Deterministic Agent-Based Path Optimization by Mimicking the Spreading of Ripples

Xiao-Bing Hu dr_xiaobinghu@hotmail.co.uk
State Key Laboratory of Earth Surface Processes and Resource Ecology, Beijing Normal University, Beijing, 100875, China; School of Engineering, University of Warwick, Coventry, CV4 7AL, UK

Ming Wang wangming@bnu.edu.cn
State Key Laboratory of Earth Surface Processes and Resource Ecology, Beijing Normal University, Beijing, 100875, China

Mark S. Leeson Mark.Leeson@warwick.ac.uk
School of Engineering, University of Warwick, Coventry, CV4 7AL, UK

Ezequiel A. Di Paolo ezequiel_dipaolo@ehu.es
Ikerbasque, Basque Science Foundation, Centre for Research on Life, Mind and Society, University of the Basque Country, San Sebastian, 20080, Spain

Hao Liu hao.liu@bjjtw.gov.cn
Beijing Metropolitan Traffic Information Center, Beijing, 100161, China

doi:10.1162/EVCO_a_00156

Abstract

Inspirations from nature have contributed fundamentally to the development of evolutionary computation. Learning from the natural ripple-spreading phenomenon, this article proposes a novel ripple-spreading algorithm (RSA) for the path optimization problem (POP). In nature, a ripple spreads at a constant speed in all directions, and the node closest to the source is the first to be reached. This very simple principle forms the foundation of the proposed RSA. In contrast to most deterministic top-down centralized path optimization methods, such as Dijkstra's algorithm, the RSA is a bottom-up decentralized agent-based simulation model. Moreover, it is distinguished from other agent-based algorithms, such as genetic algorithms and ant colony optimization, by being a deterministic method that can always guarantee the global optimal solution with very good scalability. Here, the RSA is specifically applied to four different POPs. The comparative simulation results illustrate the advantages of the RSA in terms of effectiveness and efficiency. Thanks to the agent-based and deterministic features, the RSA opens new opportunities to attack some problems, such as calculating the exact complete Pareto front in multiobjective optimization and determining the *k*th shortest project time in project management, which are very difficult, if not impossible, for existing methods to resolve. The ripple-spreading optimization principle and the new distinguishing features and capacities of the RSA enrich the theoretical foundations of evolutionary computation.

Keywords

Agent-based model, deterministic algorithms, ripple-spreading algorithm, path optimization.

Manuscript received: November 14, 2013; revised: October 14, 2014, February 6, 2015, and April 21, 2015; accepted: April 27, 2015.

1 Introduction

Learning from nature is a foundation of evolutionary computation (EC) because many successful EC methods are actually inspired by certain natural systems or phenomena (Holland, 1975; Russell and Norvig, 2010; Bäck et al., 1997). For instance, genetic algorithms (GAs) are inspired by natural selection and evolutionary processes, artificial neural networks by the animal brain, particle swarm optimization (PSO) by the learning behavior within a population, and ant colony optimization (ACO) by the foraging behavior of ants. This paper aims to investigate the optimization principle reflected by the natural ripple-spreading phenomenon, which may add fundamentally new elements to the study of EC.

We have reported on ripple-spreading models and algorithms for a wide range of problems and applications (Hu et al., 2011; Liao et al., 2013; Hu and Di Paolo, 2011; Hu et al., 2010). The motivation of these has been to take inspiration from the natural ripple-spreading phenomenon to study and resolve a variety of optimization problems in daily life. The basic hypothesis behind the ripple-spreading models and algorithms is the following. Ripple spreading reflects certain fundamental organizing principles in nature, and such principles are to be found in many systems and problems around us. Taking this inspiration when developing models and algorithms in the study of such scenarios, we are likely to better match or reflect the embedded organizing system principles, and therefore generate more effective solutions.

For instance, a theoretical ripple-spreading model for complex networks was proposed to study the spreading influence of local events in many real-world complex systems (Hu et al., 2011). The model was then successfully applied to simulate epidemic dynamics (Liao et al., 2013). The natural ripple-spreading phenomenon can also help to design algorithms to solve complex optimization problems, such as aircraft sequencing and network coding problems (Hu et al., 2011; Hu and Di Paolo, 2011). In these algorithms, a problem-dependent ripple-spreading process is integrated with a GA, in order to improve the overall optimization performance.

This paper is particularly concerned with a fundamental question for the study of ripple-spreading models and algorithms: Does the natural ripple-spreading phenomenon also reflect certain optimization principles? Put another way, can we develop an optimization algorithm that has nothing to do with GAs or any other optimization algorithm but is purely based on mimicking the natural ripple-spreading phenomenon? So far the optimization aspect of ripple-spreading models and algorithms has been provided by a GA (Hu et al., 2011; Liao et al., 2013; Hu and Di Paolo, 2011; Hu et al., 2010). This paper shows that the ripple-spreading phenomenon itself reflects an optimization principle. A ripple spreads out at the same speed in all directions, and therefore it always reaches spatial points in order according to the distance from the ripple epicenter, i.e., it always reaches the closest point first.

This very simple optimization principle can be particularly used to design a new EC method, named the *ripple-spreading algorithm* (RSA), to resolve the path optimization problem (POP) or any problem that can be converted to a POP. Actually, the POP has long been studied, and many mature methods have already been developed (Fu et al., 2006). Behind these successful methods for the POP is the Bellman optimization principle, which forms the foundation of all dynamic programming methods (Sniedovich, 1986). Therefore, we must elucidate the relations and differences between the proposed RSA and existing methods, and between the ripple-spreading optimization principle and the Bellman optimization principle. Does the RSA concept introduce anything new or valuable to optimization theories? This paper answers these questions in detail. In short, the introduction of the ripple-spreading process in the RSA makes all the difference. In fact, the RSA resembles a bottom-up decentralized agent-based simulation model rather than a top-down centralized optimization algorithm, and therefore belongs to the family of EC, but the output of the model is deterministically the optimal solution thanks to the natural optimization principle simulated. Because of the agent-based and deterministic features, the RSA opens new opportunities to address some problems, such as the calculation of the exact complete Pareto front and the determination of the *k*th shortest project times, which are very difficult, if not impossible, for existing methods to resolve.

The remainder of this paper is organized as follows. Section 2 explains the POP and reviews existing methods. Section 3 describes the proposed RSA. Section 4 gives theoretical analyses of some important properties of the new method, and Section 5 gives some simulation results. The paper ends with conclusions and discussion of future work.

2 Path Optimization Problem and Existing Algorithms

2.1 Problem Description of POPs

The POP is a classical optimization problem and has a very wide set of realistic applications such as in transportation systems, communication systems, Internet protocols, logistic optimization, process planning, and job scheduling (Fu et al., 2006). This study investigates four POPs: one-to-one POP, one-to-all POP, many-to-many POP, and the *k* shortest paths problem (*k*-SPP). A general mathematical description of a POP is given here. The POP aims to find the best route through a given route network, so as to travel from source to destination or connect them most efficiently in terms of some specific considerations. Let us assume that a route network $G(V,E)$ is composed of node set V and connection set E. V has N_N different nodes, including the source and the destination, and then this route network can be recorded as an $N_N \times N_N$ adjacency matrix A. The matrix entry $A(i, j) > 0$, $i = 1, \ldots, N_N$ and $j = 1, \ldots, N_N$, means there is a connection, i.e., a direct route from node i to node j. Otherwise, $A(i, j) = 0$ means no direct route, and it is assumed that $A(i, i) = 0$, i.e., no self-connecting route is allowed. The nonzero value of matrix entry $A(i,j)$ usually has a particular meaning. For instance, in the traditional POP, $A(i,j)$ is the physical route length, but actually it can be defined as anything, such as traveling time or fuel consumption. Therefore, for the sake of generality, here we define the meaning of the nonzero entry value as route *cost* rather than route *length*.

First we consider the one-to-one POP, where two different nodes in the route network are specified as source and destination, respectively. A feasible route connecting the source and destination can be denoted as an integer vector R. The value of each vector entry $R(i)$ is the serial number of the associated node, where $i = 1, \ldots, N_L$, and $N_L \geq 2$ indicate how many nodes are included in this route. $R(1)$ is the source and $R(N_L)$ is the destination, and the cost of route R is

$$f(R) = \sum_{i=1}^{N_L-1} A(R(i), R(i+1)). \tag{1}$$

Now we can mathematically describe the one-to-one POP as the following minimization problem:

$$\min_{R \in \Omega_R} f(R), \tag{2}$$

where Ω_R is the set of all feasible routes connecting the source and destination. In this study, a feasible route must observe (1) no node will be visited twice, i.e., $R(i) \neq R(j)$, if $i \neq j, i = 1, \ldots, N_L$, and $j = 1, \ldots, N_L$; and (2) two successive nodes in a route must have a connection between them, i.e., $A(R(i), R(i+1)) > 0$.

The one-to-all POP can be viewed as a set of $(N_N - 1)$ one-to-one POPs. In the one-to-all POP, a source node is specified, and then for every other node in the route network, we need to find the shortest path to the source. Let node n_S be the specified source node, and node n_D be the current destination node. Then the one-to-all POP can be formulated as

$$\sum_{n_D=1,\ldots,N_N, n_D \neq n_s} \min_{R \in \Omega_R} f(R). \tag{3}$$

The many-to-many POP has a set of source nodes as well as a set of destination nodes, and its goal is, for every source node, to find the associated shortest path connecting to one node, it does not matter which one, in the destination set (Drezner et al., 2002; Gastner and Newman, 2006; Yan et al., 2011). Let Ω_{SN} and Ω_{DN} be the sets of source nodes and destination nodes, respectively. If node $n \in \Omega_{SN}$, then $n \notin \Omega_{DN}$. The mathematical formulation of many-to-many POP is as follows:

$$\sum_{n_S \in \Omega_{SN}} \min_{R \in \Omega_R, n_D \in \Omega_{D_N}} f(R). \tag{4}$$

In the k shortest paths problem (k-SPP) (Yen, 1971; Eppstein, 1998; Aljazzar and Leue, 2011), for a given pair of source node n_S and destination node n_D, we need to find k routes $R_1^*, \ldots, R_k^*$, such that

$$f(R_i^*) \leq f(R), \text{for any} R \in \Omega_R - \{R_1^*, \ldots, R_i^*\}, i \in \{1, \ldots, k\}. \tag{5}$$

2.2 Review of Existing Algorithms for POPs

Existing methods for the POP can be classified into two categories: stochastic and deterministic. The former, such as EC, are often confronted with optimality and scalability problems (Holland, 1975; Russell and Norvig, 2010) and therefore are not practicably popular in real-world POP applications (the goal of proposing the RSA in this article as a new EC method is to change this situation). Since the RSA can always guarantee optimality and has a very good scalability, we compare it with other methods that can also guarantee optimality with good scalability. Therefore in this study we ignore stochastic search methods and only compare it with deterministic search methods, which exhibit much better performance in terms of both optimality and scalability, and therefore dominate real-world POP applications. Simply speaking, a deterministic search method organizes its search operations according to a certain specific rule, and as a result, for a given route network, the method outputs a unique repeatable solution (the solution given by a stochastic search method is rarely repeatable). The three most popular rules adopted in a deterministic search method are breadth-first search, depth-first search, and best-first search.

