Applications of River Formation Dynamics☆

Pablo Rabanal, Ismael Rodríguez, Fernando Rubio[a,*]

[a]*Dept. Sistemas Informáticos y Computación. Facultad de Informática*
Universidad Complutense de Madrid, 28040 Madrid, Spain

Abstract

River Formation Dynamics is a metaheuristic where solutions are constructed by iteratively modifying the values associated to the nodes of a graph. Its gradient orientation provides interesting features such as the fast reinforcement of new shortcuts, the natural avoidance of cycles, and the focused elimination of blind alleys. Since the method was firstly proposed in 2007, several research groups have applied it to a wide variety of application domains, such as telecommunications, software testing, industrial manufacturing processes, or navigation. In this paper we review the main works of the last decade where the River Formation Dynamics metaheuristic has been applied to solve optimization problems.

Keywords: Heuristic methods, Swarm Intelligence, River Formation Dynamics, Applications

1. Introduction

Optimization is a major necessity in Science and Engineering. No matter if we want to reduce the amount of needed resources to perform a task or maximize the output of some process, so often the difficulty of making the right decisions can be rephrased as some kind of optimization problem. Unfortunately, for many optimization problems finding the optimal solution is not feasible in general due to the NP-hardness of the problem. Moreover, this unfeasibility often applies to the task of finding good suboptimal solutions too. Given an approximation algorithm, let its performance ratio be the ratio between the value of optimal solutions and the value of solutions found by the algorithm in the worst case. Some NP-hard optimization problems can be approximated up to any performance ratio ε arbitrarily close to 1 in polynomial time, although the degree of that polynomial grows unboundedly as the required ratio tends to 1 (e.g. 0-1 Knapsack admits a fully polynomial-time approximation scheme, FPTAS [1]), others can be approximated in polynomial time only up to some fix suboptimal performance ratio if P$\neq$NP (e.g. MAX-3CNF cannot be approximated with a ratio higher than

☆Work partially supported by projects TIN2015-67522-C3-3-R and S2013/ICE-2731.

*Corresponding author

Email addresses: prabanal@fdi.ucm.es (Pablo Rabanal), isrodrig@sip.ucm.es (Ismael Rodríguez), fernando@sip.ucm.es (Fernando Rubio)

7/8 unless P=NP [2]), and others cannot be approximated in polynomial time up to *any* constant performance ratio if P≠NP (e.g. Min Traveling salesman problem, Max Clique or Min Set Cover [3, 4, 5]).

Despite these disheartening theoretical limits, NP-hard optimization problems appear in many kinds of environments, so we do have to face them by some means —necessarily non-exhaustive methods. Some good heuristics for NP-hard problems are specific to the problem under consideration (particularly for problems admitting at least a PTAS, that is, a polynomial-time approximation scheme), whereas others are adaptations of general optimization heuristics (metaheuristics) to the studied problem. Typically, the latter search for solutions similar to the most promising observed ones or their combinations, for example by making some simple entities interact with each other according to simple rules and collaboratively construct new solutions. Within this category we can find evolutionary computation methods and swarm optimization methods [6, 7], which are sometimes inspired by some natural process. Despite these methods cannot generally guarantee any target performance ratio in the worst case (even when facing problems in approximation class APX), they turn out to provide competitive or state-of-art results on average for typical problem instances, particularly for problems whose approximability is in APX or worse (Log-APX, Poly-APX, etc.).

New evolutionary and swarm methods are usually introduced as the implementation of some natural process in a virtual computational environment. On the one hand, this helps to present their underlying algorithms in an intuitive way, and, on the other hand, this implicitly proves that the mechanism they are based on already gives good results in some known context —a natural one. Regardless of the beauty of these metaphors, what makes a metaheuristic interesting is the mathematical process it unleashes and its properties. In [8] the *river formation dynamics* (RFD) metaheuristic was firstly introduced. Like other evolutionary and swarm heuristics, in RFD simple entities interact with each other in a simple way to collaboratively construct good solutions. However, contrarily to evolutionary methods like *genetic algorithms* (GA [9]) or swarm intelligence methods like *particle swarm optimization* (PSO [10]), in RFD each of these simple entities is not a candidate solution by itself, and entities do not traverse or evolve through the solution space itself. On the contrary, entities collaboratively *draw* solutions on a common canvas (a graph) by modifying some values spread over it as they pass through their locations, much more like *ant colony optimization* (ACO [11, 12]). Similarly as ACO, in RFD the probabilities that an entity decides to move through some edge or another depend on the values associated to the nodes reached through these edges,[1] and these values are iteratively modified by entities taking those edges and reaching those nodes. Nevertheless, in RFD these probabilities are not proportional to the values attached to each destination node or edge (as pheromone trails work in ACO). On the contrary, in RFD these probabilities are determined by the *difference* between the value attached to the origin node and value of the destination node, and entities traversing those nodes can increase or decrease these values. Hence, roughly speaking, RFD can be seen as a *gradient-oriented* version of ACO: instead

[1] In ACO these values are typically associated to the edges themselves.

of being driven by values attached to graph locations, RFD is driven by the gradients formed by these values. As we will see in the next section, this change of paradigm has some interesting properties, such as the quickly reinforcement of newly discovered shortcuts (as a shorter path immediately implies the same total value decrease in a shorter distance, which makes it preferable), the trivial avoidance of cycles (as cycles where the value always decreases are impossible), and a focused elimination of blind alleys (just by raising the value of the dead ends).

Beyond the properties derived from the rules of RFD, like most swarm intelligence metaheuristics, this method does not lack a natural process metaphor: RFD entities are *drops* falling through down slopes, values attached to nodes are ground *altitudes* (so that a higher altitude decrease through the same horizontal distance implies a steeper and preferable down slope), and the erosion/sedimentation process (i.e. decreasing/increasing the altitude of nodes) simulates the geological river formation process.

As a proof of concept, RFD was initially applied to study well-known NP-hard problems such as the traveling salesman problem (TSP) (both for static graphs [8] and for graphs where connections between nodes can change along time [13]), the construction of Steiner trees [14], and the construction of minimum variable-cost spanning/distances trees [15, 16]. Eventually, other research groups explored the application of RFD to a wide set of problems with direct industrial applications (when writing this paper, 54 researchers from 18 independent research groups, 9 countries and 4 continents have published research papers on direct applications of RFD). In this paper we collect the main applications of RFD in order to illustrate the frameworks and environments where RFD has been more successful. We hope this compilation of RFD applications will help new researchers to extrapolate the ideas developed in previous works to tackle other similar or different problems in the future.

The rest of the paper is structured as follows. In the next section we introduce the River Formation Dynamics metaheuristic, comment its properties, and describe all steps of its algorithm. In consecutive sections, from Section 3 on we present the applications of RFD to the following fields: computer networks, local area management in GSM, data warehouses, testing, the disassembly problem, the design of infrastructures, VLSI design, and robot navigation. Our conclusions are introduced in Section 11.

2. Brief introduction to RFD

Let us suppose that some drops fall on a flat landscape whose ground can be eroded. These drops will spread around that landscape until some of them reach an exit or sink, that is, a place they can fall from and disappear. Let us call that place the *sea*. In the process of falling to the sea, drops pull up some ground from places adjacent to the sea and carry it as sediments in their fall. Hence, places adjacent to the sea lose some of their ground and thus some altitude. This makes descending gradients from other (not eroded yet) adjacent places into them, so gravity will make subsequent drops fall from them and next fall into the sea, eroding more ground from new places in the process. This way, the erosion process is propagated to farther places until the whole landscape is scattered with down slopes. These slopes implicitly constitute paths from places where drops spawn (*raining* spots) to the sea —paths led just by the gravity.

