DM

Swarm Intelligence and Metaphorless Algorithms for Solving Nonlinear Equation Systems

MASTER DISSERTATION

Sérgio Gonçalves Sumares Betencourt Ribeiro
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

September | 2023

Swarm Intelligence and Metaphorless Algorithms for Solving Nonlinear Equation Systems

MASTER DISSERTATION

Sérgio Gonçalves Sumares Betencourt Ribeiro
MASTER IN INFORMATICS ENGINEERING

SUPERVISION
Luiz Carlos Guerreiro Lopes

Swarm Intelligence and Metaphorless Algorithms for Solving Nonlinear Equation Systems

Sérgio Gonçalves Sumares Betencourt Ribeiro
B.Sc. in Informatics Engineering

Supervisor:
Prof. Dr. Luiz Carlos Guerreiro Lopes

A thesis presented to the University of Madeira
in fulfillment of the requirements for the degree
of Master of Science in Informatics Engineering

Evaluation Committee:
Prof. Dr. Filipe Magno de Gouveia Quintal, FCEE/UMa (Committee Chair)
Prof. Dr. Eduardo Leopoldo Fermé, FCEE/UMa
Prof. Dr. Luiz Carlos Guerreiro Lopes, FCEE/UMa

Funchal, Portugal, September 2023

Acknowledgements

I would like to acknowledge, first and foremost, my supervisor, Luiz Guerreiro Lopes, for all the insight and support throughout this thesis, as well as Bruno Silva, whose ideas and perspectives were not only helpful but also a source of motivation.

I would also like to thank my family, in particular my wife, who encouraged me until the end.

Contents

Abstract

The simplicity, flexibility, and ease of implementation have motivated the use of population-based metaheuristic optimization algorithms. By focusing on two classes of such algorithms, particle swarm optimization (PSO) and the metaphorless Jaya algorithm, this thesis proposes to explore the capacity of these algorithms and their respective variants to solve difficult optimization problems, in particular systems of nonlinear equations converted into nonlinear optimization problems. For a numerical comparison to be made, the algorithms and their respective variants were implemented and tested several times in order to achieve a large sample that could be used to compare these approaches as well as find common methods that increase the effectiveness and efficiency of the algorithms. One of the approaches that was explored was dividing the solution search space into several subspaces, iteratively running an optimization algorithm on each subspace, and comparing those results to a greatly increased initial population. The insights from these previous experiments were then used to create a new hybrid approach to enhance the capabilities of the previous algorithms, which was then compared to preexisting alternatives.

Keywords: Computational intelligence; Particle swarm optimization; Jaya algorithm; Systems of nonlinear equations.

Resumo

A simplicidade, flexibilidade e facilidade de implementação motivou o uso de algoritmos metaheurísticos de optimização baseados em populações. Focando-se em dois destes algoritmos, optimização por exame de partículas (PSO) e no algoritmo Jaya, esta tese propõe explorar a capacidade destes algoritmos e respectivas variantes para resolver problemas de optimização de difícil resolução, em particular sistemas de equações não lineares convertidos em problemas de optimização não linear. Para que fosse possível fazer uma comparação numérica, os algoritmos e respectivas variantes foram implementados e testados várias vezes, de modo a que fosse obtida uma amostra suficientemente grande de resultados que pudesse ser usada para comparar as diferentes abordagens, assim como encontrar métodos que melhorem a eficácia e a eficiência dos algoritmos. Uma das abordagens exploradas foi a divisão do espaço de procura em vários subespaços, iterativamente correndo um algoritmo de optimização em cada subespaço, e comparar esses resultados a um grande aumento da população inicial, o que melhora a qualidade da solução, porém com um custo computacional acrescido. O conhecimento resultante dessas experiências foi utilizado na criação de uma nova abordagem hibrida para melhorar as capacidades dos algoritmos anteriores, a qual foi comparada a alternativas pré-existentes.

Keywords: Inteligência computacional; Otimização por enxame de partículas; Algoritmo Jaya; Sistemas de equações não lineares.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preamble

This master's thesis is based on the following peer-reviewed published papers:

Ribeiro, S., Lopes, L.G. (2022). Overview and Computational Analysis of PSO Variants for Solving Systems of Nonlinear Equations. In: Sharma, H., Shrivastava, V., Kumari Bharti, K., Wang, L. (eds.) *Communication and Intelligent Systems*. Lecture Notes in Networks and Systems, vol. 461. Springer, Singapore. doi: 10.1007/978-981-19-2130-8_84

Ribeiro, S., Lopes, L.G. (2022). PSO Performance for Solving Nonlinear Systems of Equations: Comparing Segmentation of Search Space and Increase of Number of Particles. In: Gervasi, O., Murgante, B., Misra, S., Rocha, A.M.A.C., Garau, C. (eds.) *Computational Science and Its Applications – ICCSA 2022 Workshops. ICCSA 2022*. Lecture Notes in Computer Science, vol. 13377. Springer, Cham. doi: 10.1007/978-3-031-10536-4_18

Ribeiro, S., Lopes, L.G. (2023). PSO–FWA: A New Hybrid Algorithm for Solving Nonlinear Equation Systems. In: Gervasi, O., Murgante, B., Rocha, A.M.A.C., Garau, C., Scorza, F., Karaca, Y., Torre, C.M. (eds.) *Computational Science and Its Applications – ICCSA 2023 Workshops. ICCSA 2023*. Lecture Notes in Computer Science, vol. 14112. Springer, Cham. doi: 10.1007/978-3-031-37129-5_5

Ribeiro, S., Silva., B., Lopes, L.G. (2023). Solving Systems of Nonlinear Equations Using Jaya and Jaya-Based Algorithms: A Computational Comparison. In: Yadav, A., Nanda, S.J., Lim, M.-H. (eds.) *Proceedings of International Conference on Paradigms of Communication, Computing and Data Analytics: PCCDA 2023*. Algorithms for Intelligent Systems. Springer, Singapore. doi: 10.1007/978-981-99-4626-6_10

The content of the four papers submitted for publication, which comprise the main chapters of this master's thesis, was left unchanged, but their original format was changed to fit the adopted standard, which led to several tables having to be modified to the landscape format.

To maintain the original numbering of the references in each paper, the references of each of them were kept at the end of the corresponding chapter, as is quite common in theses based on published work, rather than being integrated into a single list of references at the end of the thesis.

The test problems used in each of the four studies carried out are also described in each of the corresponding chapters. The fact that the problems are distinct and their order differs in each of the works is due solely to the fact that these computational studies were developed at different times without regard for maintaining exactly the same test problems and their order.

All the algorithms considered in this thesis were implemented in Julia, a dynamic, user-friendly, and open-source high-level programming language with a syntax similar to that of MATLAB, originally developed for high-performance scientific computing and data analysis [1].

1.2 Motivation

Different classes of stochastic computational intelligence algorithms have been used to solve difficult optimization problems, including swarm intelligence algorithms such as Particle Swarm Optimization (PSO) [3] and, more recently, metaphorless algorithms such as Jaya [8], whose simplicity and efficiency make them quite appealing in comparison to other classes of more complex population-based algorithms such as genetic and other evolutionary algorithms.

These stochastic metaheuristic algorithms are comparatively flexible,

efficient, and robust, which is advantageous when determining near-optimal solutions to large-scale problems within an acceptable time frame. The trade-off of not requiring good initial approximations is that the convergence of these algorithms to a solution is not guaranteed.

A large number of population-based metaheuristic algorithms, such as PSO and the Fireworks algorithm (FWA) [11], have been proposed. These algorithms allow for efficient exploration of the search space and have the ability to escape from local optima. As an added convenience, these algorithms do not rely on function's continuity and derivatives, making them also useful for discontinuous functions.

The potential advantages of these population-based computational intelligence algorithms make them natural choices for solving difficult problems, such as large nonlinear equation systems transformed into nonlinear optimization problems, which motivated the choice of such algorithms for this research.

In turn, nonlinear equation systems, the class of problems chosen for this research, appear in almost all simulations of physical processes [5] and are important in many branches of knowledge, including but not limited to Engineering, Economics, Chemistry, and Physics [7]. Nonlinear equation systems are also challenging and hard to solve, especially systems with a great number of equations and variables.

It is possible to transform a system of nonlinear equations into a nonlinear optimization problem by adopting the sum of the absolute value of each equation in the system as an objective function to be minimized, which allows the use of population-based metaheuristic algorithms to solve the resulting nonlinear equation systems.

Traditional iterative numerical methods, such as Newton's method, can also be used to solve systems of nonlinear equations, but such iterative methods require good initial approximations for convergence [2]. As the number of variables increases, so does the challenge of finding good approximations. It is frequent for numerical optimization algorithms to fail to converge if the initial approximations adopted are not sufficiently good.

Because of their complexity and difficulty in solving, these problems were chosen as good test problems to be used throughout this thesis.

1.3 Background

Only the most important swarm optimization and metaphorless algorithms, PSO and Jaya, will be briefly outlined here, as these algorithms and several of their variants will be addressed in more detail in the four main chapters of this thesis.

The Particle Swarm Optimization (PSO) algorithm was first proposed by Eberhart and Kennedy [3, 6]. It makes use of information communication between potential solutions, called particles. This exchange of information was inspired by the cooperative behaviour of social animals, such as flocks of birds and schools of fish. These particles iteratively move through the search space, slowly exploring it while moving towards promising regions until they converge into a solution.

The equations that update each particle's position at each iteration are as follows:

$$\boldsymbol{v}_i^{t+1} = w \cdot \boldsymbol{v}_i^t + {r_1}_i^t \cdot c_1 \cdot (\boldsymbol{pbest}_i^t - \boldsymbol{x}_i^t) + {r_2}_i^t \cdot c_2 \cdot (\boldsymbol{gbest}^t - \boldsymbol{x}_i^t), \tag{1.1}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}. \tag{1.2}$$

The equation that calculates the updated velocity is subject to three parameters: w is the inertia weight, used to reduce the weight of the previous velocity. c_1 and c_2 are the cognitive and social factor, respectively. The higher c_1, the more each particle will move towards $\boldsymbol{pbest}$, the best solution it found. The higher c_2, the more each particle will move to $\boldsymbol{gbest}$, the best solution found by all the particles in the flock. r_1 and r_2 are uniformly distributed random numbers between 0 and 1.

One of the drawbacks of PSO and other similar population-based metaheuristic algorithms is the number of parameters that affect the performance of the algorithm. The Jaya algorithm was proposed by Rao [8, 9] as a simple and efficient parameter-less algorithm that remains efficient and effective. The only parameters are the population size and the maximum number of iterations.

Assuming a $numVar$ number of decision variables, where v is the index of the variable, a p population index, and an iteration i, the proposed solution

$\boldsymbol{x}_{v,p,i}$ is the v^{th} proposed solution at iteration i for the decision variable at index v. The updated value at iteration $i+1$ is given by the following equation:

$$\boldsymbol{x}_{v,p,i}^{new} = \boldsymbol{x}_{v,p,i} + r_{1,v,i}\left(\boldsymbol{x}_{v,best,i} - |\boldsymbol{x}_{v,p,i}|\right) - r_{2,v,i}\left(\boldsymbol{x}_{v,worst,i} - |\boldsymbol{x}_{v,p,i}|\right), \quad (1.3)$$

where $r_{1,v,i}$ and $r_{2,v,i}$ are uniformly distributed random numbers between 0 and 1, and $\boldsymbol{x}_{v,best,i}$ and $\boldsymbol{x}_{v,worst,i}$ are the candidate solutions with the best and the worst fitness values, respectively.

In simple terms, at each iteration, each member of the population will move toward the best solution and away from the worst.

These two simple and efficient population-based algorithms will serve as the basis for the four studies that constitute the body of this thesis. In the following chapters, additional information about these two algorithms and some of their variants will be presented, along with the results of comparative analyzes of the performance of these algorithms in solving systems of nonlinear equations, which is commonly regarded as the most difficult problem in all of numerical mathematics (see, e.g., [4, 10]).

1.4 Thesis structure

The present thesis is comprised of four computational studies that can be divided into two parts. The first is focused on the Particle Swarm Optimization (PSO) algorithm and its variants, corresponding to Chapters 2, 3, and 4. The second part, which corresponds only to Chapter 5, explores the use of metaphorless algorithms, namely Jaya and some of its variants, which constitute a more recent class of population-based metaheuristics.

Chapter 2 introduces the standard PSO algorithm as well as a number of PSO variants and compares their performance on a set of nonlinear equation system problems.

In Chapter 3, a comparison of the results produced by PSO with a higher number of particles versus a segmentation of the search space is presented and discussed.

Chapter 4 presents a new population-based hybrid optimization algorithm derived from the hybridization of PSO and the Fireworks Algorithm (FWA).

The Jaya metaphorless algorithm and some of its variants are described in Chapter 5 and their performance when solving some difficult nonlinear equation system problems is compared.

Finally, in Chapter 6, the main findings of the research are presented, along with some suggestions for future work.

References

[1] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. Julia: A fresh approach to numerical computing. *SIAM Review 59*, 1 (2017), 65–98.

[2] Choi, H., Kim, S., and Shin, B.-C. Choice of an initial guess for Newton's method to solve nonlinear differential equations. *Computers & Mathematics with Applications 117* (2022), 69–73.

[3] Eberhart, R., and Kennedy, J. A new optimizer using particle swarm theory. In *6th International Symposium on Micro Machine and Human Science, Nagoya, Japan, 4–6 October 1995* (1995), IEEE, pp. 39–43.

[4] Karr, C., Weck, B., and Freeman, L. Solutions to systems of nonlinear equations via genetic algorithms. *Engineering Applications of Artificial Intelligence 11*, 3 (1998), 369–375.

[5] Kelley, C. *Solving Nonlinear Equations with Newton's Method*. SIAM, Philadelphia, PA, 2003.

[6] Kennedy, J., and Eberhart, R. Particle swarm optimization. In *International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995* (1995), vol. 4, IEEE, pp. 1942–1948.

[7] Pérez, R., and Lopes, V. Recent applications and numerical implementation of quasi-Newton methods for solving nonlinear systems of equations. *Numerical Algorithms 35*, 2–4 (2004), 261–285.

[8] Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations 7* (2016), 19–34.

[9] Rao, R. *Jaya: An Advanced Optimization Algorithm and its Engineering Applications*. Springer, Cham, Switzerland, 2019.

[10] Rice, J. *Numerical Methods, Software, and Analysis*, 2nd ed. Academic Press, Boston, 1993.

[11] Tan, Y., and Zhu, Y. Fireworks algorithm for optimization. In *International Conference on Swarm Intelligence* (2010), pp. 355–364.

Chapter 2

Overview and Computational Analysis of PSO Variants for Solving Systems of Nonlinear Equations

The problem of solving systems of nonlinear equations is one of great difficulty, especially as the scale of these systems grow. Traditional numerical methods rely on selecting good initial estimates for the roots and refine them iteratively, which is not a simple task, especially with large systems of nonlinear equations. Particle Swarm Optimization (PSO), in turn, is a nature-inspired meta-heuristic algorithm for finding the minimum of a function for which multiple improvements and different hybridizations have been proposed. These modifications range from dynamically choosing parameters and topologies to hybridization with other population-based optimization algorithms. All of these modifications have the goal of improving the basic algorithm, and a broader comparison between these proposals is necessary to further understand what might be the path forward in achieving better and more exact results. In this paper, after a brief overview of PSO-based algorithms for nonlinear equation systems, several PSO variants are tested on a number of problems in order to find the solution to non-trivial systems of nonlinear equations. Each variant was tested 100 times on each problem, in order to produce enough samples for a legitimate comparison.

2.1 Introduction

Solving systems of nonlinear equations is possibly the most difficult and challenging problem in all of numerical mathematics [15, 26] and there is not a general method sufficiently efficient and robust for its solution [24].

The difficulties associated with obtaining good numerical approximations to the solutions of nonlinear equation systems are amplified as the number of equations increases. However, it is not hard to find many examples of systems of nonlinear equations with less than ten algebraic or transcendental equations, from different areas of science and engineering, that are difficult to solve adequately by traditional numerical techniques.

In addition, although there are a variety of iterative methods in the extensive literature for the numerical solution of systems of nonlinear equations (see, e.g., the methods and references in [9, 16, 25]) much of them are severely limited by their domains of convergence, and the success of their application is strongly dependent on the initial approximations used. Cases of non-convergence to any of the solutions are relatively frequent in practice.

Nonlinear equations and nonlinear equation systems appear in nearly all simulations of physical processes [17]. An example of this are the physical models expressed mathematically by nonlinear partial differential equations which, when discretized for numerical solution, are transformed into large systems of nonlinear equations, many of them difficult to solve by traditional iterative methods.

In fact, solving systems of nonlinear equations is of fundamental importance in many fields, such as chemistry, economics, electronics, mechanics, robotics, medicine and different branches of engineering [1, 23, 28].

Due to the drawbacks mentioned above, nature-inspired metaheuristic and hybrid approaches, including PSO-based algorithms, have attracted increasing interest in more recent years due to their potential for solving systems of nonlinear equations and other difficult numerical problems. In the particular case of solving nonlinear equation systems, these approaches usually consist in transforming the original problem into a corresponding nonlinear optimization problem [3], and numerically approximating the solutions of this problem by using a pure metaheuristic search algorithm

or even a hybrid metaheuristic-based strategy.

When the use of exact optimization algorithms is not possible due to the fact that the problems are large and highly complex, which occurs for example when viewing the problem of solving a large-scale system of nonlinear equations as a multiobjective optimization problem, it is necessary to use stochastic optimization algorithms [30], which allow to find near-optimal solutions within a reasonable execution time.

Nature-inspired metaheuristic optimization algorithms, a large and important class of stochastic optimization techniques [14] that allow the efficient exploration of search spaces and the solving of complex problems with multiple conflicting objectives [4], include swarm intelligence (SI) based algorithms.

These algorithms are characterized by their robustness and the ability to escape from local optima, which although fundamental for any search algorithm, it is not found in traditional iterative methods of optimization. In addition, the complexities and discontinuities present in the equations that constitute a given system have little or no effect on their search performance.

This paper focuses in one of such SI based algorithms, known as Particle Swarm Optimization (PSO), for which multiple modifications have been proposed (see, e.g., [11]), some of which specifically to deal with the problem of solving systems of nonlinear equations, whose performance is here compared and reported.

Since the focus of this comparison is the use of PSO variants for solving systems of nonlinear equations, other alterations proposed to the PSO algorithm for generic optimization, such as the one presented in [8] were not implemented.

In the comparison carried out in this study, there are some other PSO variants that could not be used for different reasons. For example, Abraham et al. [2] proposed some alterations to the PSO algorithm to solve Diophantine equations. However, since the solution space for the problems used for testing is different, the results could not be fairly compared. Other two variants, both from [29] were not used since that paper does not mention some of the parameters used, and tests of the implementations could not reproduce the original results.

2.2 PSO-based algorithms for nonlinear equation systems

A brief overview of the main PSO-based algorithms for solving systems of nonlinear equations is presented below beginning with a short description of the standard PSO algorithm with inertia weight parameter, followed by a succinct characterization of the essence of the other PSO-based algorithms used in this study for solving nonlinear equation systems.

