16

Optimization with the Nature-Inspired Intelligent Water Drops Algorithm

Hamed Shah-Hosseini
Faculty of Electrical and Computer Engineering, Shahid Beheshti University, G.C. Tehran, Iran

1. Introduction

Scientists are beginning to realize more and more that nature is a great source for inspiration in order to develop intelligent systems and algorithms. In the field of Computational Intelligence, especially Evolutionary Computation and Swarm-based systems, the degree of imitation from nature is surprisingly high and we are at the edge of developing and proposing new algorithms and/or systems, which partially or fully follow nature and the actions and reactions that happen in a specific natural system or species.

Among the most recent nature-inspired swarm-based optimization algorithms is the Intelligent Water Drops (IWD) algorithm. IWD algorithms imitate some of the processes that happen in nature between the water drops of a river and the soil of the river bed. The IWD algorithm was first introduced in (Shah-Hosseini, 2007) in which the IWDs are used to solve the Travelling Salesman Problem (TSP). The IWD algorithm has also been successfully applied to the Multidimensional Knapsack Problem (MKP) (Shah-Hosseini, 2008a), *n*-queen puzzle (Shah-Hosseini, 2008b), and Robot Path Planning (Duan et al., 2008).

Here, the IWD algorithm and its versions are specified for the TSP, the *n*-queen puzzle, the MKP, and for the first time, the AMT (Automatic Multilevel Thresholding). Some theoretical findings have also been reviewed for the IWD algorithm. Next section reviews briefly the related works. Section 3 examines natural water drops. Section 4 states about Intelligent Water Drops (IWDs). Section 5 specifies the Intelligent Water Drops (IWD) algorithm. Next section, reviews the convergence properties of the IWD algorithm. Section 7 includes experiments with the IWD and its versions for the four mentioned problems. Final section includes the concluding remarks.

2. Related works

One of the famous swarm-based optimization algorithms has been invented by simulating the behaviour of social ants in a colony. Ants living in a nest are able to survive and develop their generations and reproduce in a cooperative way. Moreover, they can find the shortest path from their nest to a food source or vice versa. They can build complex nests capable of holding hundreds of thousands of ants and lots of other activities that show a high-level of intelligence in a colony of ants. Other social insects generally show such complex and intelligent behaviours such as bees and termites. Ant colony optimization (ACO) algorithm

Source: Evolutionary Computation, Book edited by: Wellington Pinheiro dos Santos,
ISBN 978-953-307-008-7, pp. 572, October 2009, I-Tech, Vienna, Austria

(Dorigo & et al., 1991) and Bee Colony Optimization (BCO) algorithm (Sato & Hagiwara, 1997) are among the swarm-based algorithms imitating social insects for optimization.
Evolutionary Computation is another system, which has been inspired from observing natural selection and reproduction systems in nature. Genetic algorithms (GA) (Holland, 1975) are among the most famous algorithms in this regard. Evolution Strategy (Rechenberg, 1973; Schwefel, 1981), Evolutionary Programming (Fogel et al., 1966), and Genetic Programming (Koza, 1992) are other Evolutionary-based intelligent algorithms that are often used for optimization. Memetic Algorithms (MA) (Dawkins, 1989; Ong et al., 2006) are among the most recent area of research in the field of Evolutionary Computation. Here, a meme is the basic unit of culture that can be inherited. An individual is assumed to have both genes and memes. Therefore, not only the genes of individuals evolve, but also their memes undergo evolution.
Artificial Immune Systems (AIS) are another example of swarm-based nature inspired algorithms, which follow the processes and actions that happen in the immune systems of vertebrates. Clonal Selection Algorithms (de Castro & Von Zuben, 2002), Negative Selection Algorithms (Forrest, et al., 1994), and Immune Network Algorithms (Timmis et al., 2000) are among the most common techniques in the field of AIS.
Another swarm-based optimization algorithm is the Particle Swarm Optimization (PSO), which has been introduced by (Kennedy & Eberhart, 1995). PSO uses a swarm of particles, which each one has position and velocity vectors, and they move near together to find the optimal solution for a given problem. Infact, PSO imitates the processes that exist in the flocks of birds or a school of fish to find the optimal solution.
Another swarm-based optimization algorithm is the Electromagnetism-like mechanism (EM) (Birbil & Fang, 2003). The EM algorithm uses an attraction-repulsion mechanism based on the Coulomb's law to move some points towards the optimal positions.

3. Natural water drops

In nature, flowing water drops are observed mostly in rivers, which form huge moving swarms. The paths that a natural river follows have been created by a swarm of water drops. For a swarm of water drops, the river 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. Moreover, the environment itself has substantial effects on the paths that the water drops follow. For example, against a swarm of water drops, those parts of the environment having hard soils resist more than the parts with soft soils. In fact, a natural river is the result of a competition between water drops in a swarm and the environment that resists the movement of water drops.
Based on our observation in nature, most rivers have paths full of twists and turns. Up until now, it is believed that water drops in a river have no eyes so that by using those eyes, they can find their destination, which is often a lake or sea. If we put ourselves in place of a water drop flowing in a river, we would feel that some force pulls us toward itself, which is the earth's gravity. This gravitational force pulls everything toward the center of the earth in a straight line. Therefore with no obstacles and barriers, the water drops should follow a straight path toward the destination, which is the shortest path from the source of water drops to the destination, which is ideally the earth's center. This gravitational force creates acceleration such that water drops gain speed as they come near to the earth's center. However, in reality, due to different kinds of obstacles and constraints in the way of this

ideal path, the real path is so different from the ideal path such that lots of twists and turns in a river path are seen, and the destination is not the earth's center but a lake, sea, or even a bigger river. It is often observed that the constructed path seems to be an optimal one in terms of the distance from the destination and the constraints of the environment.

One feature of a water drop flowing in a river is its velocity. It is assumed that each water drop of a river can also carry an amount of soil. Therefore, the water drop is able to transfer an amount of soil from one place to another place in the front. This soil is usually transferred from fast parts of the path to the slow parts. As the fast parts get deeper by being removed from soil, they can hold more volume of water and thus may attract more water. The removed soils, which are carried in the water drops, are unloaded in slower beds of the river. Assume an imaginary natural water drop is going to flow from one point of a river to the next point in the front. Three obvious changes happen during this transition:

- Velocity of the water drop is increased.
- Soil of the water drop is increased.
- Between these two points, soil of the river's bed is decreased.

In fact, an amount of soil of the river's bed is removed by the water drop and this removed soil is added to the soil of the water drop. Moreover, the speed of the water drop is increased during the transition.

It was mentioned above that a water drop has also a velocity. This velocity plays an important role in removing soil from the beds of rivers. The following property is assumed for a flowing water drop:

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

Therefore, the water drop with bigger speed removes more soil from the river's bed than another water drop with smaller speed. The soil removal is thus related to the velocities of water drops.

It has been said earlier that when a water drop flows on a part of a river's bed, it gains speed. But this increase in velocity depends on the amount of soil of that part. This property of a water drop is expressed below:

- The velocity of a water drop increases more on a path with low soil than a path with high soil.

The velocity of the water drop is changed such that on a path with little amount of soil, the velocity of the water drop is increased more than a path with a considerable amount of soil. Therefore, a path with little soil lets the flowing water drop gather more soil and gain more speed whereas the path with large soil resists more against the flowing water drop such that it lets the flowing water drop gather less soil and gain less speed.

Another property of a natural water drop is that when it faces several paths in the front, it often chooses the easier path. Therefore, the following statement may be expressed:

- A water drop prefers a path with less soil than a path with more soil

The water drop prefers an easier path to a harder path when it has to choose between several branches that exist in the path from the source to destination. The easiness or hardness of a path is denoted by the amount of soil on that path. A path with more soil is considered a hard path whereas a path with less soil is considered an easy path.

4. Intelligent Water Drops (IWDs)

In the previous section, some prominent properties of natural water drops were mentioned. Based on the aforementioned statements, an Intelligent Water Drop (IWD) has been

suggested (Shah-Hosseini, 2007), which possesses a few remarkable properties of a natural water drop. This Intelligent Water Drop, IWD for short, has two important properties:

- The soil it carries, denoted by soil(IWD).
- The velocity that it posses, denoted by velocity(IWD).