The formed river basin, possibly including tributaries and meanders, constitutes the solution of Nature for gathering drops from all raining spots and carrying them into the sea. Note that, due to the formation of tributaries, in general the map of paths drawn by gradients is a *tree* rather than a single path to the sea. Actually, this tree represents a tradeoff between providing the shortest path from each raining place into the sea and reducing the size of the tree itself. In computational terms, the former goal can be stated as constructing a tree formed by all shortest paths from each raining spot into the sea, and the latter goal can be rephrased as constructing the minimum spanning tree connecting all raining places to the sea. The tendency of the river formation dynamics to either goal is illustrated by the existence of tributaries and meanders, respectively. In a minimum spanning tree, paths deviate from the shortest paths to quickly join each other and provide a smaller tree, as the priority is to reduce the number of different river beds. The result is a main river bed with meanders. On the contrary, in a tree constituted by the shortest paths from some origins to a destination, individual paths (i.e. tributaries) do not have any special incentive to join each other, as each of them just has to provide a shortest path to some goal. Thus, tributaries are abundant and long in this case.

Given this behavior, any swarm intelligence method based on this natural process would be particularly appropriate for solving problems building some kind of *trees*, particularly those problems seeking for some tradeoff between shortest paths and smallest trees. However, the suitability of this method goes far beyond this mixed goal. Similarly as Nature encourages the formation of tributaries in some areas (e.g. on mountains) and meanders in others (e.g. near the sea), a computational method based on this process can be designed to tend to either goal just by controlling some method parameters [15, 16, 17]. For example, if n drops traveling together erode the soil as if n individual drops passed one after another, then big flows will have a higher impact on the landscape. Consequently, paths with big flows will be encouraged, and river beds will tend to join each other into a small number of flows, covering the water from all areas by means of meanders rather than tributaries to keep the number of flows low. This process forms small trees of river beds. On the contrary, if n drops traveling together make the erosion of a single drop, then river beds will not have any particular incentive to join each other, and the water from all areas will be covered by many separate and direct partial paths to the sea, that is, tributaries. Moreover, the same process can be used to construct *single* paths rather than trees: we just have to use a single raining spot (so the only goal is to form the shortest path from some point to another) or, alternatively, make rain at several places but discourage river basins with several river beds (e.g. by dramatically reducing the erosion caused by drops *before* joining the main river bed).

In more technical terms, RFD entities (drops) will tend to choose following steeper down slopes over plainer slopes. For each possible destination of the drop, we calculate the difference of altitudes divided by the distance (that is, the slope gradient), and this value will be the relative probabilistic weight of choosing to move to that destination instead of others. Alternatively, we can also assign some low probability to follow up slopes, which makes the algorithm more exploratory and less focused (this might be helpful in the first algorithm steps, or when the algorithm does not improve its solutions for some time). Drops pull up some soil (i.e. reduce the altitude) of the places they

go through. We can compute the amount of soil taken by drops to simulate the corresponding physical phenomenon. In this case, this amount is proportional to the speed of the drop, which in turn is proportional to the gradient the drop is passing through — either locally or more globally, i.e. the speed of the drop can be determined just by the current slope or by the last traversed slopes. Alternatively, in a more objective-oriented and less physical process, we can compute the amount of soil taken by the drop in terms of the *quality* of the global path to the sea followed by this drop. That is, once the drop reaches the sea, we assess the quality of the particular path it followed, and then this quality determines how much soil the drop takes through all spots (already) traversed by the drop to reach the sea.

Actually, we can make drops both erode the soil in some areas (i.e. take some soil as they pass through, thus reducing their altitude) and *deposit* the carried soil in other areas (thus *increasing* their altitude). Again, we can either simulate the corresponding physical phenomenon or proceed in a more objective-oriented way: we can deposit the soil when the speed is low (i.e. when the drop traverses a low gradient slope or a series of low gradient slopes), or deposit it when the drop reaches some point and it is detected that the traversed path is *not* good, respectively. The latter may happen either when the drop reaches the sea through a bad (generally, long) path, or when the drop gets stuck at some spot because there is no down slope from it and the drop fails to climb some up slope —or climbing up slopes is just disabled, according to the settings of the algorithm. When a drop gets stuck, we punish the choice of moving to that particular spot by depositing the sediments and increasing the altitude of the dead end: if that spot becomes higher then down slopes leading to it will be less steeper (or even will not be descending anymore), discouraging subsequent drops from moving there. This sedimentation may also increase the altitude of other previous spots leading to the dead end (either in the same step when the drop is stuck, or in several steps, by simple propagation), thus discouraging moves to a set of places (the whole blind alley) rather than just the dead end spot. Let us remark that this erosion and sedimentation process, which is simply guided by the gradients traversed by drops (i.e. eroding in steep down slopes and dropping sediments in plainer or up gradients), has the effect of *softening* the landscape, in a similar way as if the altitude of each point iteratively became the weighted average of the altitudes of nearby points (with weights inversely proportional to the distance). This *local* softening process (note that it is applied only to paths followed by drops, i.e. to potentially interesting paths) has the effect of eventually eliminating local minima in traversed paths of the landscape: by the erosion and sedimentation, a globally descending path including occasional ascending steps eventually becomes an ever-decreasing path.

RFD implicitly avoids the formation, and subsequent reinforcement, of round-trips, as ever-decreasing cycles are impossible by definition due to the gradient-orientation of the process. Note that cycles can be naturally formed and reinforced in non-gradient-oriented environments such as ACO, as the attraction of entities towards edges is proportional to the values associated to the edges, no matter if the values are increasing or decreasing compared to previously traversed edges (actually, in ACO ants need to remember previously traversed nodes just to avoid the formation of round-trips). Also note that, when a shorter path from some point to the sea is found, its reinforcement until it is eventually preferred by subsequent drops is fast: since the old (longer) path

and the new shortcut depart from the same origin node and reach the same ending node (the sea), the total altitude reduction along both paths is the same, but the distance traversed to reach the destination is smaller in the shorter path. Thus, the average descending *gradient* along all traversed steps is necessarily higher through the shortcut than through the old longer path. Consequently, even before subsequent drops begin to reinforce a newly discovered shortcut (by being attracted to steeper slopes and making them ever steeper via erosion), the gradients of the new shortcut make its steps immediately preferable, on average, over the steps of the older and longer path —so the reinforcement is just used to make all shortcut steps *individually* preferable, not just all together on average. This contrasts with ACO, where the pheromone trail on a new shortcut just discovered by an ant is still negligible, specially compared to the pheromone trail existing on the steps of an old and well established longer path. Hence, steps in this new shortcut are not yet even close to be preferable on average.

Compared to ACO, the gradient-orientation of RFD eases the elimination of paths leading to blind alleys, avoids cycles, and quickly reinforces new shortcuts. However, it complicates a process that is easier and faster in a value-orientation method like ACO. After a path is well established (say A-B-C-D-E-F), composing a new path from it by simple transposition (e.g. moving the segment D-E-F to the beginning of the path, forming D-E-F-A-B-C) might not be easy under the river formation scheme. The reason is that the altitude of nodes D, E and F will necessarily be lower than the altitude of nodes A, B and C due to the reinforcement of path A-B-C-D-E-F over time via erosion. In particular, a drop needs to climb up the altitude gap between nodes F and A in order to discover path D-E-F-A-B-C. As we said, actual shortcuts have better gradients on average, so the average gradient along D-E-F-A-B-C must be better than along A-B-C-D-E-F if the first one is actually a better path. However, the necessity to climb up the high ascending slope from F to A may make it hard to discover this path. Depending on the kind of problem to be solved, assigning some probability to climbing up slopes may be recommended. This can be particularly useful during those algorithm steps where the exploration of very different paths may be preferable over focusing and polishing already constructed paths by introducing small local changes. This may be the case at the beginning of the algorithm, or when the solution does not improve its solutions for some time.

