**Implementation of:**

**INTELLIGENT WATER DROPS ALGORITHM**

**By,**

**Iqra Siddiqui and Sara Intikhab**

Habib University

CS – 102

Section: 05

|  |  |  |
| --- | --- | --- |
| S.No. | Topic | Page No. |
| 1 | Abstract` | 3 |
| 2 | Introduction |  |
| 3 | Algorithm Description | 6-10 |
| 4 | Algorithm Applications | 11 |
| 5 | Implementation Details |  |
| 6 | Performance Evaluation | 13 |
| 7 | Conclusion |  |
| 8 | References | 15 |

# Abstract

The nature provides signs of inspiration for developing new intelligent systems. Intelligent Water Drops Algorithm is a product of this inspiration. It is a swarm based algorithm which is efficient in solving combinatorial and optimization problems. The algorithm uses number of intelligent water drops (IWD) which develops their solution incrementally and the best quality solution is chosen after all iterations. This quality is decided on the basis of soil and velocity that an IWD carries/ removes. After enough iterations of the IWD algorithm, the IWDs find the good paths that are decoded to good solutions of the problem.

This report provides an analysis on the python implementation of this algorithm and concludes the findings obtained while implementing it. These finding includes the algorithm complexities along with its time and space efficiency. We have also provided the algorithm essentials in this report including its flowchart, performance evaluations and its applications. This report also contains implementation details which will serve to make the IWD algorithm implementation easy to understand in python. Additionally, python implementation has been done using simpler data structures and taking consideration of weighted as well as un-weighted graph as input.

IWD Algorithm is an algorithm that can solve complex problems with great efficiency and can be modified easily according to the problem. This implementation is a minute contribution in opening room for more modifications and advancements in this algorithm and other similar nature inspired algorithms that can revolutionized the age of computing in near future.

# Introduction

Intelligent Water Drops (IWD) Algorithm is a Swarm- based optimization algorithm which is proposed by Shah Hosseini in 2007 proceedings of CEC i.e. Congress on Evolutionary Computation (Camacho-Villalón, Dorigo , & Stützle , 2019).

This algorithm is inspired by the natural swarms that exist in nature. Examples of such swarms can be ant colonies, bee colonies, rivers, etc. Intelligent Water Drops (IWD) is a way to compute a way finding technique within a swarm. This way fining technique is based on the dynamic actions of a river system and the reactions that happen within each droplet for it to find the optimum path. “The solutions are incrementally constructed by the IWD algorithm. Therefore, the IWD algorithm is a population-based constructive optimization algorithm” (Hosseini, 2009). This algorithm has been used to counter problems like knapsack problem or travelling salesman problem.

IWD Algorithm is inspired by the flow of Natural River. A natural river often finds good paths among lots of possible paths in its ways from the source to destination. These near optimal or optimal paths are obtained by the actions and reactions that occur among the water drops and the water drops with the riverbeds (Shah-Hosseini). Each water droplet in the river goes through its own obstacles to reach a particular ocean, lake or sea (destination). These obstacles vary from gravitational force of the earth, to the density of soil, and to the velocity of the droplet itself. This algorithm follows the same process and the optimum path is selected on the basis of soil and velocity of the water droplet. In the coming sections, we will determine how optimum path is taken and why it is the best path.

# Algorithm Description

The IWD algorithm is a step in the direction to model a few actions that happen in natural rivers and then to implement them in a form of an algorithm. In the IWD algorithm, IWDs are created with two main properties:

* Velocity
* Soil

A path with less soil lets the IWD become faster than a path with more soil in its route from source to destination. Therefore, paths with lower soils have higher chance to be selected by the IWD. (Shah-Hosseini)

The problem is expressed in the form of an undirected graph (N, E) where N is the nodes and E is the edges. The graph represent the environment for every IWD and they are spread randomly on the nodes of the graph.

Each IWD begins constructing its solution gradually by travelling on the nodes of the graph along the edges until an IWD finally completes its solution.

One iteration of the algorithm is complete when all IWDs have completed their solutions. After each iteration, the iteration-best solution TIB is found and it is used to update the total-best solution TTB.

The amount of soil on the edges of the iteration-best solution TIB is reduced based on the goodness (quality) of the solution. Then, the algorithm begins another iteration with new IWDs but with the same soils on the paths of the graph and the whole process is repeated.

The algorithm stops when it reaches the maximum number of iterations or the total-best solution TTB reaches the expected quality (Shah-Hosseini).

