The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm

Hamed Shah-Hosseini

Faculty of Electrical and Computer Engineering,
Shahid Beheshti University, G.C.,
Tehran, Iran
E-mail: tasom2002@yahoo.com
E-mail: h_shahhosseini@sbu.ac.ir

Abstract: 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. The intelligent water drops (IWD) algorithm is a new swarm-based optimisation algorithm inspired from observing natural water drops that flow in rivers. In this paper, the IWD algorithm is tested to find solutions of the *n*-queen puzzle with a simple local heuristic. The travelling salesman problem (TSP) is also solved with a modified IWD algorithm. Moreover, the IWD algorithm is tested with some more multiple knapsack problems (MKP) in which near-optimal or optimal solutions are obtained.

Keywords: swarm intelligence; intelligent water drops; IWDs; travelling salesman problem; TSP; multiple knapsack problem; MKP; n-queen puzzle.

Reference to this paper should be made as follows: Shah-Hosseini, H. (2009) 'The intelligent water drops algorithm: a nature-inspired swarm-based optimisation algorithm', *Int. J. Bio-Inspired Computation*, Vol. 1, Nos. 1/2, pp.71–79.

Biographical notes: H. Shah-Hosseini received his BS in Computer Engineering from Tehran University, his MS and PhD from Amirkabir University of Technology, all with high honours. He is now with the Electrical and Computer Engineering Department, Shahid Beheshti University, Tehran, Iran. His current research interests include computational intelligence especially time-adaptive self-organising maps, evolutionary computation, swarm intelligence and computer vision.

1 Introduction

The natural systems that have developed for so long are one of the rich sources of inspiration for inventing new intelligent systems. Swarm intelligence is one of the scientific fields that are closely related to natural swarms existing in nature, such as ant colonies, bee colonies, brain and rivers. Among the problem solving techniques inspired from nature are evolutionary computation (Eiben and Smith, 2003), neural networks (Haykin, 1999), time adaptive self-organising maps (Shah-Hosseini, 2006), ant colony optimisation (Dorigo and Stutzle, 2004), bee colony optimisation (Sato and Hagiwara, 1997), particle swarm optimization (Eberhart and Kennedy, 1995), DNA computing (Adleman, 1994), electromagnetism-like optimisation (Birbil and Fang, 2003) and intelligent water drops (Shah-Hosseini, 2007).

One of the recently proposed algorithms in the field of the swarm intelligence is the intelligent water drops (IWDs) algorithm (Shah-Hosseini, 2007; Shah-Hosseini, 2008). The IWD algorithm is based on the dynamic of river systems, actions and reactions that happen among the water drops in rivers. The natural water drops are used to develop IWD and the IWDs cooperate together to reach a better solution for a given problem. The IWD algorithm may be used for maximisation or minimisation problems. The solutions are incrementally constructed by the IWD algorithm. Therefore, the IWD algorithm is a population-based constructive optimisation algorithm.

The IWD algorithm has been used for the travelling salesman's problem (TSP) (Shah-Hosseini, 2007) and multiple (or multidimensional) knapsack problem (MKP) (Shah-Hosseini, 2008) with promising results. Both the TSP and the MKP are NP-hard combinatorial optimisation problems. In the TSP, a map of cities is given to the salesman and he has to visit all the cities only once to complete a tour such that the length of the tour is the shortest among all possible tours for this map. In the MKP, there are multiple knapsacks and several items and the goal is to include some of the items in the knapsacks to achieve maximum profit with the constraint that none of the knapsacks becomes overflowed. Both the TSP and MKP are often used for testing optimisation algorithms. In this paper, the IWD-TSP algorithm (Shah-Hosseini, 2007) is modified to get better results for the TSPs. Moreover, the IWD-MKP algorithm (Shah-Hosseini, 2008) is used for some other MKPs to observe its power for getting optimal or near-optimal solutions.

