2nd International Conference on Current Research Trends in Engineering and Technology
© 2018 IJSRSET | Volume 4 | Issue 5 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Eco Geography-Based Optimization : An Improved Water Wave Optimization Models to Solve NP-Hard

Digvijaysinh Mahida, Pinkal Shah, Trilok Suthar, Dipak Agrawal, Pritesh Patel
Department of Information technology, Sigma Institute of Engineering, Vadodara, Gujarat, India

ABSTRACT

Nature-inspired computing has been a hot topic in scientific and engineering fields in recent years. Inspired by the shallow water wave theory, the paper presents a novel metaheuristic method, named Water wave optimization(WWO), for global optimization problems. Two methodologies are there to find shortest path one is ant colony optimization and another is water optimization algorithm. it also fixes no of problem in real life are Scheduling problem, Vehicle routing problem, Assignment problem, Set problem, Device sizing problem in Nano electronics physical design, Image processing and TSP (Traveling sales man problem).

Keywords: Ant colony optimization (ACO), Water optimization algorithm, Evolution computing (EC), Traveling salesman problem (TSP), Swarm

I. INTRODUCTION

Ant colony optimization is a technique for optimization that was introduced in the early 1990's. The inspiring source of ant colony optimization is the foraging behaviour of real ant colonies. This behaviour is exploited in artificial ant colonies for the search of approximate solutions to discrete optimization problems, to continuous optimization problems, and to important problems in telecommunications, such as routing and load balancing [1].it also fix no of problem in real life are Scheduling problem, Vehicle routing problem, Assignment problem, Set problem, Device sizing problem in Nano electronics physical design, Image processing and TSP (Traveling sales man problem).
Many engineering optimization problems are usually quite difficult to solve, and many applications have to deal with these complex problems. In these problems, search space grows exponentially with the problem size. Therefore, the traditional optimization methods do not provide a suitable solution for them. Hence, over the past few decades, many meta-heuristic algorithm shave been designed to solve such problems. Researchers have shown good performance of meta- heuristic algorithms in a wide range of complex problems such as scheduling problems.

Here we going to find better technique for solving all above the problem which methodology is better ant colony optimization and water optimization. The way both are works are described below.

II. WATER OPTIMIZATION APPROACH

The new concept of water optimization is quite similar the ant colony optimization. It also finds path from its source to the destination.

We can take example of river, water fall, and rain water it starts from its source it dynamically chooses the path using huge moving swarms. Water in which they flow is the part of the environment that has been dramatically changed by the swarm and will also be changed in the future. We know that in nature, most paths full of twists and turns. Still it is believed that water swarm in have no eyes so that by using those eyes, they can find their destination. So simple water flow decides its path its own on which cover less obstacle, less difficulties, essay to flow to rich its destination. And it finds shortest path from its source to destination like ACO.

This concept is also useful to solve no of problem like assignment problem, travelling salesman problem, network routing and many problems can be solved using water optimization algorithm. The no of matter which observed in water flow optimization:

1. The velocity of a water swarm increases more on a path with low soil than a path with high soil.

2. A water swarm prefers a path with less soil than a path with more soil.

3. A high speed water swarm gathers more soil than a slower water drop.

III. PSEUDO CODE OF ANT COLONY OPTIMIZATIONS [6] [7]

Initialize the base attractiveness, τ, and visibility, η, For each edge;

For i < IterationMax do:

For each ant do:

Choose probabilistically (based on previous equation)

The next state to move

Into;

Add that move to the tabu list for each ant;

Repeat until each ant completed a solution;

End;

For each ant that completed a solution do:

Update attractiveness τ for each edge that the ant

Traversed;

End;

If (local best solution better than global solution)

Save local best solution as global solution;

End;
End;

A. ALGORITHMS FOR SHORTEST PATH PROBLEM

1. Dijkstra's algorithm
2. Bellman–Ford algorithm
3. A* search algorithm
4. Floyd–Warshall algorithm
5. Johnson's algorithm
6. Viterbi algorithm

