Journal of Universal Computer Science, vol. 16, no. 14 (2010), 1882-1901
submitted: 15/9/09, accepted: 29/5/10, appeared: 28/7/10 © J.UCS

Applying RFD to Construct Optimal Quality-Investment Trees

Pablo Rabanal, Ismael Rodríguez, and Fernando Rubio
(Dept. Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid, 28040 Madrid, Spain
prabanal@fdi.ucm.es, {isrodrig,fernando}@sip.ucm.es)

Abstract: River Formation Dynamics (RFD) is an evolutionary computation method based on copying how drops form rivers by eroding the ground and depositing sediments. Given a cost-evaluated graph, we apply RFD to find a way to connect a given set of origins with a given destination in such a way that distances from origins to the destination are minimized (thus improving the quality of service) but costs to build the connecting infrastructure are minimized (thus reducing investment expenses). After we prove the NP-completeness of this problem, we apply both RFD and an Ant Colony Optimization (ACO) approach to heuristically solve it, and some experimental results are reported.
Key Words: River Formation Dynamics, Ant Colony Optimization Algorithms, Heuristic Algorithms, NP-hard problems.
Category: I.2.8, G.1.6, F.2.0

1 Introduction

The *Evolutionary Computation* methods [Kirkpatrick et al.(1983), Davis(1991), Kennedy and Eberhart(1995), Eiben and Smith(2003), Jong(2006)] are based on making several simple entities evolve, according to simple local-scope rules, in such a way that globally efficient solutions are found. In particular, *River Formation Dynamics* (RFD) [Rabanal et al.(2009), Rabanal and Rodríguez(2009)] is an evolutionary computation method related to *Ant Colony Optimization* (ACO) [Dorigo et al.(1996), Dorigo and Gambardella(1997), Dorigo(2004)] that was firstly presented at [Rabanal et al.(2007)]. Roughly speaking, RFD can be seen as a gradient-oriented version of ACO where, in particular, a different nature-inspired metaphor is considered. RFD is based on copying how the water forms rivers in nature. The water transforms the environment by eroding the ground when it falls through a high decreasing slope, and it deposits carried sediments when a more flat ground is reached. In this way, altitudes of places are decreased/increased, and paths of decreasing gradient are dynamically constructed. These gradients are followed by subsequent drops to create new gradients, reinforcing the best ones. Eventually, paths consisting in consecutively taking the highest decreasing gradients constitute good paths from *raining places* to the *sea*. Though RFD has been applied to NP-hard problems of different nature, such as the *Traveling Salesman Problem* [Rabanal et al.(2007)] and the *Minimum Load Sequence* [Rabanal and Rodríguez(2009)], it has been observed that it performs particularly well in problems consisting in creating a kind of *covering tree* over a given graph (see [Rabanal et al.(2009)]). These are problems where the goal is constructing a tree covering some graph nodes, in such a way that a given property is met or a given value is minimized/maximized. Adapting RFD to these problems is particularly natural to this method: We assign the lowest altitude to one of the nodes to be covered (it becomes the *sea*) and we make drops *rain* at the rest of nodes to be covered. Eventually, the gravity makes drops form a *tree* of joining tributaries from departure points to the sea.

In this paper we take advantage of this feature of RFD to solve the following problem: Given (a) a cost-evaluated graph, i.e. a graph where costs are assigned to edges; (b) a subset of nodes required to be covered called *origin* nodes; and (c) a specific node called *destination* node, we construct a tree such that (i) paths connecting origin nodes with the destination node through the tree are as short as possible; and (ii) the tree itself is as small as possible. Trees that are optimal with respect to (i) are not optimal with respect to (ii) nor the other way around, so a kind of tradeoff between both goals will be pursued instead. Let X be the addition of costs from each origin node to the destination through the tree, and let Y be the cost of the tree itself (i.e. the addition of costs of edges forming the tree). Given some $0 \leq \alpha \leq 1$, our goal will be to minimize the expression $\alpha \cdot X + (1 - \alpha) \cdot Y$.

The previous minimization problem appears in any engineering domain where a set of origins must be joined with a destination d in such a way that, on the one hand, distances from each origin to d are minimized (in order to improve the *quality of service*, QoS) and, on the other hand, the cost of constructing the connecting infrastructure is minimized too (which reduces *investment expenses*, IE). In particular, we define the relative weight of QoS and IE in our objective by selecting an appropriate value of the α parameter. Let us suppose that we wish to construct a *local area network* for a new complex of facilities, in such a way that computers in all offices and buildings are connected to the company central server. A *tree topology* is chosen for the network due to the ease to further extend such a topology with new switches/routers/networks without needing to change networking devices (see the role of the tree topology in networking in e.g. [Perlman(1985), Peterson and Davie(2007)]). On the one hand, distances from each node to the central node should be minimized to improve the QoS. On the other hand, the length of the new wiring infrastructure should be decreased to reduce the IE. Similarly, let us consider that some cities must be joined with a capital city through a highway network. We pursue a tradeoff between minimizing distances from each city to the capital and minimizing the cost of constructing the *tree* of highways.

In order to improve our capability to express complex situations, we generalize the proposed problem as follows: We consider that the cost of traversing a given graph edge depends on the path traversed so far. That is, if we traverse e *after* following a path σ then the cost of adding e to the path is $c_{e,\sigma}$; in general, we have $c_{e,\sigma} \neq c_{e,\sigma'}$ for any other path σ'. For instance, this allows to consider that the QoS and IE costs of an edge depend on the *origin* of the path we have traversed before reaching the edge. Coming back to our previous example, the QoS of an edge denoting a *fast* but *not very reliable* wire is not high for transmitting data coming from a node typically running real-time applications (in this case, meeting some *minimal* speed in *all* situations is critical). However, the QoS of this edge may be high for transmitting data coming from an e-mail server (a high bandwidth is good for transmitting a huge amount of e-mails, though it is acceptable if the connection is rarely down). In the highway network example, the IE of building a specific highway stretch may depend on whether we expect it to be typically traversed by trucks or not.

Let us denote the proposed problem by *QoS-IE Tree* problem (`QIT`). Despite of the applicability of `QIT` to different engineering domains (networking, transportation, circuit design, assembly line design, etc), to the best of our knowledge this problem has not been defined or studied before in the literature. We will formally define `QIT` and we will show that it is an NP-complete problem (a proof consisting in a polynomial reduction from 3-SAT will be given).[1] Thus, solving `QIT` in practice requires applying a suitable heuristic method. We will show that RFD is a good choice to solve `QIT` indeed, and we will report some experimental results where the RFD method and an ACO approach are applied to solve `QIT`.