2.2.1 Standard PSO

Particle swarm optimization (PSO) was introduced by Eberhart and Kennedy [10, 18]. It is a meta-heuristic algorithm which is based on the movement of cooperative groups of animals, such as flocks of birds and schools of fish. There are a number of particles, with position $\boldsymbol{x}$, each representing a potential solution to the problem. The particles in the swarm move with velocity $\boldsymbol{v}$ through the solution space, according to the following equations (see, e.g., [5, 11]), corresponding to the standard PSO algorithm with the inertia weight parameter introduced by Shi and Eberhart [27]:

$$\boldsymbol{v}_i^{t+1} = w \cdot \boldsymbol{v}_i^t + {r_1}_i^t \cdot c_1 \cdot (\boldsymbol{pbest}_i^t - \boldsymbol{x}_i^t) + {r_2}_i^t \cdot c_2 \cdot (\boldsymbol{gbest}^t - \boldsymbol{x}_i^t), \tag{2.1}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}. \tag{2.2}$$

The algorithm is subject to three different parameters: w is the inertia weight, c_1 is called the cognitive factor, and c_2 the social factor. These parameters influence how much the movement of each particle i will tend to their best visited position up to the iteration t, $\boldsymbol{pbest}_i^t$ or to the best solution found so far, $\boldsymbol{gbest}^t$. In turn, r_1 and r_2 are vectors of uniformly distributed random numbers. The particles explore the solution space with a tendency to move towards better locations, following the particles closer to the minimum. There is no guarantee that the particles will not prematurely converge to a local minimum, and several modifications have been proposed to tackle this issue.

2.2.2 HPSO

The hybrid particle swarm optimization (HPSO) algorithm [22] was proposed as a potential improvement over the standard PSO algorithm. It hybridizes PSO with the Nelder–Mead Simplex method. The main goal of this idea was to use PSO's global search capabilities with simplex method local search.

The basic algorithm is similar to PSO but after evaluating each particle, the worst particle is replaced by a new particle using the simplex method. It has the advantage of being a simple hybridization that always improves the overall position of the particles in the swarm by adjusting the worst particle, at a low computational cost. One of the particularities of this approach is that the number of particles must be $N + 1$, where N is the number of variables of the system, so that the simplex can be calculated. Another particularity is the equation used to calculate the inertia weight,

$$w = (w_{max} - w_{min}) \cdot exp\left(\frac{t_{max} - t}{t_{max}}\right) - w_{min}. \tag{2.3}$$

Instead of using a fixed or linear inertia weight, this variant uses a weight that takes smaller steps towards the minimum as the number of iterations increases.

2.2.3 PPSO

Since the parameters of the PSO algorithm can change its performance, the Proposed Particle Swarm Optimization (PPSO) [13] chooses to bypass this choice by altering the velocity and position update equations. Using the following equations to calculate the velocity, inertia weight and position for each particle, the performance of the algorithm is not dependent on the social and cognitive factor:

$$\boldsymbol{v}_i^{t+1} = (2r_{1_i}^t - 0.5)\boldsymbol{v}_i^t + (2r_{2_i}^t - 0.5)(\boldsymbol{pbest}_i^t - \boldsymbol{x}_i^t) + (2r_{3_i}^t - 0.5)(\boldsymbol{gbest}^t - \boldsymbol{x}_i^t), \tag{2.4}$$

$$\boldsymbol{w}^{t+1} = (2r_{4_i}^t - 0.5)(\boldsymbol{gbest}^t - \boldsymbol{pbest}_i^t) + (2r_{5_i}^t - 0.5)(\boldsymbol{gbest}^t - \boldsymbol{x}_i^t), \tag{2.5}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{pbest}_i^t + (2r_{6_i}^t - 0.5)\boldsymbol{v}_i^{t+1} + (2r_{7_i}^t - 0.5)\boldsymbol{w}^{t+1}, \tag{2.6}$$

where r_1 to r_7 are uniform random numbers between 0 and 1.

One notable difference is that the position always has the *pbest* of the particle as a basis, so particles do not stray to non-promising areas of the search space. While this can be an advantage, it also limits exploration.

2.2.4 nbest PSO

The original PSO assumed a gbest topology, meaning that, for the purpose of calculating the velocity, the best point found by every particle is considered. However, this is not the only possible topology for PSO. In [10], Eberhart and Kennedy proposed an alternative topology, called lbest, where each particle only takes into account the *pbest* of k other particles, usually with $k = 2$. This has the effect of reducing the problem of premature convergence. Thus, the gbest topology is a particular case of lbest, where k is equal to the number of particles.

What the nbest PSO algorithm proposes is a dynamic topology [7], where each particle is in the same neighborhood as the k particles closest to itself. This way, even if some particles get stuck in local minima, some particles are still free to explore the search space. This can also be used to find multiple solutions to the system of nonlinear equations. This method has the disadvantage of being very computationally expensive, as for each iteration, it is necessary to calculate the distance of each particle in relation to each other and select the k closest particles.

There are multiple ways of selecting the number of particles in the neighborhood, from selecting a fixed number to using a formula to calculate the number at each iteration. One equation proposed in [6] sets a minimum and a maximum k. For each iteration t, k is calculated according to the following equation:

$$k_t = \left\lceil \frac{t_{max} - t}{t_{max}} \cdot (k_{initial} - k_{final}) + k_{final} \right\rceil. \tag{2.7}$$

2.2.5 imPSO

Improved PSO (imPSO) is a proposal for an improvement on standard PSO for its adapted inertia weight [19]. The inertia weight is calculated according

to the following equation:

$$w = a - c\,\frac{1}{b^{gbest^t}} + d\,\frac{1}{f^{bfc}+1}. \tag{2.8}$$

There is a large number of added parameters to this method. According to [19], a is a value between 0.8 and 1, b should have a value between 1 and 1.5, c a value between 0.6 and 1.2, d should be between 0.05 and 0.2, and f should have a value between 1 and 2.5. $gbest^t$ is the best value found on iteration t and bfc is the standard deviation of the position of the particles. According to the first fraction of the equation, as the $gbest$ tends towards 0 the inertia weight will also decrease. When the particles are spread out, the inertia weight will be larger, but as they converge to a single position, the inertia weight will decrease.

There is another factor associated with imPSO that will be ignored for the purpose of a fair comparison with the remaining algorithms, but must be here mentioned. The probability of getting the optimal value is calculated through multiple simulations in order to calculate the number of restarting times mb.

The algorithm is ran until the number of simulations reaches mb or a solution is found. If the number of simulations reaches mb, the output is "No results", meaning that a solution does not exist. While this is a good method to achieve reliable results in order to solve a system of nonlinear equations, for the purpose of comparing different approaches—as any algorithm will achieve a better result if it is executed multiple times and only those where it converges to the global minimum are selected—comparisons using imPSO ignore this part of the algorithm.

2.2.6 APSO–BFA

APSO–BFA is an hybridization between PSO and Bacterial Foraging Algorithm (BFA) [20]. In a similar way to PSO, BFA is a nature-inspired optimization algorithm where every member of a population represents a candidate solution in the solution space. BFA contains three main components: chemotaxis, reproduction, and elimination-dispersal. Chemotaxis is the exploration phase of the algorithm. In APSO–BFA, this

corresponds to the usual iteration of the PSO algorithm, where the velocity and position of each particle are calculated, as well as the update of the best values found. This is followed by the reproduction phase, where only the particle with the best value for the fitness function survive and duplicate. This means that the worst particles are removed every few iterations and each remaining particle is copied. Although the particles are copied, the random elements of the velocity equation guarantees that the duplicate particle will not have exactly the same velocity and position as the original one. Finally, there is the elimination-dispersal phase, where bacteria are eliminated with a small probability and replacements randomly initialized. This is the final phase, meaning it is the one that occurs the least number of times. The phases are executed in nested loops, meaning that for each reproduction phase, the chemotaxis phase will occur a fixed number of times and that the reproduction phase will occur a number of times for each elimination-dispersal phase.

PSO–BFA, unlike other PSO variants, does not keep going until a predefined maximum iteration number is reached or the minimum is found. Each phase is repeated several times, with the initial phases being nested inside the later ones. However, it would be simple to make the modification to the algorithm to include stopping criteria at the start of each chemotaxis phase. The equations to calculate velocity and position are the same as standard PSO, but each individual phase of BFA is integrated in this hybridization.

2.3 Experimental setting

The aforementioned variants, as well as the standard PSO algorithm with inertia weight were tested on a number of problems. As each variant was presented with different parameters on their respective papers, these values were followed, meaning that the only parameter in common was the number of particles used, with the exception of HPSO, since the number of particles is dependent on the number of variables of the system. This means that in terms of the raw number of particles, HPSO used much fewer particles. The number of particles for the remaining variants was 100. The number of

iterations was always 1 000, with the exception of the APSO–BFA, since it uses nested phases. The values of parameters used were the ones used in [20], meaning that the chemotaxis phase ocurred 300 times.

2.3.1 Parameters used for each PSO variant

While choosing the same parameters for each PSO variant was an option when testing the results, it was decided that using the values adopted in each variant's original article would be a fairer method to compare them, since there was the possibility that each variant performed better with a different set of parameters.

For PSO, the inertia weight was 0.7, the cognitive factor 1.8, and the social factor 1.5. For HPSO, w_{max} was 0.9 and w_{min} was 0.4. PPSO had no parameters. Nbest's inertia weight linearly decreased between 0.7 and 0.1, the cognitive factor and social factor were both 1.4, $k_{initial} = 5$, and $k_{final} = 1$. ImPSO's set of parameters were $a = 1.0$, $b = 1.8$, $c = 1.5$, $d = 0.2$ and $f = 2.0$. APSO–BFA's inertia weight linearly decreases between 0.9 and 0.4, the cognitive factor was 1.2, and the social factor was 1.8. The number of chemotaxis repetitions for each reproduction phase, Nc was 15. The number of reproduction phases for each dispersal-elimination phase, Nre was 10 and the number of dispersal-elimination phases was 2. During the dispersal-elimination phase, the probability that a particle is randomly deleted, Ped was 0.25.

2.3.2 Test problems

The test problems used on each PSO variant were the following:

Problem 1. *([28],* Interval arithmetic benchmark i1*),* $n = 10$.

$$f_1(\boldsymbol{x}) = x_1 - 0.25428722 - 0.18324757\ x_4x_3x_9$$
$$f_2(\boldsymbol{x}) = x_2 - 0.37842197 - 0.16275449\ x_1x_{10}x_6$$
$$f_3(\boldsymbol{x}) = x_3 - 0.27162577 - 0.16955071\ x_1x_2x_{10}$$
$$f_4(\boldsymbol{x}) = x_4 - 0.19807914 - 0.15585316\ x_7x_1x_6$$
$$f_5(\boldsymbol{x}) = x_5 - 0.44166728 - 0.19950920\ x_7x_6x_3$$
$$f_6(\boldsymbol{x}) = x_6 - 0.14654113 - 0.18922793\ x_8x_5x_{10}$$
$$f_7(\boldsymbol{x}) = x_7 - 0.42937161 - 0.21180486\ x_2x_5x_8$$

$f_8(\boldsymbol{x}) = x_8 - 0.07056438 - 0.17081208\ x_1 x_7 x_6$
$f_9(\boldsymbol{x}) = x_9 - 0.34504906 - 0.19612740\ x_{10} x_6 x_8$
$f_{10}(\boldsymbol{x}) = x_{10} - 0.42651102 - 0.21466544\ x_4 x_8 x_1$
$D = ([-2,2], \ldots, [-2,2])^T$

Problem 2. *([12],* Problem D1 – Modified Rosenbrock*),* $n = 12$*.*
$f_{2i-1}(\boldsymbol{x}) = \dfrac{1}{1+\exp(-x_{2i-1})} - 0.73$
$f_{2i}(\boldsymbol{x}) = 10(x_{2i} - x_{2i-1}^2),$
$i = 1, \ldots, \frac{n}{2}$
$\boldsymbol{x}^{(0)} = (-1.8, -1, \ldots, -1.8, -1)^T$
$D = ([-10,10], \ldots, [-10,10])^T$

Problem 3. *([12],* Problem D2 – Augmented Rosenbrock*),* $n = 12$*.*
$f_{4i-3}(\boldsymbol{x}) = 10(x_{4i-2} - x_{4i-3}^2)$
$f_{4i-2}(\boldsymbol{x}) = 1 - x_{4i-3}$
$f_{4i-1}(\boldsymbol{x}) = 1.25 x_{4i-1} - 0.25 x_{4i-1}^3$
$f_{4i}(\boldsymbol{x}) = x_{4i},$
$i = 1, \ldots, \frac{n}{4}$
$\boldsymbol{x}^{(0)} = (-1.2, 1, -1, 20, \ldots, -1.2, 1, -1, 20)^T$
$D = ([-10,10], \ldots, [-10,10])^T$

Problem 4. *([12],* Problem D3 – Powell badly scaled*),* $n = 12$*.*
$f_{2i-1}(\boldsymbol{x}) = 10^4 x_{2i-1} x_{2i} - 1$
$f_{2i}(\boldsymbol{x}) = \exp(-x_{2i-1}) + \exp(-x_{2i}) - 1.0001, \quad i = 1, \ldots, \frac{n}{2}$
$\boldsymbol{x}^{(0)} = (0, 100, \ldots, 0, 100)^T$
$D = ([0,100], \ldots, [0,100])^T$

Problem 5. *([12],* Problem D4 – Augmented Powell badly scaled*),* $n = 12$*.*
$f_{3i-2}(\boldsymbol{x}) = 10^4 x_{3i-2} x_{3i-1} - 1$
$f_{3i-1}(\boldsymbol{x}) = \exp(-x_{3i-2}) + \exp(-x_{3i-1}) - 1.0001$
$f_{3i}(\boldsymbol{x}) = \varphi(x_{3i}),$
$i = 1, \ldots, \frac{n}{3}$*, where:*
$$\varphi(t) = \begin{cases} \frac{t}{2} - 2, & \text{if } t \le -1 \\ \frac{1}{1998}(-1924 + 4551t + 888t^2 - 592t^3), & \text{if } t \in [-1,2] \\ \frac{t}{2} + 2, & \text{if } t \ge 2 \end{cases}$$
$\boldsymbol{x}^{(0)} = (0, 1, -4, \ldots, 0, 1, -4)^T$
$D = ([-5,5], \ldots, [-5,5])^T$

Problem 6. *([12],* Problem D5 – Tridimensional valley*),* $n = 12$.

$$f_{3i-2}(\boldsymbol{x}) = (c_2 x_{3i-2}^3 + c_1 x_{3i-1}) \exp\left(-\frac{x_{3i-2}^2}{100}\right) - 1,$$
$$f_{3i-1}(\boldsymbol{x}) = 10\,(\sin(x_{3i-2}) - x_{3i-1})$$
$$f_{3i}(\boldsymbol{x}) = 10\,(\cos(x_{3i-2}) - x_{3i}),$$
$$i = 1, \ldots, \frac{n}{3},$$
where:
$$c_1 = 1.003344481605351$$
$$c_2 = -3.344481605351171 \times 10^{-3}$$
$$\boldsymbol{x}^{(0)} = (-4, 1, 2, 1, 2, 1, 2, \ldots)^T$$
$$D = ([-10, 10], \ldots, [-10, 10])^T$$

Problem 7. *([12],* Problem D6 – Shifted and augmented trigonometric function with an Euclidean sphere*),* $n = 12$.

$$f_i(\boldsymbol{x}) = n - 1 - \sum_{j=1}^{n-1} \cos(x_j - 1) + i(1 - \cos(x_i - 1)) - \sin(x_i - 1),$$
$$i = 1, \ldots, n-1$$
$$f_n(\boldsymbol{x}) = \sum_{j=1}^{n} x_j^2 - 10000$$
$$\boldsymbol{x}^{(0)} = (0, \ldots, 0)^T$$
$$D = ([-200, 200], \ldots, [-200, 200])^T$$

Problem 8. *([12],* Problem D7 – Diagonal of three variables premultiplied by a quasi-orthogonal matrix*),* $n = 12$.

$$f_{3i-2}(\boldsymbol{x}) = 0.6x_{3i-2} + 1.6x_{3i-1}^3 - 7.2x_{3i-1}^2 + 9.6x_{3i-1} - 4.8$$
$$f_{3i-1}(\boldsymbol{x}) = 0.48x_{3i-2} - 0.72x_{3i-1}^3 + 3.24x_{3i-1}^2 - 4.32x_{3i-1} - x_{3i} + 0.2x_{3i}^3 + 2.16$$
$$f_{3i}(\boldsymbol{x}) = 1.25x_{3i} - 0.25x_{3i}^3,$$
$$i = 1, \ldots, \tfrac{n}{3}$$
$$\boldsymbol{x}^{(0)} = (50, 0.5, -1, 50, 0.5, -1, \ldots)^T$$
$$D = ([-5, 5], \ldots, [-5, 5])^T$$

Problem 9. *([12],* Problem D8 – Diagonal of three variables premultiplied by an orthogonal matrix, combined with inverse trigonometric function*),* $n = 12$.

$$f_{3i-2}(\boldsymbol{x}) = 64(x_{3i-2} + x_{3i-1} + x_{3i}) - 0.64 + 0.48 \arctan(x_{3i}) +$$
$$0.60(c_1 + c_2 x_{3i-1} + c_3 x_{3i-1}^2 + c_4 x_{3i-1}^3)$$
$$f_{3i-1}(\boldsymbol{x}) = 0.48 - 48(x_{3i-2} + x_{3i-1} + x_{3i}) + 0.36 \arctan(x_{3i}) +$$

$0.80(c_1 + c_2 x_{3i-1} + c_3 x_{3i-1}^2 + c_4 x_{3i-1}^3)$
$f_{3i}(\boldsymbol{x}) = 0.60 - 60(x_{3i-2} + x_{3i-1} + x_{3i}) + 0.80 \arctan(x_{3i})$,
$i = 1, \ldots, \frac{n}{3}$
$c_1 = 13.901020408163270000$
$c_2 = -1.4056122448979600000$
$c_3 = -2.2183673469387760000$
$c_4 = -0.27704081632653060000$
$\boldsymbol{x}^{(0)} = (10, -5.223, -1.393, 10, -5.223, -1.393, \ldots)^T$
$D = ([-200, 200], \ldots, [-200, 200])^T$

Problem 10. *([21],* 20 – Watson function*),* $n = 31$.
$$f_i(\boldsymbol{x}) = \sum_{j=2}^{m} (j-1) x_j t_i^{j-2} - \left(\sum_{j=1}^{m} x_j t_i^{j-1} \right)^2 - 1,$$
$t = \frac{i}{29}, \quad i = 1, \ldots, 29$
$f_{30}(\boldsymbol{x}) = x_1$
$f_{31}(\boldsymbol{x}) = x_2 - x_1^2 - 1$
$\boldsymbol{x}^{(0)} = (0, \ldots, 0)^T$
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 11. *([21],* 22 – Extended Powell singular function*),* $n = 12$.
$f_{4i-3}(\boldsymbol{x}) = x_{4i-3} + 10 x_{4i-2}$
$f_{4i-2}(\boldsymbol{x}) = \sqrt{5}(x_{4i-1} - x_{4i})$
$f_{4i-1}(\boldsymbol{x}) = (x_{4i-2} - 2x_{4i-1})^2$
$f_{4i}(\boldsymbol{x}) = \sqrt{10}(x_{4i-3} - x_{4i})^2$,
$i = 1, \ldots, 5$
$\boldsymbol{x}^{(0)} = (3, -1, 0, 1, \ldots, 3, -1, 0, 1)^T$
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 12. *([21],* 25 – Variable dimensioned function*),* $n = 12$.
$f_i(\boldsymbol{x}) = x_i - 1, \quad i = 1, \ldots, n$
$$f_{n+1}(\boldsymbol{x}) = \sum_{j=1}^{n} j(x_j - 1)$$
$$f_{n+2}(\boldsymbol{x}) = \left(\sum_{j=1}^{n} j(x_j - 1) \right)^2$$
$$\boldsymbol{x}^{(0)} = (1 - \frac{1}{n}, 1 - \frac{2}{n}, \ldots, 0)^T$$
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 13. *([21],* 28 – Discrete boundary value function*),* $n = 12$.