1. Dijkstra's algorithm

```
function Dijkstra(Graph, source):
 2
 3     create vertex set Q
 4
 5         for each vertex v in Graph:                 // Initialization
 6         dist[v] ← INFINITY                  // Unknown distance from source to v
 7             prev[v] ← UNDEFINED                 // Previous node in optimal path from source
 8           add v to Q                          // All nodes initially in Q (unvisited nodes)
 9
10       dist[source] ← 0                        // Distance from source to source
11
12      while Q is not empty:
13          u ← vertex in Q with min dist[u]    // Node with the least distance
14                                              // will be selected first
15          remove u from Q
16
17          for each neighbor v of u:           // where v is still in Q.
18              alt ← dist[u] + length(u, v)
19              if alt < dist[v]:               // A shorter path to v has been found
20                  dist[v] ← alt
21                  prev[v] ← u
22
23      return dist[], prev[]
```

2. Bellman–Ford algorithm

```
function BellmanFord(list vertices, list edges, vertex source)
  ::distance[],predecessor[]
   // This implementation takes in a graph, represented as
  // lists of vertices and edges, and fills two arrays
  // (distance and predecessor) with shortest-path
  // (less cost/distance/metric) information

  // Step 1: initialize graph
  for each vertex v in vertices:
     distance[v] := inf           // At the beginning , all vertices have a weight of infinity
     predecessor[v] := null              // And a null predecessor

  distance[source] := 0            // The weight is zero at the source

  // Step 2: relax edges repeatedly
  for i from 1 to size(vertices)-1:
     for each edge (u, v) with weight w in edges:
        if distance[u] + w < distance[v]:
           distance[v] := distance[u] + w
           predecessor[v] := u

  // Step 3: check for negative-weight cycles
  for each edge (u, v) with weight w in edges:
     if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"

  return distance[], predecessor[]
```

3. . A* search algorithm

```
function A*(start, goal)
   // The set of nodes already evaluated
   closedSet := {}

   // The set of currently discovered nodes that are not evaluated yet.
   // Initially, only the start node is known.
```

```
openSet := {start}

// For each node, which node it can most efficiently be reached from.
// If a node can be reached from many nodes, cameFrom will eventually contain the
// most efficient previous step.
cameFrom := an empty map

// For each node, the cost of getting from the start node to that node.
gScore := map with default value of Infinity

// The cost of going from start to start is zero.
gScore[start] := 0

// For each node, the total cost of getting from the start node to the goal
// by passing by that node. That value is partly known, partly heuristic.
fScore := map with default value of Infinity

// For the first node, that value is completely heuristic.
fScore[start] := heuristic_cost_estimate(start, goal)

while openSet is not empty
    current := the node in openSet having the lowest fScore[] value
    if current = goal
        return reconstruct_path(cameFrom, current)

    openSet.Remove(current)
    closedSet.Add(current)

    for each neighbor of current
        if neighbor in closedSet
            continue        // Ignore the neighbor which is already evaluated.

        if neighbor not in openSet    // Discover a new node
            openSet.Add(neighbor)

        // The distance from start to a neighbor
        //the "dist_between" function may vary as per the solution requirements.
        tentative_gScore := gScore[current] + dist_between(current, neighbor)
        if tentative_gScore >= gScore[neighbor]
            continue        // This is not a better path.

        // This path is the best until now. Record it!
        cameFrom[neighbor] := current
        gScore[neighbor] := tentative_gScore
        fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)

    return failure

function reconstruct_path(cameFrom, current)
    total_path := [current]
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.append(current)
    return total_path
```