2.1. RFD basic algorithm

In this section we describe the RFD algorithm. The vanilla version of the algorithm will be introduced here (along with a few very usual modifications), although the research works described in this paper typically introduce several modifications over this basic scheme. The basic algorithm of RFD consists of the following steps:

```
initializeDrops()
initializeNodes()
while (not allDropsFollowTheSamePath()) and
      (not otherEndingCondition())
   moveDrops()
   erodePaths()
   depositSediments()
   analyzePaths()
end while
```

In the first step (`initializeDrops()`) drops are put in the initial node (or initial nodes). Next, the altitude of all nodes of the graph is initialized (`initializeNodes()`): the altitude of the destination node is fixed to 0 (note that this node represents the *sea*, i.e. the final goal of all drops), and the altitude of the remaining nodes is set to some equal value.

Then, the RFD evolution loop is iterated. The algorithm is executed until either all drops find the same solution (`allDropsFollowTheSamePath()`) or another alternative finishing condition is satisfied (`otherEndingCondition()`). The former condition requires that all drops departing from the same initial nodes traverse the same sequences of nodes, whereas the latter may be used, for example, for limiting the number of iterations or the execution time, or for finishing the loop if the best solution found so far is not surpassed during the last n iterations.

We describe the operations performed within the `while` loop. The `moveDrops()` step consists in probabilistically moving the drops across the nodes of the graph. The transition rule, defining the probability that a drop k at a node i chooses the node j to move next, is given by the next expression:

$$P_k(i,j) = \begin{cases} \frac{decreasingGradient(i,j)}{\sum_{l \in V_k(i)} decreasingGradient(i,l)} & \texttt{if} \quad j \in V_k(i) \\ 0 & \texttt{if} \quad j \notin V_k(i) \end{cases} \qquad (1)$$

where $V_k(i)$ is the set of neighboring nodes of node i which can be visited by drop k and have a negative value of *decreasingGradient(i,j)*, which is the gradient between nodes i and j, and is defined as follows:

$$decreasingGradient(i,j) = \frac{altitude(j) - altitude(i)}{distance(i,j)} \qquad (2)$$

where *altitude(x)* is the altitude of the node x and *distance(i,j)* is the length of the edge from node i and node j.

Let us recall that, at the beginning of the algorithm, all nodes have the same altitude, so $\sum_{l \in V_k(i)} decreasingGradient(i,l)$ equals 0. We give a special treatment to flat gradients as follows: the probability that a drop moves through an edge with 0 gradient is set to some non-null value. This way drops can spread around a flat environment (note that this is required, in particular, at the beginning of the algorithm). We take this idea further by letting drops climb up increasing slopes with a low probability. This probability will be inversely proportional to the increasing gradient, and it will be reduced during the execution of the algorithm, similarly as in *Simulated Annealing* (see [18, 19]). Actually, the probability $P(d)$ that drop d is allowed to climb up ascending gradients is defined as follows:

$$P(d) = \frac{1}{probClimbingDrop}$$

where the natural value *probClimbingDrop* is initially set to 1, and it is increased every few loops of the algorithm. Besides, following the simulated annealing tendency to abruptly increase the exploratory tendency of the algorithm from time to time, every N iterations the value of this variable is reduced again.

The probability that a *climbing drop* k located at a node i chooses to move to node j can be naturally integrated into the general probability equation (1) just by adding

some constant to all $decreasingGradient(i,j)$ terms (so that most of gradients, even many ascending ones, naturally constitute non-negative terms). Alternatively, we can define the relative probabilistic weight of moving through plain or ascending gradients in a separate ad-hoc way, for instance as follows:

$$P_k(i,j) = \begin{cases} \delta & \texttt{if } decreasingGradient(i,j) = 0 \\ \frac{\omega}{|decreasingGradient(i,j)|} & \texttt{if } decreasingGradient(i,j) < 0 \end{cases}$$

where δ and ω are parameters of the algorithm.

Next, paths are eroded in step `erodePaths()` according to the moves of drops in the previous phase. If a drop moves from node A to node B then we erode node A. The reduction of the altitude of this node depends on the current gradient between A and B: the erosion is stronger if the descending slope between A and B is high. If the edge is flat or increasing then a small erosion is performed. Note that the altitude of the final node (i.e., the *sea*) is never modified and it remains equal to 0 during all the execution. After traversing a descending gradient, the new altitude of the eroded node A (*altitude(A)*) is defined as follows:

$$altitude(A) = altitude(A) - erosion(A,B)$$

$$erosion(A,B) = \frac{paramErosion}{(numNodes-1)\cdot numDrops} \cdot decreasingGradient(A,B)$$

where *paramErosion* is a parameter of the erosion process, *numNodes* is the number of nodes of the graph and *numDrops* is the number of drops used in the algorithm.

When the erosion process is completed, the altitude of all nodes of the graph is slightly increased (`depositSediments()`). The altitude of a node N (*altitude(N)*) is increased according to the following expression:

$$altitude(N) = altitude(N) + (erosionProduced/(numNodes-1))$$

where *erosionProduced* is the sum of the total erosion performed in the previous phase and *numNodes* is the number of nodes of the graph. This way we avoid that, after some iterations, the erosion process makes all altitudes be close to 0, which would make gradients negligible and would ruin all formed paths.

Drops can also *deposit* the sediment they are carrying in nodes. This is done when all movements available for a drop imply to climb up an increasing slope but the drop fails to climb any edge, according to the probability assigned to it. In this case, the drop is blocked and it deposits the sediments it transports, which increases the altitude of the current node. The increment is proportional to the amount of sediment cumulated by the drop during its previous moves. The altitude of the node N (*altitude(N)*) where the drop is blocked is increased by using the following formula:

$$altitude(N) = altitude(N) + paramBlockedDrop \cdot cumulatedSediment$$

where *paramBlockedDrop* is a parameter and *cumulatedSediment* is the erosion produced by this blocked drop.

The last step of the loop (`analyzePaths()`) analyzes all solutions found by drops and stores the best solution found so far.

3. Applications in Computer Networks

One of the main application areas of RFD has been the design of routing protocols in computer networks. In this particular context, an important advantage of RFD is that its gradient-oriented nature guarantees the absence of cycles, as we mentioned before. Moreover, in contrast to other metaheuristics like ACO, it is not necessary to use backpropagation to update the corresponding path after finding a solution. Thus, the communication overhead is reduced. Due to these characteristics, several groups of researchers have used RFD to design routing protocols in different contexts (see [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]). In the rest of this section we describe how RFD has been applied to two types of networks: mobile ad hoc networks and wireless sensor networks.

3.1. Mobile ad hoc networks

Mobile ad hoc networks (MANETs) lack a fix infrastructure. In fact, they are continuously changing their network topology. Thus, one important and difficult task in such networks is the design of good routing algorithms. These routing algorithms can be classified into three types: reactive (on-demand), proactive, and hybrid. In the first case, the information regarding how to reach a given destination from a given source is only updated when a message has to be sent between them. In contrast, in proactive systems the network protocol updates the routing tables periodically, even if no message has to be sent. Hybrid systems use both strategies, updating some information periodically, but also using each new message to improve the routing information.

SMART [20, 21] is an hybrid routing protocol designed to deal with the dynamic nature of MANETs. That is, it uses a reactive behavior when a route is not available, but it also behaves proactively using hello messages periodically to update information about the neighbors of each node, and to declare the presence of the node to its surroundings. In order to integrate RFD into the protocol, each node of the network manages two tables. These tables store information about the time delay needed to reach each neighbor, as well as about the relative altitude of the node itself with respect to other nodes. The maintenance of these tables is a key part of the protocol: when a message has to be transmitted to a given destination, RFD is used for stochastically selecting the route to be followed by taking into account the corresponding altitudes. The information of the tables is updated by means of drops and smart packets.

