notes.blogspot.in/2010/07/problem-reduction-with-ao-algorithm.html.
[16] http://www.cs.iastate.edu/~cs572/WWW-honavar07/cs572problem-reduction.pdf.
[17] lowa state university, deparment of computer science artificial intelligence research laboratory.
[18] ReferedFrom Artificial Intelligence TMH.
[19] http://www.eetindia.co.in/VIDEO_DETAILS_700000062.HTM Prof. PallabDasgupta of the Department of Computer Science and Engineering, IIT Kharagpurconducted this lecture.
[20] KoushalKumar ,GourSundarMitra Thakur" Advanced Applications of Neural Networks and Artificial Intelligence: A Review." Published Online June 2012 in MECS (http://www.mecs-press.org/) DOI: 10.5815/ijitcs.2012.06.08,pg.no.58.
[21]]By National Research Council, "Applications of Robotics and Artificial Intelligence to Reduce Risk and Improve Effectiveness".pg.no.2-3
[22] RC Chakraborty, "Expert Systems: AI course lecture 35-36",pg no. 03.
[23]]Didier Dubois-Henri Prade," The place of Fuzzy Logic in AI".
[24]]Andrew Ilyas," An Isearch research paper(Artificial Intelligence)", May 13, 2010.
[25] NirPochter and Jeffrey S. Rosenschein, "Requirements on Heuristic Functions when Using A* in Domains with Transpositions".
[26] Robin, "Heuristic Search(artificial intelligence articles on artificial intelligence)". December 14th,2009.
[27] Peng Dai," Heuristic Search Ideas for Deterministic and Probabilistic Problems", Journal, Pg.no.1.
[28] Sven Koenig Maxim LikhachevYaxin Liu David Furcy, "Incremental Heuristic Search in Artificial Intelligence", Pg.no. 08.