$f_1(\boldsymbol{x}) = 2x_1 - x_2 + h^2(x_1 + h + 1)^3/2$

$f_n(\boldsymbol{x}) = 2x_n - x_{n-1} + h^2(x_n + nh + 1)^3/2$

$f_i(\boldsymbol{x}) = 2x_i - x_{i-1} - x_{i+1} + h^2(x_i + t_i + 1)^3/2, \quad i = 2, \ldots, n-1,$

where $h = \frac{1}{n+1}$ *e* $t_i = ih.$

$\boldsymbol{x}^{(0)} = (\varphi_j), \quad \varphi_j = t_j(t_j - 1), \quad j = 1, \ldots, n$

$D = ([0, 5], \ldots, [0, 5])^T$

Problem 14. *([31],* Example 4.2*),* $n = 100$.

$$f_i(\boldsymbol{x}) = x_i - \frac{1}{2n}\left(\sum_{j=1}^{n} x_j^3 + i\right), \quad i = 1, \ldots, n$$

$D = ([-10, 10], \ldots, [-10, 10])^T$

2.4 Experimental results and discussion

Each of the PSO-based algorithms was run 100 times for each problem and the results were recorded. Table 1 shows the average value for each variant and for each problem. Table 2 shows the minimum value found. The minimum values for each problem are shown in bold.

The APSO–BFA variant achieved the best average results on 10 out of 14 problems, performing much better than the alternatives on problems 10, 11 and 12 in particular. However, when we look at the best fitness found, while it did achieve the best result in six problems, the standard PSO with inertia weight had the same number of minimum values found. PPSO had the best performance on problem 8 in particular. HPSO, nbest PSO and imPSO performed consistently worse than the alternatives.

It is interesting to note that the PSO with inertia weight reached the best fitness for many problems, when it did not perform particularly well on average. A possible explanation is that the PSO variants often rely on solving the problem of premature convergence of the PSO algorithm. However, when the particles are near the absolute minimum, this characteristic is a positive one, giving an advantage to the PSO algorithm.

Table 2.1: Average fitness for each variant and problem

Probl.	PSO	HPSO	PPSO	nbest PSO	imPSO	APSO-BFA
1	**0.0528181**	1.2455202	0.1184616	1.0001038	0.0908982	0.3082658
2	2.9814693	43.7396248	4.3049946	19.5092511	63.8314145	**0.0532433**
3	8.9968749	28.3936415	7.778936	9.9269534	26.5228901	**3.0211635**
4	146074.6665	18933.73947	102024.748	9636600	1748.411864	**7.5670768**
5	8.6065558	305.9715265	5.3841644	14.6271948	257.0224481	**5.1945554**
6	3.9516086	26.9582765	5.5350005	16.1843605	28.7878497	**3.4509692**
7	**12.0141506**	5436.467177	14.513556	24.5525433	156.6607548	18.4907909
8	3.1546608	14.3154373	**2.5051079**	6.8354779	10.9075963	4.7466275
9	55.2570537	9512.331841	97.1277246	9620.411493	355.9871371	**52.9168442**
10	687.0008116	4111.382325	1203.176166	1979.955606	968.4655094	**11.4743482**
11	86.4270586	1335.560895	38.544432	615.5957049	1373.850167	**0.0000172**
12	103.1314993	96.6219912	82.1260091	141.369621	317.349339	**0.00001**
13	3.6060743	0.3698597	3.723098	6.4950727	7.7535702	**0.1959979**
14	**3.7440374**	12.9426592	4.1950816	11.2938565	15.6786825	8.0368876

Table 2.2: Best fitness for each variant and problem

Probl.	PSO	HPSO	PPSO	nbest PSO	imPSO	APSO-BFA
1	**0.0001408**	0.3478782	0.0046987	0.3204995	0.0020883	0.041756
2	0.6129596	2.4106494	0.4826853	3.9993273	16.74129	**0.0063515**
3	**1.6164305**	3.3723909	2.0379767	5.3806241	9.4492288	3.0000000
4	**0.0005094**	1674.549033	3.8449885	2800000	8.386666	5.879998
5	1.5390559	5.7505782	**1.4736512**	2.1391297	20.05712	1.7346438
6	**0.4550197**	7.5608556	0.7633491	7.1926957	8.8070766	1.7496797
7	**3.0020004**	21.8360809	4.0097878	12.7831969	47.5687671	4.8217703
8	0.4606752	4.071636	**0.1132738**	3.080901	1.35673	1.246511
9	15.6940951	138.9986975	30.8773803	454.4669816	79.2954677	**10.1141256**
10	67.613611	6.5066936	158.6487704	217.3058526	4.5687597	**1.7578917**
11	9.3435249	11.0110084	0.761295	132.8155348	562.7917561	**1.41e−31**
12	33.3304672	14.8876008	22.8491931	69.2140424	197.5205035	**0**
13	1.0650655	0.2007966	1.3024803	4.8109839	3.1403194	**0.1611444**
14	**0.3559807**	6.6422099	0.6392254	4.9785431	7.3584499	1.6831947

2.5 Conclusion

Overall, the algorithms were not always successful in solving the nonlinear equation systems. One of the reasons for this is the number of variables in each system, which were considerably higher than the ones present in the variant's original articles, whose tests consisted often of systems with less than six variables.

As such, these results are a good way of comparing the variants in complex, non-trivial problems on a number of particles that was evidently not high enough to reach the solutions for each system.

The most successful PSO-based algorithm was APSO-BFA, which can be explained by being the one that changes the PSO algorithm much more drastically than the other alternatives. In stark contrast, imPSO, which only made slight alterations to the inertia weight, performed poorly, and so did HPSO, which modifies only the worst-performing particle, in addition to the modified inertia weight. The best fitness found for each variant show that the PSO algorithm, while not performing the best on average, finds the best solution at least some of the time, a result of rapid convergence if one of the particles happens to start in a promising position. This also occurs in APSO-BFA, due to the reproduction phase leading to an exploration of a promising local region.

References

[1] Abdollahi, M., Bouyer, A., and Abdollahi, D. Improved cuckoo optimization algorithm for solving systems of nonlinear equations. *The Journal of Supercomputing 72*, 3 (2016), 1246–1269.

[2] Abraham, S., Sanyal, S., and Sanglikar, M. Particle swarm optimisation based Diophantine equation solver. *International Journal of Bio-Inspired Computation 2*, 2 (2010), 100–114.

[3] Amaya, I., Cruz, J., and Correa, R. Real roots of nonlinear systems of equations through a metaheuristic algorithm. *Dyna 78*, 170 (2011), 15–23.

[4] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys 35*, 3 (2003), 268–308.

[5] Bonyadi, M., and Michalewicz, Z. Particle swarm optimization for single objective continuous space problems: A review. *Evolutionary Computation 25*, 1 (2017), 1–54.

[6] Brits, R. Niching strategies for particle swarm optimization. M.Sc. Thesis, University of Pretoria, Pretoria, South Africa, 2002.

[7] Brits, R., Engelbrecht, A., and van den Bergh, F. Solving systems of unconstrained equations using particle swarm optimization. In *IEEE International Conference on Systems, Man and Cybernetics, Yasmine Hammamet, Tunisia, 6–9 October 2002* (2002), vol. 3, 6 pp., IEEE.

[8] Clerc, M. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *1999 Congress on Evolutionary Computation – CEC99 (Cat. No. 99TH8406)* (1999), vol. 3, pp. 1951–1957.

[9] Deuflhard, P. *Newton Methods for Nonlinear Problems.* Springer-Verlag, Berlin, 2006.

[10] Eberhart, R., and Kennedy, J. A new optimizer using particle swarm theory. In *6th International Symposium on Micro Machine and Human Science, Nagoya, Japan, 4–6 October 1995* (1995), IEEE, pp. 39–43.

[11] Freitas, D., Lopes, L., and Morgado-Dias, F. Particle swarm optimisation: A historical review up to the current developments. *Entropy 22*, 3 (2020), 362.

[12] Friedlander, A., Gomes-Ruggiero, M., Kozakevich, D., Martínez, J., and Santos, S. Solving nonlinear systems of equations by means of quasi-newton methods with a nonmonotone strategy. *Optimization Methods & Software 8*, 1 (1997), 25–51.

[13] Jaberipour, M., Khorram, E., and Karimi, B. Particle swarm algorithm for solving systems of nonlinear equations. *Computers & Mathematics with Applications 62*, 2 (2011), 566–576.

[14] Jr., I. F., Yang, X.-S., Fister, I., Brest, J., and Fister, D. A brief review of nature-inspired algorithms for optimization. *Elektrotehniški Vestnik 80*, 3 (2013), 115–122.

[15] Karr, C., Weck, B., and Freeman, L. Solutions to systems of nonlinear equations via genetic algorithms. *Engineering Applications of Artificial Intelligence 11*, 3 (1998), 369–375.

[16] Kelley, C. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, Philadelphia, PA, 1995.

[17] Kelley, C. *Solving Nonlinear Equations with Newton's Method*. SIAM, Philadelphia, PA, 2003.

[18] Kennedy, J., and Eberhart, R. Particle swarm optimization. In *International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995* (1995), vol. 4, IEEE, pp. 1942–1948.

[19] Li, Y., Wei, Y., and Chu, Y. Research on solving systems of nonlinear equations based on improved PSO. *Mathematical Problems in Engineering 2015, paper 727218* (2015).

[20] Mai, X., and Li, L. Bacterial foraging algorithm based on PSO with adaptive inertia weigh for solving nonlinear equations systems. In *Advanced Materials Research*, vol. 655–657. Trans Tech Publishers, 2013, pp. 940–947.

[21] Moré, J., Garbow, B., and Hillstrom, K. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software 7*, 1 (1981), 17–41.

[22] Ouyang, A., Zhou, Y., and Luo, Q. Hybrid particle swarm optimization algorithm for solving systems of nonlinear equations. In *2009 IEEE International Conference on Granular Computing, Nanchang, China, 17–19 August 2009* (2009), IEEE, pp. 460–465.

[23] Pérez, R., and Lopes, V. Recent applications and numerical implementation of quasi-Newton methods for solving nonlinear systems of equations. *Numerical Algorithms 35*, 2–4 (2004), 261–285.

[24] Press, W., Teukolsky, S., Vetterling, W., and Flannery, P. *Numerical Recipes in C++: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, New York, 2002.

[25] Rheinboldt, W. *Methods for Solving Systems of Nonlinear Equations*, 2nd ed. SIAM, Philadelphia, PA, 1998.

[26] Rice, J. *Numerical Methods, Software, and Analysis*, 2nd ed. Academic Press, Boston, 1993.

[27] Shi, Y., and Eberhart, R. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360), 1998* (1998), IEEE, pp. 69–73.

[28] van Hentenryck, P., McAllester, D., and Kapur, D. Solving polynomial systems using a branch and prune approach. *SIAM Journal of Numerical Analysis 34*, 2 (1997), 797–827.

[29] Wang, Q., Zeng, J., and Jie, J. Modified particle swarm optimization for solving systems of equations. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques* (2007), Springer, pp. 361–369.

[30] Weise, T., Zapf, M., Chiong, R., and Nebro-Urbaneja, A. Why is optimization difficult? In *Nature-Inspired Algorithms for Optimisation* (2009), R. Chiong, Ed., Springer, pp. 1–50.

[31] Yamamura, K., Kawata, H., and Tokue, A. Interval solution of nonlinear equations using linear programming. *BIT Numerical Mathematics 38*, 1 (1998), 186–199.

Chapter 3

PSO Performance for Solving Nonlinear Systems of Equations: Comparing Segmentation of Search Space and Increase of Number of Particles

Metaheuristic algorithms have been used for different optimization problems and many modifications and hybridizations of these algorithms have been proposed. One such algorithm, Particle Swarm Optimization (PSO), has been proposed and modified for many distinct problems. Solving systems of nonlinear equations is one of its many applications, but as these systems grow, the effectiveness of PSO and PSO-based algorithms decrease. As such, there need to be modifications that impact the performance of the algorithm, such as increasing the number of particles or the number of iterations. However, there are problems where the combined use of both of these strategies does not solve all the drawbacks associated with the use of these algorithms, so a possibility would be to reduce the search space of the problems considered. In this article, the effect of the search space segmentation for solving nonlinear systems of equations using PSO is explored, and an experimental comparison is made between a simple segmentation of the search space to an increase of the number of particles.

3.1 Introduction

Finding accurate numerical approximations to the solutions of problems involving large systems of nonlinear equations remains a challenging problem [4], not only due to the inherent complexity of obtaining good initial guesses to the solutions so that the algorithm used can converge, but also because these problems can reach considerable proportions, which makes them even more difficult to solve. Nonlinear equation systems are present in most simulations of physical processes, which makes them relevant in many areas, including e.g. Physics, Chemistry and different Engineering fields.

An alternative to traditional iterative numerical algorithms are metaheuristic algorithms, which trade high accuracy for robustness and good estimates on a reasonable time. One of these algorithms is the so-called Particle Swarm Optimization (PSO) algorithm, introduced by Eberhart and Kennedy [1] and inspired on the movement of cooperative groups of birds, fishes and other social animals.

In the PSO algorithm, a fixed number of potential or candidate solutions, called particles, move around the search space, trying to find an adequate solution to the problem considered. While larger problems need a bigger number of particles to find good approximations to the solution, adding particles also adds to the computational cost of finding the solution, so blindly adding a large number of particles is not a computationally efficient approach.

A proposed alternative is to reduce the search space into more manageable space sections, and run the algorithm with a smaller number of particles on each smaller section.

The two approaches mentioned above (i.e., increase of number of particles and segmentation of search space) are compared on a set of large scale test problems arising from the literature on systems of nonlinear equations and the results are described and analysed in the following sections.

3.2 Particle swarm optimization

Particle Swarm Optimization (PSO), as mentioned above, is a metaheuristic algorithm inspired by the movement of cooperative groups of animals, such as schools of fish and flocks of birds. A number of particles is randomly spread

throughout the search space, where each particles position $\boldsymbol{x}$ is a potential solution to the problem, and information is shared between them on the best positions. Then, the particles move towards the more promising regions with velocity $\boldsymbol{v}$, hopefully converging to a solution of the problem under consideration. Each particle knows the best position it has visited, denoted as ***pbest***, as well as the best candidate solution found by all particles in the swarm, called ***gbest***.

At each iteration, the velocity and position of each particle i in the swarm is updated as follows (see, e.g., [2]):

$$v_i^{t+1} = w \cdot v_i^t + r_{1_i}^t \cdot c_1 \cdot (pbest_i^t - x_i^t) + r_{2_i}^t \cdot c_2 \cdot (gbest^t - x_i^t) \tag{3.1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{3.2}$$

where c_1 and c_2 are called the cognitive factor and social factor, respectively, which determine the attraction of each particle towards its own best position, ***pbest***, and the global best position, ***gbest***, while r_1 and r_2 are uniformly distributed random numbers between 0 and 1. Therefore, in order to guarantee that at each iteration a particle does not move further away than it was before, but is also able to explore the space adequately, c_1 and c_2 should have values between 1 and 2. The factor w is the inertia weight, introduced by Shi and Eberhart [12] in order to limit the effect of the velocity of the particle. This modification of the original algorithm is now known as the standard PSO.

The standard PSO (SPSO) algorithm can be used for general optimization and several other modifications have been made to improve its performance, from modifications to the fundamental equations that determine the velocity and position of the particles or different swarm topologies, to hybridization with other optimization algorithms, both traditional and metaheuristic [6, 9].

3.3 Evolution of search space

There are a few parameters that have a predictable effect on the performance of the PSO algorithms. While many attempts have been made to select and adjust the parameters in the velocity and position equations, ultimately, these

are often problem dependent. On the other hand, adding more particles, more iterations always improves the performance of the algorithms, at a computational cost.

The time complexity of PSO is $\mathcal{O}(n \cdot i)$, for a number of particles n and a maximum number of iterations i. Both the number of particles and the maximum number of iterations linearly increase the computation time. However, these are not equivalent. After the particles converge, they will not further explore the search space, so adding more iterations does not have a significant effect. However, adding more particles increases the probability that at least one particle will either be initialized or pass through a good position while the algorithm is running.

The problem function also has some parameters that affect the performance of the algorithm, by making the correct solution harder to find. In a nonlinear system of equations, the higher the number of variables, the more complex and more difficult to solve the system is.

Some research has been done on the dimensionality reduction for optimization using PSO, e.g. in feature selection for classification problems [5] and for the distribution system reconfiguration problem [13]. However, these solutions are not adequate for solving systems of nonlinear equations. These solutions involve alternating variables, which is not possible for solving nonlinear systems of equations, since the system would be underdetermined.

The solution cannot be found by solving for different variables, one at a time, so simplifying the problem in this way is not possible. The search space of the problem is the region where the solution resides. The larger this space, the less likely it is that a particle will find the optimal region of the search space.

Some tests were done on known optimization problems, in order to evaluate the effect of an increase of the search space. The optimization problems chosen are indicated in Table 3.1. For each of these optimization problems, PSO was run with the following parameters: $w : 0.7$; $c_1 : 1.5$; $c_2 : 1.8$.

The algorithm was run 50 times for the following combination of parameters:

number of variables: 1 to 10 with step 1;
number of iterations: 100 to 400 with step 100;
number of particles: 30 to 100 with step 10;
search space: 10 to 100, with step 10.

Table 3.1: Optimization problems

Sphere	$\sum x_i^2$
Rastigrin	$10d + \sum x_i^2 - 10\cos(2\pi x_i)$
Schwefel	$\sum x_i \sin(\sqrt{\lvert x_i \rvert})$
Salomon	$-\cos 2\pi \sum x_i^2 + 0.1 \sum x_i^2 + 1$
Griewank	$\sum \frac{x_i^2}{4000} - \prod \cos \frac{x_i}{\sqrt{i}} + 1$
Rosenbrock	$\sum 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$

The results from the simulation, when averaged for all the parameters except for the search space and problem, were plotted:

The data in Figure 3.1 shows that as the search space increases, the worse the average value found by PSO. It seems reasonable to assume that reducing the search space will increase the probability of finding the solution to a large nonlinear system of equations.

3.4 Segmented PSO

If the search space is separated into two equal sides for each variable, then the number of spaces to be searched is 2^n, where n is the number of variables in the nonlinear system of equations.

The space could be separated into more sections per variable, but since the number of sections grows exponentially with the number of variables, the cost of increasing the number of separations per variable is very high.