A breadth-first search method is an uninformed search method, and it begins at the root node and explores all the neighboring nodes; then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal (Russell and Norvig, 2010; Fu et al., 2006). Depth-first search is another uninformed search method that progresses by expanding the first child node of the search tree that appears and thence goes deeper and deeper until the destination node is found, or until it hits a node that has no children (Russell and Norvig, 2010; Cormen et al., 2001; Goodrich and Tamassia, 2006; Sniedovich, 2010; Korf, 1985). The search then backtracks, returning to the most recent node that it has not finished exploring. Best-first search explores a graph by expanding the most promising node chosen according to a specified heuristic evaluation rule or function that attempts to predict how close the end of a route is to the destination (Russell and Norvig, 2010; Pearl, 1984). Both breadth-first search and depth-first search are complete but often with a poor scalability. Therefore, limits on search depth or width may be introduced and may then cause optimality to be lost. Best-first search cannot guarantee optimality if the heuristic evaluation rule/function is not properly chosen.

One of the most acknowledged best-first search methods is the A^* algorithm, because it is optimally efficient for any heuristic evaluation function, meaning that no algorithm employing the same heuristic function will expand fewer nodes than A^* (Hart et al., 1968; Dechter and Pearl, 1985; Hart et al., 1972; Korf et al., 2001). If the heuristic function is admissible, meaning that it never overestimates the actual minimal cost of reaching the destination, then A^* is itself optimal if we do not use a closed set. If a closed set is used, then the heuristic function must also be monotonic (or consistent) for A^* to be optimal (Dechter and Pearl, 1985). However, sometimes it is difficult to find an admissible heuristic function to guarantee optimality, for example, in the case of randomly weighted route networks.

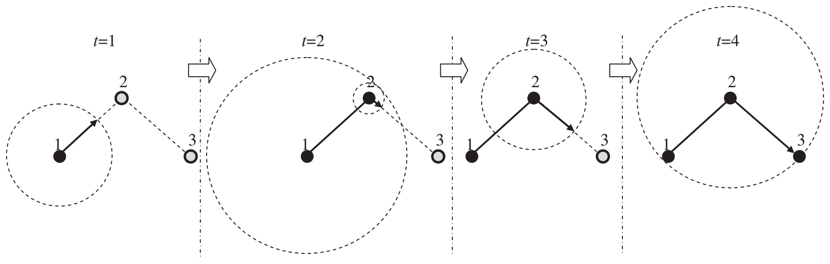
To search general weighted route networks, uniform-cost search is probably the best method to date. Typically, the search algorithm involves expanding nodes by adding all unexpanded neighboring nodes that are connected by direct connection to the set of visited nodes. The uniform-cost search is complete and optimal if the cost of each step exceeds some positive bound (Russell and Norvig, 2010). Dijkstra's algorithm, which is perhaps better known, can be regarded as a variant of uniform-cost search, where there is no goal state and processing continues until all nodes have been removed from the priority queue, i.e., until shortest paths to all nodes (not just a goal node) have been determined (Dijkstra, 1959; Misa and Frana, 2010; Sniedovich, 2006). Dijkstra's algorithm does not use any heuristic function and can be viewed as a special case of A^* where the heuristic function is the constant zero.

These deterministic search methods were originally proposed predominantly for the one-to-one POP and then were widely extended to resolve all kinds of POPs. For instance, Dijkstra's algorithm can apply to the one-to-all POP and many-to-many POP in an almost straightforward manner. For the k-SPP, as reviewed by Mohanta and Poddar (2012), the most popular methodology is to find the first shortest route and let $i = 1$; remove from the original route network each combination of those connections that appear in the first i shortest routes, and find the first shortest route in the reconstructed route network; the shortest one in all such new first shortest routes is the $(i + 1)$th shortest route; increment $i = i + 1$, and repeat reconstructing the route network and calculating the associated first shortest route, until $i = k$. Any method that can resolve the one-to-one POP can be employed in the k-SPP, and the focus often shifts to reconstructing route networks (Yen, 1971; Eppstein, 1998; Aljazzar and Leue, 2011).

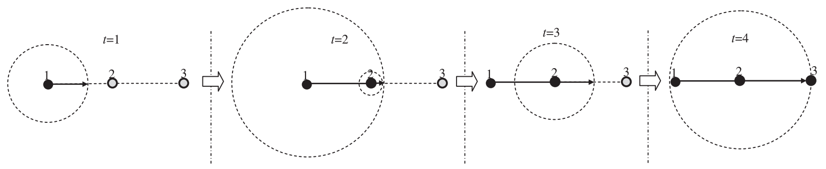
3 Ripple-Spreading Algorithm

3.1 Optimization Principle of the RSA

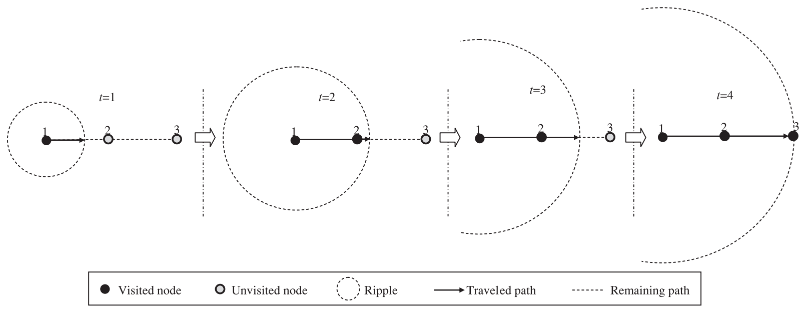
The optimization principle in the natural ripple-spreading phenomenon is very simple. A ripple spreads out on a liquid surface at the same speed in all directions, and therefore it reaches spatial points in order according to their distance from the ripple epicenter, i.e., it always reaches the closest point first. Intuitively, one might conclude that this principle could be used to find the closest interesting points (e.g., to find the closest gas station), or more generally, to find the shortest route. However, the problem is that a natural ripple



(a) Ripple spreading and activation along a zigzag route



(b) Ripple spreading and activation along a straight-line route



(c) Spreading of a single ripple along a straight line

Figure 1: Basic idea of the RSA.

travels along a straight line in all directions, while there are always zigzag routes in a route network, so how can a straight-line-based ripple travel along such routes in order to find the shortest route? Now we illustrate how this is achievable. First, we assume that when a ripple reaches a new node in the route network, it triggers a new ripple at the node. Then the new ripple travels along the new straight-line connection that starts from the node. In this way, independently of whether the connections along which the old and the new ripples travel form a straight-line route or a zigzag route, the traveling time for the new ripple to reach the next new node is always the same. For instance, in Figure 1a the new ripple reaches node 3 at time instant $t = 4$, and the same for the new ripple in Figure 1b. Therefore, zigzag routes will be no obstacle to the implementation of a straight-line-based ripple-spreading process. Comparing Figures 1b and 1c, one can easily see that the spreading and activation process is actually equivalent to the spreading process of a single ripple because the single ripple in Figure 1c also reaches node 3 at time instant $t = 4$. Note that the simulation of the ripple-spreading process is based on a discrete time framework with a specific time unit, and the time instant t means that t time units have lapsed.

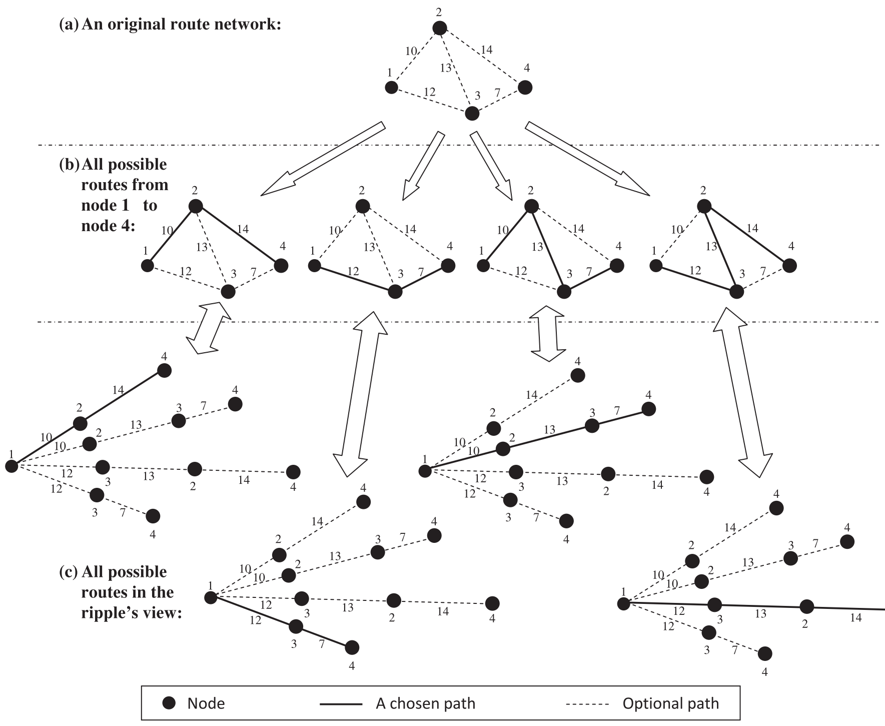
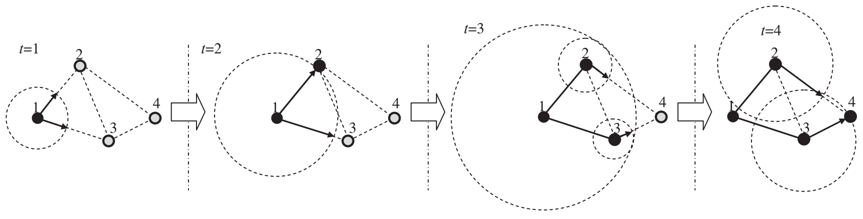

(a) An original route network:
(b) All possible routes from node 1 to node 4:
(c) All possible routes in the ripple's view:
Node
A chosen path
Optional path

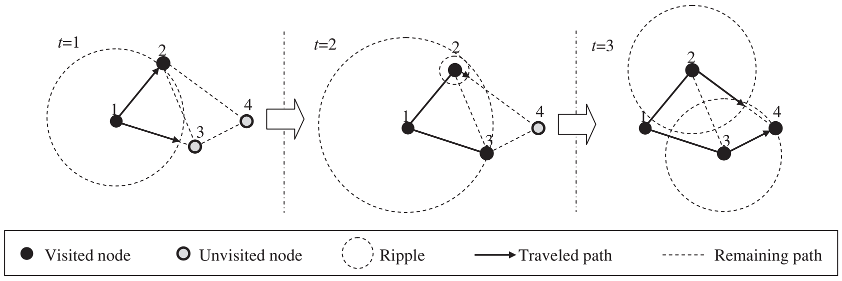
Figure 2: Ripple spreading is a very natural process to find the shortest route.

