Intelligent water drops algorithm
A new optimization method for solving the multiple knapsack problem

Hamed Shah-Hosseini
Electrical and Computer Engineering Department, Shahid Beheshti University, Tehran, Iran

Abstract
Purpose – The purpose of this paper is to test the capability of a new population-based optimization algorithm for solving an NP-hard problem, called “Multiple Knapsack Problem”, or MKP.
Design/methodology/approach – Here, the intelligent water drops (IWD) algorithm, which is a population-based optimization algorithm, is modified to include a suitable local heuristic for the MKP. Then, the proposed algorithm is used to solve the MKP.
Findings – The proposed IWD algorithm for the MKP is tested by standard problems and the results demonstrate that the proposed IWD-MKP algorithm is trustable and promising in finding the optimal or near-optimal solutions. It is proved that the IWD algorithm has the property of the convergence in value.
Originality/value – This paper introduces the new optimization algorithm, IWD, to be used for the first time for the MKP and shows that the IWD is applicable for this NP-hard problem. This research paves the way to modify the IWD for other optimization problems. Moreover, it opens the way to get possibly better results by modifying the proposed IWD-MKP algorithm.
Keywords Programming and algorithm theory, Optimization techniques, Systems and control theory
Paper type Research paper

1. Introduction
The multiple knapsack problem (MKP), is an NP-hard combinatorial optimization problem with applications such as cutting stock problems (Gilmore and Gomory, 1966), processor allocation in distributed systems (Gavish and Pirkul, 1982), cargo loading (Shih, 1979), capital budgeting (Weingartner, 1966), and economics (Martello and Toth, 1990).

Two general approaches exist for solving the MKP: the exact algorithms and the approximate algorithms. The exact algorithms are used for solving small- to moderate-size instances of the MKP such as those based on dynamic programming employed by Gilmore and Gomory (1966) and Weingartner and Ness (1967) and those based on the branch-and-bound approach suggested by Shih (1979) and Gavish and Pirkul (1985). A recent review of the MKP is given by Freville (2004).

The approximate algorithms may use metaheuristic approaches to approximately solve difficult optimization problems. The term “metaheuristic” was introduced by Glover and it refers to general purpose algorithms which can be applied to different
optimization problems with usually few modifications for adaptation to the given specific problem.

Metaheuristic algorithms include algorithms such as simulated annealing (Kirkpatrick et al., 1983), tabu search (Glover, 1989), evolutionary algorithms like genetic algorithms (Holland, 1975), evolution strategies (Rechenberg, 1973), and evolutionary programming (Fogel et al., 1966). Ant colony optimization (Dorigo et al., 1991), scatter search (Glover, 1977), greedy randomized adaptive search procedure (Feo and Resende, 1989, 1995), Iterated local search (Lourenco et al., 2003), guided local search (Voudouris and Tsang, 1995), variable neighborhood search (Mladenovic and Hansen, 1997), particle swarm optimization (Kennedy and Eberhart, 2001), electromagnetism-like optimization (Birbil and Fang, 2003), and intelligent water drops (IWD) (Shah-Hosseini, 2007). For a review on the field of metaheuristics, it is suggested to read the book by Glover and Kochenberger (2003).

Several kinds of metaheuristic algorithms have been used for the MKP to obtain near-optimal or hopefully optimal solutions including those based on evolutionary algorithms such as Glover and Kochenberger (1996) and Chu and Beasley (1998). Moreover, several variants of hybrid evolutionary algorithms have also been implemented which are reviewed in Raidl and Gottlieb (2005). Ant Colony-based algorithms are also used for the MKP including Fidanova (2002) and Leguizamon and Michalewicz (1999).

A metaheuristic algorithm can be classed as a constructive approach or a local search method. A constructive algorithm builds solutions from scratch by gradually adding solutions' components to the initially empty solutions whereas a local search algorithm starts from a complete solution and then tries to improve it over time. Evolutionary-based algorithms are local search algorithms whereas the Ant Colony-based algorithms are constructive algorithms. Moreover, a metaheuristic algorithm may use a single solution or a population of solutions to proceed at each iteration. Simulated Annealing uses a single solution whereas evolutionary algorithms are population-based algorithms.

Recently, the new metaheuristic algorithm “Intelligent Water Drops,” has been introduced in the literature and used for solving the traveling salesman problem (TSP). The TSP is also an NP-hard combinatorial optimization problem. Therefore, the new IWD algorithm should be applicable to solve the MKP. This paper tries to solve the MKP using an IWD-based algorithm. The IWD algorithm is a population-based optimization algorithm that uses the constructive approach to find the optimal solution(s) of a given problem. Its ideas are based on the water drops that flow in nature such that each water drop constructs a solution by traversing in the search space of the problem and modifying its environment.

The next section of the paper introduces the MKP. Section 3 overviews the general principles of the IWD algorithm. Section 4 proposes the modified IWD algorithm for the MKP. After that, a section on the convergence properties of the IWD algorithm is stated. Experimental results with the IWD algorithm are presented in section 6. The final section of the paper includes the concluding remarks.

2. The multiple knapsack problem
Consider we have a set of items \( i \in I \) where each item \( i \) gives the profit \( b_i \) and requires the resource (capacity) \( r_i \). The knapsack problem (KP for short) is to select a subset of
items of the set $I$ in such a way that they all fit in a knapsack of limited capacity and the sum of profits of the selected items is maximized.

The MKP generalizes the KP by considering multiple resource constraints. Therefore, the MKP is considered to have multiple knapsacks.

Assume the variable $y_i$ denotes the inclusion of the item $i$ in the knapsack such that:

$$y_i = \begin{cases} 
1 & \text{if the item } i \text{ is added to the knapsack} \\
0 & \text{otherwise} 
\end{cases}$$

Moreover, the variable $r_{ij}$ is assumed to represent the resource requirement of the item $i$ with respect to the resource constraint $j$ having the capacity $a_j$. The MKP with $m$ constraints and $n$ items can be formulated as follows:

$$\max \sum_{i=1}^{n} y_i b_i.$$  \hspace{1cm} (2)

Subject to the following constraints:

$$\sum_{i=1}^{n} r_{ij} y_i \leq a_j \quad \text{for } j = 1, 2, \ldots, m.$$  \hspace{1cm} (3)

Such that $y_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$. In the MKP, it is often assumed that the profits $b_i$ and the resources requirements $r_{ij}$ are non-negative values.