As can be seen from Figure 3.2, for two variables the search space is segmented into four sections, and for three variables into eight sections. For very large real-world problems where the solution needs to be found only once, it might be more efficient segment the space, even if the time complexity if very large, rather than using a very large number of particles. There is also the advantage that running the algorithm for each section can easily be done in a distributed manner.

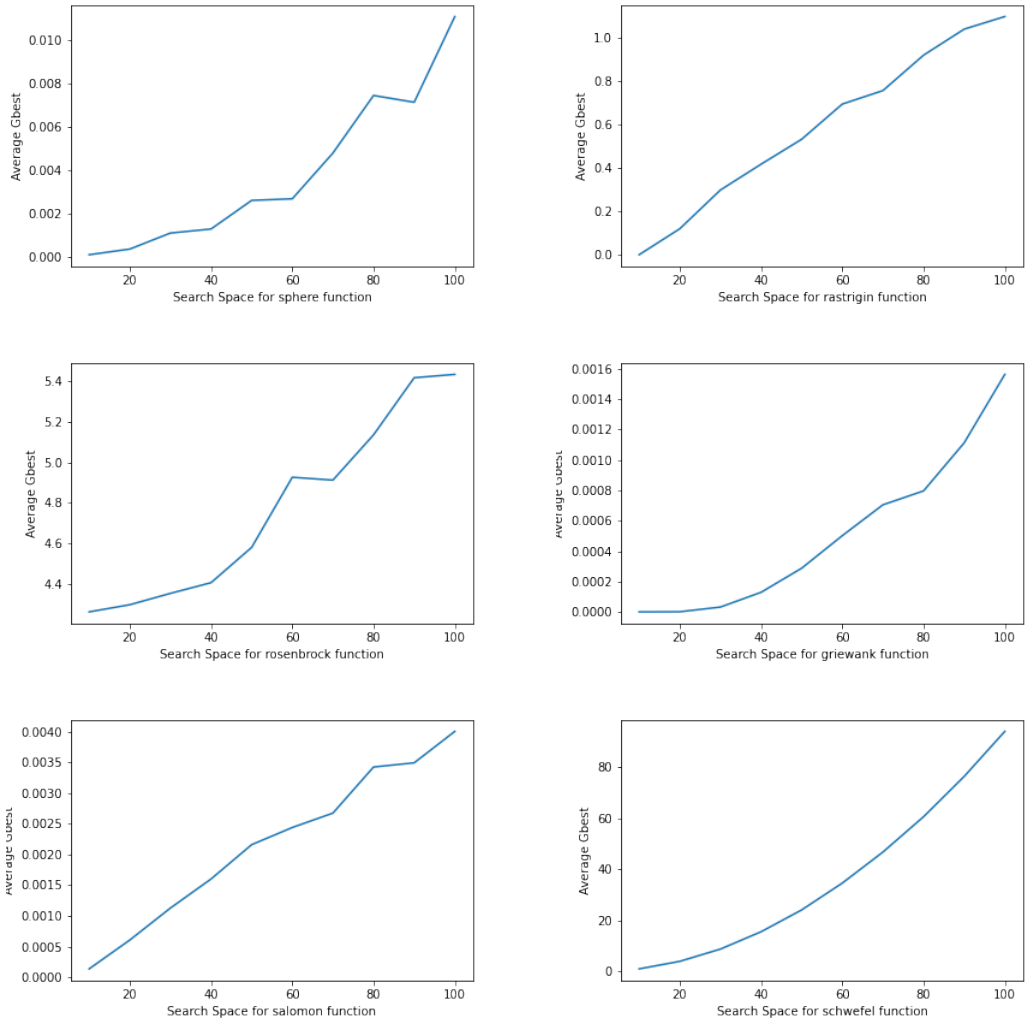

Figure 3.1: Effect of increasing the search space

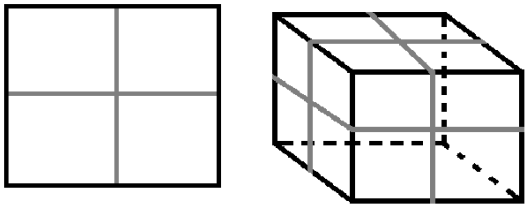

Figure 3.2: Sections of search space for two and three variables

Running PSO in this segmented way was done and compared with running PSO with a large number of particles. The parameter values for PSO were the same used in Section 3.3, and 1,000 iterations were performed.

With regard to the number of particles, each segment was run on 50 particles and 400 iterations were performed. Normally, these parameters would not result in convergence for large problems, but only a small portion of the solution space is being searched. In the experiment with a large number of particles, the number of particles was 2^n, which for the problems considered resulted in a very large number of particles that escalates with the number of variables in the same proportion as the number of segments.

Overall, in terms of raw evaluations of the objective function, in the segmented version they were evaluated $400 \times 50 \times 2^n$ times. So, for most of the problems, with an n of 12, they were evaluated 81,920,000 times. In comparison with SPSO with an increased number of particles, the evaluations were $2^n \times 1{,}000$, which for an n of 12 is 4,096,000 evaluations.

In unimodal problems, the number of particles does not need to be so large, provided the number of iterations is enough for convergence. For multimodal problems, the number of particles necessary for convergence is higher, thus requiring a much larger number of particles [11].

3.5 Test problems

The following nonlinear systems of equations were chosen as test problems for the comparison performed in this study:

Problem 1. *([14],* Interval arithmetic benchmark i1*),* $n = 10$.

$$f_1(\boldsymbol{x}) = x_1 - 0.25428722 - 0.18324757\ x_4x_3x_9$$
$$f_2(\boldsymbol{x}) = x_2 - 0.37842197 - 0.16275449\ x_1x_{10}x_6$$
$$f_3(\boldsymbol{x}) = x_3 - 0.27162577 - 0.16955071\ x_1x_2x_{10}$$
$$f_4(\boldsymbol{x}) = x_4 - 0.19807914 - 0.15585316\ x_7x_1x_6$$
$$f_5(\boldsymbol{x}) = x_5 - 0.44166728 - 0.19950920\ x_7x_6x_3$$
$$f_6(\boldsymbol{x}) = x_6 - 0.14654113 - 0.18922793\ x_8x_5x_{10}$$
$$f_7(\boldsymbol{x}) = x_7 - 0.42937161 - 0.21180486\ x_2x_5x_8$$
$$f_8(\boldsymbol{x}) = x_8 - 0.07056438 - 0.17081208\ x_1x_7x_6$$
$$f_9(\boldsymbol{x}) = x_9 - 0.34504906 - 0.19612740\ x_{10}x_6x_8$$

$f_{10}(\boldsymbol{x}) = x_{10} - 0.42651102 - 0.21466544\, x_4 x_8 x_1$
$D = ([-2,2],\ldots,[-2,2])^T$

Problem 2. *([3],* Problem D1 – Modified Rosenbrock*),* $n = 12$.
$f_{2i-1}(\boldsymbol{x}) = \dfrac{1}{1+\exp(-x_{2i-1})} - 0.73$
$f_{2i}(\boldsymbol{x}) = 10(x_{2i} - x_{2i-1}^2)$,
$i = 1,\ldots,\frac{n}{2}$
$\boldsymbol{x}^{(0)} = (-1.8,-1,\ldots,-1.8,-1)^T$
$D = ([-10,10],\ldots,[-10,10])^T$

Problem 3. *([3],* Problem D2 – Augmented Rosenbrock*),* $n = 12$.
$f_{4i-3}(\boldsymbol{x}) = 10(x_{4i-2} - x_{4i-3}^2)$
$f_{4i-2}(\boldsymbol{x}) = 1 - x_{4i-3}$
$f_{4i-1}(\boldsymbol{x}) = 1.25x_{4i-1} - 0.25x_{4i-1}^3$
$f_{4i}(\boldsymbol{x}) = x_{4i}$,
$i = 1,\ldots,\frac{n}{4}$
$\boldsymbol{x}^{(0)} = (-1.2,1,-1,20,\ldots,-1.2,1,-1,20)^T$
$D = ([-10,10],\ldots,[-10,10])^T$

Problem 4. *([3],* Problem D3 – Powell badly scaled*),* $n = 12$.
$f_{2i-1}(\boldsymbol{x}) = 10^4 x_{2i-1}x_{2i} - 1$
$f_{2i}(\boldsymbol{x}) = \exp(-x_{2i-1}) + \exp(-x_{2i}) - 1.0001, \quad i = 1,\ldots,\frac{n}{2}$
$\boldsymbol{x}^{(0)} = (0,100,\ldots,0,100)^T$
$D = ([0,100],\ldots,[0,100])^T$

Problem 5. *([3],* Problem D4 – Augmented Powell badly scaled*),* $n = 12$.
$f_{3i-2}(\boldsymbol{x}) = 10^4 x_{3i-2}x_{3i-1} - 1$
$f_{3i-1}(\boldsymbol{x}) = \exp(-x_{3i-2}) + \exp(-x_{3i-1}) - 1.0001$
$f_{3i}(\boldsymbol{x}) = \varphi(x_{3i})$,
$i = 1,\ldots,\frac{n}{3}$, *where:*
$$\varphi(t) = \begin{cases} \frac{t}{2} - 2, & \text{if } t \le -1 \\ \frac{1}{1998}(-1924 + 4551t + 888t^2 - 592t^3), & \text{if } t \in [-1,2] \\ \frac{t}{2} + 2, & \text{if } t \ge 2 \end{cases}$$
$\boldsymbol{x}^{(0)} = (0,1,-4,\ldots,0,1,-4)^T$
$D = ([-5,5],\ldots,[-5,5])^T$

Problem 6. *([3],* Problem D5 – Tridimensional valley*),* $n = 12$.

$$f_{3i-2}(\boldsymbol{x}) = (c_2 x_{3i-2}^3 + c_1 x_{3i-1}) \exp\left(-\frac{x_{3i-2}^2}{100}\right) - 1,$$
$$f_{3i-1}(\boldsymbol{x}) = 10\,(\sin(x_{3i-2}) - x_{3i-1})$$
$$f_{3i}(\boldsymbol{x}) = 10\,(\cos(x_{3i-2}) - x_{3i}),$$
$$i = 1, \ldots, \frac{n}{3},$$
where:
$$c_1 = 1.003344481605351$$
$$c_2 = -3.344481605351171 \times 10^{-3}$$
$$\boldsymbol{x}^{(0)} = (-4, 1, 2, 1, 2, 1, 2, \ldots)^T$$
$$D = ([-10, 10], \ldots, [-10, 10])^T$$

Problem 7. *([3],* Problem D6 – Shifted and augmented trigonometric function with an Euclidean sphere*),* $n = 12$.

$$f_i(\boldsymbol{x}) = n - 1 - \sum_{j=1}^{n-1} \cos(x_j - 1) + i(1 - \cos(x_i - 1)) - \sin(x_i - 1),$$
$$i = 1, \ldots, n-1$$
$$f_n(\boldsymbol{x}) = \sum_{j=1}^{n} x_j^2 - 10000$$
$$\boldsymbol{x}^{(0)} = (0, \ldots, 0)^T$$
$$D = ([-200, 200], \ldots, [-200, 200])^T$$

Problem 8. *([3],* Problem D7 – Diagonal of three variables premultiplied by a quasi-orthogonal matrix*),* $n = 12$.

$$f_{3i-2}(\boldsymbol{x}) = 0.6x_{3i-2} + 1.6x_{3i-1}^3 - 7.2x_{3i-1}^2 + 9.6x_{3i-1} - 4.8$$
$$f_{3i-1}(\boldsymbol{x}) = 0.48x_{3i-2} - 0.72x_{3i-1}^3 + 3.24x_{3i-1}^2 - 4.32x_{3i-1} - x_{3i} + 0.2x_{3i}^3 + 2.16$$
$$f_{3i}(\boldsymbol{x}) = 1.25x_{3i} - 0.25x_{3i}^3,$$
$$i = 1, \ldots, \tfrac{n}{3}$$
$$\boldsymbol{x}^{(0)} = (50, 0.5, -1, 50, 0.5, -1, \ldots)^T$$
$$D = ([-5, 5], \ldots, [-5, 5])^T$$

Problem 9. *([3],* Problem D8 – Diagonal of three variables premultiplied by an orthogonal matrix, combined with inverse trigonometric function*),* $n = 12$.

$$f_{3i-2}(\boldsymbol{x}) = 64(x_{3i-2} + x_{3i-1} + x_{3i}) - 0.64 + 0.48 \arctan(x_{3i}) +$$
$$0.60(c_1 + c_2 x_{3i-1} + c_3 x_{3i-1}^2 + c_4 x_{3i-1}^3)$$
$$f_{3i-1}(\boldsymbol{x}) = 0.48 - 48(x_{3i-2} + x_{3i-1} + x_{3i}) + 0.36 \arctan(x_{3i}) +$$

$0.80(c_1 + c_2 x_{3i-1} + c_3 x_{3i-1}^2 + c_4 x_{3i-1}^3)$
$f_{3i}(\boldsymbol{x}) = 0.60 - 60(x_{3i-2} + x_{3i-1} + x_{3i}) + 0.80 \arctan(x_{3i}),$
$i = 1, \ldots, \frac{n}{3}$
$c_1 = 13.901020408163270000$
$c_2 = -1.4056122448979600000$
$c_3 = -2.2183673469387760000$
$c_4 = -0.27704081632653060000$
$\boldsymbol{x}^{(0)} = (10, -5.223, -1.393, 10, -5.223, -1.393, \ldots)^T$
$D = ([-200, 200], \ldots, [-200, 200])^T$

Problem 10. *([10],* 22 – Extended Powell singular function*),* $n = 12$.
$f_{4i-3}(\boldsymbol{x}) = x_{4i-3} + 10x_{4i-2}$
$f_{4i-2}(\boldsymbol{x}) = \sqrt{5}(x_{4i-1} - x_{4i})$
$f_{4i-1}(\boldsymbol{x}) = (x_{4i-2} - 2x_{4i-1})^2$
$f_{4i}(\boldsymbol{x}) = \sqrt{10}(x_{4i-3} - x_{4i})^2,$
$i = 1, \ldots, 5$
$\boldsymbol{x}^{(0)} = (3, -1, 0, 1, \ldots, 3, -1, 0, 1)^T$
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 11. *([10],* 25 – Variable dimensioned function*),* $n = 12$.
$f_i(\boldsymbol{x}) = x_i - 1, \quad i = 1, \ldots, n$
$$f_{n+1}(\boldsymbol{x}) = \sum_{j=1}^{n} j(x_j - 1)$$
$$f_{n+2}(\boldsymbol{x}) = \left(\sum_{j=1}^{n} j(x_j - 1) \right)^2$$
$\boldsymbol{x}^{(0)} = (1 - \frac{1}{n}, 1 - \frac{2}{n}, \ldots, 0)^T$
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 12. *([10],* 28 – Discrete boundary value function*),* $n = 12$.
$f_1(\boldsymbol{x}) = 2x_1 - x_2 + h^2(x_1 + h + 1)^3/2$
$f_n(\boldsymbol{x}) = 2x_n - x_{n-1} + h^2(x_n + nh + 1)^3/2$
$f_i(\boldsymbol{x}) = 2x_i - x_{i-1} - x_{i+1} + h^2(x_i + t_i + 1)^3/2, \quad i = 2, \ldots, n-1,$
where $h = \frac{1}{n+1}$ *e* $t_i = ih.$
$\boldsymbol{x}^{(0)} = (\varphi_j), \quad \varphi_j = t_j(t_j - 1), \quad j = 1, \ldots, n$
$D = ([0, 5], \ldots, [0, 5])^T$

Problem 13. *([15],* Example 4.2*),* $n = 12$.

$$f_i(\boldsymbol{x}) = x_i - \frac{1}{2n}\left(\sum_{j=1}^{n} x_j^3 + i\right), \quad i = 1, \ldots, n$$

$$D = ([-10, 10], \ldots, [-10, 10])^T$$

3.6 Results obtained

The problem functions were tested with two segments. The average and minimum values found are presented in Table 3.2.

From the results, we can see that, especially for the problems where PSO has a tendency to get stuck in a local minimum, such as the Problems 2 and 3, the segmented alternative does not get stuck, but at the cost of not being able to explore much further, due to the smaller number of particles and iterations.

The premature convergence of PSO is widely acknowledged. Therefore, it is relevant to understand whether a modification to suppress this convergence would yield better results. For this, the same test problems were run, on the same conditions, with the exception of the swarm topology. Instead of using the gbest topology, where every particle communicates with every other particle, the von Neumann topology was chosen. The results obtained are presented in Table 3.3.

In the comparison of topologies for PSO, the von Neumann with rank 4 has often been found to be better performing than the alternatives [8]. In the von Neumann topology, each particle does not communicate with every other particle, but only with n particles (see Figure 3.3), with n being the rank of the topology. So, in this experiment, each particle was connected with four other particles, which delays the convergence of the algorithm, allowing for better exploration of the search space of the problems.

Table 3.2: Results for PSO with Gbest topology

Problem	Average		Minimum	
	SPSO	Segmented	SPSO	Segmented
Problem 1	1.037663071	0.698486804	0.421814871	0.408340722
Problem 2	1.62	1.608352444	1.62	1.360405712
Problem 3	3	2.903333419	3	2.373871397
Problem 4	6.637033787	6.326612765	6.282627128	6.112016672
Problem 5	5.192528753	4.62293342	4.477480166	3.882171641
Problem 6	15.53367649	10.14796265	10.07173459	7.302318603
Problem 7	17.75618911	10.94764561	12.13551478	6.732026598
Problem 8	6.405438188	4.674562396	4.725153885	3.249000098
Problem 9	77.85882864	72.15770647	72.40134433	47.19050689
Problem 10	9.5408E-15	1.86324E-14	3.38E-15	4.55E-15
Problem 11	4.517171951	3.027601273	3.111794099	1.820649757
Problem 12	0.190782092	0.175340087	0.159467829	0.163692495
Problem 13	9.202443211	6.399087192	6.05988822	4.466427714

Table 3.3: Results for segmented PSO with von Neumann topology

Problem	Average		Minimum	
	SPSO	Segmented	SPSO	Segmented
Problem 1	1.43115E-16	0.000440838	1.43115E-16	5.74899E-05
Problem 2	2.37131E-07	0.841533489	1.66395E-13	0.484539158
Problem 3	1.1831E-08	0.985771992	3.81643E-14	0.420933694
Problem 4	0.000549438	0.000611278	0.000479778	0.000485759
Problem 5	0.038483924	0.024097009	0.002670834	0.000711316
Problem 6	0.077170719	0.225986715	0.059364707	0.078127327
Problem 7	4.273857164	1.420858728	0.684416557	0.55435223
Problem 8	3.11286E-07	0.033136903	8.40785E-08	0.001336032
Problem 9	1.263340747	4.164705179	4.77684E-08	2.778112461
Problem 10	6.69532E-18	4.017634796	1.91271E-18	0.176367453
Problem 11	7.42295E-15	8.570062293	3.55271E-15	2.720638905
Problem 12	1.48766E-15	0.223516272	3.60822E-16	0.048649465
Problem 13	2.8702E-15	0.624939653	9.57567E-16	0.084652142

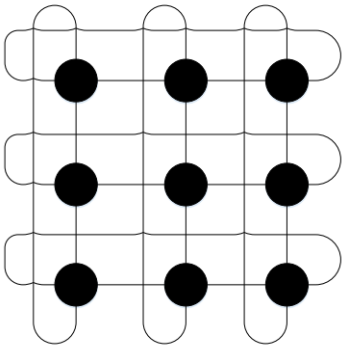

Figure 3.3: Diagram of von Neumann topology with rank 4 (extracted from [7])

3.7 Conclusion

The effects of adding many particles and of reducing the search space, and iteratively running PSO, has been compared on a set of large nonlinear systems of equations. While segmenting the search space does prevent PSO from suffering from premature converge to local minima, the computational cost of this operation is very large, and increases exponentially with the number of variables in the system. A better alternative is to increase the number of particles with a topology that makes PSO more resistant to premature convergence.