The *n*-queen problem, an NP-complete problem, is also used here to test the performance of the IWD algorithm. The IWD algorithm that is used for solving the *n*-queen is called the 'IWD-NQ' algorithm. The *n*-queen puzzle (problem) needs to place n queens on a chessboard of size n × n such that no two queens attack each other. Therefore, the goal is to obtain the global optimal solution and no near-optimal solution is acceptable.

The next section reviews some processes that occur in a river, which involves the water drops of the river and explains IWDs that have been developed based on the ideas of natural water drops. Section 3 introduces the IWD algorithm, which uses IWDs for problem solving. Section 4 shows how to use the IWD algorithm for solving three different optimisation problems. Experimental results with the proposed IWD algorithm for artificial and standard TSPs, MKPs, and *n*-queen problems form Section 5. Concluding remarks are the final section of the paper.

2 Basics of the IWDs

By looking at rivers in nature, we are surprised to see lots of twists and turns along their paths. One thing that makes us think is that why these twists have been created and is there any logic or intelligence behind them? And if that is so, can we use the mechanisms that happen in rivers and as a result, can we design and develop intelligent algorithms based on them? 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.

Both of the two properties may change during the lifetime of an IWD. An IWD flows from a source to a destination. The IWD begins its trip with an initial velocity and zero soil. During its trip, it travels in the environment from which it removes some soil and it may gain some speed. An IWD is supposed to flow in discrete steps. From its current location to its next location, the IWD velocity is increased by the amount non-linearly proportional to the inverse of the soil between the two locations. Therefore, a path with less soil lets the IWD become faster than a path with more soil.

An IWD gathers soil during its trip in the environment. This soil is removed from the path joining the two locations. The amount of soil added to the IWD is non-linearly proportional to the inverse of the time needed for the IWD to pass from its current location to the next location. This time interval is calculated by the simple laws of physics for linear motion. Thus, the time taken is proportional to the velocity of the IWD and inversely proportional to the distance between the two locations. Moreover, those parts of the environment that are used with more IWDs will have less soil. It may be said that soil is the source material of information such that the environment and water drops both have memories for soil.

An IWD needs a mechanism to select the path to its next location or step. In this mechanism, the IWD prefers the paths having low soils to the paths having high soils. This behaviour of path selection is implemented by imposing a uniform random distribution on the soils of the available paths. Then, the probability of the next path to select is inversely proportional to the soils of the available paths. Therefore, paths with lower soils have higher chance to be selected by the IWD.

3 The IWD algorithm

The IWD algorithm gets a representation of the problem in the form of a graph (N, E) with the node set N and edge set E. Then, each IWD begins constructing its solution gradually by travelling on the nodes of the graph along the edges of the graph until the 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 T^{IB} is found and it is used to update the total-best solution T^{TB}. The amount of soil on the edges of the iteration-best solution T^{IB} 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 $iter_{max}$ or the total-best solution T^{TB} reaches the expected quality.

The IWD algorithm has two kinds of parameters. One kind is those that remain constant during the lifetime of the algorithm and they are called 'static parameters'. The other kind is those parameters of the algorithm, which are dynamic and they are reinitialised after each iteration of the algorithm. It should be reminded that the values chosen for the static parameters of the IWD algorithm are the same used in Shah-Hosseini (2008) until specified otherwise.

The IWD algorithm is specified in the following steps:

1 Initialisation of static parameters. The graph (N, E) of the problem is given to the algorithm. 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. The iteration count $iter_{count}$ is set to zero.

  The number of water drops N_{IWD} is set to a positive integer value, which is usually set to the number of nodes N_c of the graph.

  For velocity updating, the parameters are $a_v = 1$, $b_v = .01$ and $c_v = 1$. For soil updating, $a_s = 1$, $b_s = .01$ and $c_s = 1$. The local soil updating parameter ρ_n, which is a small positive number less than one, is set as

$\rho_n = 0.9$. The global soil updating parameter ρ_{IWD}, which is chosen from [0, 1], is set as $\rho_{IWD} = 0.9$. 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. For the IWD-MKP, *InitVel* = 4 is used, which is the same value used in Shah-Hosseini (2008).

2 Initialisation 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.

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.