The rest of the paper is structured as follows. In the next section we present the general RFD scheme. Then, the `QIT` problem is formally defined in Section 3, where we prove its NP-completeness. Afterwards, in Section 4 we present an implementation of the algorithm and we compare the results of our method with those obtained using an ACO method. Finally, in Section 5 we present our conclusions.

2 Brief Introduction to River Formation Dynamics

In this section we briefly introduce the basic ideas of River Formation Dynamics (RFD) (for further details, see [Rabanal et al.(2007), Rabanal et al.(2009)]). Given a working graph, we associate *altitude* values to nodes. *Drops* erode the ground (they reduce the altitude of nodes) or deposit the sediment (increase it) as they move. The probability of the drop to take a given edge instead of others

[1] `QIT` is NP-hard because (a) edge costs depend on the path traversed so far and (b) trees are not required to cover all nodes but only some of them. Each of (a) and (b) is, by its own, a sufficient condition for NP-hardness. The proposed proof will be based on reason (a), and reason (b) will not be required to be exploited.

is proportional to the gradient of the down slope in the edge, which in turn depends on the difference of altitudes between both nodes and the distance (i.e. the *cost* of the edge). At the beginning, a flat environment is provided, that is, all nodes have the same altitude. The exception is the destination node, which is a hole (the *sea*). Drops are unleashed (i.e. it *rains*) at the origin node/s, and they spread around the flat environment until some of them fall in the destination node. This erodes adjacent nodes, which creates new down slopes, and in this way the erosion process is propagated. New drops are inserted in the origin node/s to transform paths and reinforce the erosion of promising paths. After some steps, good paths from the origin/s to the destination are found. These paths are given in the form of sequences of decreasing gradient edges from the origin to the destination. Several improvements are applied to this basic general scheme (see [Rabanal et al.(2007), Rabanal et al.(2009)]).

Compared to a related well-known evolutionary computation method, *Ant Colony Optimization*, RFD provides some advantages. On the one hand, local cycles are not created and reinforced because they would imply an *ever decreasing cycle*, which is contradictory. Though ants usually take into account their past path to avoid repeating nodes, they cannot avoid to be led by pheromone trails through some edges in such a way that a node must be repeated in the next step.[2] However, *altitudes* cannot lead drops to these situations. Moreover, since drops do not have to worry about following cycles, in general drops do not need to be endowed with *memory* of previous movements, which releases some computational memory and reduces some execution time. On the other hand, when a shorter path is found in RFD, the subsequent reinforcement of the path is fast: Since the same origin and destination are concerned in both the old and the new path, the difference of altitude is the same but the distance is different. Hence, the edges of the shorter path necessarily have higher down slopes and are immediately preferred (in average) by subsequent drops. Finally, the erosion process provides a method to avoid inefficient solutions because sediments tend to be cumulated in blind alleys (in our case, in *valleys*). These nodes are filled until eventually their altitude matches adjacent nodes, i.e., the valley disappears. This differs from typical methods to reduce pheromone trails in ACO: Usually, the trails of *all* edges are periodically reduced at the same rate. On the contrary, RFD intrinsically provides a *focused* punishment of bad paths where, in particular, those nodes blocking alternative paths are modified.

When there are several departing points (i.e. it rains at several points), RFD does not tend in general to provide the shortest path (i.e. river) from each point to the sea. Instead, as it happens in nature, it tends to provide a tradeoff between quickly gathering individual paths into a small number of main flows (which

[2] Usually, this implies either to repeat a node or to *kill* the ant. In both cases, the last movements of the ant were useless.

minimizes the total size of the formed *tree* of tributaries) and actually forming short paths from each point to the sea. For instance, meanders are caused by the former goal: We deviate from the shortest path just to collect drops from a different area, thus reducing the number of flows. On the other hand, new tributaries are caused by the latter one: By *not* joining the main flows, we can form tailored short paths from each origin point.[3] These characteristics make RFD a good heuristic method to solve problems consisting in forming a kind of covering tree [Rabanal et al.(2008)], which motivates using RFD to solve `QIT`.

Several improvements are applied to the basic general scheme. In particular, drops coming from different origins and coinciding at the same node are joined into a bigger drop, which allows to reduce the number of individual drop movements. Besides, in order to avoid the formation of local optima, drops are given a small probability to climb up slopes, and this probability decreases with time (see [Rabanal et al.(2007)] for further details).

3 Formal Problem Definition

In this section we formally define `QIT`, the problem we will address in Section 4 by means of RFD and ACO. As we said before, `QIT` consists in finding a tree over a given graph such that (a) all nodes belonging to a given subset of graph nodes (called *origin* nodes) are connected to a given node (*destination* node), and (b) the sum of distances from origin nodes to the destination node, as well as the size of the tree, are as small as possible (where the relative weight of the former measure is denoted by α and the latter by $1-\alpha$). Besides, we will be able to consider that the cost of taking an edge depends on the path traversed before taking the edge. In order to denote this dependance, we assume that the cost of a path of edges $e_1, \ldots, e_n$ from a given origin node o to a given destination node d depends on the evolution of a *variable* through the path. Initially, a value v_o is assigned to this variable at node o. Then, the cost added to the path due to the inclusion of edge e_1 is an amount depending on v_o. After traversing e_1, the value of the variable is updated to a new value v_1. Next, the cost of adding e_2 to the path depends on v_1. After taking e_2, the value of the variable is updated again, and the process continues so on until we obtain the whole cost of the path $e_1, \ldots, e_n$.

Following this idea, a *variable-cost graph* can be defined by attaching some information to a standard graph. Let us consider a set of origin nodes (in particular, this set could include *all* nodes of the graph). Then, (1) we assign an *initial value* to each origin node; (2) we assign a *cost function* to each edge. Depending on the value of the variable just before traversing the edge, taking the edge adds

[3] As we will see later, we can make RFD tend towards either of these choices by changing a single parameter.

a different cost; and (3) we assign a *transformation function* to each edge. Given the value of the variable before traversing the edge, it returns the new value after taking it.

Let us suppose that a variable-cost graph defined in these terms is provided and a tree t over this graph, connecting all origin nodes with the destination node, is given. On the one hand, the *QoS cost* of t is the addition of distances from each origin node to the destination. Paths departing from different origin nodes could share some edges in the tree (in particular, different sequences of edges could share some suffixes). Let us note that, in general, the cost of a shared edge is different for each path because the value of the variable when the edge is reached may be different for each path. On the other hand, the *IE cost* of tree t is the addition of costs of edges included in the tree. In this case, the cost of an edge e is computed as follows. Let us consider all paths of t connecting an origin node with the destination node and including edge e. The cost added by edge e to the IE cost of t is the *average* of the cost of e for all of these paths. Let us note that, in both problems, trees are not required to include *all* nodes from the original graph, but only the origin nodes and the destination node. Thus, other nodes of the graph are included in the tree only if they are suitable to (cheaply) connect origins and the destination. In particular, if all nodes are considered as origin nodes then the resulting tree must include all nodes indeed.