Here, an MKP is viewed as a graph $(N, E)$ where the set $N$ represents the items of the MKP and the set $E$ represents the arcs (paths) between the items. A solution is then a set of $N'$ items such that they do not violate the constraints in equation (3) and $N' \subseteq N$.

3. Basic principles of the IWD algorithm

Water drops that flow in rivers, lakes, and seas are the sources of inspiration for developing the IWD. This intelligence is more obvious in rivers which find their ways to lakes, seas, or oceans despite many different kinds of obstacles on their ways. In the water drops of a river, the gravitational force of the earth provides the tendency for flowing toward the destination. If there were no obstacles or barriers, the water drops would follow a straight path toward the destination, which is the shortest path from the source to the destination. However, due to different kinds of obstacles in their way to the destination, which constrain the path construction, the real path has to be different from the ideal path and lots of twists and turns in the river path is observed. The interesting point is that this constructed path seems to be optimum in terms of distance from the destination and the constraints of the environment.

Imagine a water drop is going to move from a point of river to the next point in the front as shown in Figure 1. It is assumed that each water drop flowing in a river can carry an amount of soil which is shown by the size of the water drop in the figure. The amount of soil of the water drop increases as it reaches to the right point shown in Figure 1 while the soil of the river bed decreases. In fact, some amount of soil of the river bed is removed by the water drop and is added to the soil of the water drop. This property is embedded in the IWDs such that each IWD holds soil in itself and removes soil from its path during movement in the environment.
A water drop has also a velocity and this velocity plays an important role in the removing soil from the beds of rivers. Let two water drops having the same amount of soil move from a point of a river to the next point as shown in Figure 2. The water drop with bigger arrow has higher velocity than the other one. When both water drops arrive at the next point on the right, the faster water drop is assumed to gather more soil that the other one. This assumption is shown in Figure 2 in which a bigger circle on the right, which has gathered more soil, denotes the faster water drop. The mentioned property of soil removing which is dependent on the velocity of the water drop is embedded in each IWD of the IWD algorithm.

It was stated above that the velocity of an IWD flowing over a path determines the amount of soil that is removed from the path. In contrast, the velocity of the IWD is also changed by the path such that a path with little amount of soil increases the velocity of the IWD more than a path with a considerable amount of soil. This assumption is shown in Figure 3 in which two identical water drops with the same velocity flow on two different paths. The 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.

What makes a water drop choose one branch of path among several choices it has in its front? Obviously, a 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 the destination.

In the IWD algorithm, the hardness is translated to the amount of soil on the path. If a branch of the path contains higher amount of soil than other branches, it becomes less desirable than the other ones. This branch selection on the path is implemented by a probabilistic function of inverse of soil, which is explained in the next section.

In nature, countless water drops flow together to form the optimal path for reaching their destination. In other words, it is a population-based intelligent mechanism. The IWD algorithm employs this mechanism by using a population of IWDs to construct paths and among all these paths over time, the optimal or near optimal path emerges.

4. The proposed IWD algorithm

The IWD (Shah-Hosseini, 2007) have been designed to imitate the prominent properties of the natural water drops that flow in the beds of rivers. Each IWD is assumed to have an amount of the soil it carries, soil(IWD), and its current velocity, velocity(IWD).

The environment in which IWDs are moving is assumed to be discrete. This environment may be considered to be composed of \( N \) nodes and each IWD needs to move from one node to another. Every two nodes are linked by an arc which holds an amount of soil. Based on the activities of the IWDs flowing in the environment, the soil of each arc may be increased or decreased.

Consider an IWD is in the node \( i \) and wants to move to the next node \( j \). The amount of the soil on the arc between these two nodes, represented by soil\( (i, j) \), is used for updating the velocity vel\( ^{IWD} \) \( t \)) of the IWD by:

\[
vel^{IWD}(t + 1) = vel^{IWD}(t) + \begin{cases} 
\frac{a_v}{b_v + c_v \cdot soil^\beta(i, j)} & \text{if } soil^\beta(i, j) \neq b_v \frac{1}{c_v} \\
0 & \text{otherwise}
\end{cases}
\]  

(4)

where vel\( ^{IWD} \) \( t + 1 \) represents the updated velocity of the IWD at the next node \( j \). Moreover, \( a_v, b_v, \) and \( c_v \) are some constant velocity parameters that are set for the given problem.

Figure 3.
Two identical IWDs flow in two different rivers

Notes: The IWD that flows in the river with less soil gathers more soil and gets more increase in speed
According to the velocity updating in equation (4), the velocity of the IWD increases if the soil \( b(i,j) \) remains in the open interval \( ](-b_v/c_v, c_v[ \). The more the amount of the soil \( b(i,j) \) the less the updated velocity \( v_{IWD}(t + 1) \) will be. In contrast, if soil \( b(i,j) \) be in the open interval \( ] - \infty, (-b_v/c_v)[ \), the velocity of the IWD decreases such that the less the amount of soil \( b(i,j) \) the less the updated velocity \( v_{IWD}(t + 1) \) will be.

In the original work on the IWD-based algorithm (Shah-Hosseini, 2007), the parameter \( \beta \) was not considered and implicitly, it was assumed that \( \beta = 1 \). Assuming that the \( a_v, b_v, \) and \( c_v \) are chosen as positive values, then selecting an even power for the soil \( b(i,j) \) in equation (4) has this advantage that the velocity \( v_{IWD}(t + 1) \) never gets negative even if the soil \( b(i,j) \) reaches below zero and the velocity updating in equation (4) reduces to the following formula:

\[
v_{IWD}(t + 1) = v_{IWD}(t) + \frac{a_v}{b_v + c_v \cdot b(i,j)}.
\]  

(5)

Such that \( \beta = 2 \alpha \). To avoid possible negative values for the velocity, in this paper, \( \beta = 2 \).