The benefit of this decision comes not only from better results, but also from a lower computational cost, because the time complexity of PSO increases linearly with the number of particles used.

References

[1] Eberhart, R., and Kennedy, J. A new optimizer using particle swarm theory. In *6th International Symposium on Micro Machine and Human Science, Nagoya, Japan, 4–6 October 1995* (1995), IEEE, pp. 39–43.

[2] Freitas, D., Lopes, L., and Morgado-Dias, F. Particle swarm optimisation: A historical review up to the current developments.

Entropy 22, 3 (2020), 362.

[3] Friedlander, A., Gomes-Ruggiero, M., Kozakevich, D., Martínez, J., and Santos, S. Solving nonlinear systems of equations by means of quasi-Newton methods with a nonmonotone strategy. *Optimization Methods & Software 8*, 1 (1997), 25–51.

[4] Gong, W., Liao, Z., Mi, X., Wang, L., and Guo, Y. Nonlinear equations solving with intelligent optimization algorithms: A survey. *Complex System Modeling and Simulation 1*, 1 (2021), 15–32.

[5] Li, A.-D., Xue, B., and Zhang, M. Improved binary particle swarm optimization for feature selection with new initialization and search space reduction strategies. *Applied Soft Computing 106* (2021), 107302.

[6] Li, Y., Wei, Y., and Chu, Y. Research on solving systems of nonlinear equations based on improved PSO. *Mathematical Problems in Engineering 2015* (2015), 727218.

[7] Lima, A., Medeiros, Y., Silva, L., and Araújo, W. Estudo comparativo sobre a convergência e o custo computacional das estruturas topológicas aplicadas à otimização por enxame de partículas (PSO). *Revista Científica Semana Acadêmica 1* (2016), 1–18.

[8] Liu, Q., Wei, W., Yuan, H., Zhan, Z.-H., and Li, Y. Topology selection for particle swarm optimization. *Information Sciences 363* (2016), 154–173.

[9] Mai, X., and Li, L. Bacterial foraging algorithm based on PSO with adaptive inertia weigh for solving nonlinear equations systems. In *Advanced Materials Research*, vol. 655–657. Trans Tech Publishers, 2013, pp. 940–947.

[10] Moré, J., Garbow, B., and Hillstrom, K. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software 7*, 1 (1981), 17–41.

[11] Piotrowski, A. P., Napiorkowski, J. J., and Piotrowska, A. E. Population size in particle swarm optimization. *Swarm and Evolutionary Computation 58* (2020), 100718.

[12] Shi, Y., and Eberhart, R. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360), 1998* (1998), IEEE, pp. 69–73.

[13] Silva, L. I., Belati, E. A., Gerez, C., and Silva Junior, I. C. Reduced search space combined with particle swarm optimization for distribution system reconfiguration. *Electrical Engineering 103*, 2 (2021), 1127–1139.

[14] van Hentenryck, P., McAllester, D., and Kapur, D. Solving polynomial systems using a branch and prune approach. *SIAM Journal of Numerical Analysis 34*, 2 (1997), 797–827.

[15] Yamamura, K., Kawata, H., and Tokue, A. Interval solution of nonlinear equations using linear programming. *BIT Numerical Mathematics 38*, 1 (1998), 186–199.

Chapter 4

PSO–FWA: A New Hybrid Algorithm for Solving Nonlinear Equation Systems

Nature-inspired optimization algorithms have been proposed for solving hard optimization problems, including the optimization-based solution of difficult systems of nonlinear equations. While there is no perfect optimization algorithm, the hybridization of such metaheuristic optimization algorithms has produced positive results by enhancing their capabilities and reducing their weaknesses. This paper presents a novel hybridization of Particle Swarm Optimization and the Fireworks Algorithm for solving nonlinear equation systems. The experimental results obtained indicate that the proposed hybrid algorithm outperforms both Particle Swarm Optimization and the Fireworks Algorithm, as well as a previously developed hybridization of these algorithms.

4.1 Introduction

Nonlinear equation systems are recognized as challenging problems to solve using traditional iterative numerical methods, whose effectiveness largely depends on the characteristics of the problem considered and the quality of the initial approximations taken, with non-convergence occurring relatively frequently.

In contrast, the use of population-based metaheuristic algorithms, such as Particle Swarm Optimization (PSO) and the Fireworks Algorithm (FWA), for solving this important class of problems, which have an extensive importance in fields such as chemistry, physics, engineering, and economics, has the advantage of being problem-independent, derivative-free, and not dependent on good initial estimates. However, there is no guarantee of convergence with these population-based stochastic algorithms, and each of them has different advantages and drawbacks, which motivates their hybridization to capitalize on their respective strengths and mitigate their weaknesses.

With this in mind, this article proposes a novel hybrid metaheuristic algorithm for solving systems of nonlinear equations that is the result of the combination of Particle Swarm Optimization and the Fireworks Algorithm.

Population-based approaches, such as the hybrid algorithm proposed in this paper, can solve a system of n nonlinear equations by converting it into an n-dimensional nonlinear optimization problem by minimizing the sum of squares of the residuals, $F(\mathbf{x}) = \sum[f_i(x_1, \ldots, x_n)]^2$, or the sum of absolute values of the residuals, $F(\mathbf{x}) = \sum |f_i(x_1, \ldots, x_n)|$. This optimization-based approach for solving nonlinear equation systems was adopted, and the performance of the proposed hybrid algorithm is here evaluated.

The structure of this paper is as follows. The next section describes the PSO and FWA algorithms, which served as basis for the proposed algorithm. Section 4.3 describes briefly some simple ways to combine PSO and FWA, as well as a previous hybridization of these two algorithms. Section 4.4 presents the proposed algorithm, and the experimental setup and test problems used in this study are described in Section 4.5. The main results obtained are presented and discussed in Section 4.6. Finally, the conclusion is given in Section 4.7.

4.2 Background

4.2.1 Particle swarm optimization

Particle Swarm Optimization (PSO), proposed by Eberhart and Kennedy [3], is a population-based stochastic algorithm, based on swarm intelligence, that draws inspiration from the cooperative behavior of groups of animals when

searching for food. In PSO, each particle represents a candidate solution to the problem, and "remembers" the best solution it found, as well as the one by its group. At each iteration, the particles update their velocity and position as follows:

$$\mathbf{v}_i^{t+1} = w \cdot \mathbf{v}_i^t + {r_1}_i^t \cdot c_1 \cdot (\mathbf{pbest}_i^t - \mathbf{x}_i^t) + {r_2}_i^t \cdot c_2 \cdot (\mathbf{gbest}^t - \mathbf{x}_i^t), \tag{4.1}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \tag{4.2}$$

where $\mathbf{v}$ is the velocity of the particle, $\mathbf{x}$ is its position, w is the inertia weight introduced by Shi and Eberhart [12], r_1 and r_2 are random numbers uniformly distributed in $[0, 1]$, and c_1 and c_2 are the cognitive and social factors, respectively. These affect how much the particle is drawn to their own previous experience, compared to the one found by their group, with **pbest** and **gbest** representing the particle's best position and the group's best position, respectively.

This sharing of information is at the core of the flexibility of PSO. It allows not only changes to the algorithm itself, but also different topologies for the group of particles that communicate between themselves. The initial topology is the gbest topology, where all particles communicate with each other. Premature convergence was observed as a result of this [8].

The pseudocode of the standard PSO is given in Algorithm 1.

Other topologies that reduce the degree of communication have been proposed. One of these topologies is the von Neumann topology [6], where every particle is connected with a small fixed number of particles, typically four. This topology was found to improve the performance of the algorithm by allowing a more efficient search space exploration [8].

4.2.2 Fireworks optimization

The Fireworks Algorithm (FWA) [13] is a population-based swarm intelligence algorithm inspired by the fireworks explosion, as each potential solution "explodes", creating a new sub-swarm. The amplitude of such explosion is dependent on the performance of the fireworks, where less-optimal solutions spread further apart, allowing for greater global

Algorithm 1 Pseudocode for the standard PSO algorithm

```
Initialize a swarm of N particles randomly;
while t < MaxIter and not terminate do
    for each particle position x_i in the swarm do
        Evaluate f(x_i)
        if f(x_i) < f(pbest) then
            Update the particle pbest
            if f(x_i) < f(gbest) then
                Update the particle gbest
            end if
        end if
    end for
    for i from 1 to N do
        Calculate v_i^{t+1} using Eq. 4.1;
        Calculate x_i^{t+1} using Eq. 4.2;
    end for
end while
Return the best particle found;
```

exploration, whereas better performing fireworks spread much less, allowing for better local exploitation.

The amplitude of explosion for each firework can be calculated as follows:

$$A_i = \hat{A} \cdot \frac{f(\mathbf{x}_i) - y_{min} + \xi}{\sum_{i=1}^{N} (f(\mathbf{x}_i) - y_{min}) + \xi}, \tag{4.3}$$

where $\hat{A}$ is the maximum explosion amplitude, $f(\mathbf{x}_i)$ is the fitness value of the firework $\mathbf{x}_i$, $y_{min} = \min f(\mathbf{x}_i)\,(i = 1, \ldots, N)$ is the fitness value of the best firework, and ξ is an arbitrarily small number used to avoid division by 0.

The number of sparks produced by the exploding firework is given by:

$$S_i = m \cdot \frac{y_{max} - f(\mathbf{x}_i) + \xi}{\sum_{i=1}^{N} (y_{max} - f(\mathbf{x}_i)) + \xi}, \tag{4.4}$$

where m is a parameter that control the number of sparks, and y_{max} is as in (4.3).

Since the number of sparks must be an integer, there is an additional step

that limits the number of sparks that can be generated as follows:

$$S_i = \begin{cases} round(a \cdot m) & \text{if } S_i < a \cdot m \\ round(b \cdot m) & \text{if } S_i > b \cdot m \quad (a < b < 1) \\ round(S_i) & \text{otherwise.} \end{cases} \tag{4.5}$$

Algorithm 2 provides a pseudocode for the Fireworks Algorithm.

Algorithm 2 Pseudocode for the FWA algorithm

```
Initialize a swarm of N fireworks randomly;
Evaluate fitness f(x) of each firework;
while t < MaxIter and not terminate do
    for each firework in the swarm do
        Calculate A_p according to Eq. 4.3;
        Calculate S_p according to Eq. 4.4 and Eq. 4.5;
        Generate and evaluate new sparks;
        for each spark do
            Evaluate the fitness of the spark;
            Select the best firework/sparks to be "Gaussian fireworks";
            Generate a few "Gaussian sparks" around the "Gaussian
    fireworks" based on a Gaussian distribution;
            Evaluate the fitness for each "Gaussian spark";
        end for
    end for
    Select the best points from the current fireworks and sparks to form
    the next generation of fireworks;
end while
Return the best firework found;
```

FWA has been proposed as a better alternative to PSO, and boasts a faster convergence. This is an advantage on some problems, but it can also lead to premature convergence, which was one of the reasons that motivated modifications to the PSO topology.

4.3 Related work

PSO and FWA are population-based optimization approaches that can be combined to leverage their respective strengths. PSO is known for its simplicity and efficiency, while FWA is known for its ability to prevent local

optima.

A simple way to combine these algorithms is to use FWA to initialize the particles in PSO, providing the swarm with a good starting point for exploration. Similarly, PSO can be used to influence the FWA explosion operation, directing the sparks to promising areas of the search space.

In the newly Dynamic Fireworks Algorithm with Particle Swarm Optimization (DFWPSO) [16], PSO was utilized in a new mechanism for update fireworks, aiming to accelerate the convergence of the FWA algorithm and reduce the computing time, thus improving the overall performance of the fireworks algorithm.

By appropriately combining the strong exploitation abilities of PSO with the strong exploration capabilities of FWA, it is possible to build more sophisticated hybrid algorithms that achieve a better balance between the exploration of the entire search space and exploitation, i.e., refinement of the best-known solutions.

In the PS–FW algorithm [2], modified FWA operators are incorporated into the PSO solution procedure. The concept behind this hybrid algorithm is that at the start of each iteration, the velocity and position of each particle are updated in the same way as in the PSO algorithm, and then the so-called abandonment and supplement mechanism is applied aiming to balance the exploration and exploitation ability of the PS–FW algorithm. These two operations follow the logic of FWA. A number of particles with the worst fit are discarded as a result of the abandonment operation, while a number of particles with better fitness fit are retained for the subsequent iteration in view of the supplement operation.

These best particles are used to implement a modified explosion operator, a new mutation operator, and a fitness-based selection operator, of which the first two were developed to accelerate global convergence and prevent premature convergence to local optima. The new particles obtained by the FWA operators are added to the original swarm in order to balance the number of particles and generate a new particle swarm for the next algorithm iteration.

4.4 Proposed PSO–FWA algorithm

Although the PS–FW algorithm [2] combines PSO and FWA and uses the local exploitation of PSO and the global exploration of FWA to achieve better results, the new PSO and FWA hybrid proposed in this paper, PSO–FWA, does not follow the same principle.

In fact, the new hybrid approach here presented does in a way the opposite, as PSO is here used for global exploration and FWA for local exploitation. In PSO–FWA, the von Neumann topology [6] is assumed to be used, but every few iterations, the *bp* best particles multiply using the amplitude and number of new particles (i.e., sparks) randomly generated within the amplitude A_i in the new swarm from FWA.

The PSO–FWA algorithm pseudocode is shown in Algorithm 3.

Algorithm 3 Pseudocode for the proposed PSO–FWA algorithm

```
Initialize N particles with von Neumann topology;
Evaluate the fitness value f(x) of each particle;
while t < maxIter and not terminate do
    if t % k = 0 then
        Sort particles by fitness;
        y_min ← particles[1];
        y_max ← particles[N/2];
        best_particles ← particles[1 : bp];
        for p in bp do
            Calculate A_p according to Eq. 4.3;
            Calculate S_p according to Eq. 4.4 and Eq. 4.5;
            Generate new sub-swarm according to Algorithm 4;
        end for
    end if
    Calculate v_n^{t+1} for each particle according to Eq. 4.1;
    Calculate x_n^{t+1} for each particle according to Eq. 4.2;
    Evaluate fitness of each particle;
end while
Return the best particle found;
```

Whenever creating new particles dynamically in PSO, these new particles need belong to a group or sub-swarm, which can create problems in recalculating a new group for every particle. To avoid this, as well as to utilize the exploitation capabilities of FWA, since only the best particles

get to reproduce in this way, there is not a large concern with premature convergence, so these new groups form a gbest topology with each other, as well as with their "parent" particle.

For a newly created particle to be able to communicate, it must know which particles belong to its group and make every existing particle aware of its existence. To maintain the von Neumann topology, this would mean recalculating the configuration of the entire swarm. Every time a particle multiplies, each new particle is connected to its parent and the other particles in the same sub-swarm via a gbest topology, in order to avoid this computational effort. As only a small number of particles multiply, and those selected to do so are the ones exploring the most promising regions, all new particles originate from such regions.

The use of the gbest topology for the new particles does not affect the general capacity of the swarm for global exploration, since the new particle only share information directly with its parent. This iterative creation of new sub-swarms connected to a single particle has an impact on the topology of the swarm. In the case of a problem with numerous local minima, where the particles are spread apart, the new sub-swarms will also be separated, allowing for a faster exploitation of the promising regions.

Algorithm 4 Pseudocode for the sub-swarm algorithm

1: **for** s in S_p **do**
2: $\quad new\_particles[s] \leftarrow p \cdot \text{Gaussian}() \cdot A_p$;
3: $\quad$ Connect all *new_particles* in same group;
4: **end for**
5: Return *new_particles*;

This does not impede the global exploration capabilities of the algorithm, as the new particles only indirectly share information to the rest of the particles while locally searching promising regions. This successive generation of new particles connected to a single parent particle slowly morphs the overall topology of the swarm. If the best particles are spread apart, indicating an objective function with many local minima, the particles that multiply will quickly explore these promising regions and further multiplications will abandon the groups that got stuck in local minima. By opposition, if the particles that multiply are close to each other, this will

result in an increase in the number of particles in the promising regions, allowing for a fast convergence.

4.5 Experimental setup

4.5.1 Experimental setting

To allow for a fair comparison, the standard PSO, FWA, PS–FW, and PSO–FWA algorithms were implemented and compared in the same experimental setting. All algorithms were run 51 times each, as suggested in [7], using the same control parameters. The maximum number of iterations allowed for each algorithm was set to 1,000, the inertia weight w was set to 0.7, and the acceleration coefficients c_1 and c_2 were both set to 1.8. In all experiments, a tolerance of 1e$-$12 was adopted.

Regarding algorithm-specific parameters, the number of best particles *bp* was set to 5 and a new sub-swarm was generated every 50 iterations.

The von Neumann topology was used in both the standard PSO and PSO–FWA, whereas the gbest topology was utilized for PS–FW, as it was used in the original algorithm [2].

The values used for the specific parameters in the FWA and PS–FW algorithms were the same as in [13] and [2], respectively.

The comparison also included the Enhanced Jaya (EJAYA) algorithm [15], a variant of the Jaya metaphor-less algorithm [10] that has recently been demonstrated to be quite effective in solving systems of nonlinear equations [11].

The implementation of the algorithms was done in Julia [1] using double-precision floating-point arithmetic. All experiments were run on a portable computer with an Intel® Core™ i7-4720HQ processor at 2.60 GHz and 8 GB RAM.

4.5.2 Test problems

The following systems of nonlinear equations, which are difficult to solve using traditional iterative methods, were chosen as test problems for the comparison performed in this study:

Problem 1. *([4], Problem D1 – Modified Rosenbrock),* $n = 12$.

$$f_{2i-1}(\boldsymbol{x}) = \frac{1}{1+\exp(-x_{2i-1})} - 0.73$$
$$f_{2i}(\boldsymbol{x}) = 10(x_{2i} - x_{2i-1}^2), \quad i = 1, \ldots, \tfrac{n}{2}$$
$$D = ([-10, 10], \ldots, [-10, 10])^T$$

Problem 2. *([4], Problem D3 – Powell badly scaled),* $n = 12$.

$$f_{2i-1}(\boldsymbol{x}) = 10^4 x_{2i-1} x_{2i} - 1$$
$$f_{2i}(\boldsymbol{x}) = \exp(-x_{2i-1}) + \exp(-x_{2i}) - 1.0001, \quad i = 1, \ldots, \tfrac{n}{2}$$
$$D = ([0, 100], \ldots, [0, 100])^T$$

Problem 3. *([4], Problem D6 – Shifted and augmented trigonometric function with an Euclidean sphere),* $n = 12$.

$$f_i(\boldsymbol{x}) = n - 1 - \sum_{j=1}^{n-1} \cos(x_j - 1) + i(1 - \cos(x_i - 1)) - \sin(x_i - 1),$$
$$i = 1, \ldots, n-1$$
$$f_n(\boldsymbol{x}) = \sum_{j=1}^{n} x_j^2 - 10000$$
$$D = ([-200, 200], \ldots, [-200, 200])^T$$

Problem 4. *([14], Economics modeling application),* $n = 12$.

$$f_i(\boldsymbol{x}) = \left(x_i + \sum_{k=1}^{n-i-1} x_k x_{i+k} \right) x_n - c_i, \quad i = 1, \ldots, n-1$$
$$f_n(\boldsymbol{x}) = \sum_{j=1}^{n-1} x_j + 1$$

where the constants c_i *can be chosen arbitrarily;*

here $c_i = 0,\ i = 1, \ldots, n-1$.

