Journal of Computational Science 20 (2017) 8–16

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Generating optimal paths in dynamic environments using River Formation Dynamics algorithm

Grzegorz Redlarski, Mariusz Dabkowski, Aleksander Palkowski*

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

ARTICLE INFO

Article history:
Received 23 March 2016
Received in revised form 17 January 2017
Accepted 1 March 2017
Available online 4 March 2017

Keywords:
Dynamic environment
Heuristic algorithm
Path planning
River Formation Dynamics
Swarm intelligence

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's 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

* Corresponding author.
E-mail addresses: grzegorz.redlarski@pg.gda.pl (G. Redlarski), mariusz.dabkowski@pg.gda.pl (M. Dabkowski), aleksander.palkowski@pg.gda.pl (A. Palkowski).

http://dx.doi.org/10.1016/j.jocs.2017.03.002
1877-7503/

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 s 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. Many new riverbeds 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 altitude 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. The authors of the original algorithm, however, proposed several changes to further increase the algorithm's efficiency. One of those changes is a rather rare chance of an agent to go against an increasing elevation, which is introduced to the system presented in the paper.

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.

Algorithm 1. River Formation Dynamics algorithm

```
1:  Height of nodes ← initial height
2:  Height of target node ← 0
3:  while end conditions are not met do
4:    Place all drops in starting node
5:    Move all drops across the graph for a maximum number of steps
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
```

Drops move one at a time (line 5), until they reach the goal or have traversed the maximum prescribed number of nodes. This maximum number of node is chosen as the total number of nodes in an environment. The probability $P_k(i,j)$ that a drop k residing in node i would select the next node j is as follows:

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