Consider that a local heuristic function \( Hud(.,.) \) has been defined for a given problem to measure the undesirability of an IWD to move from one node to another. The time taken for an IWD having the velocity \( v_{IWD}(t + 1) \) to move from the current node \( i \) to its next node \( j \), denoted by \( time(i,j; v_{IWD}(t + 1)) \), is calculated by:

\[
time(i,j; v_{IWD}) = \frac{Hud(i,j)}{v_{IWD}}.
\]  

(6)

Such that:

\[
v_{IWD} = v_{IWD}(t + 1) + \begin{cases} 
\epsilon & \text{if } |v_{IWD}(t + 1)| < \epsilon \\
0 & \text{otherwise}
\end{cases}
\]  

(7)

where \( v_{IWD} \) is obtained from \( v_{IWD}(t + 1) \) to keep its value away from zero with radius \( \epsilon \). The constant parameter \( \epsilon \) is a small positive value. Here, \( \epsilon = 0.001 \). The function \( Hud(i,j) \) denotes the heuristic undesirability of moving from node \( i \) to node \( j \).

For the TSP, the form of the HUD \( (i,j) \) denoted by \( Hud_{TSP}(i,j) \) has been suggested as follows:

\[
Hud(i,j) = Hud_{TSP}(i,j) = ||c(i) - c(j)||
\]  

(8)

where \( c(k) \) represents the two dimensional positional vector for the city \( k \). The function \( || \cdot || \) calculates the Euclidean norm. As a result, when two nodes (cities) \( i \) and \( j \) are near to each other, the heuristic undesirability measure \( Hud(i,j) \) becomes small which reduces the time taken for the IWD to pass from city \( i \) to city \( j \).

For the MKP, a few heuristics have been suggested and used in ant-based optimization algorithms (Dorigo and Stutzle, 2004), which some of the heuristics are almost complex. Here, a simple local heuristic is used which reflects the undesirability of adding an item to the current partial solution. Let the heuristic undesirability \( Hud(i,j) \) for the MKP denoted by \( Hud_{MKP}(j) \) be defined as:
\[
\text{HUD}_{\text{MKP}}(j) = \frac{\tilde{r}_j}{b_j}, \tag{9}
\]

Such that:
\[
\tilde{r}_j = \frac{1}{m} \sum_{k=1}^{m} r_{jk} \tag{10}
\]

where \(b_j\) is the profit of item \(j\) and \(\tilde{r}_j\) is the average resource requirement for item \(j\). As equation (9) shows HUD_{MKP}(j) decreases if the profit \(b_j\) is high while HUD_{MKP}(j) increases if the average resource requirements \(\tilde{r}_j\) becomes high. Therefore, among the items that can be selected for the next move of an IWD, the item which needs less resource requirements and has higher profit is more desirable.

As an IWD moves from the current node \(i\) to its next node \(j\), it removes an amount of soil from the path (arc) joining the two nodes. The amount of the soil being removed depends on the velocity of the moving IWD. For the TSP (Shah-Hosseini, 2007), it was suggested to relate the amount of the soil taken from the path with the inverse of the time that the IWD needs to pass the arc or path between the two nodes. So, a fast IWD removes more soil from the path it flows on than a slower IWD. This mechanism is an imitation of what happens in the natural rivers. Fast rivers can make their beds deeper because they remove more soil from their beds in a shorter time while slow flowing rivers lack such strong soil movements. Moreover, even in a single river, parts of the river that water drops flow faster often have deeper beds than the slower parts.

Specifically, for the TSP, the amount of the soil that the IWD removes from its current path from node \(i\) to node \(j\) is calculated by:
\[
\Delta \text{soil}(i,j) = \frac{a_s}{b_s + c_s \cdot \text{time}(i,j; \text{vel}^{\text{IWD}})} \tag{11}
\]

where \(\Delta \text{soil}(i,j)\) is the soil which the IWD with velocity \(\text{vel}^{\text{IWD}}\) removes from the path between node \(i\) and \(j\). The \(a_s\), \(b_s\), and \(c_s\) are constant velocity parameters that their values depend on the given problem. The value \(\text{time}(i,j; \text{vel}^{\text{IWD}})\) was defined in equation (6) and represents the time taken for the IWD to flow from \(i\) to \(j\).

Here, equation (11) is slightly improved to include a power for the time value in the denominator as follows:
\[
\Delta \text{soil}(i,j) = \begin{cases} 
\frac{a_s}{b_s + c_s \cdot \text{time}^\omega(i,j; \text{vel}^{\text{IWD}})} & \text{if } \text{time}^\omega(i,j; \text{vel}^{\text{IWD}}) \neq -\frac{b_s}{c_s} \\
0 & \text{otherwise} 
\end{cases} \tag{12}
\]

In the IWD algorithm for the TSP, the parameter \(\omega\) was not considered and thus implicitly \(\omega = 1\). For the MKP, the parameter \(\omega\) is set to two. Again, by assuming the parameters \(a_s\), \(b_s\), and \(c_s\) are selected as positive numbers, then selecting an even value for power \(\omega = 2\theta\) simplifies equation (12) to:
\[
\Delta \text{soil}(i,j) = \frac{a_s}{b_s + c_s \cdot \text{time}^{2\theta}(i,j; \text{vel}^{\text{IWD}})}. \tag{13}
\]

After an IWD moves from node \(i\) to node \(j\), the soil \(\text{soil}(i,j)\) on the path between the two nodes is reduced by:
\[
\text{soil}(i,j) = \rho_o \cdot \text{soil}(i,j) - \rho_h \cdot \Delta \text{soil}(i,j). \tag{14}
\]
Where $\rho_o$ and $\rho_n$ are positive numbers that should be chosen between zero and one. In the original algorithm for the TSP, $\rho_o = 1 - \rho_n$.

The IWD that has moved from node $i$ to $j$, increases the soil $\text{soil}^{\text{IWD}}$ it carries by:

$$\text{soil}^{\text{IWD}} = \text{soil}^{\text{IWD}} + \Delta\text{soil}(i, j)$$  \hspace{1cm} (15)

where $\Delta\text{soil}(i, j)$ is obtained from equation (13). Therefore, the movement of an IWD between two nodes reduces the soil on the path between the two nodes and increases the soil of the moving IWD.