Now we examine why a ripple can find the shortest route through a network. For the sake of simplicity, we assume there is a route network as shown in Figure 2a, with source node 1 and destination node 4. As shown in Figure 2b, there are four possible zigzag routes to travel from node 1 to node 4. From the ripple's view point, a zigzag route is no different from a straight-line route, as long as they are of the same length. Therefore, the four possible zigzag routes in Figure 2b are actually equivalent to four straight-line routes that radiate out from node 1 in four different directions, as shown in Figure 2c. The four straight-line routes in Figure 2c end at four different spatial points, which actually stand for the same node in the route network, i.e., node 4. Now, one can see that the problem of searching for the shortest zigzag route in the route network Figure 2a becomes a new problem of finding the straight-line route in Figure 2c so that a ripple reaches the end at the earliest time. Because of the optimization principle in the natural ripple-spreading phenomenon, we know that such a straight-line route corresponds to the shortest zigzag route in the original route network.

It must be emphasized that in the realization of the RSA for the POP, we never need to transform a route network like the one in Figure 2a to a set of radiating straight line routes like those in Figure 2c. Nor do we need to transform any single zigzag route (Fig. 1a) to a straight-line route (Fig. 1b). Figures 1 and 2 are just used to help readers understand the optimization principle of the ripple-spreading process. From the ripple's viewpoint, there is no zigzag route at all, i.e., all routes are equivalently straight-line routes. Therefore, in the realization of the RSA, we simply apply ripples to travel along zigzag routes, but the searching effect is exactly the same as traveling along equivalent straight-line routes.



(a) Basic RSA: ripple activation and constant ripple speed



(b) Improved RSA: ripple activation and variable ripple speed

Figure 3: Two different RSAs.

3.2 RSA for One-to-One POP

In this section we only consider the one-to-one POP. We develop the first RSA by implementing both ripple spreading from nodes and ripple activation between nodes. Simply speaking, this resembles a relay race between ripples of nodes, as illustrated in Figure 3a. The first ripple starts from the source, i.e., node 1. When a ripple reaches a directly connected but unvisited node, that new node is activated to generate its own ripple. When all of those nodes that are directly connected to the epicenter node of a ripple are visited (not necessarily by the same ripple), that ripple then stops and is eliminated. When the destination node, i.e., node N_N, is visited for the first time, the relay race is done. During the entire process, all ripples always travel at the same preset constant speed. This basic RSA can be mathematically described as follows. For the sake of simplicity but without losing generality, we assume hereafter that in a one-to-one POP, the source and the destination are always node 1 and node N_N, respectively. In the relay race, each node has a ripple, and the ripple of node i is called ripple i. Let $F_R(i) = j > 0$ mean that ripple i has been activated by ripple j. Let $r_R(i)$ denote the current radius of ripple i. Let Ω_{IAR} and Ω_{AR} denote the sets of inactive and active ripples, respectively.

```
Step 1  Choose a constant ripple-spreading speed $s$. Initialize $F_R(i) = 0$, and $r_R(i) = 0, i = 1, \ldots, N_N$. Initialize $\Omega_{IAR} = \{1, \ldots, N_N\}$, and $\Omega_{AR} = \emptyset$. Let the current time be $t = 0$.
Step 2  Set $F_R(1) = 1$. Let $\Omega_{IAR} = \Omega_{IAR} - \{1\}$, and $\Omega_{AR} = \Omega_{AR} + \{1\}$.
Step 3  If $F_R(N_N) = 0$, do:
        Step 3.1  Let $t = t + 1$.
```

Step 3.2 For any $i \in \Omega_{AR}$, let $r_R(i) = r_R(i) + s$. If $(A(i, j) > 0)$ and $(j \in \Omega_{IAR})$ and $(r_R(i) \geq A(i, j))$ and

$$(r_R(j) \leq r_R(i) - A(i, j)), \tag{6}$$

then let $r_R(j) = r_R(i) - A(i, j)$, and $F_R(j) = i$.

Step 3.3 For any ripple j that has just had its radius updated in step 3.2 during the current time t, let $\Omega_{AR} = \Omega_{AR} + \{j\}$, and $\Omega_{IAR} = \Omega_{IAR} - \{j\}$.

Step 3.4 For any $i \in \Omega_{AR}$, if $F_R(j) > 0$ for every j with $A(i, j) > 0$, then let $\Omega_{AR} = \Omega_{AR} - \{i\}$.

Step 4 The best route is determined by tracking back from $F_R(N_N)$, as

$$R(i) = \begin{cases} N_N, & \text{if } i = N_L \\ F_R(R(i+1)), & \text{if } i < N_L \end{cases}, \tag{7}$$

and the associated total route cost is

$$C_R = \sum_{i=1}^{N_L-1} A(R(i), R(i+1)). \tag{8}$$

For the sake of optimality, the speed in step 1 must satisfy

$$0 < s \leq a_{min}, \tag{9}$$

where a_{min} is the minimal cost associated with the connections (see Section 4). Step 2 starts the first ripple from the source, and this is a self-generated ripple as $F_R(1) = 1$. The loop of step 3 repeats until the destination, i.e., node N_N, has been reached by a ripple. Step 3.1 lets time elapse for one time unit. Step 3.2 first increases the radius of each active ripple i by the ripple-spreading speed s. Then, step 3.2 checks whether ripples i arrive at those nodes, represented by j, that connect to node i and whose ripples are currently inactive. If node j is reached by ripple i, then step 3.2 updates the radius of ripple j, and records that ripple j has been activated by ripple i. It should be noted that, assuming node j has not been reached by time $t - 1$, then node j could be reached by several ripples during the current time t. In such a case, Eq. (6) in step 3.2 guarantees that the ripple that arrives at node j first eventually is used to set $r_R(j)$ and $F_R(j)$. After all currently active ripples have been processed in step 3.2, step 3.3 updates the sets of active and inactive ripples according to newly activated ripples in step 3.2, and step 3.4 removes any active ripple whose connected nodes have all been reached.

Note that if $s \ll a_{min}$, then step 3 may be repeated many times without any new node being reached, which will diminish the computational efficiency of the basic RSA. Therefore, the best way is to simply set $s = a_{min}$. However, even with an $s = a_{min}$, if a_{min} is far smaller than route costs of most connections, it is still likely that the step 3 will be repeated many times before reaching a new node. Therefore, to improve the computational efficiency of the basic RSA, we can adopt a variable ripple-spreading speed and get an improved RSA, as follows.

Step 1 Initialize $F_R(i)$, $r_R(i)$, Ω_{IAR}, Ω_{AR}, and t, as in the basic RSA.

Step 2 Set the current ripple-spreading speed as $s = min(A(1, j))$ for any $A(1, j) > 0$. Let $F_R(1) = 1$. Update Ω_{IAR} and Ω_{AR} accordingly, as in the basic RSA.

Step 3 If $F_R(N_N) = 0$, do:

Step 3.1 Let $t = t + 1$. Set a temporary speed $s_T = \infty$.

Step 3.2 For any $i \in \Omega_{AR}$, let $r_R(i) = r_R(i) + s$, and update $r_R(j)$ and $F_R(j)$ as per the basic RSA. Also, in order to guarantee that at least one unvisited node will be reached by a ripple during next time instant $t + 1$, update the temporary speed s_T as follows. If $(A(i, j) > 0)$ and $(r_R(i) < A(i, j))$ and $(s_T > A(i, j) - r_R(i))$, let $s_T = A(i, j) - r_R(i)$. Otherwise, if $(A(i, j) > 0)$ and $(r_R(i) = A(i, j))$ and $(s_T > min(A(j, k))$ for any $A(j, k) > 0$ and $k \neq i)$, let $s_T = min(A(j, k))$.

Step 3.3 As per the basic RSA, update Ω_{IAR} and Ω_{AR} based on the result of step 3.2.

Step 3.4 Update the speed for next time instant: $s = s_T$.

Step 4 Determine the best route according to Eqs.(7) and (8).

In this improved RSA, one can see that the ripple-spreading speed s changes from time to time, so that at every time instant, at least one unvisited node is reached by a ripple. The improved RSA is illustrated in Figure 3b. Compared to Figure 3a, the basic RSA takes 4 time units to reach the destination, while the improved RSA takes only 3 time units. At the time instant $t = 3$ in Figure 3a, no new node is reached by either ripple, which means this is purely a ripple-spreading step and no actual searching work is done. Obviously, this purely ripple-spreading step diminishes the computational efficiency of the basic RSA. Thanks to variable speed, there is no purely ripple-spreading step in the improved RSA. As a result, the improved RSA may gain some computational efficiency versus the basic RSA with constant ripple-spreading speed.

From step 3.2 one can see that although the ripple speed varies from time to time, all ripples spread at the same speed s during a time instant. This fact guarantees the optimality of the improved RSA, i.e., the best route is always determined by backtracking from the ripple that reaches the destination first.

In the basic RSA and the improved RSA, each node can be viewed as an agent. We only need to define how a node should behave in terms of generating and spreading its own ripple, and do not need to do any route search (e.g., calculating the current route length from the source to an intermediate node, or comparing and sorting intermediate nodes according to their current route lengths), and the optimal route will automatically emerge as the result of the ripple relay race between nodes. This is exactly what an agent-based model does: agents behave independently of the model's purpose, and then the collective effect of all of their behaviors rather surprisingly assists the specific purpose.

3.3 Extending RSA to Other POPs

First we apply the RSA concept to the one-to-all POP. Actually, we do not need to make any modification to the two RSAs in Section 3.2, except changing the termination criterion of step 3 to the following.

Step 3 If there is at least one node that has not been reached by any ripple, then perform step 3.1 to step 3.4.

The new termination criterion means that all nodes other than the source are destinations and therefore need to be reached by a ripple. Once step 3 stops, we can get the best route from the source to any node i by tracking back from the ripple that has reached node i in the first place, as explained in step 4 of Section 3.2. Figure 3 clearly illustrates that the RSA in Section 3.2 can actually resolve the one-to-all POP. We consider Figure 3a as an example. At time $t = 2$, node 2 has been reached by the initial ripple,

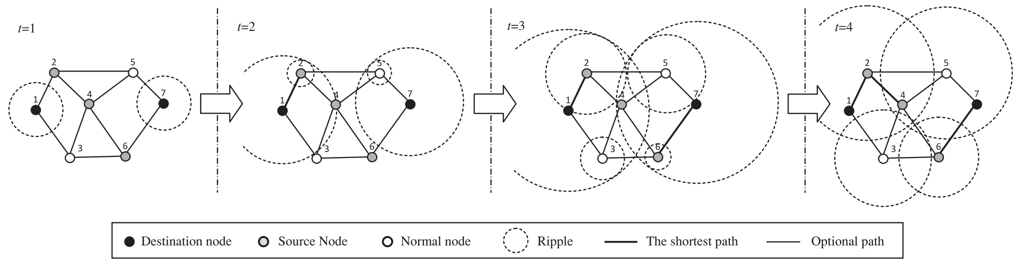


Figure 4: RSA for the many-to-many POP.

which reveals that the shortest route from node 1 to node 2 is the direct connection between them. Node 3 has been reached at time $t = 3$, which implies that the shortest route from node 1 to node 3 is the associated direct connection. At time $t = 4$, the ripple of node 3 has reached node 4, which determines that shortest route from node 1 to node 4 is $1 \to 3 \to 4$. Now it is clear that the shortest routes from node 1 to all other nodes have been identified by simulating a ripple-spreading process.