For each IWD, the values of both properties, soil(IWD) and velocity(IWD) may change as the IWD flows in its environment. From the engineering point of view, an environment represents a problem that is desired to be solved. A river of IWDs seeks an optimal path for the given problem.

Each IWD is assumed to flow in its environment from a source to a desired destination. In an environment, there are numerous paths from a given source to a desired destination. The location of the destination may be unknown. If the location of the desired destination is known, the solution to the problem is obtained by finding the best (often the shortest) path from the source to the destination. However, there are cases in which the destination is unknown. In such cases, the solution is obtained by finding the optimum destination in terms of cost or any other desired measure for the given problem.

An IWD moves in discrete finite-length steps in its environment. From its current location *i* to its next location *j*, the IWD velocity, velocity (IWD), is increased by an amount Δvelocity (IWD), which is nonlinearly proportional to the inverse of the soil between the two locations *i* an *j*, soil(*i* , *j*):

$$\Delta velocity(IWD) \propto^{NL} \frac{1}{soil(i,j)} \tag{1}$$

Here, nonlinearly proportionality is denoted by $\propto^{NL}$. One possible formula is given below in which the velocity of the IWD denoted by $vel^{IWD}(t)$ is updated by the amount of soil $soil(i,j)$ between the two locations *i* and *j*:

$$\Delta vel^{IWD}(t) = \frac{a_v}{b_v + c_v \,.\, soil^{2\alpha}(i,j)} \tag{2}$$

Here, the a_v, b_v, c_v, and α are user-selected positive parameters.

Moreover, the IWD's soil, soil(IWD), is increased by removing some soil of the path joining the two locations *i* an *j*. The amount of soil added to the IWD, Δsoil(IWD)= Δsoil(*i,j*), is inversely (and nonlinearly) proportional to the time needed for the IWD to pass from its current location to the next location denoted by time(*i,j; IWD*).

$$\Delta soil(IWD) = \Delta soil(i,j) \propto^{NL} \frac{1}{time(i,j;IWD)} \tag{3}$$

One suggestion for the above formula is given below in which $time(i,j;vel^{IWD})$ is the time taken for the IWD with velocity vel^{IWD} to move from location *i* to *j*. The soil added to the IWD is calculated by

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s \,.\, time^{2\theta}\left(i,j;vel^{IWD}\right)} \tag{4}$$

Where the parameters a_s, b_s, c_s, and θ are user-selected and should be chosen as positive numbers.

The duration of time for the IWD is calculated by the simple laws of physics for linear motion. Thus, the time taken for the IWD to move from location *i* to *j* is proportional to the velocity of the IWD, velocity (IWD), and inversely proportional to the distance between the two locations, d(*i,j*). More specifically:

$$time(i,j;IWD) \propto^{L} \frac{1}{velocity(IWD)} \tag{5}$$

Where $\propto^{L}$ denotes linear proportionality. One such formula is given below, which calculates the time taken for the IWD to travel from location *i* to *j* with velocity vel^{IWD}:

$$time\left(i,j;vel^{IWD}\right) = \frac{HUD(i,j)}{vel^{IWD}} \tag{6}$$

Where a local heuristic function $HUD(.,.)$ has been defined for a given problem to measure the undesirability of an IWD to move from one location to the next.

Some soil is removed from the visited path between locations *i* and *j*. The updated soil of the path denoted by soil(*i,j*) is proportional to the amount of soil removed by the IWD flowing on the path joining *i* to *j*, Δsoil(*i,j*)= Δsoil(IWD). Specifically:

$$soil(i,j) \propto^{L} \Delta soil(i,j) \tag{7}$$

One such formula has been used for the IWD algorithm such that $soil(i,j)$ is updated by the amount of soil removed by the IWD from the path *i* to *j*.

$$soil(i,j) = \rho_o \,.\, soil(i,j) - \rho_n \,.\, \Delta soil(i,j) \tag{8}$$

Where ρ_o and ρ_n are often positive numbers between zero and one. In the original IWD algorithm for the TSP (Shah-Hosseini, 2007), $\rho_o = 1 - \rho_n$.

The soil of the IWD denoted by $soil^{IWD}$ is added by the amount $soil(i,j)$ as shown below:

$$soil^{IWD} = soil^{IWD} + \Delta soil(i,j) \tag{9}$$

Another mechanism that exists in the behaviour of an IWD is that it prefers the paths with low soils on its beds to the paths with higher soils on its beds. To implement this behaviour of path choosing, a uniform random distribution is used among the soils of the available paths such that the probability of the IWD to move from location *i* to *j* denoted by p(i,j;IWD) is inversely proportional to the amount of soils on the available paths.

$$p(i,j;IWD) \propto^{L} soil(i,j) \tag{10}$$

The lower the soil of the path between locations *i* and *j*, the more chance this path has for being selected by the IWD located on *i*. One such formula based on Eq. (10) has been used in which the probability of choosing location *j* is given by:

$$p(i,j;IWD)=\frac{f\big(soil(i,j)\big)}{\sum_{k\notin vc(IWD)} f\big(soil(i,k)\big)} \tag{11}$$

Where $f(soil(i,j))=\dfrac{1}{\varepsilon_s+g(soil(i,j))}$. The constant parameter ε_s is a small positive number to prevent a possible division by zero in the function $f(.)$. The set $vc(IWD)$ denotes the nodes that the IWD should not visit to keep satisfied the constraints of the problem.

The function $g(soil(i,j))$ is used to shift the $soil(i,j)$ of the path joining nodes *i* and *j* toward positive values and is computed by

$$g(soil(i,j))=\begin{cases} soil(i,j) & if \min\limits_{l\notin \text{vc(IWD)}}(soil(i,l))\geq 0 \\ soil(i,j)-\min\limits_{l\notin vc(IWD)}(soil(i,l)) & else \end{cases} \tag{12}$$

Where the function $\min(.)$ returns the minimum value of its arguments.

The IWDs work together to find the optimal solution to a given problem. The problem is encoded in the environment of the IWDs, and the solution is represented by the path that the IWDs have converged to. In the next section, the IWD algorithm is explained.

5. The Intelligent Water Drops (IWD) algorithm

The IWD algorithm employs a number of IWDs to find the optimal solutions to a given problem. The problem is represented by a graph (N,E) with the node set *N* and edge set *E*. This graph is the environment for the IWDs and the IWDs flow on the edges of the graph. Each IWD begins constructing its solution gradually by traveling between the nodes of the graph along the edges until the IWD finally completes its solution denoted by T^{IWD}. Each solution T^{IWD} is represented by the edges that the IWD has visited. One iteration of the IWD algorithm is finished when all IWDs complete their solutions. After each iteration, the iteration-best solution T^{IB} is found. The iteration-based solution T^{IB} is the best solution based on a quality function among all solutions obtained by the IWDs in the current iteration. T^{IB} is used to update the total-best solution T^{TB}. The total-best solution T^{TB} is the best solution since the beginning of the IWD algorithm, which has been found in all iterations.

For a given problem, an objective or quality function is needed to measure the fitness of solutions. Consider the quality function of a problem to be denoted by $q(.)$. Then, the quality of a solution T^{IWD} found by the IWD is given by $q(T^{IWD})$. Therefore, the iteration-best solution T^{IB} is given by:

$$T^{IB}=\arg\max_{for\ all\ IWDs} q(T^{IWD}) \tag{13}$$

Where arg(.) returns its argument.