**The IWD algorithm is specified in the following steps:** (Shah-Hosseini)

1. *Initialisation of static parameters*. The graph (N, E) of the problem is given to the algorithm. The quality of the total-best solution TTB is initially set to the worst value: q(TTB) = −∞. The maximum number of iterations *itermax* is specified by the user. The iteration count *itercount* is set to zero. The number of water drops *NIWD* is set to a positive integer value, which is usually set to the number of nodes *Nc* of the graph. For velocity updating, the parameters are 1 *av* = 0.01, *bv* = 1 and *cv* = 1 . For soil updating, 1 *as* = 0.01 *bs* =1 and *cs* = 1. The local soil updating parameter *ρn* , which is a small positive number less than one, is set as *ρn* = 0.9. The global soil updating parameter *ρiwd* which is chosen from [0, 1], is set as *ρiwd* = 0.9 . Moreover, the initial soil on each path (edge) is denoted by the constant *InitSoil* such that the soil of the path between every two nodes i and j is set *by soil(i,j) = InitSoil* . The initial velocity of each IWD is set to *InitVel*. Both parameters InitSoil and InitVel are user selected and they should be tuned experimentally for the application. Here, *InitSoil* = 10000 and *InitVel* = 200.
2. *Initialisation of dynamic parameters.* Every IWD has a visited node list *Vc (IWD)*, which is initially empty: *Vc(IWD)* = {} . Each IWD’s velocity is set to *InitVel*. All IWDs are set to have zero amount of soil.
3. Spread the IWDs randomly on the nodes of the graph as their first visited nodes.
4. Update the visited node list of each IWD to include the nodes just visited
5. Repeat Steps 5.1 to 5.4 for those IWDs with partial solutions.
   1. For the IWD residing in node i, choose the next node j, which does not violate any constraints of the problem and is not in the visited node list () vc IWD of the IWD, using the following probability *pi(j)*:

Such that,

****And

Then, add newly visited node j to the list *Vc(IWD)*

* 1. For each IWD moving from node i to node j, update its velocity vel(t) by:

Where *vel(t+1)* is the updated velocity of the IWD

* 1. For the IWD moving on the path from node i to j, compute the soil Δsoil(i,j) that the IWD loads from the path by:

Where,

And the heuristic undesirability *HUD(j)* is defined appropriately for the given problem.

* 1. Update the soil *soil(i,j)* of the path from node i to j traversed by that IWD and also update the soil that the IWD carries soilIWD by:

1. Find the iteration-best solution TIB from all the solutions TIWD found by the IWDs using

Where, function q(.) gives the quality of the solution.

1. Update the soils on the paths that form the current iteration-best solution TIB by

Where, NIB is the number of nodes in the solution TIB.

1. Update the total best solution TTB by the current iteration-best solution TIB using
2. Increment the iteration number by
3. The algorithm stops here with the total-best solution TTB.

**Flowchart of IWD Algorithm**

****

The flowchart of the proposed IWD algorithm.

(a) The flowchart of the main steps of the IWD algorithm;

(b) A detailed flowchart of the sub-steps of step 5 of the IWD algorithm” (Shah‐Hosseini, 2008).

# “

**Complexity Analysis of IWD Algorithm**

Input: un-weighted graph

(V=number of nodes in graph, E= number of edges in graph, Niwd=number of iwds)

|  |  |
| --- | --- |
| Step No. | Complexity |
| 1 | O (V^2\*E) |
| 2 | O(itermax)\*O(*initializeIWD()*)  O(itermax)\*O(Niwd) |
| 3 | O(Niwd) |
| 4 | O(Niwd) |
| 5 | O(E)\*O(Niwd)\*O(itermax) |
| 6 | O(len(quality))\*O(itermax) |
| 7 | O(len(visit))\*O(itermax) |
| 8 | O(1) \*O(itermax) |
| 9 | O(1) \*O(itermax) |
| 10 | O(1) |

*Assuming V<E and V=Niwd*

Total Complexity = O(E)\*O(Niwd)\*O(itermax)

O(E\*Niwd\*itermax) or O(E\*V\*itermax)

*For further details, see the File “Complexity Analysis Unweighted Graph.py” on Git -> Code.*

We examined IWD algorithm best case and worst case situation. It can be concluded that there is no best case for this algorithm as IWD algorithm has the property of convergence and thus provided optimum result when the iterations are sufficiently large (Shah-Hosseini). Thus, its best case time complexity is same as its worst case time complexity.