$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 5. *([5], Example 1 – The Bratu problem),* $n = 12$.

$$f_1(\boldsymbol{x}) = -2x_1 + x_2 + \alpha h^2 \exp(x_1)$$
$$f_n(\boldsymbol{x}) = x_{n-1} - 2x_n + \alpha h^2 \exp(x_n)$$
$$f_i(\boldsymbol{x}) = x_{i-1} - 2x_i + x_{i+1} + \alpha h^2 \exp(x_i), \quad i = 2, \ldots, n-1,$$

where $\alpha \geq 0$ *is a parameter, assuming here* $\alpha = 3.5$, *and* $h = \dfrac{1}{n+1}$.

$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 6. *([5], Example 2 – The beam problem),* $n = 12$.

$$f_1(\boldsymbol{x}) = -2x_1 + x_2 + \alpha h^2 \sin(x_1)$$

$f_n(\boldsymbol{x}) = x_{n-1} - 2x_n + \alpha h^2 \sin(x_n)$
$f_i(\boldsymbol{x}) = x_{i-1} - 2x_i + x_{i+1} + \alpha h^2 \exp(x_i), \quad i = 2, \ldots, n-1,$
where $h = \dfrac{1}{n+1}$ *and* $\alpha \geq 0$ *is a parameter; here* $\alpha = 11$.
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 7. *([9], 21 – Extended Rosenbrock function),* $n = 12$.
$f_{2i-1}(\boldsymbol{x}) = 10(x_{2i} - x_{2i-1}^2)$
$f_{2i}(\boldsymbol{x}) = 1 - x_{2i-1}, \quad i = 1, \ldots, \frac{n}{2}$
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 8. *([9], 26 – Trigonometric function),* $n = 12$.
$f_i(\boldsymbol{x}) = n - \sum_{j=1}^{n} \cos x_j + i(1 - \cos x_i) - \sin x_i, \quad i = 1, \ldots, n$
$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 9. *([9], 27 – Brown almost-linear function),* $n = 12$.
$f_i(\boldsymbol{x}) = x_i + \sum_{j=1}^{n} x_j - (n+1), \quad i = 1, \ldots, n-1$
$f_n(\boldsymbol{x}) = \left(\prod_{j=1}^{n} x_j \right) - 1$
$D = ([-10, 10], \ldots, [-10, 10])^T$

Problem 10. *([9], 28 – Discrete boundary value function),* $n = 12$.
$f_1(\boldsymbol{x}) = 2x_1 - x_2 + h^2(x_1 + h + 1)^3/2$
$f_n(\boldsymbol{x}) = 2x_n - x_{n-1} + h^2(x_n + nh + 1)^3/2$
$f_i(\boldsymbol{x}) = 2x_i - x_{i-1} - x_{i+1} + h^2(x_i + t_i + 1)^3/2, \quad i = 2, \ldots, n-1,$
where $h = \frac{1}{n+1}$ *and* $t_i = ih$.
$D = ([0, 5], \ldots, [0, 5])^T$

4.6 Results and discussion

The average fitness value for each pair algorithm/problem was calculated and the results thus obtained are shown in Table 4.1. The best (i.e., minimum) fitness values found for each different algorithm and problem are shown in Table 4.2. The best result for each problem is bolded, while the second-best value is underlined.

Table 4.1: Average performance of each algorithm

Problem	PSO	FWA	PSO–FWA	PS–FW	EJAYA
01	0.093696602	8.231227429	**0.023829129**	2.688307333	0.786323905
02	**0.000611303**	10528.36554	0.053180425	4.929956145	5.845716462
03	**0.673563009**	17637.37896	0.813282593	7.37086e+15	24.94501395
04	7.68886e−13	121.161161	**3.50538e−13**	7.08888e+24	6.919e−13
05	2.472214572	405.3670455	**0.017626657**	0.180648169	18.40166643
06	**0.629464887**	449.2970073	0.708199283	0.554307947	16.3762457
07	**0.740239982**	1166.651425	1.946738341	4.30706e+11	11.87298727
08	**0.003830296**	0.032297347	0.03146428	0.025972228	4.244178222
09	3.08272e−05	27.63435204	**4.73538e−06**	4.77479e+66	0.019622826
10	0.001165984	0.156890307	**6.90023e−05**	7.32552e+14	0.237439165

Table 4.2: Best performance of each algorithm

Problem	PSO	FWA	PSO–FWA	PS–FW	EJAYA
01	**6.58917e−07**	0.214234267	2.67113E-05	0.616052907	0.005183943
02	0.0006113	837.2544034	**0.000413175**	2.278576919	5.000485975
03	0.00467257	3447.14932	**3.14072e−09**	3.68477048	11.79736641
04	1.64973e−13	1.93052557	2.79099e−13	**3.64338e−14**	1.2451e−13
05	2.469796642	45.3698909	**7.61427e−05**	0.061573931	1.01676417
06	0.517634554	80.45977863	**0.514741134**	0.518018226	1.212673401
07	4.71128E-07	60.61553916	**4.2838e−08**	3.047900675	2.048738198
08	0.003270911	0.021813477	**9.59711e−13**	6.0212E-05	1.282223168
09	9.7311e−13	1.41803206	**9.06386e−13**	0.29030721	3.5519e−06
10	**9.29977e−13**	0.111856778	9.8634e−13	0.045728666	0.237439165

From the results obtained, a few conclusions can be drawn. PSO–FWA achieved the best average result in half the problems, tied with PSO. While this may indicate a failure of the other algorithms, it should be noted that what these two algorithms have in common is the von Neumann topology, used to allow for a better exploration.

Due to their rapid convergence, the remaining algorithms do not perform well on challenging problems, such as nonlinear equation systems, for which exploration plays a crucial role.

Even though PSO on average performed similarly to the PSO–FWA algorithm, this approach has shown to have better convergence, as it achieved either the best result or a result quite close to the best in almost all problems.

4.7 Conclusion

PSO–FWA, a novel optimization algorithm based on a hybridization of PSO and FWA, was proposed to solve complex nonlinear equation systems. The performance of the proposed PSO–FWA hybrid algorithm was compared to PS–FW, another hybrid algorithm based on PSO and FWA, as well as to PSO and FWA.

The Enhanced Jaya (EJAYA) algorithm [15], which was recently shown quite effective at solving systems of nonlinear equations [11], was also included in the computational comparison carried out.

While the previous algorithms considered for comparison appear to perform better in simple optimization problems with few local minima, PSO–FWA was able to achieve better results in more difficult problems, as evidenced by the results obtained with the set of complex nonlinear equation systems chosen for this study.

References

[1] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. Julia: A fresh approach to numerical computing. *SIAM Review 59*, 1 (2017), 65–98.

[2] Chen, S., Liu, Y., Wei, L., and Guan, B. PS–FW: A hybrid algorithm based on particle swarm and fireworks for global optimization. *Computational Intelligence and Neuroscience 2018* (2018), 6094685.

[3] Eberhart, R., and Kennedy, J. A new optimizer using particle swarm theory. In *6th International Symposium on Micro Machine and Human Science, Nagoya, Japan* (1995), IEEE, pp. 39–43.

[4] Friedlander, A., Gomes-Ruggiero, M., Kozakevich, D., Martínez, J., and Santos, S. Solving nonlinear systems of equations by means of quasi-Newton methods with a nonmonotone strategy. *Optimization Methods and Software 8*, 1 (1997), 25–51.

[5] Kelley, C., Qi, L., Tong, X., and Yin, H. Finding a stable solution of a system of nonlinear equations. *Journal of Industrial and Management Optimization 7*, 2 (2011), 497–521.

[6] Kennedy, J., and Mendes, R. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)* (2002), vol. 2, pp. 1671–1676.

[7] Liang, J., Qu, B., Suganthan, P., and Hernández-Díaz, A. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Technical Report 201212, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China, 2013.

[8] Liu, Q., Wei, W., Yuan, H., Zhan, Z.-H., and Li, Y. Topology selection for particle swarm optimization. *Information Sciences 363* (2016), 154–173.

[9] Moré, J., Garbow, B., and Hillstrom, K. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software 7*, 1 (1981), 17–41.

[10] Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations 7* (2016), 19–34.

[11] Ribeiro, S., Silva, B., and Lopes, L. G. Solving systems of nonlinear equations using Jaya and Jaya-based algorithms: A computational comparison. In *Proceedings of the International Conference on Paradigms of Communication, Computing and Data Analytics: PCCDA 2023* (Singapore, to appear, 2023), A. Yadav, S. J. Nanda, and M.-H. Lim, Eds., Springer.

[12] Shi, Y., and Eberhart, R. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360), 1998* (1998), IEEE, pp. 69–73.

[13] Tan, Y., and Zhu, Y. Fireworks algorithm for optimization. In *International Conference on Swarm Intelligence* (2010), pp. 355–364.

[14] van Hentenryck, P., McAllester, D., and Kapur, D. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis 34*, 2 (1997), 797–827.

[15] Zhang, Y., Chi, A., and Mirjalili, S. Enhanced Jaya algorithm: A simple but efficient optimization method for constrained engineering design problems. *Knowledge-Based Systems 233* (2021), 107555.

[16] Zhu, F., Chen, D., and Zou, F. A novel hybrid dynamic fireworks algorithm with particle swarm optimization. *Soft Computing 25* (2021), 2371–2398.

Chapter 5

Solving Systems of Nonlinear Equations Using Jaya and Jaya-Based Algorithms: A Computational Comparison

Solving systems of nonlinear equations is a very challenging problem, particularly as the size of the systems increases, and there is no general numerical method that is both efficient and robust enough to tackle it. On the other hand, metaheuristic algorithms are a broad class of high-level techniques or heuristics for addressing hard and challenging problems. Some of these algorithms are known to produce high-quality approximate solutions for optimization problems, albeit without ensuring optimality. Jaya is a simple, parameter-less metaheuristic algorithm that has proven to be effective in solving a variety of real-world problems. However, its effectiveness in solving nonlinear equation systems, perhaps the hardest class of numerical problems to solve, needs to be assessed and verified. This study examines the Jaya algorithm and some of its variants with respect to the problem of solving a set of difficult scalable nonlinear equation systems. The results showed that the so-called Enhanced Jaya algorithm produced the best results overall, while the Modified Jaya algorithm had the worst outcomes, thereby not being suitable for solving the important class of numerical problems considered.

5.1 Introduction

Nonlinear equations have an extensive importance in many domains of knowledge, including Physics, Economics, Chemistry, and several branches of Engineering [18], and appear in almost all simulations of physical processes [11].

Nevertheless, there is no general numerical method that is robust and efficient enough to solve systems of nonlinear equations (SNLEs), which is perhaps the hardest problem in numerical mathematics [10]. Newton's method is a well-known and usual method for solving SNLEs [8]. As with the majority of its variants, its success is dependent on the quality of the initial approximations chosen [2].

However, a SNLE can be effortlessly converted into a homologous optimization problem by adopting the sum of each system equation's absolute value as an objective function to be minimized:

$$\min f(\boldsymbol{x}) = \sum_{j=1}^{n} \left| e_j(\boldsymbol{x}) \right|, \tag{5.1}$$

where $e_j(\boldsymbol{x}) = 0$ is the j^{th} equation, and $\boldsymbol{x} = (x_1, \ldots, x_n)^T$.

The resulting optimization problem can be solved by stochastic approaches such as pure or hybrid metaheuristic algorithms (see, e.g., the review in [5] and the references therein), which have a strong ability to determine near-optimal solutions while being more flexible, effective, and robust when comparing to deterministic optimization methods [15], particularly on large-scale optimization problems, and without requiring good initial approximations, albeit with the trade-off that the quality of the obtained solutions cannot be guaranteed.

Newton's method, with its aforementioned limitations, has been hybridized with different metaheuristic algorithms in order to combine the positive characteristics of both classes of methods when solving SNLEs. Its hybridization with the Harris hawks algorithm [22] is a recent example.

All population-based metaheuristic algorithms require the adjustment of parameters like number of iterations and population size, while other requires the same with algorithmic-specific variables. The performance of

such algorithms can be negatively affected by the poor adjustment of these parameters, with the added difficulty that there is sometimes a lack of guidance on how to properly select them.

Various population-based algorithms have been successfully adapted to efficiently solve SNLEs, including different hybridizations of the Particle Swarm Optimization algorithm (see, e.g., [13, 17, 25]) and of the Sine Cosine algorithm (see, e.g., [9, 23]). However, these algorithms require the specification of a set of initial parameters, which has the disadvantage of being determinant for the quality of the results.

To address such issue, the Jaya algorithm was proposed in [19] as a parameter-less metaheuristic approach that is both efficient and easy to set up. When compared to other population-based algorithms, Jaya has some advantages, including being generic, simple to implement, and not relying on algorithm-specific parameters [4]. As a result, it is considered a parameter-less algorithm, as it only demands the most essential control parameters of a population-based algorithm.

Different variants to the original Jaya algorithm have been proposed intending to improve its effectiveness and performance, either by tweaking some of its operators or by better balancing global and local search spaces.

This study examines some Jaya-based algorithms with various types of modifications to the original Jaya algorithm. The chosen variants include the refinement of the main equation, the use of additional population data and an enhanced search strategy, the utilization of multiple populations, and the use of an oppositional population.

The performance of Jaya and some Jaya-based variants in solving SNLEs is here investigated, and a comparative analysis is performed on the results produced by each algorithm. The aim is to assess whether the Jaya algorithm and some of its variants can be used efficiently to deal with this type of problem, and to answer which variant is more successful than others in this particular application domain.

5.2 Related work

5.2.1 Jaya algorithm

Jaya is a population-based metaheuristic for optimizing different unconstrained and constrained problems [19, 20]. The fundamental tenet of the algorithm is that the solution obtained for a particular problem should avoid the worst solution while tending towards the best one.

This strategy just relies on standard control parameters such as the dimension $popSize$ of the population and the maximum of iterations allowed ($maxIter$), in order to achieve its main goal which is to optimize (i.e., minimize or maximize) an objective function $f(\boldsymbol{x})$.

Considering the number of decision variables ($numVar$), a design variable index $v \in [1, numVar]$, a population index $p \in [1, popSize]$, and an iteration $i \in [1, maxIter]$, let $\boldsymbol{x}_{v,p,i}$ be the value of the v^{th} variable of the p^{th} population candidate during the i^{th} iteration. Then the modified value $\boldsymbol{x}^{new}_{v,p,i}$ is obtained as follows:

$$\boldsymbol{x}^{new}_{v,p,i} = \boldsymbol{x}_{v,p,i} + r_{1,v,i}\left(\boldsymbol{x}_{v,best,i} - |\boldsymbol{x}_{v,p,i}|\right) - r_{2,v,i}\left(\boldsymbol{x}_{v,worst,i} - |\boldsymbol{x}_{v,p,i}|\right), \quad (5.2)$$

where $r_{1,v,i}$ and $r_{2,v,i}$ are random numbers in $[0, 1]$ for the v^{th} variable during the i^{th} iteration, while $\boldsymbol{x}_{v,best,i}$ and $\boldsymbol{x}_{v,worst,i}$ are the candidate solutions with the best and the worst fitness values, respectively.

The Jaya pseudocode is presented in Algorithm 1.

5.2.2 Modified Jaya algorithm

One drawback of heuristic and nature-inspired algorithms is the exploration phase, in which the algorithm may get trapped into local optima. Jaya algorithm is not immune to premature convergence and this is one of the main aspects addressed by its different variants, as is the case of the Modified Jaya (MJAYA) algorithm [6].

MJAYA follows the same procedures of the original Jaya algorithm, but proposes a modified main equation given by:

$$\boldsymbol{x}^{new}_{v,p,i} = \boldsymbol{x}_{v,p,i} + r_{1,v,i}\left(\boldsymbol{x}_{v,worst,i} - |\boldsymbol{x}_{v,p,i}|\right) - L \times r_{2,v,i}\left(|\boldsymbol{x}_{v,p,i}|^2 - \boldsymbol{x}^2_{v,best,i}\right), \quad (5.3)$$

Algorithm 1 Jaya algorithm

Initialize $numVars$, $popSize$ and $maxIters$;
Generate initial population X;
Evaluate the fitness $f(\boldsymbol{x})$ of each $\boldsymbol{x} \in X$;
while $i < maxIter$ and not terminate **do**
    Determine $\boldsymbol{x}_{v,best,i}$ and $\boldsymbol{x}_{v,worst,i}$;
    for $p \leftarrow 1, popSize$ **do**
        for $v \leftarrow 1, numVars$ **do**
            Update $\boldsymbol{x}_{v,p,i}^{new}$ by Eq. (5.2);
        end for
        Calculate $f(\boldsymbol{x}_{v,p,i}^{new})$;
        if $f(\boldsymbol{x}_{v,p,i}^{new})$ is better than $f(\boldsymbol{x}_{v,p,i})$ **then**
            $\boldsymbol{x}_{v,p,i} \leftarrow \boldsymbol{x}_{v,p,i}^{new}$;
            $f(\boldsymbol{x}_{v,p,i}) \leftarrow f(\boldsymbol{x}_{v,p,i}^{new})$;
        else
            Keep $\boldsymbol{x}_{v,p,i}$ and $f(\boldsymbol{x}_{v,p,i})$ values;
        end if
    end for
end while
Report solution found;

where the objective function term with the best fitness value is only used to adjust the values of the remaining terms, and the coefficient L at every iteration is determined as follows:

$$L = \begin{cases} 1, & \text{if rand} > 0.5 \\ -1, & \text{otherwise.} \end{cases} \tag{5.4}$$

5.2.3 Enhanced Jaya algorithm

The Enhanced Jaya (EJAYA) algorithm [28] improves upon the original implementation of Jaya by making efficient use of additional information from the population to prevent exploration from being trapped in local optima, thereby minimizing the risk of premature convergence.

EJAYA uses the original Jaya algorithm parameters, such as the current best and worst solutions, while introducing new parameters such as the mean solution and historical solutions to balance its global exploration ability and local exploitation strategy.

The exploitation strategy is used to try to inhibit the algorithm from

becoming entangled in a local optimum by removing the main function's absolute value symbol, and instead using an upper local (P_u) and a lower local (P_l) attract points. P_u is determined in the following way:

$$P_u = \lambda_3 \times \boldsymbol{x}_{v,best,i} + (1 - \lambda_3) \times M, \tag{5.5}$$

where λ_3 is an uniformly distributed random number in $[0, 1]$, $\boldsymbol{x}_{v,best,i}$ is the best candidate in terms of fitness value, and M is the mean of the current population, defined as:

$$M = \frac{1}{popSize} \sum_{p=1}^{popSize} \boldsymbol{x}_p. \tag{5.6}$$

The lower point P_l is written as:

$$P_l = \lambda_4 \times \boldsymbol{x}_{v,worst,i} + (1 - \lambda_4) \times M, \tag{5.7}$$

where λ_4 is a random number in $[0, 1]$ with uniform distribution, $\boldsymbol{x}_{v,worst,i}$ is the worst candidate in terms of fitness value, and M is the current population mean, defined in Equation (5.6).