It should be noted that at the end of each iteration of the algorithm, the total-best solution T^{TB} is updated by the current iteration-best solution T^{IB} as follows:

$$T^{TB} = \begin{cases} T^{TB} & if \ \ q(T^{TB}) \geq q(T^{IB}) \\ T^{IB} & otherwise \end{cases} \tag{14}$$

Moreover, at the end of each iteration of the IWD algorithm, the amount of soil on the edges of the iteration-best solution T^{IB} is reduced based on the goodness (quality) of the solution. One such mechanism (Shah-Hosseini, 2007) is used to update the $soil(i,j)$ of each edge (i,j) of the iteration-best solution T^{IB}:

$$\begin{aligned} & soil(i,j) = \rho_s \,.\, soil(i,j) \\ & -\rho_{IWD} \cdot \frac{1}{(N_{IB}\text{-}1)} \,.\, soil_{IB}^{IWD} \quad \forall (i,j) \in T^{IB} \end{aligned} \tag{15}$$

Where $soil_{IB}^{IWD}$ represents the soil of the iteration-best IWD. The iteration-best IWD is the IWD that has constructed the iteration-best solution T^{IB} at the current iteration. N_{IB} is the number of nodes in the solution T^{IB}. ρ_{IWD} is the global soil updating parameter, which should be chosen from $[0,1]$. ρ_s is often set as $(1+\rho_{IWD})$.

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 IWD algorithm stops when it reaches the maximum number of iterations $iter_{max}$ or the total-best solution T^{TB} achieves the expected quality demanded for the given problem.

The IWD algorithm has two kinds of parameters:

- Static parameters
- Dynamic parameters

Static parameters are those parameters that remain constant during the lifetime of the IWD algorithm. That is why they are called "static". Dynamic parameters are those parameters, which are dynamic and they are reinitialized after each iteration of the IWD algorithm.

The IWD algorithm as expressed in (Shah-Hosseini, 2008b) is specified in the following ten steps:

1. Initialization of static parameters:

- The graph (N,E) of the problem is given to the algorithm, which contains N_c nodes.
- The quality of the total-best solution T^{TB} is initially set to the worst value: $q(T^{TB}) = -\infty$.
- The maximum number of iterations $iter_{\max}$ is specified by the user and the algorithm stops when it reaches $iter_{\max}$.
- The iteration count $iter_{count}$, which counts the number of iterations, is set to zero.
- The number of water drops N_{IWD} is set to a positive integer value. This number should at least be equal to two. However, N_{IWD} is usually set to the number of nodes N_c of the graph.
- Velocity updating parameters are a_v, b_v, and c_v. Here, $a_v = c_v = 1$ and $b_v = 0.01$
- Soil updating parameters are a_s, b_s, and c_s. Here, $a_s = c_s = 1$ and $b_s = 0.01$
- The local soil updating parameter is ρ_n. Here, $\rho_n = 0.9$ except for the AMT, which is $\rho_n = -0.9$.
- The global soil updating parameter is ρ_{IWD}. Here, $\rho_{IWD} = 0.9$.

- The initial soil on each edge of the graph is denoted by the constant $InitSoil$ such that the soil of the edge between every two nodes i and j is set by $soil(i,j) = InitSoil$. Here, $InitSoil = 10000$
- The initial velocity of each IWD is set to $InitVel$. Here, $InitVel = 200$ except for the MKP, which is $InitVel = 4$.
- Initialization of dynamic parameters:
- Every IWD has a visited node list $V_c(IWD)$, which is initially empty: $V_c(IWD) = \{\ \}$.
- Each IWD's velocity is set to $InitVel$.
- All IWDs are set to have zero amount of soil.

2. Spread the IWDs randomly on the nodes of the graph as their first visited nodes.
3. Update the visited node list of each IWD to include the nodes just visited.
4. Repeat steps 5.1 to 5.4 for those IWDs with partial solutions.

5.1. For the IWD residing in node i, choose the next node j, which doesn't violate any constraints of the problem and is not in the visited node list $vc(IWD)$ of the IWD, using the following probability $p_i^{IWD}(j)$:

$$p_i^{IWD}(j) = \frac{f\left(soil(i,j)\right)}{\sum_{k \notin vc(IWD)} f\left(soil(i,k)\right)} \tag{16}$$

such that

$$f(soil(i,j)) = \frac{1}{\varepsilon_s + g(soil(i,j))} \tag{17}$$

and

$$g(soil(i,j)) = \begin{cases} soil(i,j) & if \min_{l \notin vc(IWD)} (soil(i,l)) \geq 0 \\ soil(i,j) - \min_{l \notin vc(IWD)} (soil(i,l)) & else \end{cases} \tag{18}$$

Then, add the newly visited node j to the list $vc(IWD)$.

5.2. For each IWD moving from node i to node j, update its velocity $vel^{IWD}(t)$ by

$$vel^{IWD}(t+1) = vel^{IWD}(t) + \frac{a_v}{b_v + c_v \,.\, soil^2(i,j)} \tag{19}$$

where $vel^{IWD}(t+1)$ is the updated velocity of the IWD.

5.3. For the IWD moving on the path from node i to j, compute the soil $\Delta soil(i,j)$ that the IWD loads from the path by

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s \,.\, time^2\left(i, j; vel^{IWD}(t+1)\right)} \tag{20}$$

such that $time\left(i, j; vel^{IWD}(t+1)\right) = \frac{HUD(j)}{vel^{IWD}(t+1)}$ where the heuristic undesirability $HUD(j)$ is defined appropriately for the given problem.

5.4. 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 $soil^{IWD}$ by

$$\begin{aligned} soil(i,j) &= (1-\rho_n).soil(i,j) - \rho_n.\Delta soil(i,j) \\ soil^{IWD} &= soil^{IWD} + \Delta soil(i,j) \end{aligned} \tag{21}$$

6. Find the iteration-best solution T^{IB} from all the solutions T^{IWD} found by the IWDs using

$$T^{IB} = \arg\max_{\forall T^{IWD}} q(T^{IWD}) \tag{22}$$

where function $q(.)$ gives the quality of the solution.

7. Update the soils on the paths that form the current iteration-best solution T^{IB} by

$$\begin{aligned} soil(i,j) &= (1+\rho_{\text{IWD}}).soil(i,j) \\ &- \rho_{IWD}.\frac{1}{(N_{IB}-1)}.soil_{IB}^{IWD} \quad \forall (i,j) \in T^{IB} \end{aligned} \tag{23}$$

where N_{IB} is the number of nodes in solution T^{IB}.

8. Update the total best solution T^{TB} by the current iteration-best solution T^{IB} using

$$T^{TB} = \begin{cases} T^{TB} & if \;\; q(T^{TB}) \geq q(T^{IB}) \\ T^{IB} & otherwise \end{cases} \tag{24}$$

9. Increment the iteration number by $Iter_{count} = Iter_{count} + 1$. Then, go to step 2 if $Iter_{count} < Iter_{\max}$.

10. The algorithm stops here with the total-best solution T^{TB}.

The IWD algorithm may be compared to the ant-based optimization algorithms (Bonabeau et al., 1999), which is summarized below:

- Every ant in an Ant Colony Optimization (ACO) algorithm deposits pheromones on each edge it visits. In contrast, an IWD changes the amount of soil on edges.
- In the ACO algorithm, an ant cannot remove pheromones from an edge whereas in the IWD algorithm, an IWD can both remove and add soil to an edge.
- In the IWD algorithm, the changes made on the soil of an edge are not constant and they are dependent on the velocity and soil of the IWDs visiting that edge. In contrast, in the ACO algorithm, each ant deposits a constant amount of pheromone on the edge.
- Besides, the IWDs may gain different velocities throughout an iteration of the IWD algorithm whereas in ACO algorithms the velocities of the ants are irrelevant.

6. Convergence properties of the IWD algorithm

Let the graph (N,E) represents the graph of the given problem. This graph is assumed to be a fully connected graph with N_c nodes. Let N_{IWD} represents the number of IWDs in the IWD algorithm. In the soil updating of the algorithm, two extreme cases are considered:

Case one: Only those terms of the IWD algorithm, which increase soil to an edge (arc) of (N, E), are considered.

Case two: Only those terms of the IWD algorithm, which decrease soil to an edge (arc) of (N, E), are considered.

For each case, the worst-case is followed. For case one, the highest possible value of soil that an edge can hold after m iterations, $soil(edge_{\max})$, will be (Shah-Hosseini, 2008a):

$$soil(edge_{\max}) = \left((\rho_s \rho_o)^m IS_0 \right) \tag{25}$$

Where the edge is denoted by $edge_{\max}$. IS_0 is the initial soil of an edge (i, j), which is denoted by $InitSoil$ in the IWD algorithm. ρ_0 is used in Eq. (8) and ρ_s is used in Eq. (15).