Drops are sent proactively to update the information tables periodically. They behave following the basic rules of RFD, eroding/sedimenting the landscape in their way to reach the given destinations. In contrast, smart packets are only used when a given message has to be sent from an origin to a destination. In that case, the original packet to be transmitted is transformed into an *smart* packet by extending it with four extra bytes that contain information needed to appropriately erode the landscape. In principle, smart packets are transmitted following the same rules as drops. In fact, smart packets can discover new routes, which is especially important when an intermediate

node has failed or is congested (in those cases, the sedimentation process of RFD allows to quickly adapt the altitudes to avoid the problematic node), but their stochastic behavior is reduced to keep them relatively close to the best route known so far, so that the overall latency of each individual packet is not very large.

The SMART protocol has been compared with the well-known AODV protocol [33] as well as with AntHocNET [34], another well-known protocol based on a different metaheuristic: Ant Colony Optimization. The experiments show that SMART obtains better throughput (that is, the total number of successful messages delivered is higher), better end to end delay, and better jitter (that is, the variation of the packet delay is smaller), at the cost of a relatively small increment in the routing overhead with respect to AODV (although the routing overhead is much lower than in AntHocNet).

As an improvement over the SMART protocol, a couple of proposals (see [22, 23]) have slightly modified it to take into account the remaining power available in each node. Thus, when a drop or a smart packet traverse a node with low battery, its altitude is modified to reduce its use in the future, so that the node can keep working for a longer period of time without crashing due to a battery fault.

3.2. Wireless sensor networks

In Wireless Sensor Networks (WSNs), many devices are distributed among a given area. Each of them use sensors to collect information (temperature, humidity, vibration, etc.) that has to be sent periodically to the Base Station (BS), where it is processed. In this type of networks, most of the energy of the sensor nodes is spent in the transmission of the information. Thus, research has been carried out to reduce the consumption of energy. In order to reduce the energy needed to send the information, multi-hop communication is used. That is, sensors do not send the information directly to the BS, because communication at large distances is quite expensive. By contrast, information is sent through intermediate nodes to reduce the distance of each communication. Hence, a routing protocol is needed to transmit the information to the BS.

RFD was used for the first time in the context of WSNs in [26], where Guravaiah et al. introduced RFDMRP (River Formation Dynamics based Multi-hop Routing Protocol). They noticed that there was a very clear analogy between WSNs and RFD. Let us note that each sensor has to find its way to send its information (drops) to the BS (sea). During its way to the BS-sea, several drops can converge in intermediate nodes, where they join to go on traveling together (forming a path-riverbed) to the sea. Notice that, by doing so, a single message can aggregate data coming from different sensors-drops, thus reducing communications and saving energy.

In the initial stage of the method, the hop count of each node is computed. In order to do it, BS starts broadcasting a beacon message. Nearby nodes receive it and broadcast it again, increasing the hop count. After a few iterations, all nodes are reached, and the hop counts, neighbors, and relative positions of each node are known. Once the initialization is finished, RFD is used for sending messages from each node to the BS. The initial altitudes of the nodes are computed by taking into account the distance to BS and the remaining energy of each node.

In [27] it is presented an empirical study comparing RFDMRP with two well-known WSN protocols, namely LEACH [35] and MOD_LEACH [36]. The metrics

used in the comparison are the number of alive nodes along time, the total number of packets sent to BS, the energy consumption of the network, the total time that the network can operate, the node density, and the percentage of data aggregation obtained. At the beginning of the experiment, the metrics obtained by RFDMRP are a little bit worse than those obtained by LEACH and MOD_LEACH. However, as times goes on, RFDMRP obtains the best metrics in all cases. The results show that RFDMRP is a reasonable proposal to reduce the energy consumption and to enlarge the lifetime of WSNs.

The initial RFDMRP was extended in [28] to allow to deal with clustering. That is, sensors can be grouped in clusters, so that each cluster has its local BS. Then, all local BSs send their information to the global BS. The proposal allows to use RFDMRP inside each cluster, and also to use RFDMRP to send the information of the local BSs to the global BS. That is, it is a two-level RFDMRP version. The results show that the metrics obtained are again better than those obtained by other well-known proposals, both in the case of homogeneous and heterogeneous environments. In particular, it is compared with LEACH [35], DEEC [37], ERA [38], and ACH^2 [39], showing that it behaves better in terms of energy consumption and network lifetime, although at the cost of a small increase in the transmission delay (less than 10%).

4. Location Area Management in GSM

In order to deliver calls to mobile phones using the GSM standard (Global System for Mobile Communications), location management operations are needed to track their location. The space is divided into independent cells, with a base station managing each of them. When there is not any running call, location updates involve reverse control channels. That is, the mobile phone send messages to the base station to communicate its position. However, at call arrival, the base station uses the paging procedure to communicate with the mobile phone. Notice that a good location management system needs to find a trade-off between both methods: in case the mobile station updates very frequently its position, the base station always knows a recent position, so the paging cost required to locate it when a call arrives will be small. However, the cost needed to update very frequently the position will be high. Alternatively, in case the position is not updated frequently, the cost to locate the mobile station when a call arrives will be larger. In particular, when the mobile station is not found in the cell where it was in its last communication, a paging-miss occurs and other cells have to be paged until it is found, delaying the call establishment.

In [40, 41] authors use RFD to aggregate cell areas into larger location areas, trying to minimize the overall cost of location management. The cost is computed by using the mobility model defined in [42]. In particular, the paging cost for each cell is calculated by taking into account the size of the location area it belongs to (the larger the area, the larger the paging cost), while the mobility weight of each node is the addition of the mobility weight of its boundary links with other nodes. In the RFD representation, each node represents a cell, the distance between two nodes represents the cost of merging both cells, and the initial altitude of each node represents the call-to-mobility ratio (that is, query weight divided by mobility weight). The movement of drops following the RFD rules suggests the cells that are to be aggegrated to form larger location areas.

The authors present experiments comparing RFD with other approaches appearing in [42]. The results obtained with three different network topologies suggest that RFD obtains better results. Moreover, the advantage is larger when the size of the network is bigger.

5. Data Warehouse

Data warehouses store large amounts of information that can be retrieved in many different ways to obtain different reports or to analyze new data. Hence, an important issue is the answering speed when information is to be obtained. This answering speed can be improved by using materialized views. These views can be seen as precomputed summaries of relevant information, whose aim is to speedup future queries. As it can be expected, the number of materialized views has to be restricted to keep the cost of the system low. Thus, an interesting problem consists in selecting appropriate sets of materialized views to minimize the average answering speed while respecting the space constraints.

In [43] RFD is used for solving the aforementioned problem. Given the set of possible views, the size of each of them, a graph representing dependencies between them (that is, what views can help to process other views), and a maximum total size that can be used to store materialized views, the algorithm selects those views optimizing the overall processing time of all the views. In the implementation, each node represents a view, and the initial altitudes of all of them is the same. Drops are initially located at random nodes, and they traverse nodes following the standard rules of RFD. The only modification is that each path of a drop is limited, because a path represents a possible solution (that is, a set of views to be materialized, and its size must be smaller than the given maximum size). Moreover, after a drop finishes its path, it erodes the path accordingly to the quality of the solution it has found. Thus, nodes belonging to good paths are eroded more intensely, so that subsequent drops will select them with higher probability.

In addition to the RFD implementation, in [43] authors present two alternative implementations. One is based on Ant Colony Optimization (ACO), while the other one is based on Genetic Algorithms (GA). The experiments presented by the authors evaluate the three implementations using the same number of iterations for all of them. The results show that RFD obtains better results than GA and ACO.

