Generating optimal paths in dynamic environments using River Formation Dynamics algorithm

Grzegorz Redlarski, Mariusz Dabkowski, and Aleksander Palkowski*

Department of Mechatronics and High Voltage Engineering, Gdansk University of Technology, ul. G. Narutowicza 11/12, 80-233 Gdansk, Poland

*Corresponding author: aleksander.palkowski@pg.gda.pl

Abstract

The paper presents a comparison of four optimisation algorithms implemented for the purpose of finding the shortest path in static and dynamic environments with obstacles. Two classical graph algorithms — the Dijkstra complete algorithm and A* heuristic algorithm — were compared with metaheuristic River Formation Dynamics swarm algorithm and its newly introduced modified version. Moreover, another swarm algorithm has been compared — the Ant Colony Optimization and its modification. Terms and conditions of the simulation are thoroughly explained, paying special attention to the new, modified River Formation Dynamics algorithm. The algorithms were used for the purpose of generating the shortest path in three different types of environments, each served as a static environment and as a dynamic environment with changing goal or changing obstacles. The results show that the proposed modified River Formation Dynamics algorithm is efficient in finding the shortest path, especially when compared to its original version. In cases where the path should be adjusted to changes in the environment, calculations carried out by the proposed algorithm are faster than the A*, Dijkstra, and Ant Colony Optimization algorithms. This advantage is even more evident the more complex and extensive the environment is.

1 Introduction

Path planning relates to the problem of calculating a continuous or discrete path in a known or unknown environment in such a way, that the path will not violate any established constraints. An object guided along the path should achieve its goal while avoiding collision with any encountered obstacles. This process is most frequently implemented with an additional path length optimisation. However, energy consumption, time of travel or other physical factors are often subjected to an optimisation process as well.

Most often this issue is encountered in navigation systems for autonomous vehicles where safe and efficient passage is crucial and its efficiency is measured by path length (the shortest possible) and calculation time (also taking into account time to adapt to changes during the passage). As nearly all real-world applications are subject to, often very frequent, changes, the ability to rapidly correct itself according to the given changes plays a crucial role in all path planning systems.

The issue of computing an efficient path was repeatedly discussed in various contexts [1, 2]. Current solutions are based on two major principles: on classical, iterative methods, such as the Dijkstra'a algorithm [3], and on heuristic methods [4, 5]. The classical approach is computationally simple, however offers limited possibilities for more complex cases. Methods based on heuristic algorithms provide more flexibility for coping with multi-objective, complicated problems. In addition, heuristics are well suited for solving NP-complete problems, which are the case in path finding tasks, especially with time, velocity, and acceleration restrictions among polyhedral obstacles [6, 7].

Today the issue of moving in a known environment based on a complete map does not pose a major research problem. On the other hand navigation in an area partially or completely unknown or with dynamically moving obstacles and/or goals is still a problem not fully explored. This is particularly evident if one takes into account the latest artificial intelligence methods, such as swarm algorithms, to generate collision-free trajectories under uncertainty while dealing with possible terrain variations.

Employment of heuristic methods, e.g. the D* algorithm [8, 9], provides good results, however some of the methods suffer from high computation times (which eliminates their use for rapidly changing conditions), or from lack of adjustment to measurement uncertainty. Research on the D* algorithm for path planning in unknown, partially known, and completely known environment shows that even with very limited knowledge about the environment (approx. 1/64 of the whole map) the costs of passage were maintained only at a slightly higher level than for a completely known environment [8]. By further modifications of the D* algorithm, a 96% computation time reduction was achieved [9]. Another heuristic method – the A* algorithm – was used to generate collision-free trajectories in real-time for dynamic unknown environments [10]. By generating intermediate targets that depend on execution time and current values of sensory signals, the algorithm was able to generate collision-free paths in real time, even in large-scale, dynamic, unknown environments. However, the algorithm could not cope with re-planning of (upgrading) the paths.

So far, swarm algorithms were successfully used to solve the problem of static programming, i. e. when the map is known [11, 12]. A comparison between Dijkstra's algorithm, modified Dijkstra's algorithm, and a combined Dijkstra's and Particle Swarm Optimization (PSO) algorithm was made for generating collision-free paths for a fixed, unchanging environment [11]. It was proved that the PSO algorithm poses higher potential for generating shorter motion paths than the other presented methods. Zhang et al. [12] proposed a two-stage optimisation algorithm for path selection risk and length of the path. The PSO algorithm was used to optimise both functions and the results showed acceptable and safe motion paths for four three-dimensional environments affected by uncertainty, with static obstacles varying in number, location, and shape.

There is a limited number of studies on the use of other nature-inspired algorithms, like Genetic Algorithms [13, 14] or Simulated Annealing algorithm [15], as an effective tool for dynamic path optimisation. In complex environments, Genetic Algorithms consume a lot of time to generate collision-free trajectories, which limits their practical use. It was shown that for an environment with fourteen static and five dynamic obstacles, a system based on Genetic Algorithms needed about 10 seconds to find the first valid path [13]. Better results were observed in the case of the Simulated Annealing algorithm, where its enhanced version demonstrated a significant advantage over compared algorithms in speed of generating a collision-free trajectory in complex environments [15]. Another nature-inspired algorithm – Random Particle Path Optimization – was compared with a classical path planning method – Artificial Potential Field [16]. It was concluded that the proposed algorithm can generate shorter paths even in an environment with moving obstacles and varying goals.

As mentioned above, the issue of effective motion planning under dynamic conditions (i.e. with

variable terrain and objectives) remains a matter that needs to be addressed. Nature-inspired algorithms, with swarm algorithms in particular, are means to solve even most complicated problems with varying conditions, while still being effectively fast and robust. Due to increasing number of new swarm algorithms it is crucial to examine their potentiality to solve the given problem, as most of the literature focuses on implementing the Particle Swarm Optimization or Ant Colony Optimization algorithms.