5.1 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 $p_i^{IWD}(j)$:

$$p_i^{IWD}(j) = \frac{f\left(soil(i,j)\right)}{\sum_{k \notin vc(IWD)} f\left(soil(i,k)\right)} \qquad (1)$$

such that

$$f(soil(i,j)) = \frac{1}{\varepsilon_s + g(soil(i,j))} \quad \text{and}$$

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

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

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

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

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s \,.\, time^2\left(i, j; vel^{IWD}(t+1)\right)} \qquad (3)$$

such that

$$time\left(i, j; vel^{IWD}(t+1)\right) = \frac{HUD(j)}{vel^{IWD}(t+1)} \quad \text{where}$$

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

$$\begin{aligned} soil(i,j) &= (1-\rho_n) \,.\, soil(i,j) - \rho_n \,.\, \Delta soil(i,j) \\ soil^{IWD} &= soil^{IWD} + \Delta soil(i,j) \end{aligned} \qquad (4)$$

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

$$T^{IB} = \arg \max_{\forall T^{IWD}} q(T^{IWD}) \qquad (5)$$

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

$$\begin{aligned} soil(i,j) = {} & (1+\rho_{\text{IWD}}) \,.\, soil(i,j) \\ & -\rho_{IWD} \,.\, \frac{1}{(N_{IB}-1)} \,.\, soil_{IB}^{IWD} \quad \forall (i,j) \in T^{IB} \end{aligned} \qquad (6)$$

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

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

$$T^{TB} = \begin{cases} T^{TB} & if \ q(T^{TB}) \geq q(T^{IB}) \\ T^{IB} & otherwise \end{cases} \qquad (7)$$

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}.

It is reminded that the IWD has been shown to have the property of convergence in value (Shah-Hosseini, 2008). It means that the IWD algorithm is able to find the optimal solution if the number of iterations be sufficiently large.

The IWD algorithm may be compared to the ant-based optimisation algorithms (Bonabeau et al., 1999). The ants in an ant colony optimisation algorithm deposit pheromones on the paths they move on. The IWDs change soil on the paths they flow over. However, in contrast to the ants, these changes are not constant and are dependent on the velocity and soil of the IWD visiting the paths. Moreover, the IWDs may gain different velocities throughout an iteration of the IWD algorithm whereas in ant-based algorithms the velocities of the ants are irrelevant to the algorithm.

4 Solving problems by the IWD algorithm

The IWD algorithm may be used to solve optimisation problems. An IWD in the algorithm both searches and changes the environment of the given problem. By doing that, the IWD constructs incrementally a solution to the problem. The problem should be presented to the IWD algorithm in the form of a graph and the IWDs actually go node to node on the links of the graph. A swarm of IWDs flows in the graph with the guidance of a local heuristic in the hope of finding optimal or near optimal solutions. In the following, three different problems are stated and then it is shown how to use the IWD algorithm for solving them.

4.1 The travelling salesman's problem

In this subsection, we specifically express the steps for solving the travelling salesman problem or the TSP. Then, a modification to the IWD-TSP in Shah-Hosseini (2007) is proposed. In the TSP, a map of cities is given to the salesman and he is required to visit the entire cities one after the other to complete his tour such that in this tour every city is visited only once except the first city of the tour which is visited twice to form a round trip (tour). The goal in the TSP is to find the tour with the minimum total length among all such possible tours, which are obtainable 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 paths between each two cities. In this paper, the graph of the TSP is considered a complete graph. Therefore, every city has a direct path to another city. Here, it is assumed that the direct path between each two cities is an undirected path. So, in summary, the graph of the TSP is a complete undirected graph. A solution of the TSP having the graph (*N*, *E*) is then an ordered set of *n* distinct cities.