In both the one-to-one POP and the one-to-all POP, there is only one source. Therefore, at the beginning of the ripple relay race, i.e., in step 2, only one initial ripple is activated at the source. To extend to the many-to-many POP, we simply need to activate initial ripples at all sources simultaneously to start the ripple relay race. Actually, for the sake of convenience from the viewpoint of algorithm design, in the RSA for the many-to-many POP, we reverse destinations and sources. In other words, we start the ripple relay race from destination nodes, and stop the simulation once every source has been reached by at least one ripple. In the many-to-many POP, every source must travel to at least one destination, whilst a destination may connect to no source through the shortest routes. Therefore, reversing destinations and sources makes it much easier and more efficient to check whether the termination criterion is satisfied.

For brevity, the detailed modifications to extend the RSA to the many-to-many POP are not presented here, and only an illustration of the associated ripple relay race is given in Figure 4, where we assume a route network with only seven nodes. There are two destinations, i.e., node 1 and node 7; three sources, i.e., node 2, node 4, and node 6; and two normal nodes, i.e., node 3 and node 5. Two initial ripples start from the destinations. At time instant $t = 2$, the ripple of node 1 reaches the source node 2 first and therefore determines that node 2 should connect to node 1 through the direct link between node 1 and node 2. At the same time, two new ripples are triggered at node 2 and node 5 at time instant $t = 2$. At time instant $t = 3$, the ripple of node 7 reaches the source node 6 first, which determines that node 6 should connect to node 7 through the direct link between node 6 and node 7. Another two new ripples are triggered at node 3 and node 6 at time instant $t = 3$. At time instant $t = 4$, the ripples of node 2 and node 5 both reach the source node 4, but the ripple of node 2 gets there first. Therefore, the source node 4 should connect to the destination node 1 rather than to node 7, and the associated shortest route is $4 \to 2 \to 1$. Since all sources have been reached by ripples at time instant $t = 4$, indicating that every source has found a route to connect to at least one destination node, the ripple-spreading process then stops. Compared with those approaches that search for the closest destination to each source separately, the proposed algorithm achieves a parallel computing capability, as all activated ripples spread out simultaneously. As shown by the simulation results

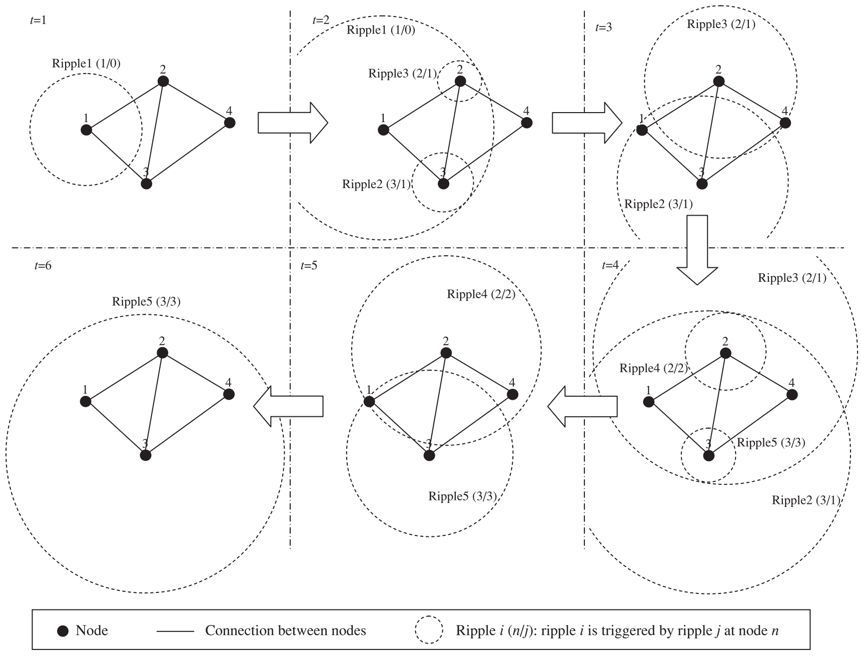


Figure 5: Ripple relay race to identify the k shortest paths.

(see Section 5), this parallel computing capability gives the proposed RSA significant computational efficiency.

So far in the simulation of ripple relay races, each node can be triggered to generate no more than one ripple, which is enough for finding the first shortest route. To extend the RSA to the k-SPP, i.e., to find the k shortest paths, we need to allow every node to be able to generate multiple ripples (Hu et al., 2013b). Actually, it is straightforward to extend the optimization principle in the natural ripple-spreading phenomenon to the k-SPP. A ripple travels at the same speed in all directions, so it reaches spatial points in order according to their distances to the ripple epicenter, i.e., the closest spatial point is the first to be reached by the ripple, and the kth closest spatial point is the kth to be reached. In the simulation of a ripple relay race where each node may generate multiple ripples, once a ripple reaches a node (not the destination), no matter whether the node has already generated any ripples, a new ripple will be generated at the node. Then, in a similar fashion as in the basic RSA for the one-to-one POP, assuming a ripple of node i reaches node j, the new ripple will spread along all connections of node j except the connection between node i and node j. The ripple will die when it has reached all nodes (except node i) connected to node j. If two ripples arrive at the same node at the same time, then two new ripples will be triggered at the node with exactly the same radius.

In the ripple relay race illustrated in Figure 5, the ripple in the kth place to reach the destination has traveled along the kth shortest path, where the solution space has four different paths from node 1 to node 4: $1 \to 2 \to 4$; $1 \to 3 \to 4$; $1 \to 2 \to 3 \to 4$; $1 \to 3 \to 2 \to 4$. At time $t = 1$, only one ripple, i.e., ripple 1, starts from node 1. During time instant $t = 2$, ripple 1 reaches node 3 and then node 2, and triggers ripple 2 at node 3, and ripple 3 at node 2. Ripple 3 reaches the destination first at time $t = 3$, identifying the first shortest path $1 \to 2 \to 4$. Ripple 2 reaches the destination at time

$t = 4$, identifying the second shortest path $1 \rightarrow 3 \rightarrow 4$. During time instant $t = 4$, ripple 2 also triggers ripple 4 at node 2, and ripple 3 triggers ripple 5 at node 3. At time $t = 5$, ripple 4 reaches the destination, discovering the third shortest path $1 \rightarrow 3 \rightarrow 2 \rightarrow 4$. At time $t = 6$, ripple 5 reaches the destination, finding the worst path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. From Figure 5, one can see clearly that we do not need to reconstruct any route network, as we never use the k best routes to calculate the $(k + 1)$th best route. This feature of the RSA distinguishes it from most existing methods for the k-SPP.

Although this paper only focuses on the four mentioned POPs, the potential of the RSA does not stop there. In fact, it opens up new opportunities to attack problems that existing methods can barely resolve. From the algorithms reported here, one can see that the key to extending the RSA to a new problem is to define problem-specific ripple-spreading behavior for agents. Hu et al. (2013b) applied the RSA to find the kth shortest project time in project management. The kth shortest project time problem is superficially similar to the k-SPP, but actually existing k-SPP methods are not applicable because a project activity often has to wait to start until some other relevant project activities have progressed to certain stage (e.g., 50% completed). This means that a node may not be accessible even though it is physically connected to the end of the current path. When a node becomes accessible depends wholly on some other relevant nodes' accessibility, and it is very likely that such other relevant nodes are not included in the current path. Therefore, traditional top-down centralized k-SPP algorithms can really help little. As a bottom-up decentralized agent-based simulation model, the RSA can address the node accessibility issue in a very straightforward way, i.e., by simply modifying the node behavior in the RSA for the k-SPP as follows. A node cannot generate its own ripples until it has been reached by the first ripple of each of those relevant nodes. Then the deterministic feature of the RSA guarantees that the kth shortest project time will be found. The capability to find the general kth best solution sheds a little light on another problem that used to look impossible: calculating the exact complete Pareto front for discrete multiobjective problems (MOPs) (Hu et al., 2013a) rather than just finding an approximation, which still dominates MOP practices (Deb et al., 2002; Knowles and Corne, 2000).

4 Theoretical Analyses of the RSA

For the sake of simplicity, our discussion in this section focuses mainly on the RSA for the one-to-one POP, which may serve as a guideline to analyze future developments in RSA.

4.1 Completeness of the RSA

In the POP, completeness means that a method can guarantee to find a route from the source to the destination as long as at least one exists. In nature, a spreading ripple will sooner or later reach any spatial point located in a reachable space. A route network is a connected graph if at least one route always exists to travel between a pair of randomly chosen nodes. A connected graph can be likened to the reachable space where a ripple will spread, and a node in the graph to a spatial point in the reachable space. As a spreading ripple will eventually sweep through the entire reachable space, with no spatial point missed, the ripples in the RSA can eventually reach any node in the connected graph by relay race traveling along the connections between nodes. Therefore, the RSA is complete in terms of finding a route from the source to the destination.

4.2 Optimality of RSA

First, let us examine how the optimization principle of the natural ripple-spreading phenomenon can guarantee the global optimal solution.

THEOREM 1: *In the RSA, the best route is the route traveled by the first ripple that reaches the destination.*

PROOF: Assume Theorem 1 is false, in other words, there exists another route that is better but takes more time to reach the destination. Now, suppose the first ripple that reaches the destination takes T_1 time units, and the cost of the route it travels along is $f(R_1)$. There is another ripple that reaches the destination at time instant $T_2 > T_1$, and the associated route cost is $f(R_2) < f(R_1)$.

In the RSA, all ripples travel at the same cost speed during a given time instant. Let $s(t) > 0$ be the ripple speed of time instant t, $t = 1, \ldots, T_2$. Because each time instant is a time unit in the RSA, then

$$f(R_1) = \sum_{t=1}^{T_1} s(t),\ f(R_2) = \sum_{t=1}^{T_2} s(t). \tag{10}$$

If Theorem 1 is false, then one has $f(R_2) < f(R_1)$, i.e., $f(R_2) - f(R_1) < 0$, which means

$$f(R_2) - f(R_1) = \sum_{t=T_1+1}^{T_2} s(t) < 0. \tag{11}$$

However, this is impossible because the ripple speed $s(t)$ is always positive. Therefore, Theorem 1 must be true.

Clearly, Theorem 1 is also the theoretical statement of the optimization principle of the natural ripple-spreading phenomenon. As mentioned in Section 3.2, the basic RSA may lose optimality if the speed does not satisfy Condition (9) due to the overtraveling problem. Here we give the theoretical analysis. Consider a general scenario as follows. Suppose that at time instant t, node n and node m are two newly visited nodes, and their ripples, i.e., ripple n and ripple m, are spreading out. The best route from the source to node n has a cost $C_{R,n}$, and to node m has a cost $C_{R,m}$. Node m has a direct connection to node h. There is a route from node n, passing K intermediate nodes (node k_1 to node k_K), to node h, and the cost associated with this route is

$$C_R = A(n, k_1) + \sum_{i=2}^{K} A(k_{i-1}, k_i) + A(k_K, h). \tag{12}$$

According to the operation process of the basic RSA described in Section 3.2, only those unvisited nodes that have a direct connection to a current newly visited node are considered to have the possibility of being visited in the next time instant. This operational restriction is actually embedded in all RSAs proposed in Section 3. Because of this operational restriction, only node k_1 and node h in this scenario have the possibility to be visited at time instant $t + 1$. Now we have the following two propositions to address the relation between the ripple speed and the optimality of the basic RSA.