One of the latest methods in this field of computation is the River Formation Dynamics (RFD) algorithm [17]. It is based on an idea to imitate the process of riverbed formation, and as an optimisation algorithm it surpasses even the Ant Colony Optimization [18, 19]. Moreover, there are indications that this algorithm is efficient in robot motion planning [20], however its features have not been thoroughly researched. Bearing in mind those facts, a question is raised whether the RFD algorithm can be used to effectively generate optimal paths in dynamically changing conditions, especially when compared to more established methods. Therefore this article aims at presenting the RFD algorithm in the task of dynamic motion planning, as well as introduces all necessary improvements to the algorithm's core. To measure its effectiveness, the original and modified versions of the RFD algorithm are compared with the Ant Colony Optimization algorithm together with its modified version and two classical Dijkstra's [21] and A* [22] algorithms by their optimised path length and computation time.

2 Methods

2.1 River Formation Dynamics algorithm

The principle of the RFD algorithm is to imitate the process of formation of riverbeds. A set of drops placed at the starting point is subjected to gravitational forces that attract the drops to the centre of the earth. As a result, these drops are distributed throughout their environment, seeking the lowest point – the sea. Riverbeds with many meanders are formed in this process. The RFD utilises this idea in graph theory problems. A set of agents-drops are created and move on edges between nodes, exploring an environment for the best solution. This is accomplished by mechanisms of erosion and soil sedimentation that relate to changes in altitude that is assigned to each node. Drops, when moving throughout an environment, modify node altitudes along their path. The transition from one node to another is carried out according to decreasing gradient of the nodes, which in fact provides many benefits (e.g., avoidance of local cycles) [17]. The RFD algorithm in this manner is a gradient-oriented variant of the Ant Colony Optimization algorithm.

A brief description of the RFD algorithm is as follows. An amount of soil is assigned to each node. Drops, as they move, erode their paths (taking some soil from nodes) or deposit the carried sediment (thus increasing the altitudes of nodes). Probability of choosing the next node depends on the gradient, which is proportional to the difference between height of the node at which the drop resides and height of its neighbour. In the beginning the environment is flat, i.e. altitudes of all nodes are equal, except the goal node which is equal to zero during the entire process. Drops are placed in the initial node to enable further exploration of the environment. At each step a group of drops sequentially traverses the space, and then performs erosion on visited nodes. Algorithm 1 presents the RFD algorithm in a form of a pseudo-code.

Drops move one at a time (line 5), until they reach the goal or are unable to make a move, in which case they evaporate to start over in a new iteration. The probability $P_k(i, j)$ that a drop k

Algorithm 1 River Formation Dynamics algorithm

1: Place all drops in starting node
2: Height of nodes ← initial height
3: Height of target node ← 0
4: **while** end conditions are not met **do**
5: Move all drops across the graph
6: Analyse complete paths
7: Height of nodes on paths −= erosion based on path costs
8: Height of all nodes += small amount of sediment
9: **end while**

residing in node i would select the next node j is as follows:

$$P_k(i,j) = \begin{cases} \frac{gradient(i,j)}{total} & \text{for } j \in V_k(i) \\ \frac{\omega/|gradient(i,j)|}{total} & \text{for } j \in U_k(i) \\ \frac{\delta}{total} & \text{for } j \in F_k(i) \end{cases} \tag{1}$$

where

$$gradient(i,j) = \frac{altitude(i) - altitude(j)}{distance(i,j)} \tag{2}$$

$$\begin{aligned} total = &\left(\sum_{l \in V_k(i)} gradient(i,l) \right) + \\ &+ \left(\sum_{l \in U_k(i)} \frac{\omega}{|gradient(i,l)|} \right) + \\ &+ \left(\sum_{l \in F_k(i)} \delta \right) \end{aligned} \tag{3}$$

$V_k(i)$ is a set of neighbouring nodes with a positive gradient (node i has a higher altitude than node j), $U_k(i)$ is a set of neighbouring nodes with a negative gradient (altitude of node j is higher), and $F_k(i)$ represents neighbours with a flat gradient. Coefficients ω and δ are certain small fixed values.

After all drops move, an erosion process is executed on all travelled paths by reducing altitudes of nodes in proportion to their gradient with a successive node (line 7). The erosion amount for each pair of nodes i and j (Equation (4)) also depends on the number of all used drops D, number of all nodes in the graph N, and a certain erosion coefficient E.

Additionally, if a drop failed to choose a subsequent node to transition onto, it deposits a fraction of carried sediment and evaporates for the rest of the algorithm iteration. This reduces the likelihood of transition to blind alleys, hence weakening bad paths.

$$erosion(i,j) = \frac{E}{(N-1) \cdot D} \cdot gradient(i,j) \tag{4}$$

After each iteration a certain, small amount of sediment is added to all nodes (line 8) in order to avoid a situation where all altitudes are close to zero, which would make gradients negligible and

would ruin all formed paths. The formula for the sediment to be added is presented in Equation (5).

$$sediment = \frac{erosionProduced}{N-1} \tag{5}$$

The algorithm is executed until the final condition is reached. This condition may represent all drops moving on the same path. Additionally, in order to reduce computation time, a top limit on the number of iterations can be introduced, as well as a condition verifying if the solution has not been improved by the last n loops.

2.2 Modified River Formation Dynamics algorithm

The original RFD algorithm has certain drawbacks which prevent the algorithm from performing well in the described path generation problem [20]. Because of a large number of coefficients, tuning of the algorithm to a particular case is highly unintuitive. And even despite this, its convergence rate is small for more complicated environments.