6. Testing

Testing [44, 45, 46, 47, 48, 49] is one of the most important issues in the development of any project, as it is the most widely used method to detect hidden errors in systems. Several researches (see [50, 51, 52, 53, 54, 55]) have used RFD to deal with different aspects of the testing process. The first work introducing RFD to solve a testing problem was presented in [50], where authors describe how to use RFD to test restorable systems. Given the specification of a system, a typical testing goal is to find a sequence of interactions that, starting from an initial state, allows to reach a given set of states or transitions. By applying this sequence to the implementation under test

(IUT), we can reach and observe the behavior of the IUT in the corresponding states (or transitions), so that we can check whether that behavior corresponds or not with the specification. The main characteristic of a *restorable* system is that a configuration previously traversed can be saved and restored (at some cost). By doing so, we can reuse some of the steps performed during an interaction (say one following steps `a-b-c-d-e`) to perform faster a subsequent interaction (say `a-b-c-f-g`) without performing again all the interactions (in particular, without repeating the steps testing the common prefix `a-b-c`). Unfortunately, in this case, finding an optimal interaction sequence (in terms of the cost of applying it) reaching all the desired states or transitions is an NP-complete problem. Note that an interaction sequence (i.e. a testing plan for a restorable system) can be seen as a kind of covering tree, where common prefixes of different interactions are not executed twice (provided that the cost of restoring the state is smaller than the cost of repeating the execution of the common prefix).

In [50] the authors show that RFD behaves particularly well in this scenario of restorable systems. The problem translates very easily into the rivers analogy, as it is only necessary to look at interaction sequences in the reverse order. Note that interactions need to start from the initial state and reach some target states or transitions. On the contrary, the RFD method will start making drops rain in those states (or transitions), and then drops will find their way from those states (or transitions) to the initial state, representing the sea. Experimental results conducted for a social network management case study show that RFD obtains better results than conventional systems. In particular, it provides a good tradeoff between the (partial) optimality of results and the time needed to achieve them.

A similar approach was taken in [55, 54]. In this case, given a program written either in C or C++, the authors automatically obtain the corresponding control flow graph (CFG) of the program. Then, they use RFD to design test sequences guaranteeing full path coverage (i.e. all possible paths of the program are followed at least once), minimizing the size of the test sequences, minimizing also the coverage redundancy (i.e. minimizing the repetition of the same paths), and prioritizing the most promising paths (i.e. paths with higher possibilities of containing an error are tested first). Experiments compare the results of RFD with two previous approaches developed by other researchers. In particular, they compare it with an implementation based on ACO, and also with an implementation based on Firefly Algorithm (FA). Although the benchmark used for comparison is relatively small, the results are promising: RFD behaves much better than ACO regarding both the size of the test sequences and the redundancy. Moreover, RFD behaves slightly better than FA regarding redundancy, while the size and prioritization of test sequences are analogous in both methods.

RFD has also been used in the context of load testing of web servers. In this area, the aim is to assess whether the server supports or not the required workload by simulating the behavior of a set of customers. In [52, 53] RFD is used for generating test cases for load testing of a web server. Initially, as it is typically done in other approaches, a random number of test cases is generated. Afterwards, RFD is used for improving such set of test cases by considering overlapping test cases and by exploring the search space to find better alternatives. The altitude of each of the nodes (tests) depends on its fitness with respect to the specification, as well as on the cost of applying the test. Experimental results show that the execution time needed to apply the tests obtained

with RFD is smaller than the time needed to apply the tests obtained with classical approaches, while the overall quality of the testing process is maintained.

7. Disassembly Problem

Recycling is an important issue not only from an ecological point of view, but also from an industrial and economical perspective. A relevant and time consuming task in product recovery is disassembly: given a discarded product, disassembly is the task of decomposing it to extract all its valuable parts. In [56, 57, 58] RFD is used to deal with sequence-dependent disassembly line balancing problem (SDDLBP), where a single product type has to be disassembled on a disassembly line. For each valuable part of the product, we have to know how much time is required to disassemble it, whether it is a hazardous part or not, and the precedence relationships with other parts. For instance, part A could not be disassembled until other parts, B and C, have already been disassembled. Moreover, it could be the case that part A could be disassembled before or after another part D, but the time required to disassemble A could be different in each case. For instance, D could partially hide A, making it more difficult (but not impossible) to disassemble A. Thus, SDDLBP deals with two type of relationships between disassembly tasks: precedence relationships and sequence dependent time increments.

Given the list of disassembly tasks of a product, given the relationships among those tasks, and given a maximum time allowed to perform the whole disassembly, the main aim of SDDLBP is to minimize the number of workstations needed to disassemble the product, fulfilling the given requirements. Notice that each task cannot be parallelized between several workstations, but different tasks can be carried out at different workstations at the same time. As a second objective, given solutions where the number of workstations is equal, SDDLBP prefers to balance the idle time at each workstation, so that none of them gets stressed. Moreover, as a third goal, the problem tries to minimize the hazard measure, trying to disassemble hazardous parts of the product as soon as possible. Finally, the forth objective is to extract the most demanded parts of the product as soon as possible.

In [56] authors provide two implementations to solve SDDLBP. The first of them uses Ant Colony Optimization (ACO), while the second one uses RFD. The problem representation in both cases is the same, as RFD and ACO metaheuristics are strongly related. In both cases, solutions are represented as a permutation of the list of tasks, denoting the order in which they are delivered to workstations. For each drop (or ant), a complete assignment is constructed by moving the drop (ant) through the solution space while applying the corresponding erosion rules of the method. Both implementations are used to solve two case studies. Results show that both algorithms are able to obtain the same minimum number of workstations, but RFD obtains statistically significantly better results balancing the work among workstations, minimizing the hazard measure, and extracting the most demanded components as soon as possible. Later on, the same authors developed a new implementation based on a genetic algorithm [58], and they showed that the new method outperforms the results obtained by RFD. Although both methods obtain the same results when optimizing the number of workstations, the genetic algorithm obtains better results considering the other three secondary objectives.

8. Designing Infrastructures

When designing any communication infrastructure, there are two important goals whose optimization cannot be achieved at the same time, so that we need to find a tradeoff between them. The first goal is to minimize the total cost of the infrastructure, while the second goal is to optimize the quality of service (QoS). In [17] RFD was used for dealing with the problem of balancing both goals. Given a cost-evaluated graph, an initial node, and a subset of nodes to be covered, the aim is to obtain a tree covering such nodes, minimizing the total cost of the tree (that is, the sum of the costs of the corresponding edges), and minimizing also the average distance from each node to the initial node. For example, in case we want to connect several towns with a capital city through a highway network, we want to minimize the investment expenses (IE) of constructing the network and we also want to minimize the time required to go from each town to the capital city. The problem is exactly the same when we want to connect several devices with a central computer: we want to minimize the cost of building the network, but also the average communication time.

In order to tackle this problem, in [17] RFD is applied by making rain at all nodes that are to be connected, whereas the initial node represents the sea. As we said in Section 2, natural rivers do not tend to minimize the optimal path to go from each point to the sea, but to form some kind of grouped solution constituting some tradeoff between short paths and small trees. It is very natural to make RFD tend to find the same tradeoff: on the one hand, after some steps, the erosion will reinforce more strongly the slopes providing the shortest paths, improving the QoS; and, on the other hand, the tendency of drops to join each other is very appropriate to optimize the IE of the tree: if drops tend to join the main flow, instead of following their respective individual shortest paths, then less edges are added to the tree and the tree cost is reduced.

Interestingly, RFD can be adapted to optimize either the QoS or the IE (or a given combination of both) just by changing a parameter: if the erosion caused by high flows is reduced, then the incentive of drops to join each other is partially reduced, and thus each drop tends to follow its own shortest path. For instance, as commented in Section 2, this effect can be achieved by changing the erosion rules in such a way that, if n drops traverse an edge, then they make the erosion effect of e.g. a single drop, rather than the effect of n drops. In this case, grouped paths are promoted by the method only when they are required to solve path conflicts. Moreover, by considering intermediate erosion effects, we construct trees partially fitting into the objectives of both problems. Thus, in order to find a tradeoff between optimizing QoS and optimizing IE, we just have to set an appropriate erosion effect value.