EJAYA's local exploitation approach is as follows:

$$\boldsymbol{x}_{v,p,i}^{new} = \boldsymbol{x}_{v,p,i} + \lambda_5 \left(P_u - \boldsymbol{x}_{v,p,i}\right) - \lambda_6 \left(P_l - \boldsymbol{x}_{v,p,i}\right), \tag{5.8}$$

where λ_5 and λ_6 are uniformly distributed random numbers in the interval $[0, 1]$.

The global exploration approach was inspired by the backtracking search algorithm [3] and uses differential vectors between the current and historical (i.e., old) populations, providing additional search space when compared to vectors from the same generation population.

In the first interaction, the historical population $X_{v,p}^{old}$ is the same as $X_{v,p}$. Afterwards, they are selected in the following way:

$$X_{v,p}^{old} = \begin{cases} X_{v,p}, & \text{if } P_{switch} \leq 0.5 \\ X_{v,p}^{old}, & \text{otherwise,} \end{cases} \tag{5.9}$$

where P_{switch} is a random number with a uniform $[0, 1]$ distribution, which defines the switching probability between the two populations.

After selecting the population, the EJAYA algorithm randomly rearranges the elements $\boldsymbol{x}_{v,p,i}^{old}$ of the historical population $X_{v,p}^{old}$ by applying a shuffling function $permuting(\cdot)$ to the entire historical population, as shown below:

$$X_{v,p}^{old} = permuting\left(X_{v,p}^{old}\right). \tag{5.10}$$

The objective function for the global exploration strategy is expressed as:

$$\boldsymbol{x}_{v,p,i}^{new} = \boldsymbol{x}_{v,p,i} + k \times \left(\boldsymbol{x}_{v,p,i}^{old} - \boldsymbol{x}_{v,p,i}\right), \tag{5.11}$$

where k is a standard normally distributed random number.

In EJAYA, the local exploitation (LES) and global exploration (GES) strategies are both equally relevant and, as such, the update strategy is selected as follows:

$$Strategy = \begin{cases} LES, & \text{if } P_{select} > 0.5 \\ GES, & \text{otherwise,} \end{cases} \tag{5.12}$$

where P_{select} is a random number in $[0, 1]$ with uniform distribution. Furthermore, the historical population $X_{v,p}^{old}$ and the current population $X_{v,p}$ are initialized by the same method.

5.2.4 SAMP–Jaya algorithm

The Self-Adaptive Multi-Population Jaya (SAMP–Jaya) combines the ideas of Jaya with the island model from Genetic Algorithms (GA) [21], although with some modifications.

Instead of dividing the population into only two groups, named master island and slave island as in the basic island model from GA, in SAMP–Jaya the number of sub-populations (or slave islands) is determined programmatically on the basis of the current problem state's characteristics. The population migration between islands is based on the quality of the fitness value and a greedy selection mechanism.

The number of sub-populations is adjusted along the search phase.

Attempting to maintain diversity and augment the exploratory process, newly created solutions (which are randomly generated) are used to replace duplicate ones. The variable m, whose value at the beginning of the execution is $m = 2$, is used to specify the number of distinct sub-populations.

5.2.5 Oppositional Jaya algorithm

The Oppositional Jaya (OJaya) approach [27] offers two improvements over the original Jaya algorithm. One is provided by the Oppositional Learning (OL), a population-based algorithm that simultaneously calculates and evaluates the current (X) and oppositional (X^o) populations to choose the best one for the following generation, and another by the Distance–Adaptive Coefficient (DAC), which is determined based on the best and worst positions. The first method provides an expansion of the search space and promotes population diversity and strength, whereas the second causes the population to move faster in the direction of the best position and away from the worst one.

In OJaya, the oppositional population elements are generated as follows:

$$\boldsymbol{x}^{o}_{v,p,i} = s \times (A_{v,i} + B_{v,i}) - \boldsymbol{x}_{v,p,i}, \tag{5.13}$$

where s is a random number in $[0, 1]$, and $A_{v,i}$ and $B_{v,i}$ are dynamic bounds for the population, which are given by:

$$A_{v,i} = \min\left(\boldsymbol{x}_{v,p,i}\right), \qquad B_{v,i} = \max\left(\boldsymbol{x}_{v,p,i}\right). \tag{5.14}$$

Both dynamic bounds $A_{v,i}$ and $B_{v,i}$ are set to be updated every 50 iterations to prevent the population from becoming stuck in a local minimum as the search space shrinks with each iteration.

As these dynamic bounds have the potential to cause $\boldsymbol{x}^{o}_{v,p,i}$ to escape the minimum and maximum limits of constrained problems, it is necessary to reset $\boldsymbol{x}^{o}_{v,p,i}$ in the following manner when this occurs:

$$\boldsymbol{x}^{o}_{v,p,i} = rand\left(A_{v,i}, B_{v,i}\right), \tag{5.15}$$

where $rand\left(A_{v,i}, B_{v,i}\right)$ is a uniformly distributed random number in

$[A_{v,i}, B_{v,i}]$.

The oppositional learning is used when generating both the initial and the current population of each iteration.

In order to achieve the benefits offered by DAC and provide fine-tuning of the population convergence in the latter stages of the exploration to find the global optima, the distance-adaptive coefficient d_i is determined as follows:

$$d_i = \begin{cases} \left(\dfrac{f(\boldsymbol{x}_{v,best,i})}{f(\boldsymbol{x}_{v,worst,i})}\right)^2, & \text{if } f\left(\boldsymbol{x}_{v,worst,i}\right) \neq 0 \\ 1, & \text{otherwise.} \end{cases} \tag{5.16}$$

The main function of OJaya is comparable to that of the original Jaya, with the addition of the d_i factor, as shown below:

$$\boldsymbol{x}_{v,p,i}^{new} = \boldsymbol{x}_{v,p,i} + r_{1,v,i}\left(\boldsymbol{x}_{v,best,i} - |\boldsymbol{x}_{v,p,i}|\right) - d_i \times r_{2,v,i}\left(\boldsymbol{x}_{v,worst,i} - |\boldsymbol{x}_{v,p,i}|\right). \tag{5.17}$$

As d_i is a function of $\boldsymbol{x}_{v,best,i}$ and $\boldsymbol{x}_{v,worst,i}$, whose distance gradually decreases during the iterations, its value is small at the beginning of the search process and gradually converges to 1 as the process approaches the end. This is the self-adaptive nature of d_i, which is achieved without the need for additional parameters.

5.3 Computational experiments

5.3.1 Experimental setting and implementation

The population size adopted varied with the dimensionality of the test problem. For all algorithms considered in this study, it was set to 10× the problem dimension, which in this study was taken as equal to 4, 8, 12, 16 and 20.

The maximum number of iterations for every algorithm under consideration was set to 1,000× the problem dimension, while the number of independent runs for each algorithm and problem was equal to 51, as suggested in [14].

The implementation was done in Julia programming language using double precision floating-point arithmetic. Computational experiments were

conducted on a computer with an AMD processor Ryzen 5 3500X and 16 GB RAM DDR4.

5.3.2 Test problems

The methaphor-less optimization algorithms considered were tested on a set of difficult nonlinear equation system problems. The benchmark problems selected from the literature are presented below.

In addition to the definition of the functions, the domain D chosen for each test problem is also indicated. The test problems are scalable with respect to the number n of variables.

Taking into account that five different dimensions were considered for each of the 14 problems shown below, the computational analysis carried out involved a total of 70 different nonlinear equation system problems.

Problem 1. *([1], Schubert–Broyden function),* $n = 4, 8, 12, 16, 20$.

$f_1(\boldsymbol{x}) = (3 - x_1)x_1 + 1 - 2x_2$

$f_i(\boldsymbol{x}) = (3 - x_i)x_i + 1 - x_{i-1} - 2x_{i+1}, \quad i = 2, \ldots, n-1$

$f_n(\boldsymbol{x}) = (3 - x_n)x_n + 1 - x_{n-1}$

$D = ([-100, 100], \ldots, [-100, 100])^T$

Problem 2. *([7], Problem D1 – Modified Rosenbrock),* $n = 4, 8, 12, 16, 20$.

$f_{2i-1}(\boldsymbol{x}) = \dfrac{1}{1 + \exp(-x_{2i-1})} - 0.73$

$f_{2i}(\boldsymbol{x}) = 10(x_{2i} - x_{2i-1}^2), \quad i = 1, \ldots, \frac{n}{2}$

$D = ([-10, 10], \ldots, [-10, 10])^T$

Problem 3. *([7], Problem D3 – Powell badly scaled),* $n = 4, 8, 12, 16, 20$.

$f_{2i-1}(\boldsymbol{x}) = 10^4 x_{2i-1}x_{2i} - 1$

$f_{2i}(\boldsymbol{x}) = \exp(-x_{2i-1}) + \exp(-x_{2i}) - 1.0001, \quad i = 1, \ldots, \frac{n}{2}$

$D = ([0, 100], \ldots, [0, 100])^T$

Problem 4. *([7], Problem D6 – Shifted and augmented trigonometric function with an Euclidean sphere),* $n = 4, 8, 12, 16, 20$.

$$f_i(\boldsymbol{x}) = n - 1 - \sum_{j=1}^{n-1} \cos(x_j - 1) + i(1 - \cos(x_i - 1)) - \sin(x_i - 1),$$

$i = 1, \ldots, n-1$

$$f_n(\boldsymbol{x}) = \sum_{j=1}^{n} x_j^2 - 10000$$
$$D = ([-200, 200], \ldots, [-200, 200])^T$$

Problem 5. *([24], Economics modeling application),* $n = 4, 8, 12, 16, 20$.

$$f_i(\boldsymbol{x}) = \left(x_i + \sum_{k=1}^{n-i-1} x_k x_{i+k} \right) x_n - c_i, \quad i = 1, \ldots, n-1$$
$$f_n(\boldsymbol{x}) = \sum_{j=1}^{n-1} x_j + 1$$

where the constants c_i *can be chosen arbitrarily;*

here $c_i = 0,\ i = 1, \ldots, n-1$.

$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 6. *([12], Example 1 – The Bratu problem),* $n = 4, 8, 12, 16, 20$.

$$f_1(\boldsymbol{x}) = -2x_1 + x_2 + \alpha h^2 \exp(x_1)$$
$$f_n(\boldsymbol{x}) = x_{n-1} - 2x_n + \alpha h^2 \exp(x_n)$$
$$f_i(\boldsymbol{x}) = x_{i-1} - 2x_i + x_{i+1} + \alpha h^2 \exp(x_i), \quad i = 2, \ldots, n-1,$$

where $\alpha \geq 0$ *is a parameter, assuming here* $\alpha = 3.5$, *and* $h = \dfrac{1}{n+1}$.

$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 7. *([12], Example 2 – The beam problem),* $n = 4, 8, 12, 16, 20$.

$$f_1(\boldsymbol{x}) = -2x_1 + x_2 + \alpha h^2 \sin(x_1)$$
$$f_n(\boldsymbol{x}) = x_{n-1} - 2x_n + \alpha h^2 \sin(x_n)$$
$$f_i(\boldsymbol{x}) = x_{i-1} - 2x_i + x_{i+1} + \alpha h^2 \exp(x_i), \quad i = 2, \ldots, n-1,$$

where $h = \dfrac{1}{n+1}$ *and* $\alpha \geq 0$ *is a parameter; here* $\alpha = 11$.

$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 8. *([16], 21 – Extended Rosenbrock function),* $n = 4, 8, 12, 16, 20$.

$$f_{2i-1}(\boldsymbol{x}) = 10(x_{2i} - x_{2i-1}^2)$$
$$f_{2i}(\boldsymbol{x}) = 1 - x_{2i-1}, \quad i = 1, \ldots, \tfrac{n}{2}$$
$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 9. *([16], 26 – Trigonometric function),* $n = 4, 8, 12, 16, 20$.

$$f_i(\boldsymbol{x}) = n - \sum_{j=1}^{n} \cos x_j + i(1 - \cos x_i) - \sin x_i, \quad i = 1, \ldots, n$$
$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 10. *([16], 27 – Brown almost-linear function),* $n = 4, 8, 12, 16, 20$.

$$f_i(\boldsymbol{x}) = x_i + \sum_{j=1}^{n} x_j - (n+1), \quad i = 1, \ldots, n-1$$

$$f_n(\boldsymbol{x}) = \left(\prod_{j=1}^{n} x_j \right) - 1$$

$$D = ([-10, 10], \ldots, [-10, 10])^T$$

Problem 11. *([16], 28 – Discrete boundary value function),* $n = 4, 8, 12, 16, 20$.

$$f_1(\boldsymbol{x}) = 2x_1 - x_2 + h^2(x_1 + h + 1)^3/2$$

$$f_n(\boldsymbol{x}) = 2x_n - x_{n-1} + h^2(x_n + nh + 1)^3/2$$

$$f_i(\boldsymbol{x}) = 2x_i - x_{i-1} - x_{i+1} + h^2(x_i + t_i + 1)^3/2, \quad i = 2, \ldots, n-1,$$

where $h = \frac{1}{n+1}$ *and* $t_i = ih$.

$$D = ([0, 5], \ldots, [0, 5])^T$$

Problem 12. *([16], 30 – Broyden tridiagonal function),* $n = 4, 8, 12, 16, 20$.

$$f_1(\boldsymbol{x}) = (3 - 2x_1)x_1 - 2x_2 + 1$$

$$f_n(\boldsymbol{x}) = (3 - 2x_n)x_n - x_{n-1} + 1$$

$$f_i(\boldsymbol{x}) = (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1, \quad i = 2, \ldots, n-1$$

$$D = ([-1, 1], \ldots, [-1, 1])^T$$

Problem 13. *([26], Example 4.1 – Nonlinear resistive circuit),* $n = 4, 8, 12, 16, 20$.

$$f_i(\boldsymbol{x}) = g(x_i) + \sum_{j=1}^{n} x_j - i, \quad i = 1, \ldots, n,$$

where $g(x_i) = 2.5x_i^3 - 10.5x_i^2 + 11.8x_i$.

$$D = ([-100, 100], \ldots, [-100, 100])^T$$

Problem 14. *([26], Example 4.2),* $n = 4, 8, 12, 16, 20$.

$$f_i(\boldsymbol{x}) = x_i - \frac{1}{2n} \left(\sum_{j=1}^{n} x_j^3 + i \right), \quad i = 1, \ldots, n$$

$$D = ([-10, 10], \ldots, [-10, 10])^T$$

5.4 Results and discussion

The average and best (i.e., minimum) fitness values found for each different algorithm and problem function are shown below in Tables 5.1 through 5.10.

Tables 5.1 to 5.5 show the average fitness values for 51 runs of each algorithm with each problem, while Tables 5.6 to 5.10 present the best fitness values for every combination algorithm/problem in each dimension. In each table, the best value obtained for every problem is bolded, while the second best is underlined.

The best results were obtained by EJAYA, both in terms of average and absolute values. This result can be explained by the nature of the class of problems under consideration.

Because the problems are difficult scalable nonlinear equation systems, an algorithm that focuses on balancing local and global exploration can better explore the search space and avoid local minima.

Box plots were also used to show the results so that the performance of each algorithm could be compared in more ways than just the average and best fitness values. Since the results are at very different scales, a logarithmic scale was used on the vertical axis to make it easier to compare them.

The Economics modeling application problem (Problem 5) is displayed on Figure 5.1. This example shows a clear difference in results from the different algorithms tested, with the worst result from Enhanced Jaya being smaller than any result from the other algorithms.

Naturally, these results are highly problem-dependent. A different example, the nonlinear resistive circuit problem (Problem 13) on Figure 5.2, shows a more balanced result, where the choice of algorithm was not such a decisive factor, with the exception of MJAYA, which performed significantly worse than the alternatives on this particular problem.

The fact that one of the algorithms consistently produces the best results indicates that there is a better strategy for approaching and solving problems of this nature. It is noteworthy that the EJAYA algorithm consistently achieved the best (minimum) fitness value, since it could be the case that a more global search would produce a less satisfactory result due to not being given enough time to lead to a more exact one.

The Friedman rank test was used to determine whether the differences in algorithms' performance observed in this study were statistically significant.

The test statistic and p-value for the mean results of each experiment were 205.828407 and 2.097028e−43, respectively. An experiment corresponds to

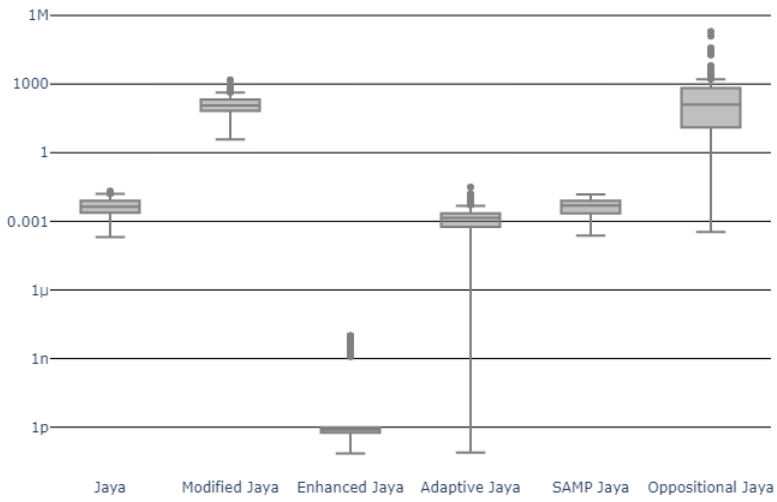

Figure 5.1: Algorithms performance for Problem 5 – Economics modeling application

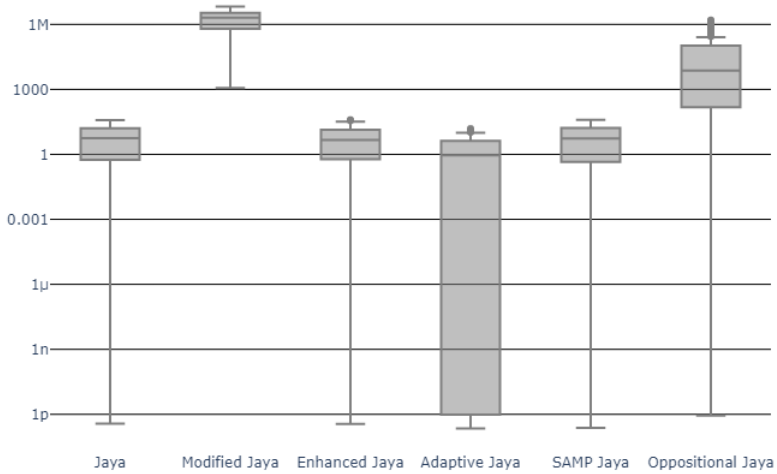

Figure 5.2: Algorithms performance for Problem 13 – Nonlinear resistive circuit

a problem with a certain number of variables. For the best results from each experiment, the obtained test statistic and p-value were 166.029499 and 7.438888e−35, respectively. These results indicate that the p-values are lower than the the significance level considered, $\alpha = 0.05$, which indicates the existence of statistically significant performance differences between the algorithms.