Because of the above, this article introduces the modified RFD algorithm. The algorithm is modified so that the formula for the transition probability for the next node to be selected is based on an exponential function. A large number of additional RFD coefficients strongly influence the algorithm's efficiency for a specific problem. Their values are difficult to tune properly. By changing the formula, this problem is disposed off. In addition, distance from the objective node heuristic is added (as in the A* algorithm). Equation (6) presents the modified probability formula and its exemplary results are presented in Figure 1. It is based on two coefficients: $pBase$, which is the base for the exponent, and α, which is a convergence tuning coefficient. The d_j indicates Cartesian distance from node j to the goal.

$$\begin{aligned} P_k(i,j) &= \frac{pBase^{gradient(i,j)}}{(d_j)^\alpha} \Big/ total, \\ total &= \sum_{l \in V_k(i) \cup U_k(i) \cup F_k(i)} \frac{pBase^{gradient(i,l)}}{(d_l)^\alpha} \end{aligned} \tag{6}$$

As can be seen in Figure 1, the probability values are uniformly distributed from very low values for negative gradients to rapidly increasing values for positive gradients. A flat gradient is assigned with a fixed value (1 in this case). It is more intuitive to tune the exponent base value for a particular case, bearing in mind how rapidly the probability should increase with increasing gradient, and what would be the maximum gradient value. The same formula can be applied to the erosion values.

2.3 Ant Colony Optimization

Ant Colony Optimization (ACO) developed by Dorigo et al. [23] is one of the first optimisation techniques inspired by the intelligence of animal swarms. The source of its inspiration was the behaviour of ant colonies, especially the mechanism of their communication. The ACO algorithm is mainly used in graph problems where it was proved to be highly efficient. The Ant System algorithm is one of the basic algorithms derived from the ACO techniques group. Algorithm 2 presents the Ant System algorithm in a form of a pseudo-code.

In each iteration, agents-ants start at the initial node. For an ant k located in node i, selection of the next node j is made according to the following likelihood:

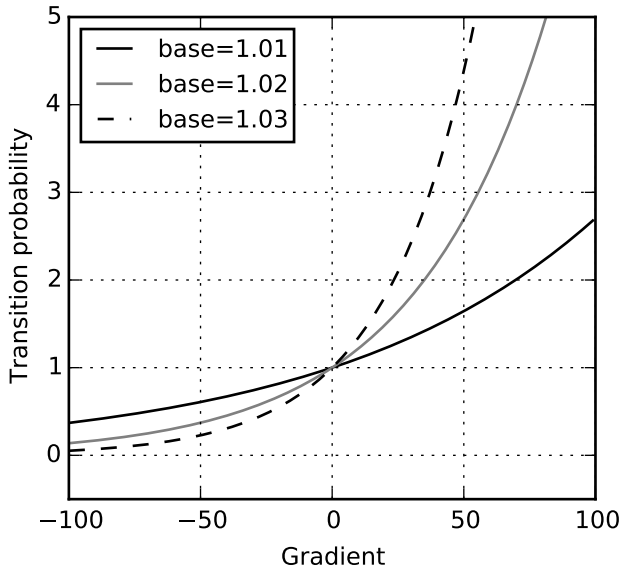

Figure 1: Probability of selecting the next node based on its gradient for three base values.

Algorithm 2 Ant Colony Optimization algorithm

1: Assign m agents to the start node.
2: Assign a certain initial amount of pheromone to each edge of the graph.
3: **while** end conditions are not met **do**
4: Construct a path for each agent.
5: Update the amount of pheromone on each edge.
6: **end while**
7: Save the best path.

$$P_k(i,j) = \begin{cases} \frac{\tau_{i,j}}{\sum_{l \in N_k(i)} \tau_{i,l}} & \text{if } j \in N_k(i) \\ 0 & \text{if } j \notin N_k(i) \end{cases} \tag{7}$$

where $N_k(i)$ is a set of nodes connected to the edges of the node in which the ant k is located, and $\tau_{i,j}$ is the amount of pheromone on the edge between nodes i and j.

Each selected node is saved in a local solution S. Upon reaching the end, an ant returns to the starting node while changing the amount of pheromone on edges contained in the solution S. This change is inversely proportional to the value of the objective function $f(S)$, i.e. the length of the travelled path. In addition, a positive factor Q is introduced. The formula for updating the amount of pheromone on a single edge between node i and j is as follows:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \frac{Q}{f(S)} \tag{8}$$

To simulate the process of volatilization of pheromones, the mechanism of evaporation is introduced. It prevents formation of very large differences between successive edges. The formula for the volatilization of pheromones on an edge is shown in the following equation:

$$\tau_{i,j} \leftarrow (1-\rho) \cdot \tau_{i,j} \tag{9}$$

where ρ stands for an evaporation coefficient.

2.4 Environment representation

To apply the above mentioned algorithms in a path generation problem, the environment must be properly modelled. The model should be in a form of a graph with nodes being the access points, and edges representing all possible transitions.

Let $G = (V, E)$ be a graph containing a set of vertices V and edges E, and $w : E \to \{1, 2, \ldots\}$, $w \in \mathbb{R}_+$ be a function that assigns an edge weight to every single edge in G. An edge $(u, v) \in E$ is indicated by $u-v$ with $u, v \in V$, whereas its weight is indicated by w_{u-v}. The sequence of vertices $p = v_0, v_1, \ldots, v_k$ is called a path in G if and only if v_i-v_{i+1} is an edge in E for all $i = 0, 1, \ldots, k-1$. If edge weights represent distances between connected vertices, a sum of edge weights between consecutive vertices in the considered path p is $d = \sum_{i=0}^{k-1} w_{v_i-v_{i+1}}$ and can be called a path length. The problem of path generation comes down to minimising the value of d for a path that connects the starting point to the goal.

This path represents successively selected nodes in the graph, the structure of which reflects the appearance of an environment. The model assumes the presence of obstacles (inaccessible nodes). The environments presented can be divided into $W \times H$ cells, thus a graph containing $W \cdot H$ nodes with proper edge connections can be its representation. Figure 2 presents such environment model for a 4×3 map. All nodes are connected by edges to their neighbours, unless an adjacent node is representing an obstacle.

Since the environment map is based on a rectangular picture, all graph nodes can be represented by pixels of this picture. Therefore, each node can have up to eight neighbours being the adjacent pixels. If a node is representing an obstacle its connections to all adjacent nodes are severed.