Definition 1. A *variable-cost graph* is a tuple $G = (N, O, d, V, A, E)$ where:

- N is a finite *set of nodes*,
- $O \subseteq N$ is the set of *origin nodes*,
- d is the *destination node*,
- $V = \{v_1, \ldots, v_n\}$ is a finite set of *values*,
- $A : O \longrightarrow V$ is the *initial value function*, that is, a function assigning an initial value to each origin node.
- E is the *set of edges*. Each edge $e \in E$ is a tuple (n_1, n_2, C, T) where $n_1, n_2 \in N$ are the *origin* and *destination* nodes, respectively, and
  - $C : V \longrightarrow \mathbb{N}$ is the *cost function* of e. Given a value in V denoting the current value of the variable, it returns the cost of traversing e.
  - $T : V \longrightarrow V$ is the *transformation function* of e. Given the current value of the variable, it returns the new value assigned to the variable if e is traversed.

Paths are sequences of edges departing at an origin node and arriving to the destination node. Formally, a path of G is a sequence of edges $\sigma = (e_1, \ldots, e_k)$

with $e_i = (n_i, n'_i, C_i, T_i) \in E$ for all $1 \leq i \leq k$ such that $n_1 \in O$, $n'_k = d$, and for all $1 \leq i \leq k-1$ we have $n'_i = n_{i+1}$. The *cost* of σ, denoted by $c(\sigma)$, is equal to

$$C_1(A(n_1))+C_2(T_1(A(n_1)))+C_3(T_2(T_1(A(n_1))))+\ldots+C_k(T_{k-1}(\ldots(T_2(T_1(A(n_1))))\ldots))$$

The term denoting the cost of traversing e_i in the previous expression, that is $C_i(T_{i-1}(\ldots(T_2(T_1(A(n_1))))\ldots))$, will be denoted by $c_{e_i}(\sigma)$. In a notation abuse, we will write $e \in \sigma$ if $e = e_i$ for some $1 \leq i \leq k$.

We say that $G' = (N', O, d, V, A, E')$ with $N' \subseteq N$ and $E' \subseteq E$ is a *tree* of G if for all $o \in O$ there exists a single path $\sigma = (e_1, \ldots, e_k)$ of G' departing from o, that is, such that $e_1 = (o, n, C, T)$ for some n, C, T. For each $o \in O$, we denote by σ_o the unique path of G' departing from o.

The *QoS cost* of G', denoted by $qos(G')$, is equal to $\sum_{o \in O} c(\sigma_o)$. The *IE cost* of G', denoted by $ie(G')$, is equal to $\sum_{e' \in E'} \frac{\sum\{c_{e'}(\sigma_o) | o \in O, e' \in \sigma_o\}}{|\{c_{e'}(\sigma_o) | o \in O, e' \in \sigma_o\}|}$. $\square$

Now we are provided with all the needed machinery to formally define the problem considered in this paper. As it is usual in Complexity Theory, this minimization problems is defined in terms of its equivalent decision problem.

Definition 2. The problem of the *Quality of Service-Investment Expenses Tree*, denoted by `QIT`, is stated as follows: Given a variable-cost graph G, a rational number α with $0 \leq \alpha \leq 1$, and a natural number K, is there any tree G' of G such that $\alpha \cdot qos(G') + (1-\alpha) \cdot ie(G') \leq K$? $\square$

`QIT` generalizes other known problems consisting in constructing a kind of covering tree from a graph by (a) considering that both the distances to a given destination node and the size of the tree itself matter; and (b) considering that the cost of traversing each edge depends on the path traversed before taking the edge. The past path is abstracted by the *value* of the variable, which particularizes the cost of each edge for each path. Let us note that, in formal terms, we do not need to consider *several* variables in the problem definition because the dependence on past paths can be denoted by using a single variable. As far as we are concerned, the tree-construction problem proposed in this paper has not been considered in the literature before. Hence, its properties must be analyzed.

Next we prove the NP-completeness of `QIT`, consisting in constructing a polynomial reduction from 3-SAT. This implies that exponential times are (very probably) required to optimally solve them. Thus, sub-optimally solving it by means of heuristic algorithms like those considered in this paper is an appropriate choice. The proof is structured as follows. First, we prove that `QIT` belongs to the `NP` class. Next, we prove that a well-known NP-complete problem, `3-SAT`, can be polynomially reduced to `QIT`, which implies that `QIT` belongs to the `NP-complete` class.

Lemma 3. `QIT` $\in$ `NP`.

Proof. We prove that QIT can be solved in polynomial time by a non-deterministic algorithm. Given a variable-cost graph G, a rational number α with $0 \leq \alpha \leq 1$, and a natural number K, this algorithm non-deterministically constructs a subgraph G' of G and next deterministically checks whether (a) G' is a tree of G, and (b) we have $\delta = \alpha \cdot qos(G') + (1 - \alpha) \cdot ie(G') \leq K$. Both operations are performed in polynomial time with respect to the size of G, α and K (measured in bits required to represent them). Given a subgraph G' of G, checking whether G' is a tree of G requires polynomial time. Next, if G' is a tree of G, calculating δ requires traversing all paths connecting each origin node to the destination node and adding the costs of all of these paths due to $qos(G')$ and $ie(G')$. The length of each of these paths is polynomial, so calculating the cost of a path requires polynomial time. Since G' is a tree, for each origin node there exists a single path connecting it to the destination node. Thus, the number of paths to be considered is polynomial. Hence, we can check whether the property $\delta \leq K$ holds or not in polynomial time. □

In order to prove the NP-completeness of `QIT`, we construct a polynomial reduction from a known NP-complete problem to `QIT`. In particular, we consider the well-known `3-SAT` problem. Next we introduce some notions related to this problem as well as the problem itself.

Definition 4. The `3-SAT` problem is stated as follows: Given a propositional logic formula φ expressed in conjunctive normal form where each disjunctive clause has at most 3 literals, is there any valuation ν satisfying φ?

Let $\varphi \equiv (l_{11} \vee l_{12} \vee l_{13}) \wedge \ldots \wedge (l_{k1} \vee l_{k2} \vee l_{k3})$ be an input for `3-SAT`. We denote by $\texttt{props}(\varphi) = \{p_1, \ldots, p_n\}$ the set of propositional symbols appearing in φ. We denote the i-th disjunctive clause of φ by c_i, that is, $c_i \equiv l_{i1} \vee l_{i2} \vee l_{i3}$.