One important mechanism that each IWD must contain is how to select its next node. An IWD prefers a path that contains less amount of soil rather than the other paths. This preference is implemented by assigning a probability to each path from the current node to all valid nodes which do not violate constraints of the given problem. Let an IWD be at the node $i$, then the probability $p_i^{\text{IWD}}(j)$ of going from node $i$ to node $j$ is calculated by:

$$p_i^{\text{IWD}}(j) = \frac{f(\text{soil}(i, j))}{\sum_{k \notin vc(IWD)} f(\text{soil}(i, k))}.$$  \hspace{1cm} (16)

Such that $f(\text{soil}(i, j))$ computes the inverse of the soil between node $i$ and $j$. Specifically:

$$f(\text{soil}(i, j)) = \frac{1}{\epsilon_s + g(\text{soil}(i, j))}.$$  \hspace{1cm} (17)

The constant parameter $\epsilon_s$ is a small positive number to prevent a possible division by zero in the function $f()$. It is suggested to use $\epsilon_s = 0.01$. $g(\text{soil}(i, j))$ is used to shift the soil $\text{soil}(i, j)$ on the path joining nodes $i$ and $j$ toward positive values and is computed by:

$$g(\text{soil}(i, j)) = \begin{cases} 
\text{soil}(i, j) & \text{if } \min_{l \notin vc(IWD)} (\text{soil}(i, l)) \geq 0 \\
\text{soil}(i, j) - \min_{l \notin vc(IWD)} (\text{soil}(i, l)) & \text{else}
\end{cases}.$$  \hspace{1cm} (18)

The function $\min()$ returns the minimum value of its arguments. The set $vc(IWD)$ denotes the nodes that the IWD should not visit to keep satisfied the constraints of the problem.

Every IWD that has been created in the algorithm moves from its initial node to next nodes till it completes its solution. For the 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 $T_i^{\text{IWD}}$ found by the IWD is given by $q(T_i^{\text{IWD}})$. One iteration of the IWD algorithm is said to be complete when all IWDs have constructed their solutions. At the end of each iteration, the best solution $T_i^{\text{IB}}$ of the iteration found by the IWDs is obtained by:

$$T_i^{\text{IB}} = \arg \max_{T_i^{\text{IWD}}} q(T_i^{\text{IWD}}).$$  \hspace{1cm} (19)

Therefore, the iteration-best solution $T_i^{\text{IB}}$ is the solution that has the highest quality over all solutions $T_i^{\text{IWD}}$. 
Based on the quality of the iteration-best solution, $q(T_{IB})$, only the paths of the solution $T_{IB}$ are updated. This soil updating should include the amount of quality of the solution. Specifically:

$$\text{soil}(i,j) = \rho_s \cdot \text{soil}(i,j) + \rho_{IWD} \cdot k(N_c) \cdot \text{soil}_{IB}^{IWD} \quad \forall (i,j) \in T_{IB}. \quad (20)$$

Where $\text{soil}_{IB}^{IWD}$ represents the soil of the iteration-best IWD. The best-iteration IWD is the IWD that has constructed the best-iteration solution $T_{IB}$. $k(N_c)$ denotes a positive coefficient which is dependent on the number of nodes $N_c$. Here, $k(N_c) = 1/(N_c - 1)$ is used. $\rho_s$ should be a constant positive value whereas the constant parameter $\rho_{IWD}$ should be a negative value. The first term on the right-hand side of equation (20) represents the amount of the soil that remains from the previous iteration. In contrast, the second term on the right-hand side of equation (20) reflects the quality of the current solution, obtained by the IWD. Therefore, in equation (20), a proportion of the soil gathered by the IWD is reduced from the total soil $\text{soil}(i,j)$ of the path between node $i$ and $j$.

This way, the best-iteration solutions are gradually reinforced and they lead the IWDs to search near the good solutions in the hope of finding the globally optimal solution.

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:

$$T_{TB} = \begin{cases} T_{TB} & \text{if } q(T_{TB}) \geq q(T_{IB}) \\ T_{TB} & \text{otherwise} \end{cases} \quad (21)$$

By doing this, it is guaranteed that $T_{TB}$ holds the best solution obtained so far by the IWD algorithm.

In summary, the proposed IWD algorithm for the MKP is specified in the following steps:

- **Step 1.** Initialization of static parameters: the number of items $N_c$ along with the profit $b_i$ for each item $i$, the number of constraints $m$ such that each resource constraint $j$ has the capacity $a_j$, the resource matrix $R$ with size $N_c \times m$, which holds the elements $r_{ij}$ are all the parameters of the given MKP.

  Set the number of water drops $N_{IWD}$ to a positive integer value. Here, it is suggested that $N_{IWD}$ is set equal to the number of items $N_c$. For velocity updating, the parameters are set as $a_v = 1$, $b_v = 0.01$, and $c_v = 1$. For soil updating, $a_s = 1$, $b_s = 0.01$, and $c_s = 1$. The local soil updating parameter $\rho_{n}$, which should be a small positive number less than one, is chosen as $\rho_{n} = 0.9$. The global soil updating parameter $\rho_{IWD}$, which should be chosen from $[-1, 0]$, is set as $\rho_{IWD} = -0.9$. Moreover, the initial soil on each path is denoted by the constant InitSoil such that the soil of the path between every two items $i$ and $j$ is set by $\text{soil}(i,j) = \text{InitSoil}$. The initial velocity of IWDs is denoted by the constant InitVel. Both parameters InitSoil and InitVel are also user selected. In this paper, InitSoil = 1,000 and InitVel = 4. The quality of the best solution $q(T_{TB})$ is initially set as: $q(T_{TB}) = -\infty$. Moreover, the maximum number of iterations $it_{\text{max}}$ that the algorithm should be repeated needs to be specified.

- **Step 2.** Initialization of dynamic parameters: For every IWD, a visited node list $V_c(IWD)$ is considered and is set to the empty list: $V_c(IWD) = \{\}$. The velocity of each IWD is set to InitVel and the initial soil of each IWD is set to zero.
Step 3. For every IWD, randomly select a node and associate the IWD to this node.

Step 4. Update the visited node list of each IWD to include the nodes just visited.

Step 5. For each IWD that has not completed its solution, repeat Steps 5.1-5.4.