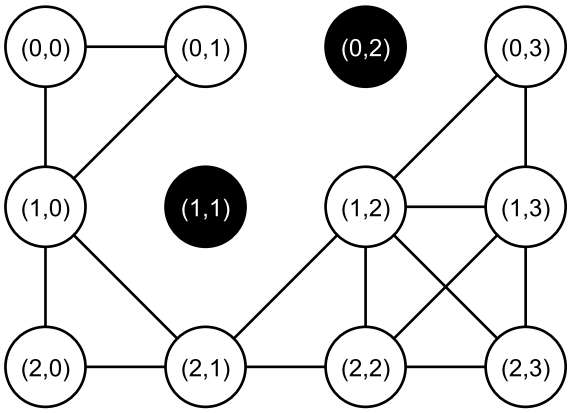

Figure 2: Representation of a 4×3 map points as a graph. Black nodes represent obstacles.

3 Results and discussion

This section presents a comparative study of five algorithms: Dijkstra's, A* (the two most popular graph algorithms, a complete one and heuristic), Ant Colony Optimization, a modified version of the Ant Colony Optimization (referred to as ACO*), River Formation Dynamics, and a modified

River Formation Dynamics algorithm (referred to as RFD*). The performance tests were carried out in three representative environments involving a large number of obstacles arranged in different configurations. Each environment was scaled to three different sizes: 20×20, 50×50, and 100×100 cells, resulting in a total of nine test cases. A graphical representation of those three types of environments is shown in Figure 3.

The modified ACO algorithm was developed in a similar manner than the RFD* algorithm. That is, The probability of choosing the next node is divided by distance to the goal, as in the denominator in Eq. (6).

Points on maps that represent tested environments were translated into graphs according to Section 2.4. Coordinates of the starting point and the objective were as follows: (0,0)→(19,19) for 20×20 maze, (1,9)→(18,9) for 20×20 mosaic, (3,9)→(16,9) for 20×20 alley, (0,0)→(49,49) for 50×50 maze, (2,24)→(46,24) for 50×50 mosaic, (9,24)→(38,24) for 50×50 alley, (0,0)→(99,99) for 100×100 maze, (2,54)→(97,54) for 100×100 mosaic, and (23,49)→(75,49) for 100×100 alley.

All results presented in the following tables summarise a series of twenty tests carried out for each case. For this purpose, a test model has been made using C# programming language. Test were carried out on a personal computer with an Intel Core i7-3770 CPU and 32 GB of RAM. The presented swarm algorithms were set to work for up to 100 iterations.

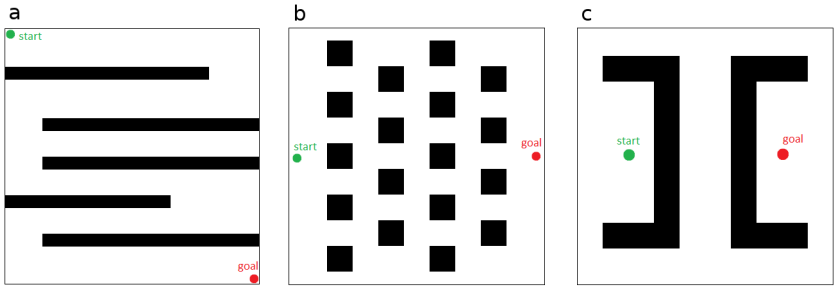

Figure 3: Graphical representation of environments used for the tests (obstacles marked black). **a**: maze. **b**: mosaic. **c**: alley.

Table 1 shows test results for a static environment, where both the starting point and destination are fixed during calculations. Records marked "n/a" indicate that for the given case an algorithm could not generate any correct path.

As can be seen from Table 1, the RFD and ACO algorithms failed to generate any correct path for the maze of all sizes. In other cases the algorithms have successfully found a path from the starting point to the objective. Path lengths computed by the Dijkstra's algorithm may be regarded as the optimal ones (as this algorithm is bound to find optimal solutions, as long as such exist). Compared to it, the RFD algorithm has found paths longer than the optimum by about 1.61 (for the 20×20 mosaic) to 23.9 times (for the 100×100 alley). In the case of execution time the RFD algorithm performed likewise. Computation times were longer by about 70 (for environments of size of 100×100 cells) to 758 times (for environments of size of 20×20 cells) than the best performing A* algorithm. This proves that the RFD algorithm in its original form is not an alternative for the classical algorithms. This is mainly due to its poor convergence when operating on such an extensive graph while the transition costs are dependent only on distance of succeeding nodes. Moreover, values of coefficients of the RFD algorithm have a strong influence on its performance and must be adapted to a specific problem, however their tuning is highly unintuitive.

After introducing distance heuristic to the ACO algorithm, its performance increased. The

Table 1: Averaged path length and computation time of compared algorithms for static, invariable environments.

		Mosaic 20×20	**Alley 20×20**	**Maze 20×20**	**Mosaic 50×50**	**Alley 50×50**	**Maze 50×50**	**Mosaic 100×100**	**Alley 100×100**	**Maze 100×100**
Path length [-]	**Dijkstra**	18.65	33.48	78.62	50.62	81.97	228.52	116.94	170.42	510.56
	A*	18.65	33.48	78.62	50.62	81.97	228.52	116.94	170.42	510.56
	ACO	51.68	138.98	n/a	450.65	1158.61	n/a	2749.60	5435.08	n/a
	ACO*	18.65	87.49	n/a	55.16	181.56	n/a	211.98	379.15	n/a
	RFD	30.21	117.12	n/a	383.1	839.24	n/a	2196.57	4070.42	n/a
	RFD*	18.65	33.48	78.62	50.62	81.97	228.52	116.94	170.42	510.56
Computation time [ms]	**Dijkstra**	6.051	5.787	6.534	57.876	59.333	46.316	542.385	753.503	445.556
	A*	1.601	4.453	6.265	24.173	56.662	48.895	335.751	661.118	448.542
	ACO	650.16	681.82	n/a	10118.72	10086.02	n/a	131071.05	132445.70	n/a
	ACO*	71.481	662.063	n/a	4005.241	6078.251	n/a	38630.724	29628.093	n/a
	RFD	1213.59	1279.466	n/a	8121.88	8704.5	n/a	33653.09	46341.6	n/a
	RFD*	59.548	103.421	84.211	268.349	291.77	337.529	4172.388	5934.1	5108.49

additional heuristic improves convergence.