PROPOSITION 1: *Suppose $C_{R,n} + C_R < C_{R,m} + A(m, h)$ in the relevant scenario. Then if the constant speed in the basic RSA does not satisfy Condition (9), the optimal route may not be found.*

PROOF: Let the speed $s > A(m, h)$, which obviously does not satisfy Condition (9). Therefore at time instant $t+1$, node h will become a visited node, but node k_2 to node k_K remain unvisited because of the operational restriction. Once a node becomes visited, it will never be considered again in a future time instant in the RSA. This means, if it is finalized that $F_R(h) = m$ at time instant $t+1$, it can never be changed again. Assuming node h is included in the final route, then obviously this route is not the optimal, because $C_{R,n} + C_R < C_{R,m} + A(m, h)$, in other words, we can replace the direct connection between node m and node h with the route from node n to node h by passing node k_1 to node k_K, and the resulting route is better. □

PROPOSITION 2: *Condition (9) can always guarantee the optimality of the basic RSA.*

PROOF: Suppose that the actual best route is composed of node n_1 to node n_{L_1} in order, where $n_1 = 1$ and $n_{L_1} = N_N$. The cost of the best route is C_{R_1}. Another route is composed of node m_1 to node m_{L_2} in order, and $m_1 = 1$ and $m_{L_2} = N_N$. This is another route from the source to the destination, and its cost is $C_{R_2} > C_{R_1}$.

First, we assume that except for node 1 and node N_N, these two routes share no other nodes. Further, we assume that node n_{L_1-1} and node m_{L_2-1} become or remain newly visited nodes at time instant t_1 and t_2, respectively, and their ripples reach node N_N at time instant t_1+1 and t_2+1, respectively. Next we need to prove that the case of $t_1 > t_2$ is impossible when Condition (9) is satisfied.

Assume $t_1 > t_2$ is the case, then there must be a node n_x, $1 \le x < L_1 - 1$, in the actual best route that becomes or remains as a newly visited node at time instant t_2. Suppose the cost from node 1 to node n_x along the actual best route is C_{R_1,t_2}, and the cost from node 1 to node m_{L_2-1} along the second route is C_{R_2,t_2}. Then because all ripples travel at the same speed in the basic RSA, we have at time instant t_2

$$C_{R_1,t_2} + r_R(n_x) = C_{R_2,t_2} + r_R(m_{L_2-1}). \tag{13}$$

Because

$$C_{R_1} = C_{R_1,t_2} + \sum_{i=x}^{L_1-1} A(n_i, n_{i+1}), C_{R_2} = C_{R_2,t_2} + A(m_{L_2-1}, N_N), \tag{14}$$

then,

$$C_{R_2} - C_{R_1} = (A(m_{L_2-1}, N_N) - r_R(m_{L_2-1})) - \left(\sum_{i=x}^{L_1-1} A(n_i, n_{i+1}) - r_R(n_x)\right). \tag{15}$$

In the basic RSA, we always have $r_R(i) < A(i, j)$ if node j has not been visited yet. Therefore, for $x < L_1 - 1$, we have

$$\sum_{i=x}^{L_1-1} A(n_i, n_{i+1}) - r_R(n_x) > \sum_{i=x+1}^{L_1-1} A(n_i, n_{i+1}) \ge a_{min}. \tag{16}$$

Because the ripple of node m_{L_2-1} reaches node N_N at time instant t_2+1, we must have, according to Condition (9),

$$A(m_{L_2-1}, N_N) - r_R(m_{L_2-1}) \le s \le a_{min}. \tag{17}$$

From Eqs. (15)–(17), we have $C_{R_2} - C_{R_1} < 0$, which is against the assumption that the route associated with C_{R_1} is the actual best route. Therefore, with Condition (9) satisfied, the case of $t_1 > t_2$ is impossible. Since $t_1 \le t_2$, one can easily prove that, similar to the proof of Theorem 1, the ripple of node n_{L_1-1} reaches the destination first. This means the basic RSA can find the actual best route when it shares no nodes with other routes except node 1 and node N_N.

In the case where the best route shares nodes besides node 1 and node N_N with other routes, we can divide the two routes into segments at shared nodes, and make sure each pair of route segments only share their starting node and end node. Then we know from the first part of this proof that, when Condition (9) is satisfied, the basic RSA can find the best segment of each pair. Therefore it can eventually find the best route from node 1 to node N_N.

Now we prove the optimality of the improved RSA of Section 3.2. To this end, we have the following proposition.

PROPOSITION 3: *The optimality of the improved RSA is guaranteed if the variable speed satisfies*

$$0 \leq s(t) \leq a_{min}(t), \tag{18}$$

where $a_{min}(t)$ is defined as the minimal remaining cost along a direct connection of every current newly visited node to its unvisited end node, i.e., $a_{min}(t) = min(A(i, j) - r_R(i))$, for all $i \in \Omega_{AR}$ that has $F_R(i) > 0$ at time instant t.

PROOF: The proof of this proposition is similar to the proof of Proposition 2, except that Eqs. (16) and (17) become

$$\sum_{i=x}^{L_1-1} A(n_i, n_{i+1}) - r_R(n_x) \geq \sum_{i=x+1}^{L_1-1} A(n_i, n_{i+1}) > a_{min}(t_2), \tag{19}$$

$$A(m_{L_2-1}, N_N) - r_R(m_{L_2-1}) \leq s(t_2) \leq a_{min}(t_2), \tag{20}$$

respectively. Then from Eq. (15), Eq. (19), and Eq. (20), we can still deduce that the case of $t_1 > t_2$ is impossible. Since in the improved RSA we have $s(t) = a_{min}(t)$ at any time t, which satisfies Condition (18), then, like the proof of Proposition 2, the algorithm can guarantee the optimal solution. □

4.3 Time Complexity of the RSA

The basic computational step in the RSA is the spreading of a ripple along a connection in a time unit. The basic computational step simply includes an addition operation and a comparison operation: increasing the radius of a ripple by the ripple-spreading speed, and then comparing the new radius with the length of a connection. Suppose a network has N_N nodes. In the basic RSA, each node can generate no more than one ripple. Assume, on average, each node has N_{AC} connections, and it takes N_{ATU} time units for a ripple to travel through a connection. Then, in the worst case, the source node needs $N_{AC} \times N_{ATU}$ basic computational steps, the destination node needs no steps, and for the rest $(N_N - 2)$ nodes, each needs no more than $(N_{AC} - 1) \times N_{ATU}$ steps. Therefore, approximately $N_{AC} \times N_{ATU} + (N_N - 2) \times (N_{AC} - 1) \times N_{ATU}$ basic computational steps need to be conducted before the shortest path can be found. This means the computational complexity of the basic RSA can be assessed as $O(N_N \times N_{AC} \times N_{ATU})$. It is well known that computational complexity of Dijkstra's algorithm is $O(N_N \times \log(N_N) + (N_N \times N_{AC})/2)$. This time complexity analysis is helpful to estimate whether the RSA is suitable for a specific network system. For example, if $\log(N_N)$ is larger than $N_{AC} \times N_{ATU}$, which is often the case in large-scale sparse networks such as transportation systems and power grids, then the RSA could be computationally more efficient than Dijkstra's algorithm.

4.4 Relation of the RSA to Other Methods

Compared with the deterministic methods reviewed in Section 2.2, which are all comprehensively defined search algorithms, the proposed RSA is actually a simulation model rather than an algorithm. Basically, a comprehensively defined search algorithm can be viewed to follow a top-down, centralized methodology of calculation. The top goal is clearly defined to find the shortest route, and a certain heuristic search rule or logic (such as breadth-first search, depth-first search, or best-first search) is explicitly executed to process all necessary nodes. In such methods, nodes are objects to be processed in a centralized manner. The RSA is different in that it follows a bottom-up, decentralized methodology of simulation, and no explicit search rules or logics are employed. Nodes are not objects to be processed but are actually agents that can behave independently in a decentralized manner. In other words, we do not define any search rule or logic in the RSA, but simply define how each node will behave, i.e., under what circumstances a node will generate a ripple and how the ripple will then spread. As is well known, in an agent-based simulation model, the behavior of an agent usually has no obvious link to the collective behavior of the model. This is exactly what happens in the RSA. The behavior of a node is to generate and spread ripples, while the collective behavior of the model is to determine the first shortest route via the ripple that reaches the destination first. In a centralized search algorithm, nodes are processed one by one serially according to a certain global ranking or sorting mechanism (e.g., calculating and sorting path lengths from the source to intermediate nodes). In the decentralized simulation model of the RSA, nodes behave in parallel or simultaneously to generate and spread ripples (there is no need to calculate any path length during the relay race). Therefore, the proposed RSA is much more like well-acknowledged agent-based evolutionary algorithms, such as ACO and PSO. However, evolutionary algorithms have a stochastic nature, which means their outputs are not repeatable and therefore cannot practically guarantee to find global optima every time. The proposed RSA is completely different, as it is a deterministic method, and its output is unique for a given route network, and the result is always globally optimal thanks to the optimization principle reflected in the natural ripple-spreading phenomenon. It should be noted that despite the fundamental decentralized nature of the revised RSA, applying variable speed is a global measure. Therefore, it is worth further investigation concerning the role and benefits of introducing suitable global measures into the decentralized RSA.

One may argue that bottom-up simulation models are often computationally less efficient than top-down search algorithms; for example, evolutionary algorithms usually take more time than dynamic programming. Fortunately, as shown in the comparative simulations in Section 5, this is not true of the RSA. There are a few factors that may contribute to the computational efficiency of the RSA. For example, the RSA never needs to calculate, compare, or sort path lengths from the source to intermediate nodes. Figure 6 illustrates another factor that explains why the RSA may well be more computationally efficient than top-down search algorithms. There are five routes from node 1 to node 4 in total in Figure 6, and the shortest route is $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. A brute-force method has to explore all the five routes in order to guarantee the finding of the shortest route (Fig. 6a). Dijkstra's algorithm needs to explore three complete routes and two incomplete routes (Fig. 6b). The proposed RSA only needs to explore one complete route and two incomplete routes (Fig. 6c), because the simulation simply stops when the first ripple reaches the destination, i.e., node 4, along the shortest route $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, and by the time of termination, other ripples have only reached node 5 and node 6. Figure 6

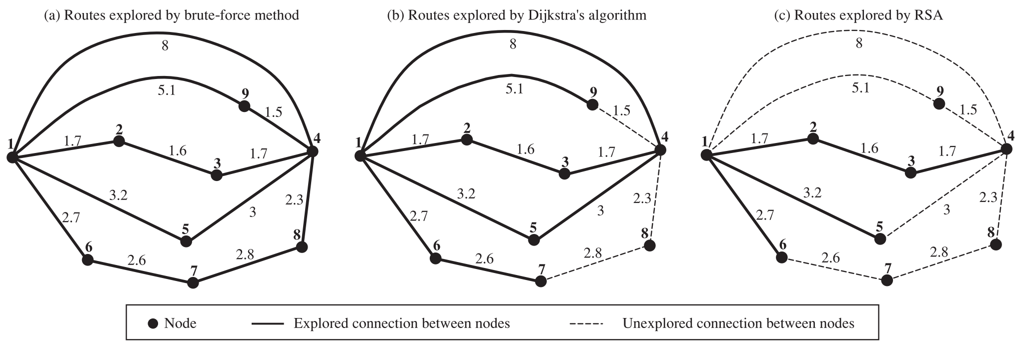

(a) Routes explored by brute-force method
(b) Routes explored by Dijkstra's algorithm
(c) Routes explored by RSA
● Node
Explored connection between nodes
Unexplored connection between nodes

Figure 6: Search efficiency of different methods.

clearly illustrates that because of the ripple-spreading process, the RSA may narrow down the solution space for search.