We say that c_i *holds when* p_j *is equal to* $x \in \{\top, \bot\}$, formally denoted by $\texttt{h}(p_j, x, c_i)$, if for all valuation ν fulfilling $\nu(p_j) = x$ we have that c_i evaluates to $\top$. That is, $\texttt{h}(p_j, \top, c_i)$ iff $l_{im} \equiv p_j$ for some $1 \leq m \leq 3$, and $\texttt{h}(p_j, \bot, c_i)$ iff $l_{im} \equiv \neg p_j$ for some $1 \leq m \leq 3$. □

We will prove $\texttt{QIT} \in \texttt{NP-complete}$ as follows. Given an input φ of `3-SAT`, we show that we can construct an input (G, α, K) of `QIT` from φ in polynomial time in such a way that the solution of `3-SAT` for φ is *yes* iff the solution of `QIT` for the variable-cost graph G, the rational number α, and the natural number K is *yes*. By the definition of the `NP-complete` class, this implies $\texttt{QIT} \in \texttt{NP-complete}$. In particular, if we were able to solve `QIT` in polynomial time then we could solve the NP-complete problem `3-SAT` in polynomial time as well: We could just transform φ into (G, α, K), next call the algorithm that solves `QIT` in polynomial time, and finally return the answer given by it.

Before formally presenting the construction of (G, α, K) from φ, let us informally introduce it. Each origin node of the constructed graph G represents a *disjunctive clause* of φ. From each of these origin nodes, a sequence of edges allows to traverse some nodes one after each other (the same sequence for all origin nodes) where each node represent a *proposition symbol* appearing in φ. Each of these proposition nodes is connected to the next proposition node through two edges. One of them represents valuating the corresponding proposition symbol to $\top$, while the other edge represents giving it the $\perp$ value. Depending on the origin node where we come from (that is, depending on the disjunctive clause we are considering), taking the edge that evaluates the proposition symbol to true or to false adds a different cost to the path. This cost is 1 *unless* the proposition valuation represented by the edge allows to make true the disjunctive clause *for the first time* in the path. In this case, the edge adds 0 to the overall path cost. In order to keep track of this information, the value of the *variable* of the variable-cost graph G codifies the considered clause, as well as whether this clause necessarily holds (according to the valuation represented by the path traversed so far). In particular, variable values follow the form $v_{j,w}$ where j is an index denoting a clause and $w \in \{already\top, notyet\top\}$. A value $v_{j,w}$ denotes that the current path departed at an origin node denoting the j-th clause of φ, and $w = already\top$ denotes that the j-th clause must be true regardless of the valuation of the remaining proposition symbols (because the valuation implicitly defined by the path traversed so far necessarily makes it true). Otherwise, we consider $w = notyet\top$. After the last proposition node is traversed, the destination node of G reached.

Recall that `QIT` seeks for a tree where a given linear combination of (a) the addition of costs from each origin node to the destination node and (b) the addition of average edge costs, is minimal (in particular, $\alpha \cdot qos(G') + (1-\alpha) \cdot ie(G')$). Let us consider that α is given the value 1 in the `QIT` problem instance we construct from φ. In this case, `QIT` seeks for a tree where the addition of costs from each origin node to the destination node is minimal (let us note that, from now on, a similar construction could be given for the case where $\alpha = 0$). Given the variable-cost graph G, a tree of G can include only *one* of the edges that connect each proposition node to the next proposition node (otherwise, it would not be a tree). Hence, given G, trees computed by `QIT` represent valuations of proposition symbols. Since $\alpha = 1$, `QIT` searches for the cheapest tree connecting all origin nodes to the destination node. Thus, `QIT` actually seeks for a tree allowing to make true as more clauses as possible. In particular, we will prove that the cost of the cheapest tree found by `QIT` is under a given threshold if and only if *all* clauses are true under the constructed valuation, that is, iff φ holds. It turns out that, due to the specific form of G, finding a tree where the addition of costs from each origin to the destination is minimal (i.e. considering $\alpha = 1$) is equivalent to finding a tree where the addition of *average* costs of edges is minimal (i.e. considering $\alpha = 0$): Due to the definition of G, in both cases the tree cost is minimized if the valuation represented by the tree makes true as more clauses as possible. Hence, we can also define a threshold such that the cost of the minimum tree for QIT assuming $\alpha = 0$ is under it iff φ is satisfiable.

Theorem 5. $\mathtt{QIT} \in \mathtt{NP\text{-}complete}$.

Proof. Due to Lemma 3, if we can polynomially reduce 3-SAT to QIT then $\mathtt{QIT} \in \mathtt{NP\text{-}complete}$. Let φ denote a conjunctive normal form $\varphi \equiv c_1 \wedge \ldots \wedge c_k$ where $\mathtt{props}(\varphi) = \{p_1, \ldots, p_n\}$. We construct a variable-cost graph $G = (N, O, d, V, A, E)$ as follows:

- $N = \{clause_1, \ldots, clause_k, prop_1, \ldots, prop_n, end\}$,
- $O = \{clause_1, \ldots, clause_k\}$,
- $d = end$,
- $V = \{v_{j,w} | 1 \leq j \leq k \wedge w \in \{already\top, notyet\top\}\}$
- For all $clause_i \in O$ we have $A(clause_i) = v_{i,notyet\top}$.
- $E = \{(clause_i, prop_1, C, T) | 1 \leq i \leq k \wedge \forall\, v \in V : (C(v) = 0 \wedge T(v) = v)\} \bigcup$

$$\left\{ \begin{pmatrix} prop_i, \\ prop_{i+1}, \\ C_i^x, \\ T_i^x \end{pmatrix} \middle| \begin{array}{l} 1 \leq i \leq k-1 \wedge x \in \{\top, \bot\} \wedge \\ C_i^x(v_{j,notyet\top}) = \left\{ \begin{array}{l} 0 \texttt{ if } \mathtt{h}(p_i, x, c_j) \\ 1 \text{ otherwise} \end{array} \right\} \wedge \\ C_i^x(v_{j,already\top}) = 1 \wedge \\ T_i^x(v_{j,notyet\top}) = \left\{ \begin{array}{ll} v_{j,already\top} & \texttt{if } \mathtt{h}(p_i, x, c_j) \\ v_{j,notyet\top} & \text{otherwise} \end{array} \right\} \wedge \\ T_i^x(v_{j,already\top}) = v_{j,already\top} \end{array} \right\} \bigcup$$

$$\left\{ \begin{pmatrix} prop_n, \\ end, \\ C_n^x, \\ T_n^x \end{pmatrix} \middle| \begin{array}{l} x \in \{\top, \bot\} \wedge \\ C_n^x(v_{j,notyet\top}) = \left\{ \begin{array}{l} 0 \texttt{ if } \mathtt{h}(p_n, x, c_j) \\ 1 \text{ otherwise} \end{array} \right\} \wedge \\ C_n^x(v_{j,already\top}) = 1 \wedge \\ T_n^x(v_{j,notyet\top}) = \left\{ \begin{array}{ll} v_{j,already\top} & \texttt{if } \mathtt{h}(p_n, x, c_j) \\ v_{j,notyet\top} & \text{otherwise} \end{array} \right\} \wedge \\ T_n^x(v_{j,already\top}) = v_{j,already\top} \end{array} \right\}$$

We show that constructing G from φ requires polynomial time. This property is a consequence of the following conditions:

(a) $|N|$ is equal to the number of clauses of φ plus the number of proposition symbols of φ plus 1 (the *end* node), which is polynomial with respect to the size of φ.

