Parallel dragonfly algorithm

High Performance Computing for Data Science course 2024/2025

Mattia Santaniello[1] and Alessio Zeni[2]

[1]University of Trento

December 18, 2024

Abstract

Dragonfly algorithm is a new kind of approach used to solve optimization problems through the emulation of dragonflies movements, but it lacks scalability. In this paper the authors try to set up an approach based on parallelization thanks to the use of API for high performance computing like MPI and OpenMP, then there will be a comparison of execution times between presented implementation and the classic approach based on serial programming, in order to see if parallelization could lead to a performance improvement and possible future extensions.

1 Introduction

Recent years have seen the raise and evolution of swarm intelligent algorithms, based on meta-heuristic approach where single individuals of a complex population follow an exploration-exploitation methodology to find the best solution in their local search space; each member of the group work independently from the others, leading to a subdivision of the labour which decrease the overall amount of execution time, working as decentralized and parallel units; but also, due to the fact that they mimic the behaviour of biological life forms, it is possible then that they could generate random patterns through the interaction with each other. The dragonfly algorithm has been proposed for the first time in 2016, and its purpose is to simulate the behaviour of a dragonfly population according to the principles of swarm intelligent algorihms seen previously. It has gained more and more popularity: it is in estimated that during 2019 more than 300 scientific papers cited the algorithm [1], due to the fact that it has demonstrated some characteristics similar to other algoritms of the same kind, however in contrast with other swarm intelligent algorithms however it present less probability to be trapped into a local optima [2]. This paper aims to define a new approach based on popular parallel APIs, and also trying to understand if parallelization could lead to a significant speed up in performances that can be considered by future researches on this algorithm

How the algorithm works

The execution begins with the initialization of some arrays which are the position P of all dragonfiles and the 'step' vector ΔP representing the velocities of each single individual,the position of source foods F and the position of enemies E; then the algorithm executes an iteration loop where, for each dragonfly, we calculate some coefficient that will be used to update the internal state of the program. This procedure is simulated through the observation of repetitive patterns observed in the behaviour of dragonflies, which are:

- Alignment of their velocity with other members of the same group, given by the formula $A_i = \frac{\sum_{j=1}^{M} V_j}{M}$, where V_j is the velocity of the j-th neighbour
- Separation of single members in order to avoid collision, and it is retrieved by the equation $S_i = -\sum_{j=1}^{M} P - P_j$, where P_j is the position of the j-th neighbour and P the current individual position.
- Cohesion towards the center of the group, given by $C_i = \frac{\sum_{j=1}^{M} P_j}{M} - P$
- Attraction to a common source food, expressed by the formula $F_i = F^+ - P$, where F^+ is the position of the current prey and P is the position of the current dragonfly
- Distraction of enemies, represented by the equation $E_i = E^- - P$ where E^- is the position of the current enemy

Once all the parameters have been calculated we can now update the current step vector of each dragonfly using the following equation:

$$\Delta P_i^{t+1} = (sS_i + aA_i + fF_i + cC_i) + \omega \Delta P_i^t$$

where s, a, f, c are respectevly the separation weight, the alignment weight, the attraction weight, the cohesion weight and ω is the inertia weight, all of them are randomly chosen in the interval $[0, 1]$. When all step vectors have been updated we can now set the new position of the dragonflies

using this equation, which takes the new step vector as an input:

$$P_i^{t+1} = P_i^t + \Delta P_i^{t+1}$$

. However if there is a dragonfly which does not have any neighbour then we assume that the overall population moves in a random way: the most common method to do that is to use a random walk function. The previous formula then must be modified into this one here:

$$P_i^{t+1} = P_i^t + Levy(d)$$

where d is the dimension of the position vector, and $Levy(d)$ is the Levy flight function, which is calculated in this way:

$$Levy(d) = 0.01 \times \frac{r_1 \times \sigma}{|r_2|^{\frac{1}{\beta}}}$$

where $r1$ and $r2$ are two random numbers uniformally distributed in the interval $[0,1]$, and σ is a vector calculated using the following formula:

$$\sigma = \frac{sin(\frac{\beta\pi}{2}) \times \Gamma(1+\beta)}{\Gamma(\frac{\beta+1}{2}) \times \beta \times 2^{(\beta-1)/2}}$$

where $\Gamma(x)$ is the gamma function [**?**], and β is a constant.Another valid approach is to apply, instead of Levy flight, the brownian motion model, used especially in gas and particle analysis: this seems to return better performances [**?**].

2 first approach

References

[1] Chnoor M. Rahman Tarik A. Rashid Abeer Alsadoon Nebojsa Bacanin Polla Fattah Seyedali Mirjalil. A survey on dragonfly algorithm and its applications in engineering. *Evolutionary Intelligence*, 2021.

[2] Yassine Meraihi Amar Ramdane-Cherif Dalila Acheli Mohammed Mahseur. Dragonfly algorithm: a comprehensive review and applications. *Neural Computing and Applications*, 2020.