The results presented in [17] for graphs of up to 300 nodes show that RFD behaves better than ACO to solve the problem. Both ACO and RFD obtain similar results when we are only interested in optimizing the QoS. However, when we are only interested in minimizing the IE, RFD behaves better than ACO. Moreover, when a tradeoff between both goals is required, RFD obtains much better results than ACO, as it is a problem that fits very easily into RFD scheme.

RFD has also been applied to deal with a different infraestructure: electrical power systems. In order to monitor electrical power systems, Phasor Measurement Units (PMUs) are used. A PMU observes not only the voltage of the bus where it is installed,

but also the currents of any other line incident to the bus where the PMU is located. Moreover, even information of buses not directly connected to the PMU bus can be obtained by applying electrical circuits laws. That is, some buses can be observed directly by the PMU and others can be observed indirectly by the PMU by applying Kirchhoff's laws. Taking into account these properties, the optimal PMU placement problem is to find the minimum number of PMUs (and their optimal locations) to observe all the electrical power system. In addition to the main optimization goal (minimizing number of PMUs), it is also important to maximize the observability redundancy. That is, it is better to observe each bus by as many PMUs as possible, so that the impact of a possible PMU failure is minimized.

In [59] RFD is used for solving the aforementioned problem. As all buses are to be observed, we have to find the paths from each bus to a PMU. Consequently, drops rain at all the buses, and each drop follows its own short path: each drop can only traverse up to three nodes, representing the maximum distance from a bus to a PMU. When all the drops have finished their respective paths, the quality of the solution is evaluated and then the erosion takes place. Afterwards, it rains again at all the buses, repeating the process for a given number of iterations.

In order to assess the usefulness of the approach, the performance of the proposal is compared with the results obtained by five other groups of researchers for the same instances, taken from a standard benchmark [60]. The results show that RFD solutions observe the systems with a slightly lower number of PMUs, obtaining also good redundancies.

Finally, RFD has also been used for designing a first-generation bioethanol plant. In [61] authors compare four metaheuristics to design a plant minimizing both the initial equipment costs and the operation costs, and maximizing the revenue throughout for the first year of operation. The initial equipment costs depend on the specific machinery to be installed in the plant. The authors consider the main (and also the most expensive) ten stages in the bioethanol production process. For each stage, from 5 to 19 available equipments are considered. Each of them has a different production capacity, initial cost, electrical consumption during operation, etc. Another relevant input data needed to run the algorithm is the expected annual production of bioethanol, as the number of equipments needed for each stage will depend on the required production. Finally, an important issue that the algorithm has to decide is the location of the plant: given a predefined set of possible locations, and given the positions where the possible suppliers and buyers are located, the algorithm selects the most convenient location to minimize the transportation costs from suppliers to the plant and from the plant to the buyers. Let us remark that each supplier (and buyer) has an associated production (or consumption) capacity, which is taken into account when computing the transportation cost.

The authors implement four solutions considering four different metaheuristics: River Formation Dynamics, Ant Colony Optimization, Particle Swarm Optimization, and Cuckoo Search. In the case of RFD, the created graph represents a decision tree with a single initial node, a single destination node, and one node for each possible option. For instance, if there are 5 different options to design the first stage of the process, and 10 options for the second stage, the initial node is connected to the 5 nodes representing the options for the first stage, then each node of the first stage is

connected to each of the 10 nodes of the second stage, and so on, until each node of the last stage is connected with the unique destination node. Thus, each time a drop has to find its way from initial to final destination, it has to find a path where each step of the path represents the decision regarding a stage in the design of the plant.

Results show that all metaheuristics obtain reasonable results, but solutions obtained by RFD are better than those obtained by the rest of methods. The advantage obtained with RFD is quite small when designing a small plant producing only 100,000 liters per year, but the advantage of RFD gets larger when designing plants with higher capacity (500,000 or 1,000,000 liters). However, the convergence of RFD is slower (around 10%). That is, RFD obtains better results at the cost of a small runtime increment.

9. VLSI Design

RFD has also been applied in the design of Very Large Scale of Integration (VLSI) circuits [62, 63, 64]. The first application of RFD in this context appeared in [62], where the authors argue that RFD can be useful to help in VLSI design. As a proof of concept, they consider a relatively simple application where a two-stage operational amplifier has to be optimized. The goal is to maximize the voltage gain subject to a given set of constraints. The design is performed using $0.5\mu m$ CMOS technology with $2.5V$ power supply, and the transconductances of each of the transistors can only change within the limits imposed by the corresponding technology.

The basic idea of the implementation is that the movement of each drop represents a modification of the design variables. Each node represents a possible configuration of variables (transconductances, channel resistances, channel lengths, etc.), and drops are randomly initialized at different nodes (variables configurations). In each drop step, modifying the configuration variables by moving to another node is considered, and the erosion of the nodes depends on the value of the objective function that is being optimized.

The results are compared with those obtained by using Mentor Graphics Pyxis Tool, as well as with the results achieved by using other metaheuristics like Particle Swarm Optimization, Simulated Annealing, and Genetic Algorithms. Experiments suggest that RFD obtains competitive results, suggesting that it is a good option to consider RFD for other more complex issues in VLSI design.

Following their conclusions in [62], the same research group continued their work in [63] by considering a more realistic problem. The problem of analyzing voltage fluctuations across power rails of a chip is tackled. Note that even relatively small voltage fluctuations (0.2V) can create incorrect logic outputs inside the chip, and also note that typical chips include millions of nodes to be analyzed. Thus, finding hotspots in large power grid networks requires running simulations consuming large computational resources. Authors evaluate the performance of an RFD approach for a benchmark defined in [65], where the size of the input grids ranges from 1 million to 25 million nodes. Experiments show that RFD analyzes the problems faster than classic strategies, including random walk, hierarchical random walk, an iterative solver gauss-seidel, and a direct solver KLU. The speedup obtained increases with the size of the problem, obtaining speedups of up to 6 with the larger inputs. Regarding the accuracy

of the simulation, experiments show that the results obtained are under a 3% margin of error. In fact, the accuracy is better than the one obtained by using random walk. Thus, RFD obtains good results in both runtime and accuracy.

Finally, RFD has also been applied to VLSI design by hybridizing it with ACO. In fact, the idea of exploring this hybridization is borrowed from [66], where an hybrid RFD-ACO algorithm was introduced, showing that the interaction between ACO and RFD can help to avoid local optima. The goal in [64] is to optimize the routing paths of the circuit. The authors compare the results with those obtained by other researchers, and show that they obtain better results than those obtained by using another well-known metaheuristics, Artificial Bee Colony.

10. Robot navigation

In navigation systems for autonomous vehicles, finding a path according to some constraints, e.g. to avoid collisions or unsafe conditions, is a usual problem. Typically, this problem is solved to minimize the path length, the energy consumption or the travel time. Real-world applications usually have to deal with changes, and a quick response is crucial in these systems.

In [67, 68, 69] the authors apply RFD to mobile robot navigation, in particular to generate optimal paths in dynamic environments. In [67, 68] RFD is implemented to find short paths between two points in a map with obstacles, and tested with real robots. Results are compared with Dijkstra's algorithm, and they show that RFD surpasses Dijkstra when working in a changing environment. The time needed to compute the optimal path is reduced considerably, because RFD takes advantage from the information obtained before the changes are introduced, whereas Dijkstra cannot. Later on, in [69] the authors present a variation of RFD where they modify the transition rule in order to simplify it. The new proposed transition formula is not as general as the original one, because they need to know the distance to the objective node, and it is not always known (it depends on the problem being solved). Results found by this RFD version are compared with the original RFD, an ACO algorithm, and two classic algorithms: Dijkstra and A*. Results obtained with static graphs show that RFD finds the optimal path, but it takes longer than Dijkstra and A*. However, when the new RFD is dealing with changes in the graphs, it becomes faster than the rest of algorithms. That is, RFD takes advantage from its previous knowledge to be able to react faster when a change is introduced in the graph. Several changing situations are considered in the paper, such as including new obstacles, removing obstacles, or moving the goal point. In all cases RFD recomputes faster the new optimal path. Morevoer, the advantage of RFD with respect to the other algorithms is larger when dealing with larger graphs.