The modified RFD* algorithm has found the optimal path in all cases, however its computation times were still 5 to 37 times longer than the fastest obtained from the A* algorithm. Despite a considerable improvement with respect to the original RFD algorithm, its nature (heuristic calculations performed sequentially by many agents) hinders high-speed calculations for a completely unknown environment. This disadvantage may be alleviated by performing concurrent computations by all swarm agents. Nevertheless a significant improvement of its computation time and path-seeking ability is clearly visible. It is a result of the added goal distance heuristic and more intuitive parameters. In this way the swarm behaviour is more easily controllable.

The modified RFD* algorithm however shows high potential when encountered with dynamic conditions after an initial path calculation. Table 2 presents test results collected after changing the initial goal point. In this case the ACO and RFD algorithms rely on previously collected data, i.e. on pheromone levels and node heights, respectively, modified after the first stage (for which results are shown in Table 1). The new positions of goal points were set at: (19,17) for 20×20 maze, (19,8) for 20×20 mosaic, (14,9) for 20×20 alley, (49,46) for 50×50 maze, (47,22) for 50×50 mosaic, (35,24) for 50×50 alley, (99,96) for 100×100 maze, (99,54) for 100×100 mosaic, and (72,49) for 100×100 alley. As for the RFD algorithm, the old goal node was given the average height of neighbouring nodes. The new goal-node height was set to, naturally, zero.

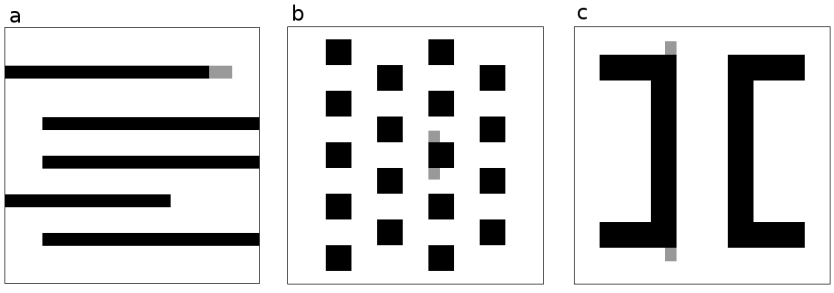

Figure 4: Localisation of new obstacles added to the testing environments (marked grey). **a**: maze. **b**: mosaic. **c**: alley.

Table 2: Averaged path length and computation time of compared algorithms after changing the goal coordinates.

		Mosaic 20×20	**Alley 20×20**	**Maze 20×20**	**Mosaic 50×50**	**Alley 50×50**	**Maze 50×50**	**Mosaic 100×100**	**Alley 100×100**	**Maze 100×100**
Path length [-]	**Dijkstra**	20.07	34.31	77.79	50.79	83.21	227.28	117.76	171.66	509.32
	A*	20.07	34.31	77.79	50.79	83.21	227.28	117.76	171.66	509.32
	ACO	60.45	135.32	n/a	471.88	1175.75	n/a	2651.17	5458.90	n/a
	ACO*	20.07	64.83	n/a	69.52	210.56	n/a	241.62	380.73	n/a
	RFD	28.42	98.16	n/a	295.24	798.88	n/a	2133.11	3824.62	n/a
	RFD*	20.07	34.31	77.79	50.79	83.21	227.28	117.76	171.66	509.32
Computation time [ms]	**Dijkstra**	4.658	4.655	6.215	55.931	62.062	45.988	513.412	751.669	443.453
	A*	1.953	4.375	6.326	25.148	56.548	47.941	320.462	654.803	442.611
	ACO	652.73	685.97	n/a	11139.51	10873.72	n/a	130845.01	132885.49	n/a
	ACO*	557.349	775.123	n/a	3966.235	6183.59	n/a	35302.022	28573.663	n/a
	RFD	197.364	201.56	n/a	183.474	355.38	n/a	1040.421	981.498	n/a
	RFD*	4.351	5.234	5.102	10.299	14.446	15.103	70.356	90.322	121.452

Table 3: Averaged path length and computation time of compared algorithms after addition of new obstacles.

		Mosaic 20×20	**Alley 20×20**	**Maze 20×20**	**Mosaic 50×50**	**Alley 50×50**	**Maze 50×50**	**Mosaic 100×100**	**Alley 100×100**	**Maze 100×100**
Path length [-]	**Dijkstra**	19.48	34.31	82.62	51.45	83.62	232.52	118.59	172.91	514.56
	A*	19.48	34.31	82.62	51.45	83.62	232.52	118.59	172.91	514.56
	ACO	49.54	112.01	n/a	438.39	1075.32	n/a	2690.44	5370.58	n/a
	ACO*	19.48	124.74	n/a	127.35	257.26	n/a	296.22	331.62	n/a
	RFD	53.88	105.43	n/a	301.66	810.58	n/a	1856.91	3788.83	n/a
	RFD*	19.48	34.31	82.62	51.45	83.62	232.52	118.59	172.91	514.56
Computation time [ms]	**Dijkstra**	5.518	4.601	5.654	67.314	85.992	57.822	557.943	753.041	440.751
	A*	2.037	4.938	5.936	28.153	55.631	47.522	362.752	637.612	434.634
	ACO	642.97	669.38	n/a	10900.92	10048.88	n/a	131001.71	133701.91	n/a
	ACO*	61.851	730.941	n/a	4928.543	6028.818	n/a	34866.798	30177.845	n/a
	RFD	120.677	581.521	n/a	804.213	1184.2	n/a	4772.492	5204.32	n/a
	RFD*	6.166	6.921	5.989	11.319	19.662	15.926	81.774	92.482	101.404