(b) $|V|$ is equal to the number of disjunctive clauses of φ multiplied by 2. Thus, for each edge in E, defining functions C and T by means of extensional arrays (relating each input value with its output value) requires polynomial size and time.

(c) $|E|$ is equal to the number of clauses plus the number of propositions multiplied by 2, which is polynomial with respect to the size of φ.

Finally, we prove that the *answer* of `QIT` for the graph G, the `QIT` objective parameter $\alpha = 1$, and a threshold $K = k \cdot (n-1)$ is *yes* iff φ is satisfiable. That is, we prove that φ is satisfiable iff there exists a tree G' of G such that $\alpha \cdot qos(G') + (1-\alpha) \cdot ie(G') = qos(G') \leq k \cdot (n-1)$. We consider each implication of this statement:

$\Rightarrow$: Let us note that a tree G' of G must include all edges connecting each node $clause_i$ with $prop_1$. All of these edges have 0 cost. Besides, for each pair of edges connecting each node $prop_i$ with node $prop_{i+1}$, the tree G' must include exactly one of these edges. Let us consider a valuation ν such that for all $1 \leq i \leq n$ we have $\nu(p_i) = \top$ if G' includes the edge $(prop_i, prop_{i+1}, C_i^{\top}, T_i^{\top})$ and $p_i = \bot$ if G' includes $(prop_i, prop_{i+1}, C_i^{\bot}, T_i^{\bot})$. For all $clause_i \in O$, the cost of the path from $clause_i$ to end in G' is $n-1$ if ν makes true c_i, and n otherwise. This is because if ν makes c_i true then all edges in the path but *one* add 1 cost to this path. The exception is the edge that makes true c_i for the first time, which adds 0 cost. If φ is satisfiable then there exists a valuation ν' making true *all* clauses c_i. Thus, there exists a way to choose the edges connecting each $prop_i$ with $prop_{i+1}$ in such a way that, for all $clause_i$, the unique path from $clause_i$ to end has $n-1$ cost. In this case, $qos(G') = k \cdot (n-1)$.

$\Leftarrow$: Let us consider a valuation ν defined as in the previous case. If $qos(G')$ is $k \cdot (n-1)$ then the cost from each $clause_i$ to end must be $n-1$. This implies that, for each $1 \leq i \leq n$, ν makes true the clause c_i. Hence, φ is satisfiable.

Though reducing the `3-SAT` instance to a `QIT` instance with $\alpha = 1$ is enough to prove the NP-completeness of `QIT`, let us note that if we considered $\alpha = 0$ then we could make an alternative proof with very similar arguments. In particular, we could construct a polynomial reduction from `3-SAT` to `QIT` by using the same variable-cost graph G defined before. In this case, we have that φ is satisfiable iff there exists a tree G' of G such that $\alpha \cdot qos(G') + (1-\alpha) \cdot ie(G') = ie(G') \leq n-1$. Let us recall that if $\alpha = 0$ then the cost of each edge e of G' is the average cost of e for all paths traversing e. Due to the structure of G, it is easy to check that $ie(G') = \frac{qos(G')}{k}$. Thus, φ is satisfiable iff $ie(G') = \frac{k \cdot (n-1)}{k} = n-1$. □

It is worth to point out that the goal of the previous construction is proving the NP-completeness of `QIT`, not providing a suitable graph construction to solve `3-SAT` by means of RFD or ACO. In particular, if the variable-cost graph

G were used to find solutions to `3-SAT` by means of RFD, then we would need to introduce a *barrier node* at each edge connecting a node $prop_i$ with $prop_{i+1}$ (see details about barrier nodes in [Rabanal et al.(2007)]).

4 `QIT` Implementation and Results

In this section we describe the application of RFD to solve `QIT` and we report some experimental results. Furthermore, we compare the results obtained by using our algorithm and those obtained by using an ACO method. All experiments were performed in an Intel Core Duo T7250 with a 2.00 GHz processor.

Let us consider the application of RFD to the `QIT` problem. We make *rain* at all origin nodes, and the destination node is the *sea*. After executing RFD for some time, for each graph node we only take the edge with the highest decreasing gradient, and we discard the rest of edges. Since a path of decreasing gradients cannot lead from a node to itself, the (unique) paths from each origin node to the destination node remaining after the removal must depict a tree indeed. As discussed before, natural rivers do not tend to form solutions where each drop goes to the sea through its shortest path, but they tend to form grouped solutions. This allows RFD to implicitly deal with *path conflicts*, i.e. situations where, at a given node, two drops coming from different origins have different preferences regarding which edge should be taken next (because costs are different for each of them; recall that we are considering that costs depend on previously followed paths). In these situations, the tendency of RFD to form grouped solutions implicitly leads to forming paths with a suitable cost tradeoff between available choices: After some steps, the erosion will reinforce more strongly the slopes providing the lowest overall cost. In addition, the tendency of drops to join each other is very appropriate to optimize the IE cost 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, we can adapt RFD to optimize the QoS cost just by changing a parameter: If we reduce the erosion caused by *high flows*, 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, we can achieve this effect 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 as required by α, we just have to set an appropriate erosion effect value.

Regarding the ACO implementation, let us note that solving `QIT` is not natural for ACO. For instance, ACO does not provide an implicit way to deal with the convergence of paths. Let us consider a node where two paths coming from different origin nodes converge. Ants coming from an origin node could be confused by pheromone trails and go on to the *other* origin node, rather than following to the destination node, because pheromone trails are not directed (on the contrary, in RFD formed edge gradients are implicitly directed towards the destination). In order to solve this problem in ACO, a different *kind* of pheromone trail will be considered for ants departing at each origin node, and ants will follow only their own pheromone kind.[4] After the algorithm is executed for some time, paths formed from each origin point are combined to form a tree. If paths constructed for two origin points reach the same node and next leave it through a *different* edge, then only the path with highest pheromone trail leaving the node (considering all kinds of pheromone) is added to the solution tree.