11. Conclusions

We have introduced the main applications of RFD to optimization problems during its first decade of existence. Application fields as different as computer networks, robot navigation, industrial or infrastructure construction, testing, or data warehousing have been tackled by RFD. The variety of these applications shows that RFD can be successfully applied to many optimization problems where the goal is to construct some

kind of short path or combination of short paths. The analyzed research works show that, compared to other well known similar heuristics such as Ant Colony Optimization, Particle Swarm Optimization or Genetic Algorithms, the best solutions were very often found by RFD, sometimes in return for a slight longer execution time, typically around 5-10%. These results are consistent with our own experience with RFD in our previous works, in particular in our comparisons with ACO: solutions found by ACO after its convergence has to be assumed (i.e. when it is unable to improve its solutions for some reasonable time) are typically surpassed a bit later by RFD, which usually takes longer to get stuck with some solution. Thus, typically RFD tends to a more exploratory and deep search, whereas typically ACO tends to a more focused search.

As optimization problems will remain as a major and non-trivial issue for many fields within Science an Engineering during the next years, RFD (and its present and future variants) will likely keep providing researchers tackling NP-hard optimization problems with a suitable heuristic to find reasonable sub-optimal solutions in reasonable time.

Acknowledgements

This work has been partially supported by projects TIN2012-39391-C04-04, TIN2015-67522-C3-3-R, and S2013/ICE-2731. The authors would like to thank Mercedes Hidalgo-Herrero and the anonymous reviewers for valuable comments on a previous version of the paper.

References

[1] O. H. Ibarra, C. E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems, Journal of the ACM 22 (4) (1975) 463–468.

[2] J. Håstad, Some optimal inapproximability results, Journal of the ACM 48 (4) (2001) 798–859.

[3] P. Orponen, H. Mannila, On approximation preserving reductions: Complete problems and robust measures (revised version), Tech. rep., Department of Computer Science, University of Helsinki (1990).

[4] D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, in: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, ACM, 2006, pp. 681–690.

[5] I. Dinur, D. Steurer, Analytical approach to parallel repetition, in: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, ACM, 2014, pp. 624–633.

[6] T. Back, D. B. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, IOP Publishing, Bristol, UK, 1997.

[7] J. Kennedy, R. C. Eberhart, Y. Shi, Swarm intelligence, Morgan Kaufmann, 2001.

[8] P. Rabanal, I. Rodríguez, F. Rubio, Using river formation dynamics to design heuristic algorithms, in: International Conference on Unconventional Computation, Springer, 2007, pp. 163–177.

[9] D. E. Goldberg, Genetic algorithms, Pearson Education, 2006.

[10] J. Kennedy, Particle swarm optimization, in: Encyclopedia of machine learning, Springer, 2011, pp. 760–766.

[11] M. Dorigo, G. Di Caro, Ant colony optimization: a new meta-heuristic, in: Proceedings of the 1999 Congress on Evolutionary Computation, CEC'99, Vol. 2, IEEE, 1999, pp. 1470–1477.

[12] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE computational intelligence magazine 1 (4) (2006) 28–39.

[13] P. Rabanal, I. Rodríguez, F. Rubio, Solving dynamic TSP by using river formation dynamics, in: Fourth International Conference on Natural Computation (ICNC'08), IEEE, 2008, pp. 246–250.

[14] P. Rabanal, I. Rodríguez, F. Rubio, Studying the application of ant colony optimization and river formation dynamics to the steiner tree problem, Evolutionary Intelligence 4 (1) (2011) 51–65.

[15] P. Rabanal, I. Rodríguez, F. Rubio, Finding minimum spanning/distances trees by using river formation dynamics, in: International Conference on Ant Colony Optimization and Swarm Intelligence, Springer, 2008, pp. 60–71.

[16] P. Rabanal, I. Rodríguez, F. Rubio, Applying river formation dynamics to solve NP-complete problems, in: Nature-inspired algorithms for optimisation, Springer, 2009, pp. 333–368.

[17] P. Rabanal, I. Rodríguez, F. Rubio, Applying RFD to construct optimal quality-investment trees, J. Universal Computer Science 16 (14) (2010) 1882–1901.

[18] S. Kirkpatrick, C. G. Jr., M. Vecchi, Optimization by Simulated Annealing, Science 220 (4598) (1983) 671.

[19] M. Fleischer, Simulated annealing: past, present, and future, Proceedings of the 27th conference on Winter simulation (1995) 155–161.

[20] S. H. Amin, H. Al-Raweshidy, R. S. Abbas, Smart data packet ad hoc routing protocol, Computer Networks 62 (2014) 162–181.

[21] S. H. Amin, Optimising routing and trustworthiness of ad hoc networks using swarm intelligence, Ph.D. thesis, Brunel University School of Engineering and Design (2014).

[22] L. Y. Vishnu, V. R. Prasad, Power aware routing protocol for MANETs based on swarm algorithm, Computer Journal of Computer Science and Technology 15 (3) (2015) 19–23.

[23] N. M. Shankar, P. Neelima, A. N. Kumar, Optimization of routing protocol using river formation dynamics (RFD) algorithm, International Journal of Computational Science, Mathematics & Engineering 3 (7) (2016) 44–46.

[24] S. Sharma, N. Gupta, Routing in wireless mesh network using river formation dynamics, International Journal of Innovative Research in Science and Engineering 2 (7) (2016) 30–46.

[25] S. Sharma, N. Gupta, River formation dynamics routing protocol for wireless mesh network, Integrated Research Advances 4 (1) (2016) 9–13.

[26] K. Guravaiah, R. L. Velusamy, RFDMRP: River formation dynamics based multi-hop routing protocol for data collection in wireless sensor networks, Procedia Computer Science 54 (2015) 31–36.

[27] K. Guravaiah, R. L. Velusamy, Performance analysis of RFDMRP: River formation dynamics based multi-hop routing protocol in WSNs, arXiv preprint arXiv:1602.04980 (2016) 1–11.

[28] K. Guravaiah, R. L. Velusamy, Energy efficient clustering algorithm using RFD based multi-hop communication in wireless sensor networks, Wireless Personal Communications (2017) 1–28.

[29] S. Mehrjoo, F. Khunjush, River formation dynamics based routing in wireless sensor network, Soft Computing Journal 3 (2) (2015) 54–67. arXiv:http://scj.kashanu.ac.ir/article-1-210-en.

[30] S. Mehrjoo, F. Khunjush, Optimal data aggregation tree in wireless sensor networks based on improved river formation dynamics, Computational Intelligence (2017) In press.

[31] K. Venkateswarlu, T. V. Rao, RFDDA: River formation dynamics based data aggregation in wireless sensor networks, International Journal of Emerging Trends in Engineering Research 3 (6) (2015) 100–105.

[32] S. Kaur, G. Kaur, River based formation routing in WSN: A survey, International Journal of Advanced Research in Computer Science and Software Engineering 6 (4) (2016) 626–631.

[33] C. Perkins, E. Belding-Royer, S. Das, Ad hoc on-demand distance vector (AODV) routing, Tech. rep., RFC 3561 (2003).

[34] G. Di Caro, F. Ducatelle, L. M. Gambardella, AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks, European Transactions on Telecommunications 16 (5) (2005) 443–455.

[35] W. R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, IEEE, 2000.

[36] D. Mahmood, N. Javaid, S. Mahmood, S. Qureshi, A. M. Memon, T. Zaman, MODLEACH: a variant of LEACH for WSNs, in: Eighth International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA'13), IEEE, 2013, pp. 158–163.