For case two, the lowest possible value of soil for an edge is computed. That edge is denoted by $edge_{\max}$. Then, after m iterations, the soil of $edge_{\max}$ is (Shah-Hosseini, 2008a):

$$soil(edge_{\min}) = \left(m\left(\rho_{IWD} - \rho_n N_{IWD}\right)\left(\frac{a_s}{b_s}\right) \right) \tag{26}$$

Where N_{IWD} is the number of IWDs. Soil updating parameters a_s and b_s are defined in Eq. (20). ρ_{IWD} is the global soil updating parameter used in Eq. (23). ρ_n is the local soil updating parameter used in Eq. (21).

Based on Eqs. (25) and (26), the following proposition is stated:

Proposition 6.1. The soil of any edge in the graph (N, E) of a given problem after m iterations of the IWD algorithm remains in the interval $\left[soil(edge_{\min}), soil(edge_{\max})\right]$.

The probability of finding any feasible solution by an IWD in iteration m is $(p_{lowest})^{(N_c - 1)}$ where the probability of any IWD, going from node *i* to node *j*, is always bigger than p_{lowest}. It has been shown (Shah-Hosseini, 2008a) that p_{lowest} is always a positive value. Since there are N_{IWD} IWDs, then the probability $p(s; m)$ of finding any feasible solution s by the IWDs in iteration m is:

$$p(s; m) = N_{IWD} (p_{lowest})^{(N_c - 1)} \tag{27}$$

The probability of finding any feasible solution s at the end of M iterations of the algorithm is:

$$P(s; M) = 1 - \prod_{m=1}^{M} \left(1 - p(s; m)\right) \tag{28}$$

Because $0 < p(s; m) \le 1$, then by making M large enough, it is concluded that: $\lim_{M \to \infty} \prod_{m=1}^{M} \left(1 - p(s; m)\right) = 0$. Therefore, the following proposition is true:

Proposition 6.2. If $P(s; M)$ represents the probability of finding any feasible solution s within M iterations of the IWD algorithm. As M gets larger, $P(s; M)$ approaches to one:

$$\lim_{M \to \infty} P(s; M) = 1 \tag{29}$$

Knowing the fact that the optimal solution s^* is a feasible solution of the problem, from above proposition, the following proposition is concluded.

Proposition 6.3. The IWD algorithm finds the optimal solution s^* of any given problem with probability one if the number of iterations M is sufficiently large.

It is noticed that the required M to find the optimal solution s^* should be decreased by careful tuning of parameters of the IWD algorithm for the given problem.

7. Experimental results

Every IWD in the IWD algorithm both searches and changes its environment. During this search, the IWD incrementally constructs a solution. The problem definition is presented to the IWD algorithm in the form of a graph and IWDs visit nodes of the graph by travelling on the edges of the graph. A swarm of IWDs flows in the graph often with the guidance of a local heuristic in the hope of finding optimal or near optimal solutions. In the following, the IWD algorithm is used for four different problems: the TSP, the *n*-queen puzzle, the MKP, and for the first time, the IWD algorithm is used for Automatic Multilevel Thresholding (AMT).

7.1 The Travelling Salesman Problem (TSP)

In the TSP (travelling salesman problem), a map of cities is given to the salesman and he is required to visit every city only once one after the other to complete his tour and return to its first city. The goal in the TSP is to find the tour with the minimum total length among all such possible tours for the given map.

A TSP is represented by a graph (N, E) where the node set N denotes the n cities of the TSP and the edge set E denotes the edges between cities. Here, the graph of the TSP is considered a complete graph. Thus, every city has a direct link to another city. It is also assumed that the link between each two cities is undirected. So, in summary, the graph of the TSP is a complete undirected graph. A solution of the TSP having the graph (N, E) is an ordered set of n distinct cities. For such a TSP with n cities, there are $(n-1)!/2$ feasible solutions in which the global optimum(s) is sought.

A TSP solution for an n-city problem may be represented by the tour $T = (c_1, c_2, ..., c_n)$. The salesman travels from city c_1 to c_2, then from c_2 to c_3, and he continues this way until it gets to city c_n. Then, he returns to the first city c_1, which leads to the tour length $TL(.)$, defined by:

$$TL(c_1, c_2, ..., c_n) = \sum_{i=1}^{n} d(c_i, c_{i+1}) \tag{30}$$

such that $c_{n+1} = c_1$, and $d(.,.)$ is the distance function, which is often the Euclidean distance. The goal is to find the optimum tour $T^* = (c_1^*, c_2^*, ..., c_n^*)$ such that for every other feasible tour T:

$$\forall T: \quad TL(T^*) \le TL(T) \tag{31}$$

In order to use the IWD algorithm for the TSP, the TSP problem as mentioned above is viewed as a complete undirected graph (N, E). Each link of the edge set E has an amount

of soil. An IWD visits nodes of the graph through the links. The IWD is able to change the amount of the soils on the links. Moreover, cities of the TSP are denoted by nodes of the graph, which hold the physical positions of cities. An IWD starts its tour from a random node and it visits other nodes using the links of the graph until it returns to the first node. The IWD changes the soil of each link that it flows on while completing its tour.

For the TSP, the constraint that each IWD never visits a city twice in its tour must be kept satisfied. Therefore, for the IWD, a visited city list $V_c(IWD)$ is employed. This list includes the cities visited so far by the IWD. So, the next possible cities for an IWD are selected from those cities that are not in the visited list $V_c(IWD)$ of the IWD.

One possible local heuristic for the TSP, denoted by $HUD_{TSP}(i, j)$, has been suggested (Shah-Hosseini, 2008a) as follows:

$$HUD_{TSP}(i, j) = \| \mathbf{c}(i) - \mathbf{c}(j) \| \tag{32}$$

where $\mathbf{c}(k)$ denotes the two dimensional positional vector for the city k. The function $\| . \|$ denotes the Euclidean norm. The local heuristic $HUD_{TSP}(i, j)$ measures the undesirability of an IWD to move from city i to city j. For near cities i and j, the heuristic measure $HUD(i, j)$ becomes small whereas for far cities i and j, the measure $HUD(i, j)$ becomes big. It is reminded that paths with high levels of undesirability are chosen fewer times than paths with low levels of undesirability. In the IWD algorithm, the time taken for the IWD to pass from city i to city j, is proportional to the heuristic $HUD_{TSP}(i, j)$.

A modification to the IWD-TSP has been proposed in (Shah-Hosseini, 2008b), which finds better tours and hopefully escape local optimums. After a few number of iterations, say N_I, the soils of all paths (i, j) of the graph of the given TSP are reinitialized again with the initial soil $InitSoil$ except the paths of the total-best solution T^{TB}, which are given less soil than $InitSoil$. The soil reinitialization after each N_I iterations is expressed in the following equation:

$$soil(i, j) = \begin{cases} \alpha_I \Gamma_I InitSoil & for\ every\ (i, j) \in T^{TB} \\ InitSoil & otherwise \end{cases} \tag{33}$$

where α_I is a small positive number chosen here as 0.1. Γ_I denotes a random number, which is drawn from a uniform distribution in the interval $[0, 1]$. As a result, IWDs prefer to choose paths of T^{TB} because less soil on its paths is deposited. Here, $N_I = 15$.

We may refer to the IWD algorithm for the TSP as the "IWD-TSP" algorithm. Moreover, the modified IWD algorithm for the TSP is called the "MIWD-TSP" algorithm.

The MIWD-TSP algorithm is tested with cities on a circle such that the cities are equally placed on the perimeter of the circle (Shah-Hosseini, 2008b). The number of cities is chosen from 10 to 100 cities incremented by 10. The results of this experiment are shown in Table 1 in which the average numbers of iterations to get to the global optimums are depicted. The number of IWDs is kept at the constant value 50. As the number of cities increases, the average number of iterations to find the global optimum almost monotonically increases.

No. of cities on the circle	10	20	30	40	50	60	70	80	90	100
Ave. iterations	10.4	39.6	78.6	85.5	134.5	181.9	253.1	364.7	387.4	404.5

Table 1. The average number of iterations to find the global optimum for the TSP where cities are equally spaced on the perimeter of a circle. The results are the average numbers of iterations of ten runs of the MIWD-TSP algorithm.