Step 5.1. Choose the next node $j$ to be visited by the IWD among those that are not in its visited node list and do not violate the $m$ constraints defined in equation (3). When there is no unvisited node that does not violate the constraints, the solution of this IWD has been completed. Otherwise, choose next node $j$ when the IWD is in node $i$ with the probability $p_{i}^{IWD}(j)$ defined in equation (16) and update its visited node list.

Step 5.2. For each IWD moving from node $i$ to node $j$, update its velocity $vel_{IWD}^{IWD}(t)$ by setting $a = 1$ in equation (5) which yields:

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

Such that $vel_{IWD}^{IWD}(t + 1)$ is the updated velocity of the IWD.

Step 5.3. Compute the amount of the soil, $\Delta soil(i,j)$, that the current water drop IWD with the updated velocity $vel_{IWD}^{IWD} = vel_{IWD}^{IWD}(t + 1)$ loads from its current path between two nodes $i$ and $j$ by setting $\theta = 1$ in equation (13):

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s \cdot time^2(i,j; vel_{IWD}^{IWD})}.$$  

Such that:

$$time(i,j; vel_{IWD}^{IWD}) = \frac{HUD_{MKP}(j)}{vel_{IWD}^{IWD}}$$

where the heuristic undesirability $HUD_{MKP}(j)$ is computed by equation (9).

Step 5.4. Update the soil of the path traversed by that IWD, $soil(i,j)$, and the soil that the IWD carries, $soil_{IWD}^{IWD}$, using equations (14) and (15) as follows:

$$soil(i,j) = (1 - \rho_n) \cdot soil(i,j) - \rho_n \cdot \Delta soil(i,j)$$

$$soil_{IWD}^{IWD} = soil_{IWD}^{IWD} + \Delta soil(i,j).$$  

Step 6. Find the iteration-best solution $T^IB$ from all the solutions found by the IWDs using equation (19).

Step 7. Update the soils of the paths that exist in the current iteration-best solution $T^IB$ using equation (20) by setting $\rho_s = (1 - \rho_{IWD})$:

$$soil(i,j) = (1 - \rho_{IWD}) \cdot soil(i,j) + \rho_{IWD} \cdot \frac{1}{(N_c - 1)} \cdot soil_{IB}^{IWD} \quad \forall(i,j) \in T^IB.$$  

Step 8. Update the total best solution $T^TB$ by the current iteration-best solution $T^IB$ using equation (21).

Step 9. Go to Step 2 until the maximum number of iterations is reached.

Step 10. The algorithm stops here with the final solution $T^TB$. 

It is possible to use only $T_M$ and remove Step 8 of the IWD algorithm. But, by doing this, some good solutions may temporarily be lost and it takes more time of the algorithm to find them again. Therefore, it is better to keep the total best solution $T^{TB}$ of all iterations than to count only on the iteration-best solution $T^{IB}$.

The steps of the proposed IWD algorithm are expressed in two flowcharts shown in Figure 4. The flowchart in Figure 4(a) shows the main steps of the algorithm. The Step 5 of the IWD algorithm is depicted with more details in the flowchart of Figure 4(b).

![Flowchart of proposed IWD algorithm](image)
5. Convergence properties of the IWD algorithm

In this section, the purpose is to show that the IWD algorithm is able to find the optimal solution at least once during its lifetime if the number of iterations that the algorithm is run be sufficiently big. For a few particular ACO algorithms and careful setting of parameters of the ACO, such property has been shown to exist and this kind of convergence is called convergence in value (Dorigo and Stutzle, 2004). In the following, the convergence in value for the IWD algorithm is investigated.

For any IWD in the proposed algorithm, the next node of the IWD is found probabilistically by using equation (16). Therefore, as long as the probability of visiting any node is above zero, in the long run, it is expected with probability one that an IWD of the algorithm will choose that node at some iteration.

Any solution $S$ of the given problem is composed of a number of nodes $\{n_p, n_q, \ldots, n_r\}$ selected by an IWD during an iteration of the algorithm. As a result, if it is shown that the chance of selecting any node $n_k$ in the graph $(N, E)$ of the problem is above zero, then the chance of finding any feasible solution from the set of all solutions of the problem is nonzero. As a consequence, if it is proved that there is positive chance for any feasible solution to be found by an IWD in an iteration of the algorithm, it will be guaranteed that the optimal solution is found. Because, once an IWD finds an optimal solution, that solution becomes the iteration-best solution in the algorithm and thus the total-best solution is updated to the newly found optimal solution as expressed in Step 8 of the algorithm. In summary, the convergence in value is proven to exist if the probability of choosing any node of the problem’s graph in a solution is nonzero.

Let the graph $(N, E)$ represents the graph of the given problem. This graph is a fully connected graph with $N_c$ nodes. Also, let $N_{IWD}$ represents the number of IWDs in the algorithm. In the soil updating of the algorithm, two extreme cases are considered. Case 1 which includes only those terms that increase soil to an arc of $(N, E)$ and case 2 which includes only those terms that decrease the soil to an arc of $(N, E)$. For each case, the worst-case is followed. For case 1, the highest possible value of soil that an arc can hold is computed. For case 2, the lowest possible value of soil for an arc is computed. The equations (14) and (20) contain the formulas that update the soil of an arc. In the following, each case is studied separately.

**Case 1.** For simplicity, the initial soil of an arc $(i, j)$ is denoted by $IS_0$. This arc $(i, j)$ is supposed to contain the maximum possible value of soil and is called “arcmax”. For equation (14), the first term on the right hand side, $\rho_o soil(i, j)$, is the only term with positive sign. To consider the extreme case, it is assumed that in one iteration of the algorithm, this term is applied just once to the arc because the parameter $\rho_o$ is supposed to have its value between zero and one. For equation (20), the first term on the right hand side, $\rho_s soil(i, j)$, has positive sign. In the extreme case, this term is applied once in one iteration of the algorithm. As a result, by replacing soil$(i, j)$ with $IS_0$ in the mentioned terms, the amount of soil of arcmax will be $((\rho_o^M) IS_0)$ after one iteration.