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. He then returns to the first city c_1. The tour length, $TL(.)$ is calculated by

$$TL\ (c_1, c_2, ..., c_n) = \sum_{i=1}^{n} d\ (c_i, c_{i+1}) \tag{8}$$

such that $c_{n+1} = c_1$ and the distance function $d(.,.)$ which computes the distance between two cities is often selected as the Euclidean distance. The goal of any optimisation algorithm for the TSP is to find the tour $T^* = \left(c_1^*, c_2^*, ..., c_n^*\right)$ with the minimum length among all possible tours:

$$\begin{aligned} &TL\ (T^*) \le TL\ (c_1, c_2, ..., c_n) \\ &\text{for every tour } (c_1, c_2, ..., c_n) \end{aligned} \tag{9}$$

where $TL\ (.)$ returns the total length of the given tour. The tour T^* is called the global optimum tour.

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 can travel between nodes of the graph through these links and 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.

The local heuristic for the TSP, denoted by $HUD_{TSP}(i, j)$, has been suggested as follows:

$$HUD_{TSP}(i, j) = \| \mathbf{c}(i) - \mathbf{c}(j) \| \tag{10}$$

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. The time, which is 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 is proposed here to get better tours and hopefully escape local optimums. After every constant number of iterations, N_I, the soils of all paths of the graph problem are reinitialised again such that the paths of the total-best solution T^{TB} are given less soil than the other paths:

$$soil(i, j) = \begin{cases} \alpha_I \Gamma_I\, InitSoil & \text{for every } (i, j) \in T^{TB} \\ InitSoil & \text{otherwise} \end{cases} \tag{11}$$

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.

4.2 The n-queen problem

The 8-queen puzzle is the problem of putting eight chess queens on an 8 × 8 chessboard such that no two queens are able to attack each other. Thus, a solution requires that no two queens occupy the same row, column, or diagonal. The 8-queen puzzle can be generalised to the *n*-queen puzzle in which *n* queens must be placed on an *n* × *n* chessboard such that no two queens attack each other (Watkins, 2004). The solution exists for $n = 1$ and $n \ge 4$.

One strategy to reduce the huge search space 64^n in the *n*-queen problem is to place the *n* queens 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 *i*th queen is placed on any row of the *i*th 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 *i*th row chosen by the *i*th queen is considered the *i*th 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 problem, any feasible solution is also the optimal solution because any feasible solution for an *n*-queen problem is the solution in which no two queens attack each other and that is the desired solution. For this problem, only the final positions of queens on the chessboard are desired to be found while the path to reach the final feasible (optimal) solution(s) is not wanted.

For an IWD to solve the *n*-queen problem, the local heuristic undesirability $HUD_{NQ}(i, j)$, which is used in the IWD algorithm, is proposed as follows:

$$HUD_{NQ}(i, j) = (1+r)\left| \, |i-j| - \frac{n}{2} \right| \tag{12}$$

where $HUD_{NQ}(i, j)$ is the undesirability of an IWD to go from current row (city) *i* to the next row (city) *j*. 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 $n \times n$. The heuristic of equation (12) favours the distance between the rows of neighbouring columns to be near the length *n*/2.

For the *n*-queen problem, it is observed experimentally that the IWD algorithm with the proposed local heuristic 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. Using a good local search algorithm may help to come out of such local optima faster. In this regard, a simple local search algorithm is proposed. This local search algorithm called 'n-queen local search' or NQLS is activated only when the iteration-best solution of the IWD algorithm contains only two queens attacking each other.

For the *n*-queen problem, the quality of a solution *T* is given by

$$q(T) = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} attack(c_i, c_j) \tag{13}$$

such that

$$attack(c_i, c_j) = \begin{cases} 1 & \text{if } c_i \text{ and } c_j \text{ attack each other} \\ 0 & \text{else} \end{cases} \tag{14}$$

The optimal solution *T** has the quality value zero: $q(T^*) = 0$. It is hoped that the total-best iteration T^{TB} reaches the quality zero. As a result, the proposed NQLS algorithm is activated when the quality of the iteration-best solution T^{TB} becomes –1.

In the following, the proposed NQLS is expressed in four steps:

1 Get the iteration-best solution with tour $T^{IB} = \left(c_1^{IB}, c_2^{IB}, ..., c_n^{IB}\right)$ with the quality $q(T^{IB}) = -1$.