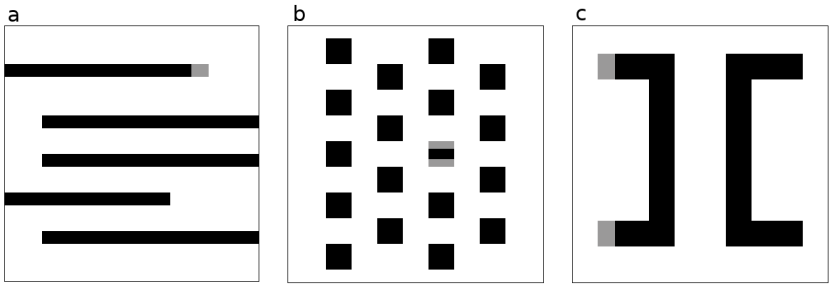

Figure 5: Marking of obstacles that have been removed from the testing environments (in gray). **a**: maze. **b**: mosaic. **c**: alley.

Table 4: Averaged path length and computation time of compared algorithms after removing certain obstacles.

		Mosaic 20×20	**Alley 20×20**	**Maze 20×20**	**Mosaic 50×50**	**Alley 50×50**	**Maze 50×50**	**Mosaic 100×100**	**Alley 100×100**	**Maze 100×100**
Path length [-]	**Dijkstra**	17.82	32.07	76.62	48.97	79.14	224.52	115.28	167.59	504.56
	A*	17.82	32.07	76.62	48.97	79.14	224.52	115.28	167.59	504.56
	ACO	37.72	127.66	n/a	429.91	1106.74	n/a	2717.84	5129.76	n/a
	ACO*	18.65	169.39	n/a	68.48	237.54	n/a	289.69	295.25	n/a
	RFD	30.21	117.12	n/a	383.1	839.24	n/a	2196.57	4070.42	n/a
	RFD*	17.82	32.07	76.62	48.97	79.14	224.52	115.28	167.59	504.56
Computation time [ms]	**Dijkstra**	4.593	4.794	5.642	56.846	71.968	58.254	541.029	757.242	438.629
	A*	1.092	4.429	5.328	20.467	57.568	48.106	338.702	665.238	440.981
	ACO	643.59	667.04	n/a	10873.22	10183.79	n/a	130503.92	132996.64	n/a
	ACO*	62.35	769.54	n/a	4814.421	5924.835	n/a	32334.028	21751.601	n/a
	RFD	205.739	471.739	n/a	838.274	747.91	n/a	2572.473	1942.22	n/a
	RFD*	5.183	7.511	8.181	9.371	11.482	12.669	47.429	50.163	70.171

Based on the results presented in Table 2 it can be reconfirmed that the ACO algorithm performs worst in all three environments. Even paths produces by the second worst algorithm – the RFD – are longer than the optimum by about 1.4 to 22.3 times. This may be a result of poor coefficient tuning. At this stage it is crucial for the swarm algorithms to be capable of searching for new, possible paths despite already having a primary path established and reinforced. In the case of the RFD algorithm some agents must be able to travel upwards a slope between two nodes (i.e. to a node with negative gradient). Maladjusted values of certain algorithm coefficients may hinder this ability. As for the ACO algorithm the reason for this situation may be agents failing into local cycles.

The A* algorithm is significantly faster for small environments (by about 100 times for 20×20 matrices). However, the more complex the environment, the faster the RFD algorithms become compared to the other two. For the 100×100 alley the A* algorithm is only about 1.4 times faster. The two classical algorithms represent a polynomial time complexity, strongly related to the number of nodes (with an additional heuristic dependency for the A* algorithm), while the swarm algorithms perform a stochastic space search, therefore in some cases being able to rapidly converge towards the goal.

In accordance with previous work [20] the RFD algorithms calculated new paths in a considerably lower time. In contrast to the Dijkstra's and A* algorithms, which must generate a path over again in every case, the RFD algorithm uses information gathered in previous iterations. Based on modified node heights, swarm agents are able to quickly travel across the graph and search for new solutions. The RFD* algorithm exceeds in this matter. Although being still slower than the two classical algorithms for small environments (A* algorithm is still about 2.2 times faster for 20×20 mosaic), its advantage increases with increasing environment size. In case of the 100×100 alley, the RFD* algorithm performs even 7.2 times faster than the best of the other.

Tables 3 and 4 present results of similar tests performed after adding new obstacles (Figure 4), and after removing certain obstacles from the initial environments (Figure 5). The aim of these tests was to either block previously calculated optimal paths or form new, possible ones, thus forcing the algorithms to adapt to new conditions. In technical terms, if an obstacle is removed or added the graph connections are modified. Addition of a new obstacle severs existing neighbour connections for all nodes adjacent to the new obstacle-node. Elimination of an obstacle creates new connections for all adjacent nodes. As for the RFD algorithm, the newly created available

node is given the default soil amount. Likewise these tests were performed after reinforcing a first path for the starting conditions (Figure 3).

The results are similar to those obtained in previous tests. All algorithms except the original RFD and ACO were able to find optimal paths. Analogously, the RFD* algorithm exceeds the other algorithms, especially in extensive environments.

As an additional test, the algorithms were tested against introducing a major change in the environment. As opposed to the tests presented above, the major changes mean either moving the goal far from the original location, nearly on the other side of the map, or deleting a significant portion of obstacles to open entirely new, shorter paths. The mosaic map new positions of goal points were set at: (19,17) for the 20×20 map, (47,45) for the 50×50 map, and (99,85) for the 100×100 map. The other maps had their obstacles removed. Details of those changes are presented in Figure 6.