Let $m$ denotes the number of iterations that the algorithm has been repeated so far. Therefore, the soil of arcmax, soil(arcmax), will have the soil $((\rho_o^M) IS_0)$ at the end of $m$ iterations of the algorithm:

$$soil(\text{arc max }) = ((\rho_o^M) IS_0). \tag{26}$$

**Case 2.** In this case, the lowest amount of soil of an arc $(i, j)$ is estimated. Let arcmin denote the aforementioned arc $(i, j)$. Here, only the negative terms of
equations (14) and (20) are considered. From the equation (14), the term \(-\rho_c \Delta \text{soil}(i, j)\) is supposed to be applied \(N_{\text{IWD}}\) times to the arcmin in one iteration, which is the extreme case for making the soil as lowest as possible. The extreme high value for \(\Delta \text{soil}(i, j)\) is obtained from equation (13) by setting the time in the denominator to zero, which yields the positive value \(a_s/b_s\). Therefore, the most negative value in one iteration that can come from equation (14) is the value: \((-\rho_c N_{\text{IWD}}(a_s/b_s))\).

From equation (20), the term \(\rho_{\text{IWD}} \cdot k(N_c) \cdot \text{soil}_{\text{IB}}^{\text{IWD}}\) is the negative term. The highest value of \(\text{soil}_{\text{IB}}^{\text{IWD}}\) can be \((N_c - 1)(a_s/b_s)\) and since \(k(N_c) = 1/(N_c - 1)\), the most negative value for the term will be \((\rho_{\text{IWD}}(a_s/b_s))\). As a result, in one iteration of the algorithm the arcmin has the amount of soil that is greater than or equal to the value \(((\rho_{\text{IWD}} - \rho_c N_{\text{IWD}})(a_s/b_s))\). Similar to case 1, \(m\) denotes the number of iterations that the algorithm has been repeated. Therefore, the soil of arcmin, \(\text{soil}^{\text{arc min}}\), has the soil \((m(\rho_{\text{IWD}} - \rho_c N_{\text{IWD}})(a_s/b_s))\) after \(m\) iterations:

\[
\text{soil}^{\text{arc min}} = \left(m(\rho_{\text{IWD}} - \rho_c N_{\text{IWD}}) \frac{a_s}{b_s}\right).
\]

The \(\text{soil}^{\text{arcmin}}\) and \(\text{soil}^{\text{arcmax}}\) are the extreme lower and upper bounds of the soil of arcs in the graph \((N, E)\) of the given problem, respectively. Therefore, the soil of any arc after \(m\) iterations of the IWD algorithm remains in the interval \([\text{soil}^{\text{arcmin}}, \text{soil}^{\text{arcmax}}]\).

Consider the algorithm is at the stage of choosing the next node \(j\) for an IWD when it is in node \(i\). The value \(g(\text{soil}(i, j))\) of arc \((i, j)\) is calculated from equation (18), which positively shifts \(\text{soil}(i, j)\) by the amount of the lowest negative soil value of any arc, \(\min_{j \in \text{arc(IWD)}} (\text{soil}(i, j))\) as explained before. To consider the worst-case, let this lowest negative soil value be \(\text{soil}^{\text{arcmin}}\) and the \(\text{soil}(i, j)\) be equal to \(\text{soil}^{\text{arcmax}}\). As a result, the value of \(g(\text{soil}(i, j))\) becomes \((\text{soil}^{\text{arcmax}} - \text{soil}^{\text{arcmin}})\) with the assumption that \(\text{soil}^{\text{arcmin}}\) is negative, which is the worst case. Equation (16) is used to calculate the probability of an IWD going from node \(i\) to \(j\). For this purpose, \(f(\text{soil}(i, j))\) needs to be computed by equation (17) which yields:

\[
f(\text{soil}(i, j)) = \frac{1}{\varepsilon_s + (\text{soil}^{\text{arc max}} - \text{soil}^{\text{arc min}})}.
\]

The denominator of formula (16) becomes its largest possible value when it is assumed that each \(\text{soil}(i, k)\) in equation (16) is zero. Consequently, the probability of the IWD going from node \(i\) to node \(j\), \(p_{i}^{\text{IWD}}(j)\), will be bigger than \(p_{\text{lowest}}\) such that:

\[
p_{i}^{\text{IWD}}(j) > p_{\text{lowest}} = \frac{\varepsilon_s}{(N_c - 1)(\varepsilon_s + (\text{soil}^{\text{arc max}} - \text{soil}^{\text{arc min}}))}.
\]

The value of \(p_{\text{lowest}}\) is above zero.

With some assumptions on the relations between parameters of the algorithm, \(p_{\text{lowest}}\) can become even bigger. For example, if it is assumed that \((\rho_c \rho_o) < 1\), then \(\text{soil}^{\text{arcmax}}\) in equation (26) goes to zero as \(m\) increases. Moreover, if \(\rho_o = \rho_{\text{IWD}} / N_{\text{IWD}}\), then \(\text{soil}^{\text{arcmin}}\) becomes zero. These two assumptions yield that \(\text{soil}^{\text{arc max}} - \text{soil}^{\text{arc min}} = 0\). Therefore, \(p_{\text{lowest}} = 1/(N_c - 1)\), which is again above zero and it is the biggest value that \(p_{\text{lowest}}\) can get in the worse-case.
The probability of finding any feasible solution by an IWD in the iteration $m$ will be $(p_{\text{lowest}})^{(N_c-1)}$. Since there are $N_{\text{IWD}}$ IWDs, then the probability $p(s; m)$ of finding any feasible solution $s$ by the IWDs in iteration $m$ is:

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

(30)

Now, the probability of finding any feasible solution $s$ at the end of $M$ iterations of the algorithm will be:

$$P(s; M) = 1 - \prod_{m=1}^{M} (1 - p(s; m)).$$

(31)

Because $0 < p(s; m) \leq 1$, then by making $M$ large, the term becomes small toward zero:

$$\lim_{M \to \infty} \prod_{m=1}^{M} (1 - p(s; m)) = 0.$$

Therefore:

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

This fact indicates that any solution $s$ of the given problem can be found at least once by at least one IWD of the algorithm if the number of iterations of the algorithm, $M$, is big enough. The following proposition summarizes the above finding.

**Proposition 5.1.** 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:

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

(32)

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

**Proposition 5.2.** The IWD algorithm finds the optimal solution $s^*$ of the 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 a given problem.