The IWD-TSP algorithm (Shah-Hosseini, 2007) for the TSPs of Table 1 often gets trapped into local optimums. As the number of cities of the TSP increases, the average numbers of iterations to get to the global optimums become high. For the TSPs with high numbers of cities, it is impractical to use the IWD-TSP for them, and the MIWD-TSP is recommended. In Fig. 1, such a local optimum is shown for the TSP with 10 cities along the tour obtained by the MIWD-TSP algorithm.

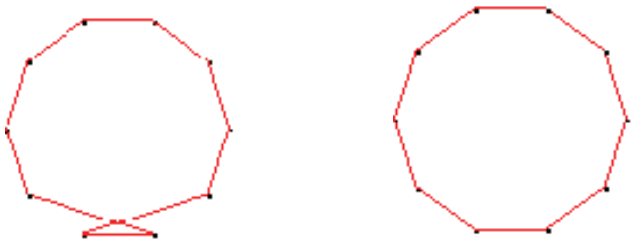


Fig. 1. The IWD-TSP converges to a good local optimum in the left image. However, it has a small self-crossing in its tour for the TSP with 10 cities. In contrast, by using the MIWD-TSP, the self-crossing is removed and it converges to the tour in the right image.

Four TSPs are taken from the TSPLIB95 (the TSP Library in the Internet) to test the performance of the MIWD-TSP algorithm (Shah-Hosseini, 2008b). The lengths of average and best tours in five runs of the MIWD-TSP algorithm are reported in Table 2. For comparison, the lengths of best tours of some other nature-inspired algorithms are also mentioned in Table 2. The table shows that the tours obtained by the MIWD-TSP algorithm are satisfactorily close to the known optimums and are comparable to the other nature-inspired algorithms.

Problem Name	Optimum length	Method					
		MMAS	BCO	EA	Improved ACO	MIWD-TSP Best	MIWD-TSP Average
eil51	426	426	431.13	---	428.87	428.98	432.62
eil76	538	---	---	544.36	---	549.96	558.23
st70	675	---	678.62	677.10	677.10	677.10	684.08
kroA100	21282	21282	21441.5	21285.44	---	21407.57	21904.03

Table 2. The comparison between the Modified IWD-TSP and four other nature-inspired algorithms MMAS (Stutzle & Hoos, 1996), BCO (Teodorovic et al., 2006), EA (Yan et al., 2005), Improved ACO (Song et al., 2006) for the four TSPs mentioned below. The MIWD-TSP iterations: 3000 for eil51, 4500 for eil76, and 6000 for st70 and kroA100.

The convergence curve for the TSP "eil51" with the MIWD-TSP algorithm is shown in Fig. 2 in which the tour length of the total-best solution versus iterations is depicted. Here, the algorithm converges to the tour of length 428.98, which is shown in the figure.

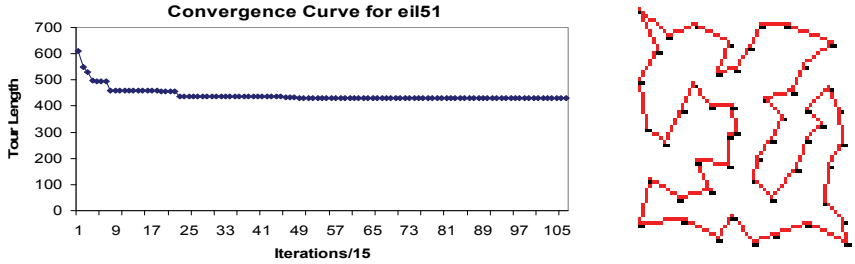


Fig. 2. The MIWD-TSP algorithm converges to the tour length 428.98 for eil51. The converged tour is shown on the right side.

7.2 The *n*-queen puzzle

The *n*-queen puzzle is the problem in which *n* chess queens must be placed on an n×n chessboard such that no two queens attack each other (Watkins, 2004). The n-queen puzzle is the generalization of the 8-queen puzzle with eight queens on an 8×8 chessboard. The 8-quuen puzzle was originally proposed by the chess player "Max Bezzel" in 1848. Many mathematicians such as "Gauss" and "Cantor" have worked on the 8-queen puzzle and its generalized n-queen puzzle. The first solutions were provided by "Franz Nauck" in 1850. He also generalized it to the n-queen puzzle on an n×n chessboard. The 8-queen puzzle has 92 distinct solutions. If the solutions that differ only by rotations and reflections of the board are counted as one, the 8-queen puzzle has 12 unique solutions. Except for n=6, as the n increases, the number of unique (or distinct) solutions increases. A solution to the 8-queen puzzle is depicted in Fig. 3.

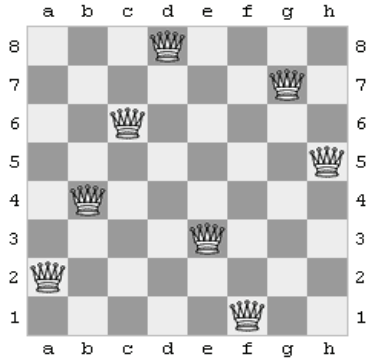


Fig. 3. A solution to the 8-queen puzzle on the chessboard.

One strategy is to put all n queens instantly on the n×n chessboard, which leads to the hug search space n^{2n} (wrongly written 64^n in (Shah-Hosseini, 2008b)). For $n=8$, the search space contains $64^8 = 2^{48} \cong 2.8\times10^{14}$. To reduce the size of the search space of the *n*-queen

problem, the n queens are placed one by one on the chessboard such that the first queen is placed on any row of the first column. Then, the second queen is placed on any row of the second column except the row of the first queen. Following this strategy, the ith queen is placed on any row of the ith column except those rows that previous queens have occupied. This incremental strategy of putting queens on the chessboard reduces the search space to $n!$ where the symbol "!" denotes the factorial.

In the incremental strategy, if every row of the chessboard is considered a city, then the n-queen problem may be considered as a TSP. The first row chosen by the first queen is considered the first city of the tour. The second row chosen by the second queen is called the second city of the tour. Continuing this way, the ith row chosen by the ith queen is considered the ith city of the tour. The constraint that no two queens are in the same row is viewed as no two cities of the TSP graph are visited by the salesman. In summary, for the n-queen problem, a complete undirected TSP graph is created.

In the n-queen puzzle, any feasible solution is also an optimal solution. Any feasible solution for an n-queen puzzle is the solution in which no two queens attack each other and that is the optimal solution. For this problem, the path to reach the final feasible (optimal) solution(s) is not wanted and in fact only the final positions of queens on the chessboard are desired.

The local heuristic undesirability $HUD_{NQ}(i, j)$, which has been proposed in (Shah-Hosseini, 2008b) to solve the n-queen puzzle, is stated as follows:

$$HUD_{NQ}(i, j) = (1 + r)\left| |i - j| - \frac{n}{2} \right| \tag{34}$$

where $HUD_{NQ}(i, j)$ is the undesirability of an IWD to go from current row (city) i to the next row (city) j. Symbol $|.|$ denotes the absolute value. The variable r is a random number chosen uniformly from the interval $[0, 1]$. The symbol n denotes the number of cities (columns or rows) of the chessboard of size $n \times n$. In the above equation, the heuristic favours the distance between the rows of neighbouring columns to be near the length $\frac{n}{2}$.