It is well established that Bellman's optimization principle is the theoretical foundation of dynamic programming, which guarantees the optimality of many search methods such as the A^* algorithm and Dijkstra's algorithm (Sniedovich, 1986). Now we turn to the examination of the relation between Bellman's optimization principle and the optimization principle of the ripple-spreading phenomenon. Bellman's principle is stated as follows: "An optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" (Bellman, 1957). In the context of route optimization, after starting from the source, no matter which intermediate node we approach, the remaining route must be the shortest route from that intermediate node to the destination. Therefore, in dynamic programming, we first calculate the shortest distances from the destination to all those nodes that have a direct connection to the destination. Then, based on the results, we progress to calculate the shortest distances from the destination to all those nodes that have a direct connection to those already calculated nodes, until the source is reached. Note that in route optimization the source and the destination are usually reversible. Then both the A^* algorithm and Dijkstra's algorithm exactly follow the dynamic programming practice, and calculate the shortest distances from the source to intermediate nodes from the nearest to the farthest, until the destination is reached. We can view the simulation process of the RSA at the macro level as following coincidentally the centralized strategy described by the Bellman's optimization principle. However, at the micro level, the RSA never calculates any shortest distance from the source to any intermediate node, but the decentralized behavior of nodes defined according to the ripple-spreading optimization principle can automatically ensure the realization of optimal remaining decisions in Bellman's statement.

Here we use the k-SPP to illustrate that the centralized nature of dynamic programming is disadvantageous when compared with the decentralized nature of the simulation model of the RSA. To apply a dynamic programming method, such as the A^* algorithm and Dijkstra's algorithm, to the k-SPP, we usually have to keep reconstructing route networks (Yen, 1971; Eppstein, 1998; Aljazzar and Leue, 2011). For example, first we apply, say, Dijkstra's algorithm, to find the first shortest path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ (Figure 7a). Then we proceed to calculate the second shortest path based on the first shortest path. To this end, we have to reconstruct three new route networks, each of

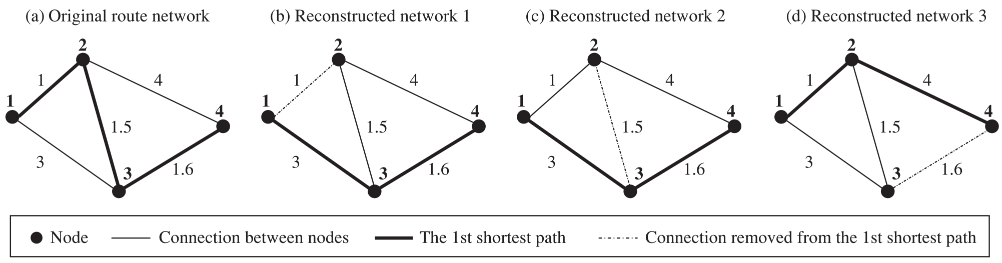

(a) Original route network
(b) Reconstructed network 1
(c) Reconstructed network 2
(d) Reconstructed network 3
1
2
3
4
1
4
1.5
3
1.6
● Node
Connection between nodes
The 1st shortest path
Connection removed from the 1st shortest path

Figure 7: Reconstructing route networks for Dijkstra's algorithm.

which is generated by removing one of the three connections that together compose the first shortest path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ (Figures 7b–7d). Then based on each reconstructed route network, we apply Dijkstra's algorithm to calculate the associated first shortest path. The associated first shortest paths in Figures 7b–7d are $1 \rightarrow 3 \rightarrow 4$, $1 \rightarrow 3 \rightarrow 4$, and $1 \rightarrow 2 \rightarrow 4$, respectively. Among these associated first shortest paths, $1 \rightarrow 3 \rightarrow 4$ is the shortest. Therefore, we know the second shortest path in the original route network of Figure 7a is $1 \rightarrow 3 \rightarrow 4$. The methods following the practice of dynamic programming often focus on how to reconstruct route networks efficiently (Yen, 1971; Eppstein, 1998; Aljazzar and Leue, 2011). The RSA for the *k*-SPP, discussed in Section 3.3, is completely different because we never need to reconstruct any route network but simply simulate micro ripple-activating and spreading behaviors at nodes in the original route network, and the macro outputs of the simulation process are automatically the *k* shortest paths (e.g., see Fig. 5).

Furthermore, the decentralized RSA can achieve what is beyond the capability of centralized dynamic programming. In the dynamic network of Hu et al. (2014), whether an unexplored connection is feasible or not depends on which connections have been explored. Figure 8 gives an example, where the feasibility of unexplored connections from node 4 to node 7 depends on how node 4 is reached by the current route. If node 4 is reached from node 3, then only long connections are feasible to node 7, while if node 4 is reached from node 2, then some short connections become feasible. This violates the Bellman's optimization condition. Therefore, all dynamic programming methods such as A^* and Dijkstra's algorithm will lose the global optimality, and the best path they find has a cost of 22 in the case of Figure 8. Surprisingly, the RSA for the *k*-SPP can still guarantee the global optimality in this dynamic network to find the globally optimal path with a cost of 16. This is another "impossible becomes possible" example thanks to the RSA, which also implies that there is a fundamental difference between the RSA and those methods based on the Bellman optimization principle.

5 Simulation Results

5.1 Simulation Setup

In this section we conduct analysis of the RSA based on four sets of comparative simulation experiments, i.e., one-to-one, one-to-all, many-to-many and *k*-SPP. In the one-to-one set, the two RSAs proposed in Section 3.2, i.e., the basic RSA and the improved RSA, are compared with the A^* algorithm and Dijkstra's algorithm (Cormen et al., 2001; Sniedovich, 2010). In the basic RSA, the ripple-spreading speed is set as half of the minimal connection cost in a route network. Since the A^* algorithm may fail to find global optima if the heuristic function is not admissible (Dechter and Pearl,

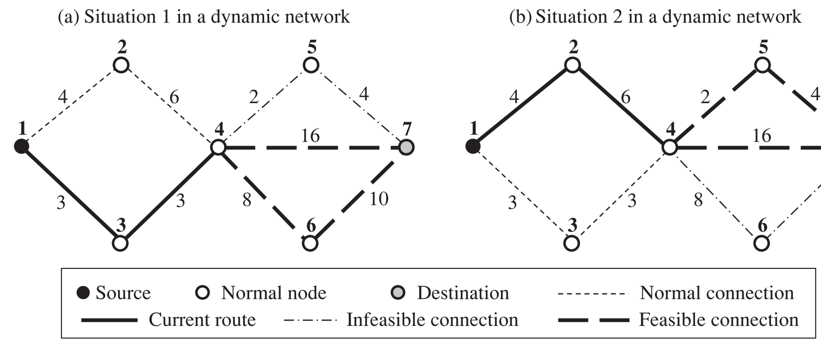


Figure 8: A dynamic network where the feasibility of unexplored connections depends on current route.

1985), in the other three sets of experiments, we only use Dijkstra's algorithm for comparative purposes. Actually, in the *k*-SPP set, the RSA is compared with the well-known Yen's algorithm (Yen, 1971) and Dijkstra's algorithm is employed by Yen's algorithm to calculate the first shortest paths in reconstructed route networks.

The networks used in the experiments are generated as follows. First, we randomly disturb the locations of N_N nodes, which are originally evenly distributed in a rectangular area defined by $[-1000\ 1000\ \ -1000\ 1000]$. We let the minimal distance between any two of the evenly distributed nodes be d_{EDN}. Then a random disturbance is within the range $[-d_{EDN}/3,\ d_{EDN}/3]$. Then we establish connections between nodes according to two network models. For some networks (denoted WS), the model in Watts and Strogatz (1998) is used, in order to get Poisson degree distribution. For other networks (denoted BA), the model in Barabási and Albert (1999) is used, in order to generate scale-free topologies with power-law degree distribution. In some WS networks, each node on average has four connections, i.e., $N_{AC} = 4$, while in other WS networks, $N_{AC} = 8$. In some BA networks, the power coefficient is set as $e = 1$, while in other BA networks, $e = 2$. Basically, a larger e means a more significant power-law degree distribution. Finally, we calculate the cost of each established connection. There are two different definitions of route cost: straight-line distance between nodes and random weight. Networks with random weights (within the range of [1 10] in this study) as route cost are especially used to illustrated that the A^* algorithm may lose optimality, while both Dijkstra's algorithm and the proposed RSA can always guarantee finding global optima.

For a given set of experimental parameters (WS model or BA model, N_{AC}, e, N_N, and so on), we randomly generate 100 route networks. Since all algorithms in this study are deterministic, we apply them to each network only once. Then we compare their average results based on the 100 networks of the given set of experimental parameters. Note that in all experiments, a route network is recorded not as an adjacency matrix as described in Section 2.1, but instead to reduce the memory demand a list of established connections between nodes is adopted. All methods are coded and all tests are conducted in a MATLAB® environment on a personal computer with a 2.6 GHz CPU, 2 GB memory and the Windows XP operating system.

The combined influence of experimental parameters, such as the density of connections, the topology of network, and the unevenness in connection costs, is complicated. There is thus a case for further investigation of such influence. This is particularly important to a specific real-world application study. In this study, we mainly focus on fundamental differences between the RSA and existing methods.

Table 1: Test results for the one-to-one POP (straight-line distance, $N_N = 1{,}000$).

	WS Model (Poisson Distribution)						BA Model (Power-Law Distribution, $N_{AC} = 4$)					
	$N_{AC} = 4$			$N_{AC} = 8$			$e = 1$			$e = 2$		
	Runtime(s)			Runtime(s)			Runtime(s)			Runtime(s)		
	Mean	SD	Obj.	Mean	SD	Obj.	Mean	SD	Obj.	Mean	SD	Obj.
RSA1-1	0.10	0.02	**3415.38**	0.12	0.02	**3326.15**	0.01	0.02	**3472.03**	0.12	0.03	**3586.47**
RSA1-2	0.08	**0.01**	**3415.38**	0.09	0.02	**3326.15**	0.08	0.02	**3472.03**	0.10	0.02	**3586.47**
A^*1-1	0.06	**0.01**	**3415.38**	0.09	**0.01**	**3326.15**	0.08	**0.01**	**3472.03**	0.10	**0.01**	**3586.47**
A^*1-2	0.14	0.02	**3415.38**	0.17	0.02	**3326.15**	0.16	0.02	**3472.03**	0.22	0.03	**3586.47**
Dijkstra1	**0.04**	**0.01**	**3415.38**	**0.06**	**0.01**	**3326.15**	**0.05**	**0.01**	**3472.03**	**0.06**	**0.01**	**3586.47**

Table 2: Test results for the one-to-one POP (random weight, $N_N = 1{,}000$).