This approach is suitable for finding trees in such a way that paths from each origin to the destination are short, i.e., for optimizing with respect to QoS. However, a different strategy must be considered to find good trees in terms of their overall cost, i.e. in terms of IE, because this strategy does not promote joining paths to reduce the number of edges to be included in trees. We will optimize IE costs in ACO by using a strategy inspired by [Bui and Zrncic(2006)]. In particular, a *single* pheromone kind is considered for all ants. This allows pheromone trails of a given path to attract ants coming from a *different* path, which in turn allows to join different paths together and reduces IE costs. This leads to the problem commented before: Ants coming from a path can get confused at a convergence node and go on through the other path. Moreover, ants can be led by pheromone trails in such a way that it is impossible not to repeat a node next, thus making the path traversed so far useless.

Thus, the strategy followed by ACO to optimize IE costs is very different from the strategy followed to optimize QoS. This contrasts with RFD, where a simple and modular modification (a change on the erosion effect parameter) allows it to easily reach any required tradeoff between QoS and IE optimization. If both QoS costs and IE costs must be optimized in ACO (i.e. if $0 < \alpha < 1$), then we combine the two strategies proposed in ACO to optimize QoS and IE, respectively, which complicates the algorithm execution. Based on the experimentation with different approaches, the following mechanism is followed. The algorithm is executed in two stages. First, ACO is executed according to the QoS optimizing strategy described before, i.e. we consider different pheromone kinds. After some execution time, only edges whose pheromone trail reaches a given threshold are taken in this graph, and next ACO is executed for the resulting graph according to the IE optimizing strategy given before, i.e. we consider a single pheromone kind. Depending on the value of α, the duration of each of these stages is enlarged/reduced. This contrasts with the RFD method, where a

[4] This is equivalent to launching a different ACO execution for each origin point.

the same strategy, parameterized by the erosion effect, is followed in all cases.

Three randomly generated variable-cost graphs with 100, 200, and 300 nodes are used in experiments as benchmark problems, altough we have also tested other graphs obtaining analogous results. In the three graphs, each node is connected to approximately 40% of the rest of nodes. Variables used to particularize the cost of edges can take up to 5 possible values. Cost functions and transformation functions attached to edges are randomly generated. In particular, features such as monotonicity or injectivity are not required in these functions. Table 1 shows the results of the experiments, where the input of both algorithms were the graphs described before. The third and fourth columns of the table describe whether we are optimizing only QoS, IE, or a combination of both of them. Then, the fith column shows the best solution found to optime QIT for the concrete combination of QoS and IE values described in the previous columns. As we are performing several executions for each example, we also show the average and variance of the obtained results (columns sixth and seventh).

Let us analyze the results shown in Table 1. When we consider $\alpha = 1$ (that is, when the weight of QoS in the measure to be optimized is 100% and the weight of IE is 0%), RFD obtains better average results than ACO, although the difference is not big. In fact, the best solution found by ACO in 10 executions is better than the best solution found by RFD in the same number of executions. Regarding the case where $\alpha = 0$ (i.e. only costs due to IE are considered) the advantage of RFD over ACO is greater than in the $\alpha = 1$ case. The reason is that ACO is well suited for searching short paths between given points (in fact, the basic ACO method is devoted to find short paths between two points indeed), though it is not very suitable for *joining* different short paths into a main flow, in such a way that the size of the constructed tree is reduced. On the contrary, this goal is natural to RFD.

The greatest advantage of RFD over ACO is observed in the case of intermediate α values, that is, when a tradeoff between QoS and IE is pursued. When we need to find appropriate trees with respect to QoS and IE, RFD clearly outperforms ACO in all situations. In fact, the only case where ACO obtains acceptable results is in the smallest graph (100 nodes), where the advantage of RFD is not too big. However, when the problem size is larger (200 or 300 nodes), the difference of performance between both methods becomes larger. Thus, the scalability of RFD in these cases is much better than the scalability of ACO.

The main reason for obtaining these results is basically the same reason why RFD works better than ACO to find good trees with respect to IE costs: ACO is well suited to find short paths, though the goal of constructing *grouped* solutions towards a common destination is more natural in RFD. Moreover, ACO results are worse when a QoS-IE combination is required than when only an optimization with respect to IE is required. It could be argued that a QoS-

Table 1: Summary of `QIT` results.

Method	Graph size	% Qos	% IE	Best solution	Arithmetic mean	Variance
ACO	100	0	100	582.62	618.26	552.36
RFD	100	0	100	599.25	610.26	51.94
ACO	100	25	75	603.93	643.31	756.17
RFD	100	25	75	529.18	535.35	23.14
ACO	100	50	50	509.98	541.67	315.68
RFD	100	50	50	444.82	459.83	65.02
ACO	100	75	25	409.84	421.30	86.43
RFD	100	75	25	376.34	389.40	65.83
ACO	100	100	0	285.90	305.79	93.57
RFD	100	100	0	290.87	310.88	95.99
ACO	200	0	100	771.64	933.20	5275.66
RFD	200	0	100	854.78	884.05	111.41
ACO	200	25	75	1270.38	1484.18	38149.49
RFD	200	25	75	771.02	779.11	26.86
ACO	200	50	50	1029.60	1189.63	8297.23
RFD	200	50	50	689.18	698.19	41.72
ACO	200	75	25	819.58	917.88	3025.96
RFD	200	75	25	602.54	623.54	154.83
ACO	200	100	0	542.32	577.54	365.01
RFD	200	100	0	506.36	538.76	160.33
ACO	300	0	100	1239.73	1414.31	8804.20
RFD	300	0	100	1330.00	1362.05	244.50
ACO	300	25	75	1592.38	1751.85	83816.83
RFD	300	25	75	1140.84	1161.03	127.71
ACO	300	50	50	1370.84	1512.30	19654.78
RFD	300	50	50	1035.38	1053.18	124.00
ACO	300	75	25	1132.21	1191.59	2145.74
RFD	300	75	25	907.62	934.46	201.03
ACO	300	100	0	1069.99	1128.73	641.79
RFD	300	100	0	802.44	829.22	128.40

IE combination should perform better for ACO because of the QoS part of the optimization goal, which is a natural goal for ACO. However, the strategy needed to optimally cover some nodes with respect to *both* measures is more complex than in the case of considering only IE. On the other hand, seeking for a tradeoff between QoS and IE is easy for RFD because a simple parameter adjustment