2 Set $T_0 = T^{IB}$.

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 jump 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 *mod* 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 jump 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.

The IWD algorithm for the *n*-queen problem is called 'IWD-NQ' algorithm. The proposed IWD-NQ algorithm uses the standard IWD algorithm mentioned in Section 3 with the local heuristic $HUD_{NQ}(i, j)$ defined in equation (12) and the local search algorithm NQLS, which has been proposed above.

4.3 The multiple knapsack problem

The knapsack problem or KP (Kellerer et al., 2004) 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 maximised.

The multiple or multidimensional knapsack problem, MKP, is a generalisation 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 maximise 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:

$$\max \sum_{i=1}^{n} y_i b_i \tag{15}$$

subject to the following constraints:

$$\sum_{i=1}^{n} r_{ij} y_i \le a_j \ for\ j = 1, 2, ..., m. \tag{16}$$

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.

To solve the MKP using the IWD algorithm, the search space of the problem is viewed as a graph (*N*, *E*) where the node set *N* denotes the items of the MKP and the edge set *E* denotes the arcs (paths) between the items (nodes). A feasible solution is a set of N' items such that they do not violate the constraints in equation (16) and $N' \subseteq N$. For the MKP, the optimal solution is also a feasible solution and it is composed of a subset of *n* items, which maximises the profit defined in equation (15). Therefore, the order of selecting items in the solution of the MKP is not important and not all items may be included in the solution.

The heuristic undesirability $HUD_{MKP}(j)$ that has been used in the IWD-MKP algorithm (Shah-Hosseini, 2008) is defined as follows:

A simple local heuristic is used which reflects the undesirability of adding an item to the current partial solution. Let the $HUD_{MKP}(j)$ for the MKP be defined as

$$HUD_{MKP}(j) = \frac{1}{mb_j} \sum_{k=1}^{m} r_{jk}. \tag{17}$$

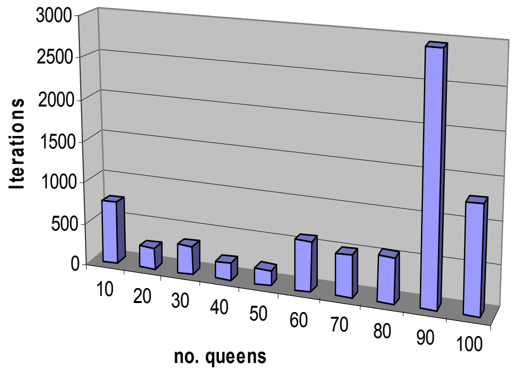
Where b_j denotes the profit of item *j* and r_{jk} is the resource requirement for item *j* from knapsack *k*. Equation (17) shows that $HUD_{MKP}(j)$ decreases if the profit b_j is high whereas $HUD_{MKP}(j)$ increases if the resource requirements of item *j* are high. As a result, the items with less 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.

The IWD-MKP algorithm uses the standard IWD algorithm mentioned in Section 3 and the local heuristic $HUD_{MKP}(j)$ defined in equation (17) to solve the MKP.

5 Experimental results

The IWD algorithm is used for three different problems: *n*-queen puzzle, the TSP and the MKP. The first set of experiments is used to test the capabilities of the proposed IWD-NQ algorithm introduced in Section 4.2 for the *n*-queen puzzle. The IWD-NQ has a simple heuristic and a plain local search algorithm to escape from local optimums. The IWD-NQ is tested with ten different *n*-queens puzzle where *n* is increased from ten to 100 by increments of ten. The average number of iterations for the ten runs of each *n*-queen puzzle is depicted in Figure 1. In these experiments, the number of IWDs is kept constant with 50 IWDs.