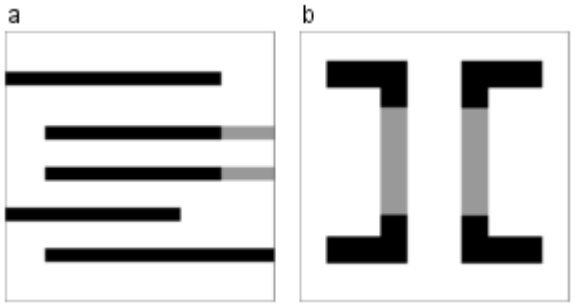

Figure 6: Marking of obstacles that have been removed from the testing environments (in gray) when a major change was introduced. **a**: maze. **b**: alley.

Table 5: Averaged path length and computation time of compared algorithms after removing certain obstacles when a major change was introduced.

		Mosaic 20×20	**Alley 20×20**	**Maze 20×20**	**Mosaic 50×50**	**Alley 50×50**	**Maze 50×50**	**Mosaic 100×100**	**Alley 100×100**	**Maze 100×100**
Path length [-]	**Dijkstra**	21.31	13	58.97	55.11	29	167.7	117.5	52	359.11
	A*	21.31	13	58.97	55.11	29	167.7	117.5	52	359.11
	ACO	85.6	238.48	n/a	506.03	1468.16	n/a	2481.27	6085.31	n/a
	ACO*	21.31	65.19	n/a	60.53	239.12	n/a	187.095	281.45	n/a
	RFD	176.27	54.01	n/a	1319.55	128.43	n/a	n/a	2616.64	n/a
	RFD*	21.31	13	58.97	55.11	29	167.7	117.5	52	359.11
Computation time [ms]	**Dijkstra**	2.858	2.296	3.515	30.237	30.966	28.663	226.148	305.574	229.698
	A*	0.988	0.592	3.359	11.637	3.618	28.475	112.069	16.251	226.841
	ACO	460.031	449.92	n/a	4368.948	4326.924	n/a	33090.306	33682.913	n/a
	ACO*	68.041	786.81	n/a	3303.432	5728.462	n/a	32817.862	22746.026	n/a
	RFD	686.816	966.442	n/a	4935.295	4919.252	n/a	n/a	22889.838	n/a
	RFD*	35.498	26.229	412.697	105.466	60.564	622.767	466.014	124.209	1067.273

In the case of a major change, the advantage of the RFD* algorithm is decreased. The algorithm has found appropriate paths, however, its computation time — as fast as it is — is still much slower than the Dijkstra's and A* algorithms.

In the above cases, in contrast to the ACO algorithm, the RFD algorithms benefit from information gathered in previous path calculations. Information about the environment is stored

as modified node heights, and can be transferred to future iterations. Therefore agents of the RFD algorithms can rapidly move across a graph along reinforced paths, while still being able to find new paths and adapt to small changes in the environment. This may be the case for other graph-searching algorithms, e.g. the Ant Colony Optimization, however the RFD algorithm is free from certain drawbacks that may cause, e.g., falling into local cycles. The ACO algorithm performs very similar to the original RFD algorithm, however can be deemed as the worst of the algorithms presented. This is also proof that for this particular kind of environment all swarm algorithms that operate like the ACO should be modified to rival such deterministic algorithms as the Dijkstra's or A*.

It may be noticed that the dynamic conditions proposed for the above test (i.e., removal and addition of obstacles) are based on minor changes in the environment. The idea behind this is that real-world environments change likewise on a basic level. In most cases all bigger changes can be divided into smaller, incremental changes. In fact, if the changes in the environment is more substantial (e.g., when a new optimal path on the other side of the map would be opened) the highlighted advantages of the RFD algorithm would not matter so much because the algorithm would have to compute an entirely new path from the beginning. Therefore the system presented at this stage is not versatile.

It is apparent that the test do not provide full potential of the RFD algorithm. As a heuristic swarm algorithm, it can be modified to work with problems of a greater complexity. It is possible to add and test another layer of restrictions and more fitness functions according to a more complex model of an environment (e.g. with varying terrain height). Additional time, velocity, or acceleration optimisation using the RFD algorithm should be the topic of future research.

4 Conclusions

This paper presents a comparison of the River Formation Dynamics algorithm and its modified versions with the Ant Colony Optimization algorithm, its modified version, and two classical Dijkstra's and A* algorithms based on their optimised path lengths and computation times. The original River Formation Dynamics algorithm was modified to increase its convergence towards a solution. During its operation, a set of agents-drops explores the environment in search for the best path from a starting node to an objective. While searching for a solution, agents modify node heights, which helps to create stronger tendencies towards a given goal. Moreover, properties of the algorithm (in particular, relying on gradient between nodes) ensure its resistance to local cycles.

The results obtained in this study have shown that modern swarm algorithms provide an alternative in the process of dynamic path planning to conventional, heuristic and complete graph algorithms, such as the Dijkstra's and A*. Superiority of the proposed, modified River Formation Dynamics algorithm is apparent for complex environments with an increasing number of cells, while in some cases its calculations are over 13 times faster than the fastest conventional A* algorithm. Moreover, the time of trajectory planning in case of dynamic changes in the environment is in the range of about 70–121 ms, which allows its use in real-time systems. However, further work is needed in order to make the path generation time comparable to conventional algorithms for completely unknown environments. A solution to this problem would be to use some other algorithm (e.g., Dijkstra's or A*) in the first iteration for an unknown environment, while in the later phases use the modified River Formation Dynamics algorithm. Another solution for improving the computation times would be to perform concurrent computations by all swarm agents.

Further work is also required to verify the applicability of swarm algorithms, such as the

modified River Formation Dynamics, to generate paths in partly unknown environments. Given the described research, such work is justified, and the results may be promising.

References

[1] D.-J. Huh et al. "Path planning and navigation for autonomous mobile robot". In: *IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the]*. Vol. 2. Nov. 2002, pp. 1538–1542. DOI: 10.1109/IECON.2002.1185508.

[2] N. Sariff and N. Buniyamin. "An Overview of Autonomous Mobile Robot Path Planning Algorithms". In: *Research and Development, 2006. SCOReD 2006. 4th Student Conference on*. June 2006, pp. 183–188. DOI: 10.1109/SCORED.2006.4339335.