### 6. Experimental results

The proposed IWD algorithm for solving the MKP is tested here with a set of MKPs mentioned in the OR-Library (OR-Library, http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files). For each test problem, the algorithm is run for ten times. It is reminded that all experiments are implemented on a Personal Computer having Pentium 4 CPU, 1.80 GHz, and Windows XP using C# language in the environment Microsoft Visual Studio 2005.

The first data set that is used for testing the proposed IWD-MKP algorithm comes from the file “mknap1.txt” of the OR-Library which contains seven test problems of the MKP. For these seven problems, the qualities of the optimal solutions are known. Therefore, the IWD-MKP algorithm is tested with these problems to see whether the algorithm is able to find the optimal solutions or not.
Table I reports the quality of the total best of each run of the IWD-MKP algorithm for each test problem in the file “mknap1.txt”. The IWD-MKP reaches the optimal solutions for five of the problems in the average number of iterations reported in Table I. For the other two test problems, the algorithm reaches very near-optimal solutions after 100 iterations. For the problem with ten constraints and 20 items, the qualities of iteration-best solutions for the ten runs of the IWD algorithm are shown in Figure 5. The best run of the algorithm converges to the optimum solution 6,120 in four iterations whereas its worst run converges in 39 iterations. Similar convergence curves are shown in Figure 6 for the problem with ten constraints and 28 items. The best run converges to the optimum solution 12,400 in five iterations whereas the worst run of the algorithm converges in 20 iterations.

The second data set is taken from the file “mknapcb1” of the OR-Library in which each test problem has five constraints and 100 items. Table II shows the results of applying the proposed IWD-MKP to the first ten problems of the set. For each problem, the best and the average quality of ten runs of the IWD-MKP is reported. For comparison, the results of the two Ant Colony Optimization-based algorithms of Leguizamon and Michalewicz (1999) (for short, L&M) and Fidanova (2002) are mentioned. Moreover, the results obtained by the LP relaxation method that exist in the

<table>
<thead>
<tr>
<th>Constraints × variables</th>
<th>Quality of optimum solution</th>
<th>The solution quality of the IWD-MKP</th>
<th>Average no. of iterations of the IWD-MKP</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 × 6</td>
<td>3,800</td>
<td>3,800</td>
<td>3.3</td>
</tr>
<tr>
<td>10 × 10</td>
<td>8,786.1</td>
<td>8,786.1</td>
<td>12.9</td>
</tr>
<tr>
<td>10 × 15</td>
<td>4,015</td>
<td>4,015</td>
<td>30.9</td>
</tr>
<tr>
<td>10 × 20</td>
<td>6,120</td>
<td>6,120</td>
<td>18.7</td>
</tr>
<tr>
<td>10 × 28</td>
<td>12,400</td>
<td>12,400</td>
<td>11.9</td>
</tr>
<tr>
<td>5 × 39</td>
<td>10,618</td>
<td>10,563.6</td>
<td>100</td>
</tr>
<tr>
<td>5 × 50</td>
<td>16,537</td>
<td>16,405</td>
<td>100</td>
</tr>
</tbody>
</table>

**Note:** The actual optimal qualities are known for these problems and are shown below.

Intelligent water drops algorithm

---

Table I. The problems of the OR-Library in file “mknap1.txt”, which are solved by the IWD-MKP algorithm

**Figure 5.** Convergence curves of ten runs of the IWD algorithm for an MKP in Table I with the optimum 6,120

**Notes:** Each color in the figure shows one run of the algorithm
The solutions of the IWD-MKP are often better than the solutions obtained by Fidanova. They are also near to the results of the L&M and LP relaxation methods. These near-optimal results of the proposed IWD-MKP are obtained by using a simple local heuristic that has been used in the algorithm whereas the results of the L&M ACO-based algorithm are obtained by defining a rather complex heuristic definition. Generally, it is seen that the qualities of solutions of LP relaxation are better than other algorithms in Table II. Therefore, there is much space to improve these population-based optimization algorithms, including the proposed IWD-MKP algorithm, to reach the qualities of the specialized optimization algorithms such as LP relaxation or the algorithms in Vasquez and Hao (2001) and Vasquez and Yannick (2005) which are a combination of LP relaxation and tabu search.

<table>
<thead>
<tr>
<th>Constraints × 100</th>
<th>Variables-problem number</th>
<th>LP optimal</th>
<th>L&amp;M Best</th>
<th>Fidanova Best</th>
<th>Quality of the IWD-MKP's solutions_best</th>
<th>Quality of the IWD-MKP's solutions_average</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 × 100-00</td>
<td></td>
<td>24,585</td>
<td>24,381</td>
<td>23,984</td>
<td>24,295</td>
<td>24,175.4</td>
</tr>
<tr>
<td>5 × 100-01</td>
<td></td>
<td>24,538</td>
<td>24,274</td>
<td>24,158</td>
<td>24,158</td>
<td>24,031.3</td>
</tr>
<tr>
<td>5 × 100-02</td>
<td></td>
<td>23,895</td>
<td>23,551</td>
<td>23,523</td>
<td>23,518</td>
<td>23,404</td>
</tr>
<tr>
<td>5 × 100-03</td>
<td></td>
<td>23,724</td>
<td>23,527</td>
<td>22,874</td>
<td>23,218</td>
<td>23,120.9</td>
</tr>
<tr>
<td>5 × 100-04</td>
<td></td>
<td>24,223</td>
<td>23,991</td>
<td>23,751</td>
<td>23,802</td>
<td>23,737.2</td>
</tr>
<tr>
<td>5 × 100-05</td>
<td></td>
<td>24,884</td>
<td>24,613</td>
<td>24,601</td>
<td>24,601</td>
<td>24,554</td>
</tr>
<tr>
<td>5 × 100-06</td>
<td></td>
<td>25,793</td>
<td>25,591</td>
<td>25,293</td>
<td>25,521</td>
<td>25,435.6</td>
</tr>
<tr>
<td>5 × 100-07</td>
<td></td>
<td>23,657</td>
<td>23,410</td>
<td>23,204</td>
<td>23,374</td>
<td>23,344.9</td>
</tr>
<tr>
<td>5 × 100-08</td>
<td></td>
<td>24,445</td>
<td>24,204</td>
<td>23,762</td>
<td>24,148</td>
<td>24,047</td>
</tr>
<tr>
<td>5 × 100-09</td>
<td></td>
<td>24,635</td>
<td>24,411</td>
<td>24,255</td>
<td>24,366</td>
<td>24,317</td>
</tr>
</tbody>
</table>

Notes: The results are compared with the LP optimal solutions and best solutions of two ant-based algorithms: Leguizamon and Michalewicz (L&M), and Fidanova.