However, it has been observed that the IWD algorithm with the mentioned local heuristic in Eq. (34) is usually trapped in the local optima in which only two queens attack each other. Sometimes, coming out of such local optima takes considerable iterations of the algorithm. For this purpose, a local search algorithm called "N-Queen Local Search" or NQLS has been proposed in (Shah-Hosseini, 2008b). This local search algorithm is activated when the current iteration-best solution T^{IB} of the IWD algorithm contains only two queens attacking each other. Specifically, the NQLS algorithm is activated when the quality of the iteration-best solution $T^{IB}, q(T^{IB})$, becomes -1. The quality of a solution T denoted by $q(T)$ is measured by

$$q(T) = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} attack(c_i, c_j) \tag{35}$$

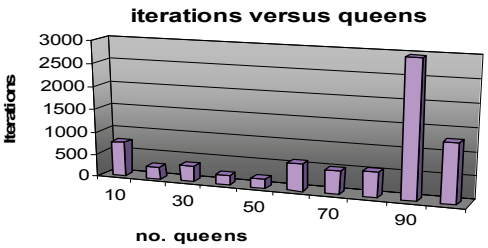
Where $attack(c_i, c_j) = 1$ if queens c_i and c_j attack each other. Otherwise; $attack(c_i, c_j) = 0$. It is reminded that the optimal solution T^* has the highest quality value zero: $q(T^*) = 0$. In the following, the NQLS is expressed in four steps:

1. Get the iteration-best solution T^{IB} with tour $\left(c_1^{IB}, c_2^{IB}, ..., c_n^{IB}\right)$ and $q(T^{IB}) = -1$.
2. initialize $T_0 = T^{IB}$.
2. For $k = 1,2,..., n-1$ do the following steps (steps 2.1 to 2.3):

   2.1. Shift the cities in the tour one position to the right such that the last city becomes the first city in the tour: $T_k = shift_{Right}(T_{k-1})$.
   2.2. If $q(T_k) = 0$, then set $T_0 = T_k$ and go to step 4.
   2.3. End loop.
3. For $k = 1,2,..., n-1$ do the following steps (steps 3.1 to 3.3):

   3.1. Increment each city's number (row) by one such that the highest row becomes the lowest row in the chessboard: $T_k = (T_{k-1} + k) \bmod n$, where $\bmod$ is the modulus function. Moreover, the increment inside the parenthesis is applied to each city of the tour T_{k-1}.
   3.2. If $q(T_k) = 0$, then set $T_0 = T_k$ and go to step 4.
   3.3. End loop.
4. If $q(T_0) = 0$, then the total-best iteration solution T^{TB} has been obtained and is updated by $T^{TB} = T_0$; otherwise, no updating is implemented by this algorithm.

For simplicity, the IWD algorithm for the n-queen problem using the local search NQLS is called "IWD-NQ" algorithm. The IWD-NQ algorithm is tested with ten different n-queens puzzle (Shah-Hosseini, 2008b) where n is increased from 10 to 100 with step ten. The average number of iterations needed to find the optimal solution for ten runs of each n-queen puzzle is depicted in Fig. 4. It is reminded that 50 IWDs are used in the experiments. It is seen that the number of iterations to find the optimal solution(s) does not necessarily depend on the number of queens. For example, the average number of iterations for the 90-queen problem is bigger than the 100-queen problem.



	10	20	30	40	50	60	70	80	90	100
ave. iterations	759	261	348	210	190	602	510	547	2951	1294

Fig. 4. The average number of iterations to find the global optimums versus the number of queens for the n-queen puzzle. The results are the average iterations of ten runs of the IWD-NQ algorithm. The number of queens changes from 10 to 100.

7.3 The multidimensional knapsack problem

The knapsack problem or KP is to select a subset of items i of the set I, each item i with the profit b_i and resource (capacity) requirement r_i such that they all fit in a knapsack of limited capacity and the sum of profits of the selected items is maximized.

The multidimensional knapsack problem, MKP, is a generalization of the KP. In the MKP, there exists multiple knapsacks and thus there are multiple resource constraints. The inclusion of an item i in the m knapsacks is denoted by setting the variable y_i to one, otherwise y_i is set to zero. Let the variable r_{ij} represents the resource requirement of an item i with respect to the resource constraint (knapsack) j having the capacity a_j. In other words, r_{ij} represents the amount of capacity that item i requires from knapsack j. The MKP with m constraints (knapsacks) and n items wants to maximize the total profit of including a subset of the n items in the knapsacks without surpassing the capacities of the knapsacks. For the MKP, in more specific terms:

$$\max \sum_{i=1}^{n} y_i b_i \tag{36}$$

subject to the following constraints:

$$\sum_{i=1}^{n} r_{ij} y_i \le a_j \ \ for\ j = 1,2,...,m \tag{37}$$

where $y_i \in \{0,1\}$ for $i = 1,2,...,n$. Here, the profits b_i and the resources requirements r_{ij} are non-negative values.

The MKP is an NP-hard combinatorial optimization problem with applications such as cutting stock problems, processor allocation in distributed systems, cargo loading, capital budgeting, and economics.

Two general approaches maybe used for solving the MKP: the exact algorithms and the approximate algorithms. The exact algorithms are useful for solving small to moderate-size instances of the MKP such as those based on dynamic programming and those based on the branch-and-bound approach. For a more detail review on the MKP, Freville (2004) is suggested.

The approximate algorithms may use nature-inspired approaches to approximately solve difficult optimization problems. Nature-inspired algorithms include algorithms such as Simulated Annealing (Kirkpatrick et al., 1983), Evolutionary Algorithms like Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Ant Colony Optimization, Particle Swarm Optimization, Electromagnetism-like optimization, and Intelligent Water Drops (IWDs).

For the MKP, the local heuristic undesirability, denoted by $HUD_{MKP}(j)$, is computed by:

$$HUD_{MKP}(j) = \frac{1}{mb_j} \sum_{k=1}^{m} r_{jk} \tag{38}$$

Where b_j denotes the profit of item j and r_{jk} is the resource requirement for item j from knapsack k. The above equation shows that $HUD_{MKP}(j)$ decreases if the profit b_j is high

whereas $HUD_{MKP}(j)$ increases if the resource requirements of item j are high. As a result, the items with less resource requirements and higher profits are more desirable. $HUD_{MKP}(j)$ represents how undesirable is the action of selecting item j as the next item to be included in the knapsacks. We may refer to the IWD algorithm used for the MKP as the IWD-MKP algorithm.

Problem Name	Constraints × Variables	Quality of Optimum solution	Quality of the IWD-MKP's solution		no. of iterations of the IWD-MKP	
			Best	Average	Best	Average
WEING1	2 × 28	141278	141278	141278	59	1243.8
WEING2	2 × 28	130883	130883	130883	154	618.4
WEING3	2 × 28	95677	95677	95677	314	609.8
WEING4	2 × 28	119337	119337	119337	4	48.5
WEING5	2 × 28	98796	98796	98796	118	698.5
WEING6	2 × 28	130623	130623	130623	71	970.3
WEING7	2 × 105	1095445	1094736	1094223	100	100
WEING8	2 × 105	624319	620872	617897.9	200	200

Table 3. Eight problems of the OR-Library in file "mknap2.txt", solved by the IWD-MKP algorithm. The global optimal solutions have also been mentioned.

Constraints × Variables-Problem Number	LP optimal	L & M	Fidanova	Quality of the IWD-MKP's solutions	
		best	best	best	average
5 × 100-00	24585	24381	23984	24295	24175.4
5 × 100-01	24538	24274	24145	24158	24031.3
5 × 100-02	23895	23551	23523	23518	23404
5 × 100-03	23724	23527	22874	23218	23120.9
5 × 100-04	24223	23991	23751	23802	23737.2
5 × 100-05	24884	24613	24601	24601	24554
5 × 100-06	25793	25591	25293	25521	25435.6
5 × 100-07	23657	23410	23204	23374	23344.9
5 × 100-08	24445	24204	23762	24148	24047
5 × 100-09	24635	24411	24255	24366	24317

Table 4. The MKPs with five constraints and 100 items of the OR-Library in file "mknapcb1.txt" solved by 100 iterations of the IWD-MKP algorithm. The results are compared with the LP optimal solutions and best solutions of two ant-based algorithms: Leguizamon and Michalewicz (L & M), and Fidanova.