Figure 1 The average number of iterations to get to the global optimal solution versus the number of queens for the *n*-queen puzzle (see online version for colours)



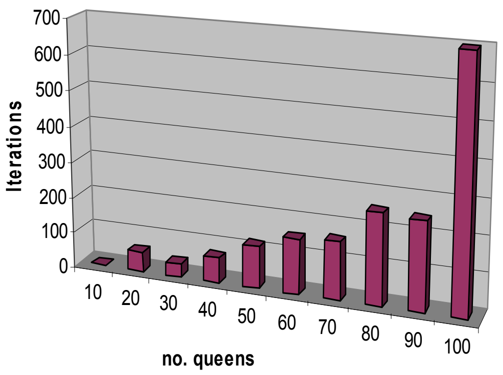
Note: The results shown here are the average iterations of ten runs of the proposed IWD-NQ algorithm

It is seen that the number of iterations to get to the optimal solution(s) does not depend necessarily to the number of queens. For example, the average number of iterations for the 90-queen problem is bigger than the 100-queen problem. One reason is that getting out of the local optimal solution for the case with 90 queens is more difficult than the case with 100. In other words, the proposed local heuristic of equation (12) works better for some number of queens than the others.

For the aforementioned experiments, the lowest numbers of iterations in ten runs of the IWD-NQ algorithm are shown in Figure 2. In this figure, the number of iterations increases as the number of queens is increased. Therefore, the results for best performance (lowest iterations) in ten runs depend on the number of queens and often increase with increase in the number of queens.

The IWD-NQ algorithm is tested with the 200-queens problem using 50 IWDs and it gets an average number of iterations 4,893 in ten runs. The minimum and maximum number of iterations in these ten runs are 1,964 and 13,597, respectively. It is seen that there is a wide gap between the minimum and maximum number of iterations. One reason of such a wide gap in the number of iterations is that some of the local optimums are harder to escape from than other local optimums.

Figure 2 The minimum number of iterations to get to the global optimal solution versus the number of queens for the *n*-queen puzzle in ten runs of the proposed IWD-NQ algorithm (see online version for colours)

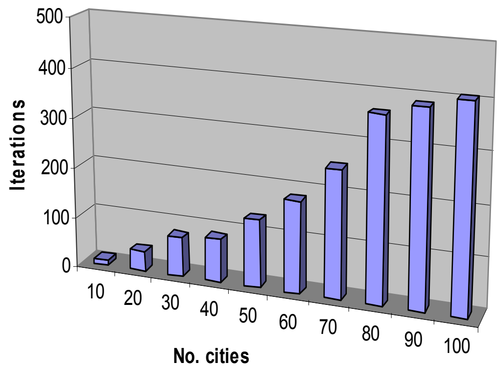


The local optimums of the *n*-queen problem should be thoroughly analysed and based on the analyses some more efficient local search algorithms should be designed to help the IWD to escape from the local optimums. It is reminded that better local heuristics may make the algorithm converge faster to the global optimum and/or may reduce the number of hard local optimums.

The next set of experiments is implemented with the IWD-TSP algorithm with the local heuristic mentioned in Section 4.1 and the modification introduced in equation (11) with $N_I = 15$. To avoid the confusion, we call this Modified IWD-TSP or 'MIWD-TSP'. The first experiment with the MIWD-TSP algorithm is executed for a number of cities that are placed on the perimeter of a circle in equal distances from each other. The algorithm is tested for different number of cities on the circle from ten to 100 cities incremented by ten. The results of this experiment are shown in Figure 3 in which the average numbers of iterations to get to the global optimums are depicted. The number of IWDs used for all experiments of Figure 3 is 50. It is seen that as the number of cities increases, the average number of iterations to find the relevant optimal solution almost monotonically increases. For example, for the 10-city problem, 10.4 average iterations are needed whereas for the 20-city problem, 39.6 average iterations are needed.

It is reminded that the IWD-TSP in Shah-Hosseini (2007) gets stuck in some hard local optimums for the cities on a circle. For example, in one of the runs of the IWD-TSP for the 10-city problem, the large iteration number 22275 is observed whereas the largest iteration number of the MIWD-TSP in ten runs is 33. Moreover, for the 20-city problem, the average number of iterations 2046 is obtained whereas for the MIWD-TSP, the average number 387 is obtained. For the circle TSP with more cities, the average numbers of iterations to get to the global optimums become so large that it is impractical to use the IWD-TSP for them. However, the IWD-TSP works well in finding a good local optimum for the toy problem as reported in Shah-Hosseini (2008). In summary, the MIWD-TSP algorithm is much more efficient for the toy TSP problem.