The Nemenyi post-hoc test was then utilized to validate the existence of statistically significant differences between each pair of algorithms and compare their performance. At $\alpha = 0.05$, the p-values for each pairwise comparison of means returned by the Nemenyi test indicate that all groups with EJAYA have statistically significantly different means (p-value for EJAYA–Jaya: 0.00423; EJAYA–MJAYA: 0.001; EJAYA–SAMP-Jaya: 0.011719; EJAYA-OJaya: 0.001). The Enhanced Jaya had the lowest mean rank, confirming that it is the algorithm with the best performance.

In turn, the Modified Jaya algorithm was consistently outperformed. Despite the fact that this Jaya variant also attempts to reduce population premature convergence, the comparatively simple method used to achieve this did not result in the desired effects on the set of problems used. The other three algorithms were also quite ineffective in solving this important class of numerical problems.

Table 5.1: Average fitness for each algorithm and problem with dimension 4

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	0.037301	759.5075	**7.8e−13**	0.082208	3.60415
02	0.061348	4.366276	**0.013026**	0.099944	1.98836
03	1.101062	1.700416	1.529441	0.9208	**0.012071**
04	3.172041	1285.801	**0.794585**	2.942294	3.060386
05	0.002189	97.55153	**7.03e−13**	0.002149	4.28393
06	0.02041	127.4752	**0.020138**	0.020458	28.08892
07	0.867158	112.8549	**0.86427**	0.867175	21.75779
08	2.293608	112.3845	**0.211815**	1.238739	18.14579
09	0.232556	1.066933	0.113705	0.231329	**0.108791**
10	6.56e−5	1.370445	**7.96e−13**	0.000155	1.100248
11	0.4248	0.472382	0.4248	0.4248	**0.200767**
12	1.106871	1.301978	**0.823576**	1.016749	0.916372
13	**0.055345**	76634	0.060916	0.061493	32.60203
14	0.005805	2.591199	**7.8e−13**	0.008878	2.037

Table 5.2: Average fitness for each algorithm and problem with dimension 8

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	1.82763	7726.655	**1.7393**	1.861864	152.2006
02	0.68266	72.27876	**0.021869**	0.716674	22.01862
03	**2.367052**	3.520389	3.196196	2.647138	863.877
04	10.48261	20889.06	**4.073712**	10.27806	197.7471
05	0.008329	121.0815	**6.96e−13**	0.009389	465.9061
06	73.93172	158.8989	**0.001618**	72.14528	163.2751
07	73.67055	158.4651	**0.690457**	72.24566	157.4344
08	21.74472	4420.537	**0.590003**	12.81173	418.4348
09	5.08573	7.992402	**0.304068**	5.011239	2.560319
10	0.432841	21.90249	**9e−13**	0.308484	7.468406
11	**0.312071**	0.702193	**0.312071**	**0.312071**	0.819976
12	1.401421	4.072288	**1.383169**	1.401567	2.409392
13	0.786868	906616.7	0.997937	**0.716111**	4334.69
14	0.030862	24.76336	**8.59e−13**	0.066765	8.036433

Table 5.3: Average fitness for each algorithm and problem with dimension 12

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	2.027671	15607.16	2.110943	**2.026045**	569.9055
02	2.715694	295.517	**0.03217**	1.562613	65.5981
03	**3.777186**	5.245614	5.058997	4.337887	807.2976
04	23.54298	48843.81	**7.137143**	23.94317	1777.321
05	0.007516	187.1631	**7.38e−13**	0.006327	2073.987
06	82.86831	445255.7	**1.724116**	82.52612	505.4965
07	82.21373	156.6737	**0.528442**	82.70787	387.7884
08	69.17203	28507.12	**1.004769**	70.81044	1078.883
09	18.32003	20.72244	**0.73744**	18.10252	17.07631
10	0.390616	74.52302	**0.057542**	0.413794	28.72603
11	**0.237439**	1.458474	**0.237439**	**0.237439**	2.653205
12	1.410814	8.467387	1.43527	**1.409355**	4.396191
13	5.564458	2058324	**5.516597**	5.433279	47757.1
14	0.066187	42.94972	**9.01E−13**	0.019483	20.28755

Table 5.4: Average fitness for each algorithm and problem with dimension 16

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	**2.007063**	24388.38	2.22222	2.008302	1845.694
02	4.779463	689.847	**0.198708**	3.872578	124.2643
03	**5.631415**	7.283231	6.980571	6.049226	322.6964
04	43.47981	85029.14	**12.09548**	45.29773	3149.952
05	0.004715	214.6964	**5.5e−13**	0.004928	3530.811
06	90.76954	1.07e+17	**5.085716**	91.11857	922.8598
07	91.57444	3.31e+14	**5.248573**	90.94256	748.7884
08	152.1592	60370.24	**3.377784**	133.6855	4192.898
09	37.74819	39.81277	**0.907257**	37.46247	44.7763
10	0.33721	456.7778	**0.164283**	0.258828	75.9037
11	0.192521	3.828483	**0.190461**	**0.190461**	3.588161
12	**1.414091**	13.25389	1.476021	1.414669	8.31018
13	14.43239	3176498	**12.88087**	14.87231	172534.6
14	0.351574	59.89378	0.039723	**1.17e−12**	30.05229

Table 5.5: Average fitness for each algorithm and problem with dimension 20

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	2.001243	32930.64	2.883153	**1.999806**	3107.198
02	7.587837	1118.439	**0.492324**	7.51904	236.5383
03	**7.113975**	9.746856	9.01987	7.835969	367279.4
04	74.74562	118316.4	**13.83089**	75.53196	10691.88
05	0.00403	212.8566	**6.94e−9**	0.004408	7312.667
06	109.2742	3.49e+21	**5.42168**	107.1104	1825.048
07	108.1948	1.46e+21	**7.008227**	108.5383	1441.275
08	284.1983	94117.88	**8.325698**	207.4755	8610.55
09	63.20463	65.21878	**1.579544**	63.49481	104.6752
10	0.286777	21340.98	**0.08651**	0.490721	132.5569
11	1.151586	6.011312	**0.158707**	0.701521	3.94693
12	**1.413574**	17.73365	1.650891	1.413914	12.11648
13	29.76446	4800462	**22.13837**	29.19197	295083.2
14	0.301565	76.89122	**9.35e−9**	9.47e−9	43.24111

Table 5.6: Best fitness value for each algorithm and problem with dimension 4

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	0.013514	23.91481	**3.74e−13**	0.008449	0.259312
02	0.015	1.261467	**2.53e−13**	0.011323	0.111472
03	0.076134	0.284877	1	0.000835	**0.000125**
04	0.639886	14.33572	**2.48e−8**	1.021443	0.333206
05	0.000202	10.17526	**1.19e−13**	0.000238	0.000344
06	0.020156	48.56258	**0.020138**	0.020166	0.020288
07	0.864726	41.85269	**0.86427**	0.864893	0.864809
08	0.000199	28.30774	**4.4e−13**	0.000171	0.060556
09	0.139507	0.407813	**1.89e−8**	0.032742	0.026646
10	4.37e−13	0.387378	3.2e−13	**2.43e−13**	9.33e−13
11	0.4248	0.4248	0.4248	0.4248	**0.008097**
12	0.408424	0.553353	**5.3e−13**	0.346904	0.048358
13	3.72e−13	1160.649	3.66e−13	**2.37e−13**	8.71e−13
14	**3.13e−13**	0.456107	4.11e−13	5.64e−13	7.74e−13

Table 5.7: Best fitness value for each algorithm and problem with dimension 8

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	1.746513	1163.029	**1.642072**	1.748557	1.855896
02	0.246311	28.06783	**7.45e−13**	0.236563	1.725815
03	1.098038	2.231975	2.000208	0.186106	**0.00033**
04	4.969094	3377.885	**0.413205**	4.770179	4.972168
05	0.000722	3.844443	**1.11e−13**	0.001467	0.405516
06	63.05564	100.3864	**0.001618**	35.50098	6.966559
07	59.56525	126.8712	**0.690457**	21.63332	43.54873
08	0.262853	1190.088	**6.24e−13**	0.417598	3.024152
09	3.030025	6.545709	**9.08e−5**	3.788699	0.379582
10	**5.64e−13**	5.82194	5.89e−13	5.85e−13	0.001798
11	0.312071	0.312071	0.312071	0.312071	**0.174061**
12	1.343339	2.189377	1.068989	1.343156	**0.957273**
13	0.258026	173194.6	**8.79e−13**	0.259604	5.784104
14	6.13e−13	8.706569	4.65e−13	**4.42e−13**	8.46e−13

Table 5.8: Best fitness value for each algorithm and problem with dimension 12

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	1.985728	8431.876	**1.947732**	1.995528	2.123706
02	0.707785	103.0104	**7.09e−13**	0.635289	4.309436
03	1.251606	4.093437	3.000423	1.032287	**0.000523**
04	18.03753	15107.65	**0.131327**	9.392153	14.96514
05	0.000994	11.22811	**8.98e−14**	0.000644	0.116536
06	66.17686	103.9643	**0.008119**	66.98294	144.4244
07	71.25037	101.1358	**0.514594**	68.91963	134.912
08	1.660379	4933.791	**1.41e−8**	1.747052	28.36734
09	12.73162	17.11407	**7.52e−13**	12.71641	6.227708
10	2.12e−9	23.16159	**2.79e−12**	4.82e−10	6.273952
11	**0.237439**	**0.237439**	**0.237439**	**0.237439**	1.01749
12	1.398442	5.837022	**1.339655**	1.398353	1.398094
13	3.090889	497417.5	**1.595609**	2.831569	41.32905
14	7.66e−13	28.86674	6.82e−13	**6.26e−13**	0.096067

Table 5.9: Best fitness value for each algorithm and problem with dimension 16

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	1.992515	18145.68	**1.971647**	1.992711	3.853688
02	1.467945	309.3021	**1.29e−12**	1.339854	8.824352
03	3.604961	5.762362	6.000153	0.067755	**0.000767**
04	25.55577	57483.21	**0.150535**	28.60737	12.6763
05	0.000681	14.83673	**7.13e−14**	0.000368	10.04305
06	77.52219	150	**0.248488**	70.53718	306.507
07	74.80778	101.4781	**0.524374**	78.48955	261.161
08	3.801844	17369.55	**0.000242**	3.580692	140.3054
09	32.29294	35.66817	**9.39e−13**	30.27802	14.05743
10	**4.02e−7**	46.58691	4.24e−7	5.3e−7	2.006934
11	**0.190461**	**0.190461**	**0.190461**	**0.190461**	1.029605
12	1.410731	9.530622	**1.410497**	1.410538	4.018768
13	9.35898	1258665	**4.635214**	9.351241	273.8042
14	8.03e−13	44.99119	7.65e−13	**7.63e−13**	1.215313

Table 5.10: Best fitness value for each algorithm and problem with dimension 20

Problem	Jaya	MJAYA	EJAYA	SAMP-Jaya	OJaya
01	1.997252	23424.63	**1.996577**	1.996677	5.199616
02	2.140845	488.0374	**1.26e−6**	2.039682	23.07343
03	4.001134	8.016777	8.000024	4.280304	**0.000924**
04	47.35422	78182.15	**3.49e−7**	50.47265	50.79653
05	0.000287	24.98126	**1.21e−9**	0.000554	2.709221
06	87.26843	102.406	**0.724969**	81.69802	662.5349
07	88.5846	101.4305	**1.406232**	94.15044	448.7978
08	25.19813	39450.97	**0.074645**	13.811	444.3823
09	56.32278	60.36546	**9.02e−9**	54.59069	43.4929
10	2.63e−5	112.4798	**4.38e−6**	3.89e−5	20.92637
11	**0.158707**	0.400519	**0.158707**	**0.158707**	2.098237
12	1.413378	12.25678	**1.409187**	1.41338	3.627609
13	21.80754	2819758	**10.7863**	19.81322	1063.36
14	**7.85e−9**	65.1341	8.21e−9	8.2e−9	20.35027

5.5 Conclusion

The Jaya algorithm and some of its variants were tested against a set of difficult scalable nonlinear equation systems in order to evaluate their performance, as well as understanding if a specific Jaya variant could be better suited to this class of problems.

The Enhanced Jaya algorithm consistently performed better than the other variants, both in finding the best result and obtaining good results on average when compared to the other variants. The ability of EJAYA to solve this class of problems demonstrates that, with enough iterations, a degree of global exploration leads to better results without sacrificing local exploration.

References

[1] Bodon, E., Del Popolo, A., Lukšan, L., and Spedicato, E. Numerical performance of ABS codes for systems of nonlinear equations. Technical Report DMSIA 01/2001, Universitá degli Studi di Bergamo, Bergamo, Italy, 2001.

[2] Choi, H., Kim, S., and Shin, B.-C. Choice of an initial guess for Newton's method to solve nonlinear differential equations. *Computers & Mathematics with Applications 117* (2022), 69–73.

[3] Civicioglu, P. Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation 219*, 15 (2013), 8121–8144.

[4] Degertekin, S., Lamberti, L., and Ugur, I. Sizing, layout and topology design optimization of truss structures using the Jaya algorithm. *Applied Soft Computing 70* (2018), 903–928.

[5] Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., and Cosar, A. A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering 137* (2019), 106040.

[6] Elattar, E. E., and ElSayed, S. K. Modified JAYA algorithm for optimal power flow incorporating renewable energy sources considering the cost, emission, power loss and voltage profile improvement. *Energy 178* (2019), 598–609.

[7] Friedlander, A., Gomes-Ruggiero, M., Kozakevich, D., Martínez, J., and Santos, S. Solving nonlinear systems of equations by means of quasi-Newton methods with a nonmonotone strategy. *Optimization Methods and Software 8*, 1 (1997), 25–51.

[8] Grau-Sánchez, M. Improving order and efficiency: Composition with a modified Newton's method. *Journal of Computational and Applied Mathematics 231*, 2 (2009), 592–597.

[9] Jui, J., and Ahmad, M. A hybrid metaheuristic algorithm for identification of continuous-time Hammerstein systems. *Applied Mathematical Modelling 95* (2021), 339–360.

[10] Karr, C., Weck, B., and Freeman, L. Solutions to systems of nonlinear equations via genetic algorithms. *Engineering Applications of Artificial Intelligence 11*, 3 (1998), 369–375.

[11] Kelley, C. *Solving Nonlinear Equations with Newton's Method*. SIAM, Philadelphia, PA, 2003.

[12] Kelley, C., Qi, L., Tong, X., and Yin, H. Finding a stable solution of a system of nonlinear equations. *Journal of Industrial and Management Optimization 7*, 2 (2011), 497–521.

[13] Kumar, N. An alternative computational optimization technique to solve linear and nonlinear Diophantine equations using discrete WQPSO algorithm. *Soft Computing 26*, 22 (2022), 12531–12544.

[14] Liang, J., Qu, B., Suganthan, P., and Hernández-Díaz, A. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Technical Report 201212, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China, 2013.

[15] Lin, M.-H., Tsai, J.-F., and Yu, C.-S. A review of deterministic optimization methods in engineering and management. *Mathematical Problems in Engineering 2012* (2012), 756023.

[16] Moré, J., Garbow, B., and Hillstrom, K. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software 7*, 1 (1981), 17–41.

[17] Pan, L., Zhao, Y., and Li, L. Neighborhood-based particle swarm optimization with discrete crossover for nonlinear equation systems. *Swarm and Evolutionary Computation 69* (2022), 101019.

[18] Pérez, R., and Lopes, V. Recent applications and numerical implementation of quasi-Newton methods for solving nonlinear systems of equations. *Numerical Algorithms 35*, 2 (2004), 261–285.

[19] Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations 7* (2016), 19–34.

[20] Rao, R. *Jaya: An Advanced Optimization Algorithm and its Engineering Applications*. Springer, Cham, Switzerland, 2019.

[21] Rao, R., and Saroj, A. A self-adaptive multi-population based Jaya algorithm for engineering optimization. *Swarm and Evolutionary Computation 37* (2017), 1–26.

[22] Sihwail, R., Solaiman, O., Omar, K., Ariffin, K., Alswaitti, M., and Hashim, I. A hybrid approach for solving systems of nonlinear equations using Harris hawks optimization and Newton's method. *IEEE Access 9* (2021), 95791–95807.

[23] Suid, M., and Ahmad, M. A novel hybrid of nonlinear sine cosine algorithm and safe experimentation dynamics for model order reduction. *Automatika 64*, 1 (2023), 34–50.

[24] van Hentenryck, P., McAllester, D., and Kapur, D. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis 34*, 2 (1997), 797–827.

[25] Verma, P., and Parouha, R. Solving systems of nonlinear equations using an innovative hybrid algorithm. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering 46*, 4 (2022), 1005–1027.

[26] Yamamura, K., Kawata, H., and Tokue, A. Interval solution of nonlinear equations using linear programming. *BIT Numerical Mathematics 38*, 1 (1998), 186–199.

[27] Yu, J., Kim, C., and Rhee, S.-B. Oppositional Jaya algorithm with distance-adaptive coefficient in solving directional over current relays coordination problem. *IEEE Access 7* (2019), 150729–150742.

[28] Zhang, Y., Chi, A., and Mirjalili, S. Enhanced Jaya algorithm: A simple but efficient optimization method for constrained engineering design problems. *Knowledge-Based Systems 233* (2021), 107555.

Chapter 6

Conclusions

6.1 Main findings

The use of PSO and Jaya shows good results for solving nonlinear equation systems, but this relies heavily on both the population size and the parameters of the functions, as well as the particular characteristics of the problems.

Some proposed variants and hybridizations can achieve better results than the original algorithms. It can be observed from the experimental results that some algorithms perform consistently better than others. While this cannot be extrapolated to different types of problems, it shows that for solving nonlinear equation systems, some common strategies should be used for better results. A pattern for the most successful algorithms seems to be a good balance between local and global exploration. For particle swarm optimization, the use of less interconnected topologies, like the von Neumann topology, achieves this balance, while the EJaya algorithm provides both a local and a global search strategy.

The comparison between the original algorithms and their variants shows that the originals often perform quite well, which is an indication that some variants sacrifice exploration in favor of exploitation and are tested on simpler problems with fewer local minima.

A key parameter is the number of population, which greatly increases the quality of the results at some computational cost. Because some algorithms perform expensive computations at every iteration, it is reasonable to

attempt to solve a problem using a larger number of particles with a simpler algorithm.

A hybrid algorithm between PSO and FWA was proposed to achieve this balance between an algorithm that performs computationally inexpensive operations for most iterations while expanding the population at regular intervals, as well as finding a balance between the exploration/exploitation dichotomy.

6.2 Future work

One of the key aspects that was not explored in this thesis was a comparison using not only the results but also the time expended in finding the result, or possibly an average time per iteration of the algorithm.

The use of parallelization with GPUs, thus making the scaling of population-based solutions more efficient, is another promising topic that deserves further exploration.

Another potential approach is the use of a highly exploratory algorithm/ parameter combination, followed by an analysis of the distance between the elements of the population. These locations could be clustered to find approximations to be used as starting points in greedier or non-stochastic approaches.