The IWD-MKP has been used for eight problems (Shah-Hosseini, 2008b) in file "mknap2.txt" of the OR-Library (the OR-Library in the Internet). For each MKP, the best and the average qualities of ten runs of the IWD-MKP are reported in Table 3. For comparison,

the qualities of optimal solutions are also mentioned in the table. The IWD-MKP algorithm finds the global optimums for the first six MKPs with two constraints and 28 items. However, the qualities of solutions of the problems "WEING7" and "WEING8" with two constraints and 105 items obtained by the IWD-MKP are very close to the qualities of optimal solutions. It is reminded that 50 IWDs are used in the mentioned experiments.
The first ten MKP problems in file "mknapcb1" of the OR-Library are solved by the IWD-MKP and the results of ten runs of the algorithm are shown in Table 4. For comparison, the results of the two Ant Colony Optimization-based algorithms of Leguizamon and Michalewicz (for short, L & M) (Leguizamon and Michalewicz, 1999) and Fidanova (Fidanova, 2002) are mentioned. Moreover, the results obtained by the LP relaxation method that exist in the file "mkcbres.txt" of the OR-Library are also included. The solutions of the IWD-MKP are often better than the solutions of those obtained by Fidanova.

7.4 Automatic multilevel thresholding

Image segmentation is often one of the main tasks in any Computer Vision application. Image segmentation is a process in which the whole image is segmented into several regions based on similarities and differences that exist between the pixels of the input image. Each region should contain an object of the image at hand. Therefore, by doing segmentation, the image is divided into several sub-images such that each sub-image represents an object of the scene.
There are several approaches to perform image segmentation (Sezgin & Sankur, 2004). One of the widely used techniques for image segmentation is multilevel thresholding. In doing multilevel thresholding for image segmentation, it is assumed that each object has a distinct continuous area of the image histogram. Therefore, by separating the histogram appropriately, the objects of the image are segmented correctly.
Multilevel thresholding uses a number of thresholds $\{t_1, t_2, ..., t_M\}$ in the histogram of the image $f(x, y)$ to separate the pixels of the objects in the image. By using the obtained thresholds, the original image is multithresholded and the segmented image $T(f(x, y))$ is created. Specifically:

$$T(f(x,y)) = \begin{cases} g_0 & if\ f(x,y) < t_1 \\ g_1 & if\ t_1 \le f(x,y) < t_2 \\ \dots & \dots \\ g_M & if\ f(x,y) \ge t_M \end{cases} \tag{39}$$

Such that g_i is the grey-level assigned to all pixels of the region i, which eventually represents object i. As it is seen in the above equation, the $M+1$ regions are determined by the M thresholds $\{t_1, t_2, ..., t_M\}$. The value of g_i may be chosen to be the mean value of gray-levels of the region's pixels. However, in this paper we use the maximum range of

gray-levels, 255, to distribute the gray-levels of regions equally. Specifically, $g_i = i.\left[\frac{255}{M}\right]$ such that the function $[.]$ returns the integer value of its argument.

To multithreshold an image, the number of thresholds should be given in advance to a multilevel thresholding algorithm. However, a few algorithms have been designed to automatically determine the suitable number of thresholds based on the application at hand such as the Growing Time-Adaptive Self-Organizing Map (Shah-Hosseini & Safabakhsh, 2002). Such algorithms are called automatic multilevel thresholding. In automatic multilevel thresholding, the problem becomes harder because the search space increase hugely as the number of thresholds increases. For example, if the brightness of each image's pixel be selected from G different gray levels and the image is thresholded by M thresholds, then the search space will be of size $\frac{G!}{(G-M)!}$. For $G = 256$ and $M = 30$, the search space is approximately of size 3×10^{71}. It is reminded that in this paper, it is assumed that $G = 256$.

The AMT (Automatic Multilevel Thresholding) problem is converted to a TSP problem such that the number of cities is assumed to be 256. Each IWD begins its trip from city zero to city 255. There are exactly two directed links from city i to city $i+1$, which are named "Above Link" and "BelowLink". When the IWD is in city i, the next city of the IWD is city $i+1$. If AboveLink is selected by the IWD in city i, it means that $i+1$ is not the next threshold for the AMT. In contrast, if BelowLink is selected by the IWD, it means that the next threshold is the value $i+1$.

Here, a local heuristic is not suggested. As a result, the amount of soil that each IWD removes from its visiting path , $\Delta soil(i,j)$, is computed by:

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s.1000} \tag{40}$$

Where $time^2\left(i,j;vel^{IWD}(t+1)\right)$ in Eq. (20) has been replaced by the constant amount 1000.

Moreover, the local soil updating parameter ρ_n in Eq. (21) is chosen as a negative value. For the AMT, $\rho_n = -0.9$

The quality of each solution found by an IWD, denoted by $q(T^{IWD})$, is suggested to be a generalization of the Haung's method (Huang & Wang, 1995) in which a fuzziness measure is employed for bilevel thresholding. Here, the Haung's method is generalized to be used for multilevel thresholding. The first step is to define the membership function, which returns the degree of membership of any pixel to the class (object) of the image. Again, the thresholds obtained by an IWD are represented by the set $\{t_1, t_2, ..., t_M\}$. The membership function $u_f(I(x,y))$ for each pixel (x,y) of image $I(.,.)$ is defined as follows:

$$u_f(I(x,y)) = \frac{1}{1+\left|I(x,y)-\bar{I}_{k+1}\right|/255} \qquad if \;\; t_k < I(x,y) \le t_{k+1} \tag{41}$$

Such that $\bar{I}_{k+1}$ denotes the average gray-level in a region of the image with gray-levels $\{t_k+1, t_k+2, ..., t_{k+1}\}$. Therefore, $\bar{I}_{k+1} = \frac{\sum_{i=t_k+1}^{t_{k+1}} i\ p(i)}{\sum_{i=t_k+1}^{t_{k+1}} p(i)}$. The function $|.|$ returns the absolute value of its argument. The values of the membership function $u_f(I(x,y))$ vary within the interval $[0.5, 1]$. When M is set to one, the formula of Eq. (41) is reduced to the Haung's method for bilevel thresholding.

Given the membership function, a measure must be introduced to determine the fuzziness of the thresholded image with the original one. For this purpose, the idea introduced by Yager (Yager, 1979) is employed. The measure of fuzziness is defined as how much the fuzzy set and its complement set are indistinct. Having the membership function $u_f(I(x,y))$, the fuzziness measure is defined as:

$$D_f = \sqrt{\sum_{i=0}^{255} \left(2\,u_f(i) - 1\right)^2} \tag{42}$$

For a crisp set, the value of D_f is 16 whereas for the fuzziest set the value of D_f is zero. Thus, the IWD algorithm should try to maximize D_f by finding the best threshold values with the optimum number of thresholds. It is reminded that the quality of each IWD's solution T^{IWD}, denoted by $q(T^{IWD})$, is computed by D_f in Eq. (42).

We may refer to the IWD algorithm used for the AMT as the IWD-AMT algorithm.

The IWD-AMT is tested with three gray-level images, House, Eagle, and Lena shown in Fig. 5. For this purpose, 50 IWDs are used and the results are shown in Fig. 6.

Fig. 5. The three test gray-level images. From left to right: House, Eagle, and Lena.

The histogram of the original images and their thresholded images obtained by the IWD-AMT are shown in Figs. 7-8. The most important features of the images have been preserved whereas the numbers of gray-levels have been reduced dramatically. Specifically, the house image is reduced to 12 gray-levels, the eagle image is reduced to 18 gray-levels and the Lena image is reduced to eight regions. The experiments show that some regions have very small

numbers of pixels. Therefore, it would be useful to remove very small regions by a simple postprocessing while keeping the qualities of the thresholded images intact.

Fig. 6. The three thresholded images obtained by the proposed IWD-AMT algorithm. From left to right: thresholded House with 12 regions, thresholded Eagle with 18 regions, and the thresholded Lena with eight regions.

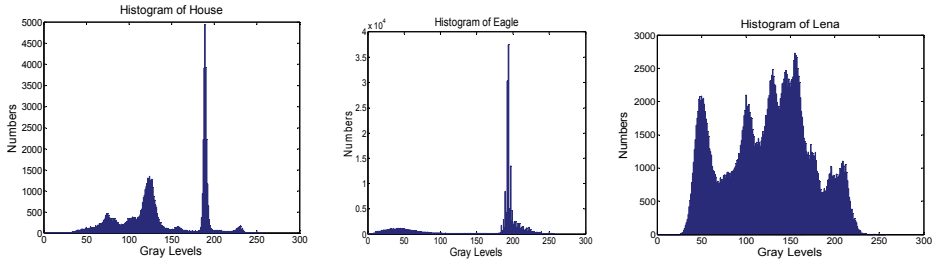


Fig. 7. The histograms of the House, the Eagle, and Lena image.