Figure 3 The average number of iterations to get to the global optimal solution versus the number of cities of the TSP problem where cities equally spaced on the perimeter of a circle (see online version for colours)



Note: The results shown here are the average numbers of iterations of ten runs of the MIWD-TSP algorithm

Four TSPs are chosen from the TSPLIB95 (the TSP library in the internet) to test the capability of the MIWD-TSP algorithm. The lengths of average and best tours in five runs of the MIWD-TSP algorithm are reported in Table 1. For comparison, the lengths of best tours of some other metaheuristics are also mentioned in Table 1. The table shows that the tours obtained by the MIWD-TSP algorithm are satisfactorily close to the known optimum solutions and are comparable to the other metaheuristics. The best tours of the TSPs in five runs of the MIWD-TSP algorithm are depicted in Figure 4. It is seen in Figure 4 that the tours have no self-crossing regions and thus the MIWD-TSP solves the self-crossing that sometimes happen in the IWD-TSP experiments in Shah-Hosseini (2007).

The final set of experiments is executed with the IWD-MKP algorithm having the local heuristic (Shah-Hosseini, 2008) mentioned in Section 4.3. The IWD-MKP is tested with eight problems 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 2. The qualities of optimal solutions are known for the eight MKPs and are mentioned in the table for comparison. 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. For the problem 'WEING4' with two constraints and 28 items, the qualities of iteration-best solutions for five runs of the IWD algorithm are depicted in Figure 5. In the figure, the best run of the IWD-MKP algorithm converges to the optimum solution 119377 in 12 iterations whereas its worst run converges in 60 iterations.

Table 1 The comparison between the MIWD-TSP and four other metaheuristics MMAS (Stutzle and Hoos, 1996), BCO (Teodorovic et al., 2006), EA (Yan et al., 2005), Improved ACO (Song et al., 2006) for the four TSPs mentioned below

Problem name	*Optimum length*	*Method*					
		MMAS	*BCO*	*EA*	*Improved ACO*	*MIWD-TSP*	
						Best	*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

Notes: The MIWD-TSP iterations: 3000 for eil51, 4500 for eil76 and 6000 for st70 and kroA100

Table 2 Some of the problems of the OR-library in file 'mknap2.txt' which are solved by 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

Note: The global optimal solutions are also mentioned.

Figure 4 The best tours of five runs of the MIWD-TSP mentioned in Table 1, (a) the tour of eil51 (b) the tour of eil76 (c) the tour of st70 (d) the tour of kroA100

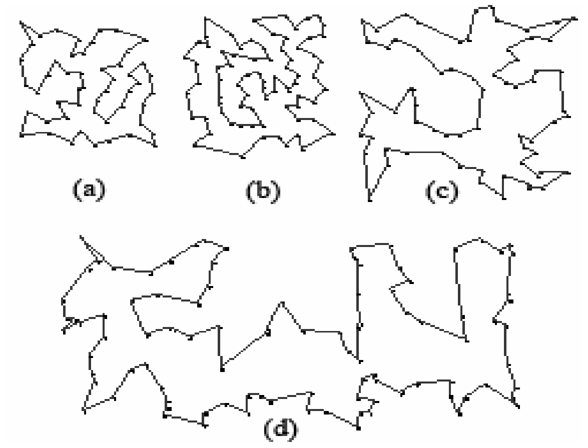
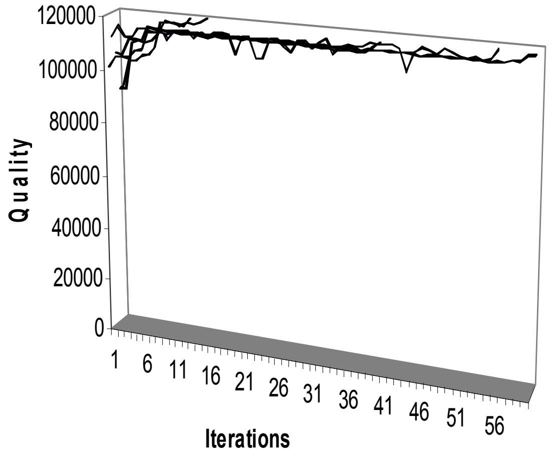