4. Floyd–Warshall algorithm

let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
2 **for each** edge (*u*,*v*)
3 dist[*u*][*v*] ← w(*u*,*v*) *// the weight of the edge (*u,v*)*
4 **for each** vertex *v*

```
5    dist[v][v] ← 0Ant Colony Optimization and
Water Optimazition Algorithm
6 for k from 1 to |V|
7   for i from 1 to |V|
8     for j from 1 to |V|
9       if dist[i][j] > dist[i][k] + dist[k][j]
10          dist[i][j] ← dist[i][k] + dist[k][j]
11      end if
```

5. Johnson's algorithm

1. First, a new node q is added to the graph, connected by zero-weight edges to each of the other nodes.
2. Second, the Bellman–Ford algorithm is used, starting from the new vertex q, to find for each vertex v the minimum weight $h(v)$ of a path from q to v. If this step detects a negative cycle, the algorithm is terminated.
3. Next the edges of the original graph are reweighted using the values computed by the Bellman–Ford algorithm: an edge

   from u to v, having length , is given the new length $w(u,v) + h(u) − h(v)$.
4. Finally, q is removed, and Dijkstra's algorithm is used to find the shortest paths from each node s to every other vertex in the reweighted graph.

IV. CONCLUSION

Water optimization is helpful for solving many real life problems like Scheduling problem, Vehicle routing problem, Assignment problem, Set problem, Device sizing problem in Nano electronics physical design, Image processing and TSP (Traveling sales man problem). TSP is NP hard problem and we can find nearest solution for such problem and Water optimization approach may be more helpful for research in such problem.

V. REFERENCES

[1]. Christian Blum L. Perlovsky, “Ant colony optimization: Introduction and recent t rends”, ALBCOM, LSI, U niversitat Politècnica de Catalunya, Jordi G irona 1-3, Campus Nord, 08034 Barcelona, Spain October 2005 .

[2]. A. Colorni, M. Dorigo et V. Maniezzo, Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134-142, 1991.

[3]. Jump up to:a b M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy, 1992.

[4]. Zlochin, Mark; Birattari, Mauro; Meuleau, Nicolas; Dorigo, Marco (1 October 2004). "Model-Based Search for Combinatorial Optimization: A Critical Survey". Annals of Operations Research. 131 (1-4): 373–395. doi:10.1023/B:ANOR.0000039526.52305.af. ISSN 0254-5330.

[5]. Osaba, E. Diaz, F.,” Comparison of a memetic algorithm and a tabu search algorithm for the traveling salesman

[6]. Problem”IEEE Computer Science and Information Systems (FedCSIS), 2012 Federated Conference, 9 Sept. 2012

[7]. Kirti Pandey,Pallavi Jain,”Comparision of different heuristic,metaheuristic,traveling salesman problem solution”, , Proceedings of 1 6th IRF International Conference, 1 4th December 2014, Pune, India, ISBN: 978-93 - 84209-74-2

[8]. Viterbi AJ (April 1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". IEEE Transactions on Information Theory. 13 (2): 260–269. doi:10.1109/TIT.1967.1054010.

[9]. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), Introduction to Algorithms, MIT Press and McGraw-Hill, ISBN 978-0-262-03293-3. Section 25.3, "Johnson's algorithm for sparse graphs", pp. 636–640.

[10]. Bang-Jensen, Jørgen; Gutin, Gregory (2000). "Section 2.3.4: The Bellman-Ford-Moore algorithm". Digraphs: Theory, Algorithms and Applications (First ed.). ISBN 978-1-84800-997-4.

[11]. Bang-Jensen, Jørgen; Gutin, Gregory (2000). "Section 2.3.4: The Bellman-Ford-Moore algorithm". Digraphs: Theory, Algorithms and Applications (First ed.). ISBN 978-1-84800-997-4

[12]. Digvijaysinh Mahida1, Keyur Patel 2, Dipak Agrawal" Ant Colony Optimization and Water Optimazition Algorithm" 3International Journal of Advance Engineering and Research Development (IJAERD) Special Issue SIEICON-2017, April -2017,e-ISSN: 2348 - 4470 , print-ISSN:2348-6406