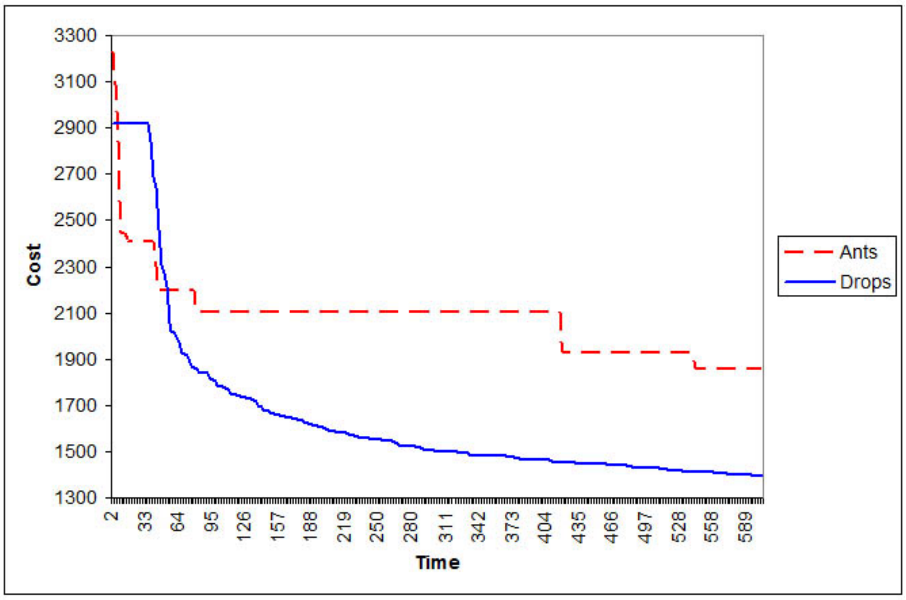

Figure 1: `QIT` results for a 300 nodes graph with $QoS = 100$ and $IE = 0$

allows RFD to define its tendency towards each kind of tree.

In order to analyze the time needed to find good solutions, let us consider Figures 1–3. These figures show the evolution of the quality of the solutions found by ACO and RFD along time (measured in seconds) in a single execution using a 300 nodes graph. In all cases, ACO finds a first solution faster than RFD, although RFD always surpasses ACO after some time. The reason is that RFD performs a deeper exploration of the graph before finding the first solution, but then it behaves better in the mid and long terms. Regarding the comparison between the four figures we can see that, when only QoS costs are considered (see Figure 1), it takes longer for RFD to surpass the quality of the solutions found by ACO. However, when only IE costs are considered (see Figure 3), RFD surpasses ACO in less time. Moreover, when we try to optimize with respect to a combination of both measures (see Figures 2 and 4), the time needed by RFD to surpass ACO is even smaller. It is interesting to note that in these cases (Figures 2 and 4) the evolution in time of ACO performance is not so smooth as in the other two cases. The reason is that $0 \neq \alpha \neq 1$, so that the problem is harder for ACO and a single small discovery by an ant can have a greater influence in the overall result.

The reasons for these results are similar. On the one hand, RFD is better

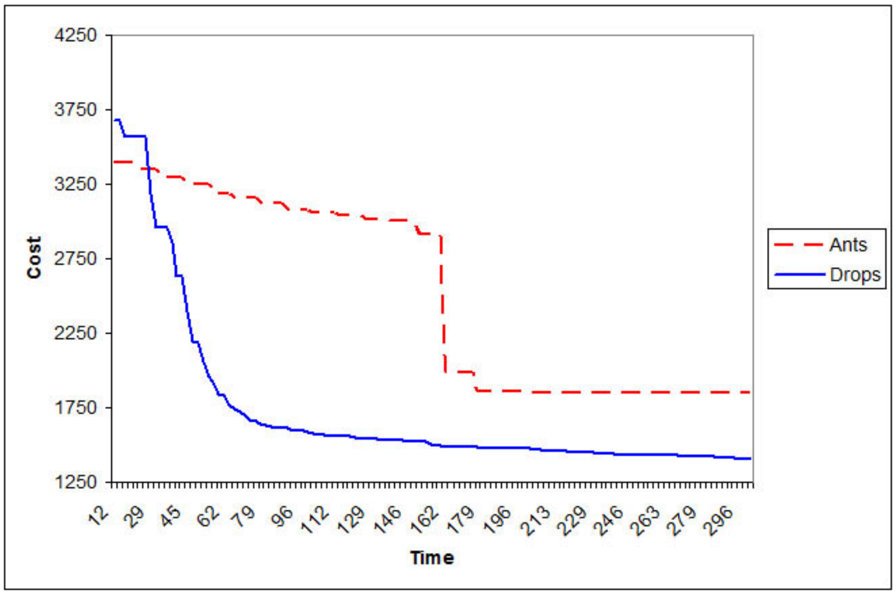

Figure 2: `QIT` results for a 300 nodes graph with $QoS = 75$ and $IE = 25$

suited than ACO for creating covering trees where individual paths must be combined and grouped into common paths. On the other hand, ACO is faster in simpler cases (such as when only QoS is considered), and it gets slower when it deals with more complex problems such as only considering IE, or considering a balance between QoS and IE. We conclude that RFD is a better choice when the problem to be solved requires to construct a solution where individual solutions must be (partially) gathered.

5 Conclusions and Future Work

In this paper we have applied a River Formation Dynamics method to find a way to connect a set of origins with a given destination in such a way that (a) distances from origins to the destination are minimized (which improves the quality of service) and (b) costs to build the connecting infrastructure are minimized (which reduces investment expenses). Though this problem has applications to several engineering domains (networking, transportation, circuit design, assembly line design, etc), to the best of our knowledge it has not been defined or studied in computational terms before. We have shown its NP-completeness, and we have solved it by means of both an RFD approach and an ACO approach. We have observed that the `QIT` problem fits particularly well into the