Figure 5 Convergence curves of five runs of the IWD-MKP algorithm for the MKP 'WEING4' in file 'mknap2.txt' of the OR-library with the global optimum 119337



Note: Each curve in the figure shows one run of the algorithm.

6 Conclusions

The IWD algorithm is an optimisation algorithm that uses a swarm of water drops to collectively search for optimal solutions in the environment of the given problem. In fact, each IWD constructs incrementally a solution to the problem by moving on the graph representation of the problem. Then, among the obtained solutions, the best one in terms of quality is chosen and its total path is reinforced by soil removal. During each iteration of the IWD algorithm, an IWD gains some velocity and removes some soil from the path it flows on. After enough iterations of the IWD algorithm, the IWDs find the good paths that are decoded to good solutions of the problem.

Three different problems are used to experiment the IWD algorithm, the *n*-queen puzzle, the TSP and the MKP. Here, for the first time, the IWD algorithm is used for solving *n*-queen problems having a simple heuristic and a local search algorithm. It is shown that the MKP-NQ algorithm can find the global optimal solutions of the *n*-queen puzzle. However, by suggesting better heuristics and local search algorithms the number of iterations may be reduced.

The IWD algorithm is modified for the TSP problem and this MIWD-TSP gets better tours with shorter lengths in comparison to the standard MKP-TSP algorithm. Moreover, some new MKPs are tested with the IWD-MKP algorithm and it is observed that the algorithm is able to find optimal or near optimal solutions for the given MKPs.

This paper indicates that the IWD algorithm is capable to deal with optimisation problems in finding solutions with good or optimal qualities. However, there is an open space for modifications in the standard algorithm, embedding other mechanisms that exist in natural rivers and/or inventing local heuristics that fit better with the IWD algorithm. It also demonstrates that the nature is an excellent teacher for designing and inventing new swarm-based optimisation algorithms.

Acknowledgements

This work has been supported by a research grant of Shahid Beheshti University.

References

Adleman, L.M. (1994) 'Molecular computation of solutions to combinatorial problem', *Science*, pp.1021–1023.

Birbil, I. and 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. and Theraultz, G. (1999) *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press.

Dorigo, M. and Stutzle, T. (2004) *Ant Colony Optimization*, Prentice-Hall.

Eberhart, R.C. and Kennedy, J. (1995) 'A new optimizer using particle swarm theory', *Proc. Sixth Int. Symposium on Micro Machine and Human Science*, Nagoya, Japan, pp.39–43.

Eiben, A.E. and Smith, J.E. (2003) *Introduction to Evolutionary Computing*, Springer-Verlag.

Haykin, S. (1999) *Neural Networks: A Comprehensive Foundation*, 2nd ed., Prentice-Hall.

Kellerer, H., Pferschy, U. and Pisinger, D. (2004) *Knapsack Problems*, Springer, New York, NY.

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

Sato, T. and Hagiwara, M. (1997) 'Bee system: finding solution by a concentrated search', *IEEE Int. Conf. on Computational Cybernetics and Simulation*, pp.3954–3959.

Shah-Hosseini, H. (2006) 'The time adaptive self-organizing map is a neural network based on artificial immune system', *Proc. IEEE World Congress on Computational Intelligence*, Vancouver, Canada, July, pp.1007–1014.

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

Shah-Hosseini, H. (2008) '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.

Song, X., Li, B. and 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. and Hoos, H. (1996) 'Improving the ant system: a detailed report on the MAX-MIN ant system', *Technical Report AIDA 96-12*, FG Intellektik, TU Darmstadt, Germany.

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

TSP library, available at http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/STSP.html.

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

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