	WS Model (Poisson Distribution)						BA Model (Power-Law Distribution, $N_{AC} = 4$)					
	$N_{AC} = 4$			$N_{AC} = 8$			$e = 1$			$e = 2$		
	Runtime(s)			Runtime(s)			Runtime(s)			Runtime(s)		
	Mean	SD	Obj.	Mean	SD	Obj.	Mean	SD	Obj.	Mean	SD	Obj.
RSA1-1	1.76	0.27	**122.74**	2.52	0.33	**113.52**	1.46	0.17	**97.61**	1.28	0.14	**82.90**
RSA1-2	0.68	0.11	**122.74**	0.89	0.12	**113.52**	0.47	0.07	**97.61**	0.35	0.06	**82.90**
A^*1-1	0.10	0.02	181.42	0.13	0.02	173.83	0.09	0.02	134.42	0.11	0.02	112.44
A^*1-2	0.14	0.02	**122.74**	0.20	0.02	**113.52**	0.13	0.02	**97.61**	0.14	0.02	**82.90**
Dijkstra1	**0.03**	**0.01**	**122.74**	**0.04**	**0.01**	**113.52**	**0.03**	**0.01**	**97.61**	**0.03**	**0.01**	**82.90**

5.2 One-to-One POP Experimental Results

First, we consider the set of one-to-one POP experiments by comparing the RSA with the A^* algorithm and Dijkstra's algorithm. In this set, the total number of nodes in a route network is $N_N = 1{,}000$. The source is the left bottom node, and the destination is the right top node. Both RSAs, i.e., the basic RSA and the improved RSA, are tested here, and they are denoted RSA1-1 and RSA1-2, respectively. There are also two versions of the A^* algorithm used in this set: one version, denoted A^*1-1, chooses the heuristic function to be the straight-line distance from the last node of the current route to the destination node, and the other version, denoted A^*1-2, always sets the heuristic function to zero. Dijkstra's algorithm here is denoted Dijkstra1. The results of runtime (given as mean value and standard deviation) and average route cost (the value of the objective function, Obj) are presented in Table 1 (straight-line distance) and Table 2 (random weight), which show the following.

- Since RSA1-1, RSA1-2, A^*1-2, and Dijkstra1 have theoretical proofs of global optimality for a given network, they should yield the same Obj value.

Therefore, no matter how many networks are used, the average Obj values of the four algorithms should be the same. Tables 1 and 2 show that these four algorithms always have the same average Obj values, which is consistent with the theoretical analysis. This empirically confirms the optimality of these four algorithms.

- A^*1-1 can guarantee optimality when the connection cost is defined as the physical straight-line distance between nodes (Table 1) but often loses optimality in the case of random weights (Table 2). This is because the heuristic function of A^*1-1 is admissible in the case of Table 1: the straight-line distance from a node to the destination is always no larger than the actual minimal cost from that node to the destination. However, in the case of Table 2, the random weights of connections have nothing to do with straight-line distances. Therefore, the heuristic function of A^*1-1 is no longer admissible and then cannot guarantee the optimality of A^*1-1 (Table 2).
- In terms of computational time, on average, Dijkstra1 is the most efficient. In the case of physical straight-line distance as connection cost, RSA1-1, RSA1-2, A^*1-1, and A^*1-2, are comparable to each other. In the case of random weight, the performance of either RSA1-1 or RSA1-2 drops greatly, which can be explained as follows. Connections between nodes usually have different costs, and this is described as unevenness in connection costs, which can be roughly measured by the ratio between the maximal connection cost and the minimal one in a network. This ratio is about 3 in the experiments of Table 1, and about 10 in those of Table 2. A large ratio usually means a larger N_{ATU}. As analyzed in Section 4.3, the time complexity of A^* or Dijkstra's algorithm is only determined by N_N and N_{AC}, while the time complexity of RSA is directly influenced by N_{ATU}. Therefore, the A^* and Dijkstra algorithms are quite robust against the unevenness in connection costs, while the RSA is sensitive.
- As analyzed in Section 3.2, because of using a variable ripple-spreading speed, RSA1-2 is more computationally efficient than RSA1-1.
- In terms of the absolute value of SD, the RSA in general has the largest SD. This is also understandable based on the theoretical result in Section 4.3. The time complexity of RSA is $O(N_N \times N_{AC} \times N_{ATU})$. For either A^* or Dijkstra, the time complexity is mainly determined by N_N and N_{AC}. Therefore, once the pair of (N_N, N_{AC}) is given and fixed, other network parameters such as node locations and degree distribution have relatively much smaller influence on the runtime. However, in the case of the RSA, even for a fixed pair of (N_N, N_{AC}), different node locations or degree distribution may lead to rather different N_{ATU} and then cause the runtime to vary more significantly. One might argue that the RSA has smaller values of SD/mean. However, SD/mean makes little sense here because the runtimes in this section are too small, which might be largely biased by computer performance uncertainties.
- For a given definition of connection cost and network models, the values of N_{AC} and e have similar influences on all algorithms. For example, in WS networks, no matter which definition of connection cost is employed, all algorithms have larger runtimes in the $N_{AC} = 8$ case than in the $N_{AC} = 4$ case because a larger N_{AC} generally means a larger search space. However, they have smaller Obj values for $N_{AC} = 8$ than for $N_{AC} = 4$ because more connections between

Table 3: Runtimes (second) of different algorithms in one-to-all POP (straight-line distance).

	$N_N = 2{,}000$		$N_N = 4{,}000$		$N_N = 6{,}000$		$N_N = 8{,}000$		$N_N = 10{,}000$	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
RSA2-1	0.18	0.03	0.38	0.05	0.69	0.06	1.01	0.09	1.29	0.11
RSA2-2	0.14	0.03	0.34	0.04	0.62	0.06	0.85	0.08	1.14	0.09
Dijkstra2	**0.05**	**0.01**	**0.09**	**0.02**	**0.16**	**0.02**	**0.25**	**0.03**	**0.34**	**0.03**

nodes usually means smaller shortest paths. In BA networks with straight-line distance as connection cost, all algorithms have larger runtimes and Obj values for $e = 2$ than for $e = 1$ because a larger e implies the shortest path between a pair of nodes is more likely to go around some hub nodes, and going around hub nodes usually means a long physical distance. However, in BA networks with random weight as connection cost, all algorithms have smaller runtimes and Obj values for $e = 2$ than for $e = 1$. This is because going around hub nodes usually means fewer connections are included in the shortest path, and because of the random weights and smaller connection numbers, this usually produces smaller path costs.

- Tables 1 and 2 show that for the one-to-one POP, Dijkstra's algorithm is better than the A^* algorithm; the proposed RSA could be comparable to Dijkstra's algorithm; and the performance gap between the RSA and Dijkstra's algorithm is the most significant in the case of the WS model with $N_{AC} = 4$. Therefore, in the next three sets of experiments, because of limited space, we compare the RSA with Dijkstra's algorithm using the WS model with $N_{AC} = 4$ and changing N_N to form different problem scales, and we only analyze runtimes, as optimality is always guaranteed.

5.3 One-to-All POP Experimental Results

The second set is one-to-all tests, and the network scale is increased to $N_N = 2{,}000, 4{,}000, 6{,}000, 8{,}000$ and $10{,}000$, respectively. In each test case, the source is randomly chosen from the N_N nodes. As discussed in Section 3.3, it is very easy to extend the RSA to the one-to-all POP just by changing their termination criteria, and the extended RSAs are denoted as RSA2-1 and RSA2-2, respectively. The one-to-one Dijkstra's algorithm can also be applied to the one-to-all POP by simply changing the termination criterion, and the resulting algorithm is labeled Dijkstra2. The main results are given in Table 3, which shows the following.

- In terms of the mean value of runtime, Dijkstra2 is better than RSA2-2 and than RSA2-1.
- In terms of the absolute value of SD, Dijkstra2 is also better than RSA2-2.

5.4 Many-to-Many POP Experimental Results

In one-to-one and one-to-all tests, Dijkstra's algorithm is faster than the basic RSA and the improved RSA. Here with the many-to-many POP, we demonstrate that both RSAs may outperform Dijkstra's algorithm in terms of computational efficiency. The

Table 4: Runtimes (second) of different algorithms in many-to-many POP (straight-line distance).

	$N_N = 20{,}000$		$N_N = 40{,}000$		$N_N = 60{,}000$		$N_N = 80{,}000$		$N_N = 100{,}000$	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
RSA3-1	1.01	0.09	2.10	0.19	3.89	0.27	8.01	0.53	17.37	0.86
RSA3-2	**0.97**	**0.08**	**1.96**	**0.17**	**3.24**	**0.24**	**5.85**	**0.41**	**11.69**	**0.69**
Dijkstra3-1	7.13	0.41	25.32	1.38	51.85	2.41	128.32	4.95	276.90	8.47
Dijkstra3-2	6.42	0.37	21.93	1.21	46.69	2.12	98.13	4.02	165.82	5.62

network scale is further increased to $N_N = 20{,}000$, 40,000, 60,000, 80,000 and 100,000, respectively. For a given N_N, we randomly choose 20% nodes as destinations, and set the rest as sources. As explained in Section 3.3, the basic RSA and the improved RSA can be extended to the many-to-many POP by simply starting initial ripples at all destination nodes simultaneously, and the modified RSAs are denoted RSA3-1 and RSA3-2, respectively. There are two ways to extend the one-to-one Dijkstra's algorithm, i.e., Dijkstra1, to the many-to-many POP. We can run Dijkstra1 for every unvisited source node, and each run of Dijkstra1 will stop once a destination node is reached (no matter which one). We denote this Dijkstra's algorithm as Dijkstra3-1. We can also start Dijkstra's algorithm from the set of all destination nodes. In the one-to-one Dijkstra's algorithm, there is only one node (either the source or the destination) in the initial search front. Now we can set the initial search front as the whole set of destinations and then keep expanding the search front until all sources are visited. The resulting method is called Dijkstra3-2. In contrast to Dijkstra3-1, we can get the results in only one run of Dijkstra3-2, just as in RSA3-1 and RSA3-2. The average results are given in Table 4, which clearly shows the following.