[3] H. Wang, Y. Yu, and Q. Yuan. "Application of Dijkstra algorithm in robot path-planning". In: *Mechanic Automation and Control Engineering (MACE), 2011 Second International Conference on*. July 2011, pp. 1067–1069. DOI: 10.1109/MACE.2011.5987118.

[4] E. Masehian and D. Sedighizadeh. "Classic and Heuristic Approaches in Robot Motion Planning – A Chronological Review". In: *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering* 1.2 (2007), pp. 249–254. ISSN: 1307-6892.

[5] M. P. Garcia et al. "Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation". In: *Applied Soft Computing* 9.3 (2009), pp. 1102–1110. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2009.02.014. URL: http://www.sciencedirect.com/science/article/pii/S1568494609000349.

[6] C. Lum and R. Rysdyk. "Time constrained randomized path planning using spatial networks". In: *American Control Conference*. June 2008, pp. 3725–3732. DOI: 10.1109/ACC.2008.4587073.

[7] H. Li, S. X. Yang, and M. Seto. "Neural-Network-Based Path Planning for a Multirobot System With Moving Obstacles". In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 39.4 (July 2009), pp. 410–419. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2009.2020789.

[8] A. Yahja, S. Singh, and A. Stentz. "An Efficient On-line Path Planner for Outdoor Mobile Robots Operating in Vast Environments". In: *Robotics and Autonomous Systems* 33.2/3 (Aug. 2000), pp. 129–143.

[9] D. Ferguson and A. Stentz. "Field D*: An Interpolation-Based Path Planner and Replanner". In: *Robotics Research*. Ed. by S. Thrun, R. Brooks, and H. Durrant-Whyte. Vol. 28. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2007, pp. 239–253. ISBN: 978-3-540-48110-2. DOI: 10.1007/978-3-540-48113-3_22. URL: http://dx.doi.org/10.1007/978-3-540-48113-3_22.

[10] H. Liu, W. Wan, and H. Zha. "A dynamic subgoal path planner for unpredictable environments". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. May 2010, pp. 994–1001. DOI: 10.1109/ROBOT.2010.5509324.

[11] H. I. Kang, B. Lee, and K. Kim. "Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm". In: *Computational Intelligence and Industrial Application, 2008. PACIIA '08. Pacific-Asia Workshop on*. Vol. 2. Dec. 2008, pp. 1002–1004. DOI: 10.1109/PACIIA.2008.376.

[12] Y. Zhang, D.-w. Gong, and J.-h. Zhang. "Robot path planning in uncertain environment using multi-objective particle swarm optimization". In: *Neurocomputing* 103 (2013), pp. 172–185. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2012.09.019. URL: http://www.sciencedirect.com/science/article/pii/S0925231212007722.

[13] Y. Wang, I. Sillitoe, and D. Mulvaney. "Mobile Robot Path Planning in Dynamic Environments". In: *Robotics and Automation, 2007 IEEE International Conference on*. Apr. 2007, pp. 71–76. DOI: 10.1109/ROBOT.2007.363767.

[14] C.-M. Lin. "Multicriteria–multistage planning for the optimal path selection using hybrid genetic algorithms". In: *Applied Mathematics and Computation* 180.2 (2006), pp. 549–558. ISSN: 0096-3003. DOI: 10.1016/j.amc.2005.12.048. URL: http://www.sciencedirect.com/science/article/pii/S0096300306000439.

[15] H. Miao and Y.-C. Tian. "Dynamic robot path planning using an enhanced simulated annealing approach". In: *Applied Mathematics and Computation* 222 (2013), pp. 420–437. ISSN: 0096-3003. DOI: 10.1016/j.amc.2013.07.022. URL: http://www.sciencedirect.com/science/article/pii/S0096300313007728.

[16] B. Mohajer et al. "A New Online Random Particles Optimization Algorithm for Mobile Robot Path Planning in Dynamic Environments". In: *Mathematical Problems in Engineering* 2013 (2013), p. 9. DOI: 10.1155/2013/491346. URL: http://www.hindawi.com/journals/mpe/2013/491346/abs/.

[17] P. Rabanal, I. Rodriguez, and F. Rubio. "Using River Formation Dynamics to Design Heuristic Algorithms". In: *Unconventional Computation*. Ed. by S. G. Akl et al. Vol. 4618. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 163–177. ISBN: 978-3-540-73553-3. DOI: 10.1007/978-3-540-73554-0_16. URL: http://dx.doi.org/10.1007/978-3-540-73554-0%5C_16.

[18] P. Rabanal, I. Rodriguez, and F. Rubio. "Solving Dynamic TSP by Using River Formation Dynamics". In: *2008 Fourth International Conference on Natural Computation*. Vol. 1. IEEE, 2008, pp. 246–250. ISBN: 978-0-7695-3304-9. DOI: 10.1109/ICNC.2008.760. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4666848.

[19] P. Rabanal, I. Rodriguez, and F. Rubio. "Applying River Formation Dynamics to Solve NP-Complete Problems". In: *Nature-Inspired Algorithms for Optimisation*. Ed. by R. Chiong. Vol. 193. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2009, pp. 333–368. ISBN: 978-3-642-00266-3. DOI: 10.1007/978-3-642-00267-0_12. URL: http://dx.doi.org/10.1007/978-3-642-00267-0%5C_12.

[20] G. Redlarski, A. Palkowski, and M. Dabkowski. "Using River Formation Dynamics Algorithm in Mobile Robot Navigation". In: *Solid State Phenom.* 198 (2013), pp. 138–143. ISSN: 1662-9779. DOI: 10.4028/www.scientific.net/SSP.198.138. URL: http://www.scientific.net/SSP.198.138.

[21] E. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390. URL: http://dx.doi.org/10.1007/BF01386390.

[22] P. Hart, N. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (July 1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[23] M. Dorigo, M. Birattari, and T. Stutzle. "Ant colony optimization". In: *Computational Intelligence Magazine, IEEE* 1.4 (2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4129846.