[37] L. Qing, Q. Zhu, M. Wang, Design of a distributed energy-efficient clustering algorithm for heterogeneous wireless sensor networks, Computer communications 29 (12) (2006) 2230–2237.

[38] T. Amgoth, P. K. Jana, Energy-aware routing algorithm for wireless sensor networks, Computers & Electrical Engineering 41 (2015) 357–367.

[39] A. Ahmad, N. Javaid, Z. A. Khan, U. Qasim, T. A. Alghamdi, ACH^2: Routing scheme to maximize lifetime and throughput of wireless sensor networks, IEEE Sensors Journal 14 (10) (2014) 3516–3532.

[40] D. Dholakiya, P. B. Swadas, B. V. Chawda, Location management in GSM using river formation dynamics, International Journal of Mobile & Adhoc Network 4 (2) (2014) 136–142.

[41] D. Dholakiya, T. Doshi, S. Ghiya, P. Patel, Advanced river formation dynamics for location area management in GSM, International Journal of Engineering Research & Technology (IJERT) 4 (9) (2015) 611–615.

[42] H. Wang, G. Fan, J. Zhang, Performance comparison of location areas and reporting centers under aggregate movement behavior mobility models, in: International Conference on Parallel Processing (ICPP'02), IEEE, 2002, pp. 445–452.

[43] P. S. Boroujeni, N. Daneshpour, Materialized view selection using river formation dynamics algorithm, in: 1st International Conference on Computer Sciences Communication and Information Technology (ICCSCIT'14), 2014.

[44] J. Tretmans, Conformance testing with labelled transition systems: Implementation relations and test generation, Computer Networks and ISDN Systems 29 (1996) 49–79.

[45] E. Brinksma, J. Tretmans, Testing transition systems: An annotated bibliography, in: 4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067, Springer, 2001, pp. 187–195.

[46] A. Petrenko, Fault model-driven test derivation from finite state models: Annotated bibliography, in: 4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067, Springer, 2001, pp. 196–205.

[47] I. Rodríguez, M. Merayo, M. Núñez, $\mathscr{HOTL}$: Hypotheses and observations testing logic, Journal of Logic and Algebraic Programming 74 (2) (2008) 57–93.

[48] I. Rodríguez, A general testability theory, in: CONCUR 2009 - Concurrency Theory, 20th International Conference, LNCS 5710, Springer, 2009, pp. 572–586.

[49] I. Rodríguez, L. Llana, P. Rabanal, A general testability theory: classes, properties, complexity, and testing reductions, IEEE Transactions on Software Engineering 40 (2014) 862–894.

[50] P. Rabanal, I. Rodríguez, F. Rubio, Testing restorable systems: formal definition and heuristic solution based on river formation dynamics, Formal Aspects of Computing 25 (5) (2013) 743–768.

[51] G. Kumar, Dynamic test case generation using river formation dynamics algorithm, International Journal of Computing and Corporate Research 5 (6) (2015) 1–14.

[52] G. K. Sandhu, R. Kaur, Effective test case generation for load testing of web server using river formation dynamics, International Journal of Advances in Electronics and Computer Science 3 (8) (2016) 17–21.

[53] G. K. Sandhu, Effective test case generation for load testing of web server using river formation dynamics, Master's thesis, Thapar University Patiala (2016).

[54] Y. Khatri, A. Sharma, A. Kumar, Optimal test sequence generation using river formation dynamics, in: International Conference on System Modeling & Advancement in Research Trends (SMART), IEEE, 2016, pp. 24–31.

[55] Y. Khatri, Application of river formation dynamics in search-based software engineering, Master's thesis, Delhi Technological University (2016).

[56] C. B. Kalayci, S. M. Gupta, River formation dynamics approach for sequence-dependent disassembly line balancing problem, in: Reverse Supply Chains: Issues and Analysis, CRC Press, 2013, pp. 289–312.

[57] C. B. Kalayci, Algorithms for sequence-dependent disassembly line balancing problem, Ph.D. thesis, Northeastern University Boston, Massachusetts (2012).

[58] C. B. Kalayci, O. Polat, S. M. Gupta, A hybrid genetic algorithm for sequence-dependent disassembly line balancing problem, Annals of Operations Research 242 (2) (2016) 321–354.

[59] H. G. Abood, V. Sreeram, Y. Mishra, Optimal placement of PMUs using river formation dynamics (RFD), in: 2016 IEEE International Conference on Power System Technology (POWERCON), 2016, pp. 1–6.

[60] R. D. Christie, Power systems test case archive, Tech. rep., Electrical Engineering dept., University of Washington (2000).

[61] G. Redlarski, M. Krawczuk, A. Kupczyk, J. Piechocki, D. Ambroziak, A. Palkowski, Swarm-assisted investment planning of a bioethanol plant, Pol. J. Environmental Studies 26 (3) (2017) 1203–1214.

[62] S. Dash, D. Joshi, G. Trivedi, CMOS analog circuit optimization via river formation dynamics, in: 2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA), 2016, pp. 51–55.

[63] S. Dash, K. L. Baishnab, G. Trivedi, Applying river formation dynamics to analyze VLSI power grid networks, in: VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), 2016 29th International Conference on, IEEE, 2016, pp. 258–263.

[64] B. Sinha, S. Nath, K. L. Baishnab, A hybrid RFD-ACO approach for routing optimization in VLSI physical design, in: Int. Conference on Smart Technologies in Computer and Comunication (SMARTTECH'17), 2017.

[65] Y. Zhong, M. D. Wong, Fast algorithms for IR drop analysis in large power grid, in: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design, IEEE Computer Society, 2005, pp. 351–357.

[66] P. Rabanal, I. Rodríguez, F. Rubio, An ACO-RFD hybrid method to solve NP-complete problems, Frontiers of Computer Science 7 (5) (2013) 729–744.

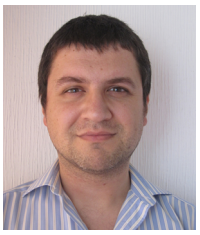
[67] A. Palkowski, G. Redlarski, Swarm algorithms in modern engineering optimization problems, in: XV International PhD Workshop OWD, 2013, pp. 501–506.

[68] G. Redlarski, A. Palkowski, M. Dabkowski, Using river formation dynamics algorithm in mobile robot navigation, in: Solid State Phenomena, Vol. 198, Trans Tech Publ, 2013, pp. 138–143.

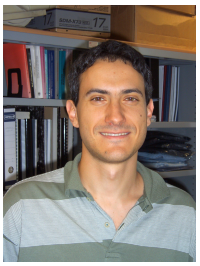
[69] G. Redlarski, M. Dabkowski, A. Palkowski, Generating optimal paths in dynamic environments using river formation dynamics algorithm, Journal of Computational Science 20 (2017) 8–16.

Biosketches

Pablo Rabanal is an Assistant Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2004 and his PhD in the same subject in 2010, devoted to the development of nature-inspired techniques to solve NP-complete problems. Dr. Rabanal has published more than 30 papers in international refereed conferences and journals. His research interests cover swarm and evolutionary optimization methods, formal methods, testing techniques, and web services.



Ismael Rodríguez is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2001 and his PhD in the same subject in 2004. Dr. Rodríguez received the Best Thesis Award of his faculty in 2004. Dr. Rodríguez has published more than 100 papers in international refereed conferences and journals. His research interests cover formal methods, testing techniques, swarm and evolutionary optimization methods, and functional programming.



Fernando Rubio is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 1997, and he was awarded by the Spanish Ministry of Education with "Primer Premio Nacional Fin de Carrera". He finished his PhD in the same subject four years later. Dr. Rubio received the Best Thesis Award of his faculty in 2001. Dr. Rubio has published more than 80 papers in international refereed conferences and journals. His research interests cover formal methods, swarm and evolutionary optimization methods, parallel computing, and functional programming.