- In terms of computational efficiency, both RSA3-1 and RSA3-2, i.e., the basic RSA and the improved RSA, are faster than either Dijkstra3-1 or Dijkstra3-2 (the RSA is about ten times faster than Dijkstra's algorithm). By analyzing the values of SD/mean, one can see again that Dijkstra's algorithm has more stable runtimes.
- In the one-to-one POP and the one-to-all POP, the runtimes of RSA1-1 and RSA1-2 are both worse than those of Dijkstra's algorithm. This then poses the question as to what makes RSA3-1 and RSA3-2 become faster than Dijkstra's algorithm in the many-to-many POP. It is partly the search efficiency (see Figure 6) that makes the difference. In the one-to-one POP, there is only one destination, and therefore the solution space explored by the RSA may be just a few routes fewer than with Dijkstra's algorithm. A one-to-all POP is largely equivalent to a one-to-one POP where the destination is the farthest node from the source. In the many-to-many POP, there are tens of thousands of sources (since RSA3-1, RSA3-2, and Dijkstra2 all progress from destinations to sources), which means the solution space explored by the RSA could be hundreds of thousands of routes smaller than that of Dijkstra's algorithm. Another reason for the better computational efficiency of the RSA is its parallel computing capability (see Section 4.4).

Table 5: Runtimes (second) of different algorithms in the *k*-SPP (straight-line distance).

		$k = 2$		$k = 4$		$k = 6$		$k = 8$		$k = 10$	
N_N		Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
16	Yen	**0.01**	**0.00**	**0.01**	**0.00**	0.03	**0.01**	0.07	**0.01**	0.16	0.03
	RSA4	**0.01**	**0.00**	0.02	**0.00**	**0.02**	**0.01**	**0.03**	**0.01**	**0.03**	**0.02**
25	Yen	**0.01**	**0.00**	**0.02**	**0.00**	**0.05**	**0.01**	**0.11**	**0.02**	0.36	0.05
	RSA4	0.05	0.02	0.10	0.01	0.13	0.04	0.15	0.04	**0.18**	**0.04**
36	Yen	**0.02**	**0.01**	**0.08**	**0.02**	**0.33**	**0.05**	1.79	0.10	7.14	0.38
	RSA4	0.15	0.04	0.39	0.06	0.65	0.07	**0.93**	**0.08**	**1.17**	**0.12**
49	Yen	**0.08**	**0.02**	**0.38**	**0.06**	**1.86**	**0.11**	8.92	**0.41**	41.33	1.72
	RSA4	0.64	0.06	1.19	0.09	3.52	0.22	**6.37**	0.42	**8.87**	**0.69**
64	Yen	**0.26**	**0.04**	**1.48**	**0.09**	**8.49**	**0.49**	49.52	2.07	293.27	10.27
	RSA4	2.89	0.21	6.29	0.48	16.25	0.99	**27.90**	**1.83**	**41.29**	**2.35**

- One may notice that the RSA for a many-to-many POP with $N_N = 20{,}000$ (Table 4) is faster than the RSA for a one-to-all POP with $N_N = 10{,}000$ (Table 3). This is because in a one-to-all POP with $N_N = 10{,}000$, the largest number of nodes in the shortest routes is easily over 100, while in a many-to-many POP with $N_N = 20{,}000$, the largest number of nodes in the shortest routes is often just a few tens. Therefore, based on the analysis in Section 4.3, the time complexity of the RSA in a one-to-all POP with $N_N = 10{,}000$ is greater than that in a many-to-many POP with $N_N = 20{,}000$.

5.5 k-SPP Experimental Results

Now we move to the *k*-SPP. In this set of tests, the network scale is set as $N_N = 16$, 25, 36, 49, and 64, respectively, because the scalability of the algorithms tested still needs to be improved in future work. The experimental results with these small N_N values can fairly demonstrate the fundamental difference between the RSA and existing methods for the *k*-SPP. Another parameter of this set is the number of best paths that need to be calculated, and it is set as $k = 2$, 4, 6, 8, and 10, respectively. In each test case, the source is the left bottom node, and the destination is the right top node. Here for the sake of simplicity in programming, only the basic RSA is extended for the *k*-SPP, denoted RSA4. Yen's algorithm, denoted Yen, is used for comparative purposes, and the one-to-one Dijkstra's algorithm is integrated in Yen to calculate the first shortest path. The results are given in Table 5, from which one may make the following observations.

- In terms of computational efficiency, RSA4 is clearly better than Yen in those test cases with a large k, such as $k = 10$. Basically, Yen's algorithm is more sensitive to the change in k because a larger k implies an exponentially heavier computational load to reconstruct route networks. As discussed in Section 5.2, the runtimes of RSA4 are relatively more unstable than those of Yen, as shown by the SD values.
- Both the RSA and Yen's algorithm have a scalability problem, and therefore more efforts are still needed to improve them. Table 5 shows that the RSA may have a better application potential when k is large, which is more likely

the concern of the *k*-SPP. For example, in the method of Hu et al. (2013a) to calculate the complete Pareto front of a multiobjective optimization problem, it is crucial to have a method capable of finding the *k* best solutions to each of the single-objective optimization problems, and the *k* may vary within a large range.

6 Conclusions and Future Work

Inspiration from nature often brings surprises, which has been demonstrated repeatedly by many successful advances in evolutionary computation (EC). This paper focuses on the natural ripple-spreading phenomenon and attempts to apply its underlying principle to design effective algorithms for the path optimization problem (POP). Inspired by the fact that a ripple travels at the same speed in all directions and therefore always reaches the closest spatial point first, a deterministic bottom-up decentralized agent-based simulation model is proposed to search the optimal path through a route network. The proposed RSA is complete and optimal, and it also has a good scalability because of its computational efficiency. Through comparison with some of the best route optimization algorithms via simulation, the proposed RSA is tested in four kinds of POPs and demonstrates a very effective performance. This paper shows that from the ripple-spreading optimization principle, to the combination of agent-based and deterministic features, to the new opportunities of tackling some complex problems, the proposed RSA is largely distinguished from existing methods and therefore enriches the theoretical foundations of EC.

More theoretical, technical, and implemental efforts are needed in the future to explore the full potentials of the RSA, for example, studying in depth where the strength of the RSA comes from, investigating the roles of local node behaviors and global measures of setting time-varying ripple-spreading speed, improving the computational efficiency of the RSA for the *k*-SPP, developing effective purpose-designed RSAs for real-world POPs as well as other kinds of optimization problems, analyzing the computational complexity of such purpose-designed RSAs, technically realizing the parallel computing capability of the RSA, and understanding the influence of experimental parameters such as the topology of networks, the unevenness of connection costs, and the percentage of destination nodes. Basically, the proposed RSA is a promising concept, and many things can and need to be done.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 61472041), the National Basic Research Program of China (Grant No. 2012CB955404), the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant No.41321001), and the Seventh Framework Programme (FP7) of the European Union (Grant No. PIOF-GA-2011-299725). The authors would also like to thank Dr. Matthew D. Higgins and Miss Tianyao Miao, who assisted with preparing the final version of this article with LaTex.

References

Aljazzar, H., and Leue, S. (2011). K*: A heuristic search algorithm for finding the *k* shortest paths. *Artificial Intelligence*, 175(18): 2129–2154.

Bäck, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1): 3–17.

Barabási, A.-L., and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439): 509–512.

Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to algorithms*. 2nd ed. Cambridge, MA: MIT Press.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182–197.

Dechter, R., and Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3): 505–536.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): 269–271.

Drezner, T., Drezner, Z., and Salhi, S. (2002). Solving the multiple competitive facilities location problem. *European Journal of Operational Research*, 142(1): 138–151.

Eppstein, D. (1998). Finding the *k* shortest paths. *SIAM Journal on Computing*, 28(2): 652–673.

Fu, L., Sun, D., and Rilett, L. R. (2006). Heuristic shortest path algorithms for transportation applications: State of the art. *Computers and Operations Research*, 33(11): 3324–3343.

Gastner, M. T., and Newman, M. (2006). Optimal design of spatial distribution networks. *Physical Review E*, 74(1): 016117.

Goodrich, M. T., and Tamassia, R. (2006). *Algorithm design: Foundation, analysis and Internet examples*. New York: Wiley.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*, 37:28–29.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Ann Arbor: University of Michigan Press.

Hu, X.-B., and Di Paolo, E. (2011). A ripple-spreading genetic algorithm for the aircraft sequencing problem. *Evolutionary Computation*, 19(1): 77–106.

Hu, X.-B., Leeson, M. S., and Hines, E. L. (2010). A ripple-spreading genetic algorithm for the network coding problem. In *IEEE Congress on Evolutionary Computation*, pp. 1–8.

Hu, X.-B., Wang, M., and Di Paolo, E. (2013a). Calculating complete and exact Pareto front for multiobjective optimization: A new deterministic approach for discrete problems. *IEEE Transactions on Systems, Man and Cybernetics (B)*, 43(3): 1088–1101.

Hu, X.-B., Wang, M., Leeson, M. S., Hines, E. L., and Di Paolo, E. (2011). Deterministic ripple-spreading model for complex networks. *Physical Review E*, 83(4): 046123.

Hu, X.-B., Wang, M., Sun, Q., Leeson, M. S., and Di Paolo, E. (2013b). A ripple-spreading algorithm to calculate the *k* best solutions to the project time management problem. In *IEEE Symposium on Computational Intelligence in Scheduling*, pp. 75–82.

Hu, X.-B., Wang, M., Ye, Q., Han, Z., and Leeson, M. S. (2014). Multi-objective new product development by complete Pareto front and ripple-spreading algorithm. *Neurocomputing*, 142:4–15.

Knowles, J. D., and Corne, D. W. (2000). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2): 149–172.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1): 97–109.

Korf, R. E., Reid, M., and Edelkamp, S. (2001). Time complexity of iterative-deepening-A*. *Artificial Intelligence*, 129(1): 199–218.

Liao, J.-Q., Hu, X.-B., Wang, M., and Leeson, M. S. (2013). Epidemic modelling by ripple-spreading network and genetic algorithm. *Mathematical Problems in Engineering*, article 506240.

Misa, T. J., and Frana, P. L. (2010). An interview with Edsger W. Dijkstra. *Communications of the ACM*, 53(8): 41–47.

Mohanta, K., and Poddar, B. (2012). Comprehensive study on computational methods for *k*-shortest paths problem. *International Journal of Computer Applications*, 40(14): 22–26.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Reading, MA: Addison-Wesley.

Russell, S., and Norvig, P. (2010). *Artificial intelligence: A modern approach*. 3rd ed. Englewood Cliffs, NJ: Prentice Hall.

Sniedovich, M. (1986). A new look at Bellman's principle of optimality. *Journal of Optimization Theory and Applications*, 49(1): 161–176.

Sniedovich, M. (2006). Dijkstra's algorithm revisited: The dynamic programming connexion. *Control and Cybernetics*, 35(3): 599.

Sniedovich, M. (2010). *Dynamic programming: Foundations and principles*. Boca Raton, FL: CRC Press.

Watts, D. J., and Strogatz, S. H. (1998). Collective dynamics of small-world networks. *Nature*, 393(6684): 440–442.

Yan, S., Chen, C.-Y., and Lin, Y.-F. (2011). A model with a heuristic algorithm for solving the long-term many-to-many car pooling problem. *IEEE Transactions on Intelligent Transportation Systems*, 12(4): 1362–1373.

Yen, J. Y. (1971). Finding the *k* shortest loopless paths in a network. *Management Science*, 17(11): 712–716.