Table II.  
The problems with five constraints and 100 items of the OR-Library in file “mknapch1.txt” are solved using 100 iterations of the proposed IWD-MKP algorithm.

Figure 6.  
Convergence curves of ten runs of the IWD algorithm for an MKP in Table I with the optimum 12,400.
The problems with ten constraints and 100 items of the OR-Library in file “mknapcb4.txt” are also solved using 100 iterations of the IWD-MKP algorithm and the solutions are reported in Table III. The results of the IWD-MKP are compared with the LP optimal solutions and best solutions of two other Ant Colony Optimization-based algorithms: L&M (Leguizamon and Michalewicz, 1999), and Ant-Knapsack (Alaya et al., 2004). Again, the solutions of the LP relaxation are better than the other methods in the table. The solutions of the IWD-MKP are close to those of LP relaxation and other two ACO algorithms.

The experiments show that the proposed IWD-MKP algorithm is capable to obtain optimal or near optimal solutions for different kinds of MKPs.

### 7. Conclusion

In this paper, a new population-based optimization algorithm called “Intelligent Water Drop” algorithm, which is based on the mechanisms that exist in natural rivers and between the water drops, is proposed for the MKP and thus is called “IWD-MKP”. The IWD-MKP considers an MKP as a graph and lets each IWD traverse the arcs between the nodes of the graph and change their amounts of soil according to the mechanisms embedded in the algorithm. In fact, each IWD constructs a solution while modifying its environment. Then, at the end of each iteration of the algorithm, the iteration-best solution is found and rewarded by reducing an amount of soil from all the arcs that form the solution. The amount of soil that is reduced is proportional to the amount of soil that IWD has gathered from the arcs of the solution in this iteration.

A simple local heuristic is used in the proposed IWD-MKP algorithm and the IWD-MKP algorithm is tested with different kinds of MKPs. The solutions that are obtained by the IWD-MKP are optimal or near-optimal solutions. The convergence properties of the IWD algorithm is also discussed and showed that it has the property of convergence in value.

<table>
<thead>
<tr>
<th>Constraints × variables-problem number</th>
<th>LP optimal</th>
<th>L&amp;M Best</th>
<th>Ant-knapsack Best</th>
<th>Quality of the IWD-MKP’s solutions Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 × 100-00</td>
<td>23,480</td>
<td>23,057</td>
<td>23,064</td>
<td>22,936</td>
<td>22,754</td>
</tr>
<tr>
<td>10 × 100-01</td>
<td>23,220</td>
<td>22,801</td>
<td>22,801</td>
<td>22,591</td>
<td>22,517.8</td>
</tr>
<tr>
<td>10 × 100-02</td>
<td>22,493</td>
<td>22,131</td>
<td>22,131</td>
<td>21,969</td>
<td>21,793</td>
</tr>
<tr>
<td>10 × 100-03</td>
<td>23,087</td>
<td>22,772</td>
<td>22,717</td>
<td>22,416</td>
<td>22,269.2</td>
</tr>
<tr>
<td>10 × 100-04</td>
<td>23,073</td>
<td>22,654</td>
<td>22,654</td>
<td>22,466</td>
<td>22,270.9</td>
</tr>
<tr>
<td>10 × 100-05</td>
<td>23,053</td>
<td>22,652</td>
<td>22,716</td>
<td>22,475</td>
<td>22,369.2</td>
</tr>
<tr>
<td>10 × 100-06</td>
<td>22,257</td>
<td>21,875</td>
<td>21,875</td>
<td>21,731</td>
<td>21,634.8</td>
</tr>
<tr>
<td>10 × 100-07</td>
<td>22,964</td>
<td>22,551</td>
<td>22,551</td>
<td>22,542</td>
<td>22,226.4</td>
</tr>
<tr>
<td>10 × 100-08</td>
<td>22,882</td>
<td>22,418</td>
<td>22,511</td>
<td>22,218</td>
<td>22,099.3</td>
</tr>
<tr>
<td>10 × 100-09</td>
<td>23,090</td>
<td>22,702</td>
<td>22,702</td>
<td>22,702</td>
<td>22,527</td>
</tr>
</tbody>
</table>

Notes: The results are compared with the LP optimal solutions and best solutions of two ant-based algorithms: Leguizamon and Michalewicz (L&M), and Ant-Knapsack
Different local heuristics may be proposed to be used in the IWD algorithm for the MKP in order to improve the quality of the solutions. Some new mechanisms, preferably those that have roots in nature may also be employed in the algorithm to help it reach the globally optimal solutions. It is emphasized that there are other mechanisms and interactions in rivers and among natural water drops that has not been employed in the IWD algorithm. As a result, the way is open to new ideas to be used in the IWD algorithm.

Moreover, the mechanisms that have been used in the IWD algorithm need to be analyzed both theoretically and experimentally. The IWD algorithms should be modified to be used for other combinatorial problems. It also should be modified to be employed for continuous optimization problems. Local searches are often used in other optimization algorithms. Therefore, in the IWD, a local search algorithm may also be used.

References


Further reading
About the author

Hamed Shah-Hosseini was born in Tehran, Iran, in 1970. He received the BS degree in Computer Engineering from Tehran University, the MS, and the PhD degrees from Amirkabir University of Technology, all with high honors. He is now with the Electrical and Computer Engineering Department, Shahid Beheshti University, Tehran, Iran. His research interests include Computational Intelligence especially Time-Adaptive Self-Organizing Maps, Evolutionary Computation, Swarm Intelligence, and Computer Vision. Hamed Shah-Hosseini can be contacted at: tason2002@yahoo.com; h_shahhosseini@sbu.ac.ir and his personal homepage is www.drshahhoseini.com