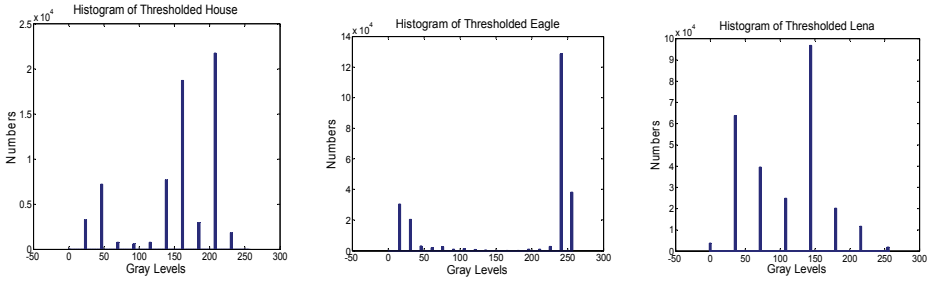


Fig. 8. The histograms of the thresholded images of House, Eagle, and Lena.

8. Conclusion

Four different problems are used to test the IWD algorithm: the TSP (Travelling Salesman Problem), the n-queen puzzle, the MKP (Multidimensional Knapsack Problem), and the AMT (Automatic Multilevel Thresholding). The experiments indicate that the IWD algorithm is capable to find optimal or near optimal solutions. However, there is an open space for modifications in the standard IWD algorithm, embedding other mechanisms that exist in natural rivers, inventing better local heuristics that fit better with the given problem,

and suggesting new representations of the given problem in form of graphs. The IWD algorithm demonstrates that the nature is an excellent guide for designing and inventing new nature-inspired optimization algorithms.

9. References

Birbil, I. & Fang, S. C. (2003). An electro-magnetism-like mechanism for global optimization, *Journal of Global Optimization*, Vol. 25, pp. 263-282.

Bonabeau, E.; Dorigo, M. & Theraultz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press.

Dawkins, R. (1989). *The Selfish Gene*, Oxford University Press.

de Castro, L. N. & Von Zuben, F. J. (2002). Learning and Optimization Using the Clonal Selection Principle, *IEEE Trans. on Evolutionary Computation*, Vol. 6, No. 3, pp. 239-251.

Dorigo, M.; Maniezzo, V. & Colorni, A. (1991). Positive feedback as a search strategy, *Technical Report 91-016*, Dipartimento di Elettronica, Politecnico di Milano, Milan.

Duan, H.; Liu S. & Lei, X. (2008). Air robot path planning based on Intelligent Water Drops optimization, *IEEE IJCNN 2008*, pp. 1397 – 1401.

Fidanova, S. (2002). Evolutionary algorithm for multidimensional knapsack problem, *PPSNVII-Workshop*.

Fogel, L.J.; Owens, A.J. & Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*, New York: John Wiley.

Forrest, S.; Perelson, A.S.; Allen, L. & Cherukuri, R. (1994). Self-nonself discrimination in a computer, *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pp. 202-212.

Freville, A. (2004). The multidimensional 0–1 knapsack problem: An overview, *European Journal of Operational Research*, Vol. 155, pp. 1–21.

Huang, L.K. & Wang, M.J.J. (1995). Image thresholding by minimizing the measures of fuzziness, *Pattern Recognition*, Vol. 28, No. 1, pp. 41-51.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization, *Proc. of the IEEE Int. Conf. on Neural Networks*, pp. 1942–1948, Piscataway, NJ.

Kirkpatrick, S.; Gelatt, C. D. & Vecchi, M. P. (1983). Optimization by simulated annealing, *Science*, Vol. 220, pp. 671-680.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by means of Natural Evolution*, MIT Press, Massachusetts.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Leguizamon, G. & Michalewicz, Z. (1999). A new version of ant system for subset Problem, *Congress on Evolutionary Computation*, pp. 1459-1464.

Ong, Y. S.; Lim, M. H.; Zhu, N. & Wong, K. W. (2006). Classification of Adaptive Memetic Algorithms: A Comparative Study, *IEEE Transactions on Systems Man and Cybernetics*, Part B, Vol. 36, No. 1, pp. 141-152.

OR-Library, available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files.

Rechenberg, I. (1973). Evolutionstrategie-Optimierung Technischer Systeme nach Prinzipien der Biologischen Information, Fromman Verlag, Freiburg, Germany.

Sato, T. & Hagiwara, M. (1997). Bee System: finding solution by a concentrated search, *IEEE Int. Conf. on Computational Cybernetics and Simulation*, pp.3954-3959.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*, John Wiley & Sons, New-York.

Sezgin, M. & Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation, *Journal of Electronic Imaging*, Vol. 13, No. 11, pp. 146–165.

Shah-Hosseini, H. & Safabakhsh, R. (2002). Automatic multilevel thresholding for image segmentation by the growing time adaptive self-organizing map, *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 24, No. 10, pp. 1388-1393.

Shah-Hosseini, H. (2007). Problem solving by intelligent water drops. *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 3226-3231, Swissotel The Stamford, Singapore, September.

Shah-Hosseini, H. (2008a). Intelligent water drops algorithm: a new optimization method for solving the multiple knapsack problem. *Int. Journal of Intelligent Computing and Cybernetics*, Vol. 1, No. 2, pp. 193-212.

Shah-Hosseini, H. (2008b). The Intelligent Water Drops algorithm: A nature-inspired swarm-based optimization algorithm. *Int. J. Bio-Inspired Computation*, Vol. 1, Nos. 1/2, pp. 71–79.

Song, X.; Li, B. & Yang, H. (2006). Improved ant colony algorithm and its applications in TSP, *Proc. of the Sixth Int. Conf. on Intelligent Systems Design and Applications*, pp. 1145 – 1148.

Stutzle, T. & Hoos, H. (1996). Improving the ant system: a detailed report on the MAX-MIN ant system, *Technical Report AIDA*, pp. 96-12, FG Intellektik, TU Darmstadt, Germany.

Teodorovic, D.; Lucic, P.; Markovic, G. & Orco, M. D. (2006). Bee colony optimization: principles and applications, *8th Seminar on Neural Network Applications in Electrical Engineering*, NEUREL-2006, Serbia, September 25-27.

Timmis, J.; Neal, M. & Hunt, J. (2000). An artificial immune system for data analysis, *BioSystems*, Vol. 55, No. 1, pp. 143–150.

TSP Library (TSPLIB95), Available at: http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/STSP.html

Watkins, J. (2004). *Across the Board: The Mathematics of Chess Problems*, Princeton: Princeton University Press.

Yan, X.-S.; Li, H.; CAI, Z.-H. & Kang, L.-S. (2005). A fast evolutionary algorithm for combinatorial optimization problem, P*roc. of the Fourth Int. Conf. on Machine Learning and Cybernetics*, pp. 3288-3292, August.

Yager, R. R. (1979). On the measures of fuzziness and negation. Part 1: Membership in the unit interval, *Int. Journal of Gen. Sys.*, Vol. 5, pp. 221-229.

Evolutionary Computation

Edited by Wellington Pinheiro dos Santos

ISBN 978-953-307-008-7
Hard cover, 572 pages
Publisher InTech
Published online 01, October, 2009
Published in print edition October, 2009

This book presents several recent advances on Evolutionary Computation, specially evolution-based optimization methods and hybrid algorithms for several applications, from optimization and learning to pattern recognition and bioinformatics. This book also presents new algorithms based on several analogies and metafores, where one of them is based on philosophy, specifically on the philosophy of praxis and dialectics. In this book it is also presented interesting applications on bioinformatics, specially the use of particle swarms to discover gene expression patterns in DNA microarrays. Therefore, this book features representative work on the field of evolutionary computation and applied sciences. The intended audience is graduate, undergraduate, researchers, and anyone who wishes to become familiar with the latest research work on this field.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821