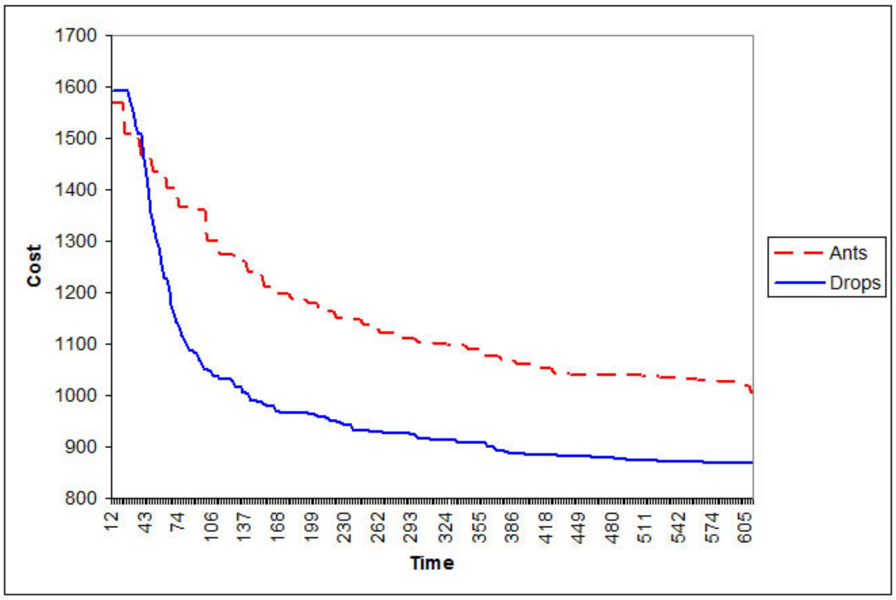

Figure 3: QIT results for a 300 nodes graph with $QoS = 0$ and $IE = 100$

RFD scheme. This is because RFD naturally tends to construct covering trees where, on the one hand, paths from raining points to the sea are short (in particular, tributaries providing partially tailored solutions from each origin point are formed) and, on the other hand, the size of the tree is reduced (in particular, meanders deviate from the shortest path to cover other areas without the necessity of further branching). The natural tendency of drops in RFD towards the lowest point avoids that drops get confused at points where two paths join together. This contrasts with ACO: If an ant reaches a convergence point, it could start to climb up the *other* tributary rather than going down through the main down flow. This forces to modify the ACO scheme as explained in the previous section. On the contrary, no rule modification is required in RFD to make drops *go down*. Thus, handling several paths and properly composing them is a natural task to RFD. As commented in Section 2, other advantages of RFD over ACO are the following: Cycles are implicitly avoided; shorter paths are quickly reinforced; and sediment cumulation provides a focalized way to punish bad paths. The results shown in experiments described in Section 4 corroborate the usefulness of these features in the RFD approach.

As future work, we plan to apply our method to solve a broader set of input cases, covering different kinds of graphs. In particular, we are very interested

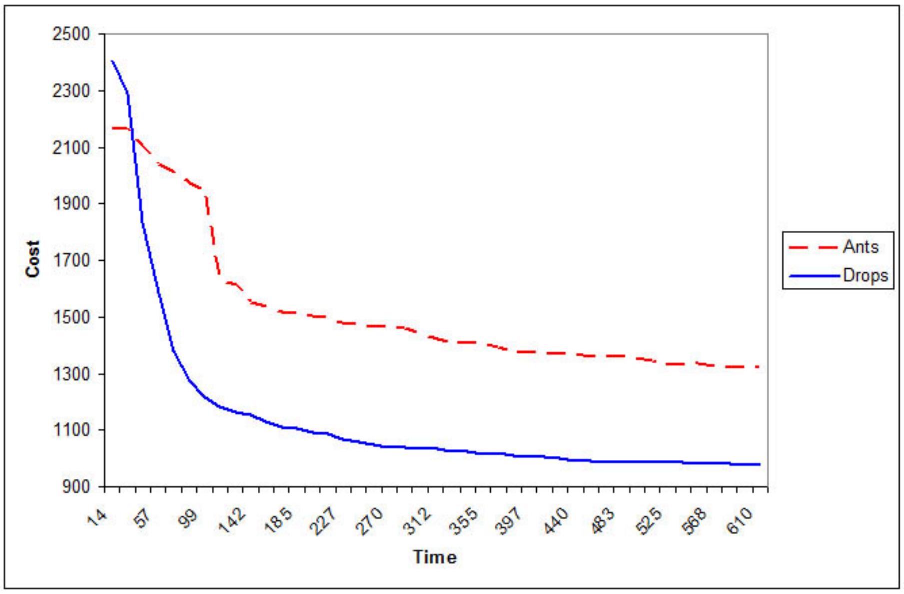

Figure 4: `QIT` results for a 300 nodes graph with $QoS = 25$ and $IE = 75$

in considering a concrete real world problem to illustrate the usefulness of the approach. In addition to that, we plan to create an hybrid ACO-RFD method to try to obtain the best of both worlds. Let us remark that ACO typically requires less time to find a solution, though RFD typically behaves better in the long term. Thus, an hybrid method could obtain both advantages.

Acknowledgments

This research has been partially supported by projects TIN2009-14312-C02-01 and UCM-BSCH GR58/08 - group number 910606.

References

[Bui and Zrncic(2006)] Bui, T., Zrncic, C.: "An ant-based algorithm for finding degree-constrained minimum spanning tree"; GECCO'06; 11–18; ACM Press, 2006.

[Davis(1991)] Davis, L., ed.: Handbook of genetic algorithms; Van Nostrand Reinhold New York, 1991.

[Dorigo(2004)] Dorigo, M.: Ant Colony Optimization; MIT Press, 2004.

[Dorigo and Gambardella(1997)] Dorigo, M., Gambardella, L.: "Ant colonies for the traveling salesman problem"; BioSystems; 43 (1997), 2, 73–81.

[Dorigo et al.(1996)] Dorigo, M., Maniezzo, V., Colorni, A.: “Ant system: optimization by a colony of cooperating agents”; IEEE Transactions on Systems, Man and Cybernetics, Part B; 26 (1996), 1, 29–41.

[Eiben and Smith(2003)] Eiben, A., Smith, J.: Introduction to Evolutionary Computing; Springer, 2003.

[Jong(2006)] Jong, K. D.: Evolutionary computation: a unified approach; MIT Press, 2006.

[Kennedy and Eberhart(1995)] Kennedy, J., Eberhart, R.: “Particle swarm optimization”; Proceedings of IEEE International Conference on Neural Networks, 1995; 4 (1995).

[Kirkpatrick et al.(1983)] Kirkpatrick, S., Jr., C. G., Vecchi, M.: “Optimization by Simulated Annealing”; Science; 220 (1983), 4598, 671.

[Perlman(1985)] Perlman, R.: “An algorithm for distributed computation of a spanningtree in an extended lan”; Symposium on Data communications, SIGCOMM’85; 44–53; ACM, 1985.

[Peterson and Davie(2007)] Peterson, L., Davie, B.: Computer Networks: A Systems Approach, 3rd Edition; Morgan Kaufmann, 2007.

[Rabanal and Rodríguez(2009)] Rabanal, P., Rodríguez, I.: “Testing restorable systems by using RFD”; Int. Work Conference on Artificial Neural Networks, IWANN’09; Springer, 2009.

[Rabanal et al.(2007)] Rabanal, P., Rodríguez, I., Rubio, F.: “Using river formation dynamics to design heuristic algorithms”; Unconventional Computation, UC’07, LNCS 4618; 163–177; Springer, 2007.

[Rabanal et al.(2008)] Rabanal, P., Rodríguez, I., Rubio, F.: “Finding minimum spanning/distances trees by using river formation dynamics”; Ant Colony Optimization and Swarm Intelligence, ANTS’08, LNCS 5217; 60–71; Springer, 2008.

[Rabanal et al.(2009)] Rabanal, P., Rodríguez, I., Rubio, F.: “Applying river formation dynamics to solve NP-complete problems”; R. Choing, ed., Nature-Inspired Algorithms for Optimisation; volume 193 of Studies in Computational Intelligence; 333–368; Springer, 2009.